

Automatische Wiederherstellung von Nachverfolgbarkeit zwischen Anforderungen und Quelltext

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Tobias Hey

aus Bergisch Gladbach

Tag der mündlichen Prüfung: 24. August 2023

1. Referent/Referentin: Prof. em. Dr. Walter F. Tichy
2. Referent/Referentin: Prof. Dr.-Ing. Anne Koziolk



Dieses Dokument ist lizenziert unter einer Creative Commons Namensnennung - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (CC BY-SA 4.0): <https://creativecommons.org/licenses/by-sa/4.0/deed.de>

Zusammenfassung

Für die effiziente Entwicklung, Wartung und Pflege von Softwaresystemen spielt ein umfassendes Verständnis der Zusammenhänge zwischen den Softwareentwicklungsartefakten eine entscheidende Rolle. Die Nachverfolgbarkeit dieser Zusammenhänge ermöglicht es beispielsweise, vergangene Entwurfsentscheidungen nachzuvollziehen oder die Auswirkungen von Änderungen zu berücksichtigen. Das manuelle Erstellen und Pflegen dieser Nachverfolgbarkeitsinformationen ist allerdings mit hohem manuellem Aufwand und damit potenziell hohen Kosten verbunden, da meist menschliche Expertise zum Verständnis der Beziehungen erforderlich ist. Dies sorgt dafür, dass in den meisten Softwareprojekten diese Informationen nicht zur Verfügung stehen. Könnten Nachverfolgbarkeitsinformationen zwischen Softwareartefakten allerdings automatisch generiert werden, könnte die Entwicklung, Wartung und Pflege einer Vielzahl von Softwaresystemen effizienter gestaltet werden. Bestehende Ansätze zur automatischen Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext sind nicht in der Lage, die semantische Lücke zwischen den Artefakten zu überbrücken. Sie erzielen zu geringe Präzisionen auf akzeptablen Ausbeuteniveaus, um in der Praxis eingesetzt werden zu können.

Das in dieser Arbeit vorgestellte Verfahren FTLR zielt durch einen semantischen Ähnlichkeitsvergleich auf eine Verbesserung der automatischen Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext ab. FTLR setzt hierzu vortrainierte fastText-Worteinbettungen zur Repräsentation der Semantik ein. Außerdem macht es sich strukturelle Informationen der Anforderungen und des Quelltextes zunutze, indem es anstatt auf Artefaktebene auf Ebene der Teile der Anforderungen und des Quelltextes abbildet. Diese Abbildung geschieht durch den Einsatz der Wortüberführungsdistanz, welche einen semantischen Ähnlichkeitsvergleich, der nicht durch Aggregation verfälscht wird, ermöglicht. Die eigentliche Bestimmung der Nachverfolgbarkeitsverbindungen erfolgt daraufhin durch einen Mehrheitsentscheid über alle feingranularen Zusammenhänge eines Artefakts, um die vorherrschenden Aspekte zu bestimmen und ggf. irrelevante Zusammenhänge zu ignorieren. In einem Experiment auf sechs Vergleichsdatensätzen konnte gezeigt werden, dass der Einsatz der Wortüberführungsdistanz gegenüber einer einfachen, aggregierten Vektorabbildung zu einer signifikanten Verbesserung der Identifikation von Nachverfolgbarkeitsverbindungen führt. Ebenso zeigte die Abbildung auf feingranularer Ebene mit anschließender Aggregation durch einen Mehrheitsentscheid signifikante Verbesserungen gegenüber der direkten Abbildung auf Artefaktebene.

Um die Präzision FTLRs weiter zu erhöhen, wird ein Ansatz zum Filtern von irrelevanten Teilen von Anforderungen eingesetzt. Dieser basiert auf einer Klassifikation der Anforderungselemente mittels eines sprachmodellbasierten Klassifikators. Entscheidend für

die Anwendung in FTLR ist dabei eine Anwendbarkeit auf ungesehene Projekte. Der vorgestellte Klassifikator NoRBERT nutzt Transferlernen, um große vortrainierte BERT-Sprachmodelle auf die Klassifikation von Anforderungen feinanzupassen. Hierdurch ist NoRBERT in der Lage, vielversprechende Ergebnisse auf ungesehenen Projekten zu erzielen. Das Verfahren war in der Lage, auf ungesehenen Projekten eine Abbildungsgüte von bis zu 89,8 % im F_1 -Maß zu erzielen. Durch das Bestimmen, ob ein Anforderungselement keine funktionalen Aspekte enthält, lassen sich irrelevante Teile der Anforderungen vor der Verarbeitung durch FTLR herausfiltern. Ein Vergleich der Leistung FTLRs mit und ohne einen derartigen Anforderungselementfilter ergab, dass ein signifikanter Leistungszuwachs im F_1 -Maß durch das Filtern erzielt werden kann. FTLR erzielt hierbei Werte im F_1 -Maß von bis zu 55,5 % und im Mittelwert der durchschnittlichen Präzision von 59,6 %.

Neben der Repräsentation der Semantik durch ausschließlich auf natürlichsprachlichem Text vortrainierten Worteinbettungen wurden außerdem bimodale Sprachmodelle für den Einsatz in FTLR untersucht. Diese auf großen dualen Korpora, bestehend aus Quelltextmethoden und ihrer natürlichsprachlichen Dokumentation, vortrainierten Sprachmodelle erzielen in verwandten Aufgabenstellungen aus der Softwaretechnik, wie Quelltextsuche oder Fehlerlokalisierung, vielversprechende Ergebnisse. Um die Eignung für die automatische Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext zu untersuchen, wurden zwei Integrationsmöglichkeiten des bimodalen Sprachmodells UniXcoder in FTLR entwickelt. In einem Vergleich auf fünf Datensätzen zur Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext konnte kein Leistungszuwachs durch den Einsatz dieser Art von Modellen gegenüber den leichtgewichtigeren Worteinbettungen festgestellt werden.

Abschließend wurde die Leistung FTLRs in Bezug zu bestehenden Ansätzen zur unüberwachten automatischen Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext gesetzt. Hierbei zeigt sich, dass FTLR auf Projekten, die ausschließlich objektorientierten Quelltext enthalten, eine höhere durchschnittliche Präzision und ein höheres F_1 -Maß als bestehende Verfahren erzielt. Allerdings verdeutlichen die Ergebnisse auch, dass, insbesondere auf großen Projekten, alle bestehenden Ansätze und auch FTLR noch weit von einer Abbildungsgüte entfernt sind, die es für eine vollständige Automatisierung der Wiederherstellung von Nachverfolgbarkeitsverbindungen in der Praxis benötigt.

Abstract

For the efficient development, maintenance and management of software systems, a comprehensive understanding of the relationships between software artifacts plays a crucial role. The traceability of these relationships makes it possible, for example, to comprehend past design decisions or to assess the impacts of changes. However, manually creating and maintaining traceability information entails high manual effort and therefore potentially high costs, as human expertise is usually required to understand the relationships. This is why this information is not available in most software projects. However, if traceability information between software artifacts could be generated automatically, the development, maintenance and management of a wide range of software systems could be made more efficient. Existing approaches to automatically recover trace links between requirements and source code are not able to bridge the semantic gap between artifacts. They achieve too low precision at acceptable recall levels to be used in practice.

The FTLR approach presented in this dissertation aims at improving the automatic traceability link recovery between requirements and source text by performing a semantic similarity comparison. Therefore, FTLR uses pre-trained fastText word embeddings to represent semantics. It leverages structural information of the requirements and source code by using mapping on the level of the components of the requirements and source code instead of on the artifact level. This mapping uses the Word Movers Distance, which provides a semantic similarity comparison that is not skewed by aggregation. The actual identification of trace links is achieved by a majority vote on all fine-grained links of an artifact to determine the most prevalent aspects and ignore irrelevant relationships. In an experiment on six benchmark datasets, the use of the Word Movers Distance showed a significant improvement of the identification of traceability connections over basic aggregated vector mappings. Similarly, the fine-grained mapping followed by an aggregation with majority vote showed significant improvements over a direct artifact level mapping.

To further increase FTLR's precision, an approach for filtering irrelevant parts of requirements is applied. The approach is based on a classification of the requirement elements using a language model-based classifier. Crucial for the application in FTLR is an applicability to unseen projects. The presented classifier NoRBERT uses transfer learning to fine-tune large pre-trained BERT language models to the classification of requirements. Hereby, NoRBERT is able to achieve promising results on unseen projects. The approach was able to achieve a mapping quality of up to 89.8 % in F_1 -score on unseen projects. Determining whether a requirement element contains no functional aspects allows irrelevant parts of the requirements to be filtered out before processing by FTLR. A comparison of FTLR's performance with and without such a requirement element filter showed that a

significant performance increase in F_1 -score can be achieved by filtering. FTLR achieves F_1 -scores of up to 55.5 % and results in mean average precision of up to 59.6 %.

In addition to representing semantics through word embeddings that are pre-trained exclusively on natural language text, this work also investigates bimodal language models for the use in FTLR. These language models are pre-trained on large dual corpora, consisting of source code methods and their natural language documentation, and achieve promising results in related software engineering tasks, such as code search or bug localization. To investigate the applicability of these models for the automatic recovery of trace links between requirements and source code, two options for integrating the bimodal language model UniXcoder in FTLR were developed. In a comparison on five datasets for recovering traceability links between requirements and source code, this type of model showed no increase in performance over the more lightweight word embeddings.

Finally, the performance of FTLR was compared to the performance of existing approaches for unsupervised automatic traceability link recovery between requirements and source code. FTLR achieves higher mean average precision and F_1 -scores than existing approaches on projects that only contain object-oriented source code. However, the results also illustrate that, especially on large projects, all existing approaches including FTLR are still far from achieving the quality that is needed to fully automate traceability link recovery in practice.

Danksagung

Geschätzter Leser, es wird Sie wohl kaum wundern, wenn ich Ihnen sage, dass der Abschluss einer Promotion kaum ohne die Unterstützung anderer Menschen möglich ist. Daher möchte ich in diesem Kapitel meine Dankbarkeit gegenüber all den Menschen zum Ausdruck bringen, die einen entscheidenden Beitrag zum Gelingen dieser Promotion geleistet haben.

Zuallererst möchte ich hierbei Walter meinen tiefsten Dank aussprechen. Er ermöglichte es mir, mein Interesse an der Sprachverarbeitung mit meiner Leidenschaft für die Softwaretechnik im Rahmen meiner Promotion zu verbinden. Dabei gewährte er mir eine bemerkenswerte Freiheit in der inhaltlichen Gestaltung, sei es während der verschiedenen Themenwechsel in den ersten beiden Jahren oder bei der Entwicklung des Weges zur Erreichung des angestrebten Ziels. Zugleich hatte er dabei aber ebenso immer ein offenes Ohr für aufkommende Fragen oder war bereit, Diskussionen über die Sinnhaftigkeit bestimmter Vorgehensweisen zu führen. Vor allem jedoch ist Walter für mich bis heute eine unerschöpfliche Quelle der Inspiration. Seine Fähigkeit, zukünftige Trends in der Softwaretechnik vorausszusehen und in eine Vision zu gießen, hat mich von Anfang an fasziniert. Darüber hinaus erwies sich Walter als das ideale Vorbild für das Erlernen guter wissenschaftlicher Arbeitsweisen, die mich zu dem Wissenschaftler geformt haben, der ich heute bin.

Ein großer Dank gebührt auch meinen geschätzten Kollegen am Lehrstuhl Tichy. Unser bunter Haufen von doch teils sehr unterschiedlichen Charakteren mit ebenso unterschiedlichen Forschungsrichtungen und Denkweisen half mir immer wieder über den Tellerrand hinauszuschauen und Dinge kritisch zu hinterfragen, welches eine so entscheidende Fähigkeit für einen Wissenschaftler darstellt. Aber auch abseits der Forschung war die gemeinsame Zeit mit euch sehr bereichernd, besonders in kulinarischer Hinsicht. Hervorheben möchte ich hierbei vor allem Mathias, Martin und Sebastian. Ohne Mathias wäre ich wohl nie am IPD und vor allem nicht in der Sprachverarbeitung gelandet. Martin schaffte es uns trotz der teils hohen Arbeitsbelastung in der Lehre doch immer wieder ein Schmunzeln auf die Lippen zu zaubern und unsere Moral hochzuhalten. Der wohl größte Beitrag zum Erfolg meiner Promotion ist allerdings Sebastian zuzuschreiben. Erst ertrug er meine nicht zu stoppenden Ideen und Verbesserungsvorschläge während der Betreuung meiner Masterarbeit. Danach war er die beste Person, die ich mir hätte vorstellen können, um mich an das wissenschaftliche Arbeiten heranzuführen und den Einstieg in die Promotion zu erleichtern. In den vier gemeinsamen Jahren als Büro-Gegenüber waren wir als viermaliges SWT1-Übungsleiterduo, mit 12 gemeinsamen Publikationen und beständigen NLP- und Gott und die Welt-Diskussionen wohl durchaus irgendwie

ein kongeniales Duo (um es in seinen Worten auszudrücken). Diese Unterstützung und Freundschaft haben für mich diese akademische Reise zu einem unvergesslichen Erlebnis werden lassen. Gemeinsam haben wir Höhen und Tiefen gemeistert und ich bin dankbar, dass ich Teil dieser inspirierenden Gemeinschaft sein durfte.

Neben den Kollegen am Lehrstuhl Tichy möchte ich mich aber auch ganz herzlich bei Anne und Ralph bedanken, die mir nach Walters Emeritierung ein neues Zuhause am SDQ geboten haben. Trotz des Wechsels mitten in Corona fühlte ich mich bei euch direkt herzlichst aufgenommen. Auch die anderen Kollegen machten mir den Übergang leicht. Zuallererst ist hier sicherlich Jan zu nennen, welcher zuerst als exzellenter Abschlussarbeiter, dann als mitleidender NLP- und Rückverfolgbarkeitsenthusiast den perfekten inhaltlichen Sparring-Partner für Ideen und Konzepte, seien sie für Veröffentlichungen oder die Dissertation selbst, darstellte. Christopher möchte ich für den erfrischenden neuen Wind danken, den er in die letzten beiden Jahre als SWT1-Übungsleiter eingebracht hat. Auch die anderen Kaffeeliebhaber (Heiko, Frederik, Timur, Nicholas, ...) und die unzähligen mehr oder minder fachlichen Diskussionen an der Kaffeemaschine haben diese Dissertation bereichert und meinen (akademischen) Horizont erweitert. Sebastian ließ sich dabei sogar davon überzeugen, mit mir Sport zu treiben und sich gemeinsam beim Bouldern die Finger kaputtzumachen (und zur Belohnung natürlich die ein oder andere Weinschorle zu genießen). Mit Sandro hatte ich außerdem für die letzten Monate des Aufschreibens einen passenden Leidensgenossen im Büro, welcher diese doch durchaus schwierige Zeit erträglicher gemacht hat.

Ohne all die Studierenden, die mich bei der Verwirklichung meiner Ideen unterstützt haben, hätte die Fertigstellung dieser Arbeit allerdings wohl sicher noch um einiges länger gebraucht. Mein besonderer Dank geht daher auch an Fei, Thomas, Daniel und Tim, deren Abschlussarbeiten und anschließende Hiwi-Tätigkeiten entscheidende Vorarbeiten für diese Dissertation geleistet haben.

Nicht zuletzt möchte ich mich von Herzen bei meiner Familie und insbesondere meiner Frau Marianne bedanken. Ihr unerschütterlicher Glaube an mich, ihre Geduld und Liebe waren meine ständige Inspiration und Motivation. Sie hat mich ermutigt, wenn ich Zweifel hatte und mir den Rückhalt gegeben, den ich gebraucht habe, um diesen teils doch nicht immer einfachen Weg zu gehen. Sie sorgte aber auch immer wieder dafür, dass ich neben der Arbeit auch genügend Zeit für eine erfüllende Freizeitgestaltung fand, welches zweifellos sowohl meinem Körper als auch meinem Geist zugutekam. Diese Dissertation ist nicht nur mein Erfolg, sondern auch ihrer.

Diese Dissertation ist das Ergebnis der kollektiven Anstrengungen und Unterstützung dieser großartigen Menschen. Ihre Beiträge, sei es in Form von Fachwissen, moralischer Unterstützung oder einfach durch das Teilen meiner Begeisterung für die Forschung, haben diesen Meilenstein in meinem Leben erst möglich gemacht.

Inhaltsverzeichnis

Zusammenfassung	i
Abstract	iii
Danksagung	v
Abbildungsverzeichnis	xv
Tabellenverzeichnis	xvii
Quelltextausschnittsverzeichnis	xxi
Abkürzungsverzeichnis	xxiii
I. Prolog	1
1. Einleitung	3
2. Grundlagen	13
3. Verwandte Arbeiten	65
II. Automatische Nachverfolgbarkeit von Anforderungen und Quelltext	99
4. Lösungsorientierte Analyse	101
5. Feingranulare Wiederherstellung von Nachverfolgbarkeitsverbindungen	119
6. Evaluation des grundlegenden Verfahrens	139
7. Zusammenfassung	161
III. Anforderungsklassifikation als Vorfilter	163
8. Anforderungsklassifikation zur Filterung von Eingaben in FTLR	165
9. Transferlernen für Anforderungsklassifikation	175
10. Evaluation der Anforderungselementfilterung	203
11. Zusammenfassung	217
IV. Einsatz bimodaler Sprachmodelle in FTLR	219
12. Bimodale, kontextsensitive Sprachmodelle als Repräsentationsbasis	221
13. Evaluation mit bimodalen Sprachmodellen	227
14. Zusammenfassung	237

V. Epilog	239
15. Vergleich mit bestehenden Ansätzen	241
16. Zusammenfassung und Ausblick	255
Literaturverzeichnis	265
Anhang	286
A. Etikettensätze	289
B. Stoppwortlisten	293
C. Javadoc-Etiketten	297
D. Weitere Informationen zu den Datensätzen	301
E. Weitere Evaluationsergebnisse	305

Inhaltsverzeichnis

Zusammenfassung	i
Abstract	iii
Danksagung	v
Abbildungsverzeichnis	xv
Tabellenverzeichnis	xvii
Quelltextausschnittsverzeichnis	xxi
Abkürzungsverzeichnis	xxiii
I. Prolog	1
1. Einleitung	3
1.1. Zielstellung und Einschränkungen	5
1.2. Wiederherstellung von Nachverfolgbarkeitsverbindungen	7
1.3. Thesen	10
1.4. Aufbau der Arbeit	10
2. Grundlagen	13
2.1. Nachverfolgbarkeit von Softwareentwicklungsartefakten	13
2.1.1. Nachverfolgbares Artefakt	14
2.1.2. Nachverfolgbarkeitsverbindung	15
2.1.3. Nachverfolgbarkeit	16
2.1.4. Formen des Nachverfolgens	18
2.2. Anforderungen	19
2.3. Quelltextinformationen	20
2.3.1. Bezeichner	21
2.3.2. Quelltextkommentare	21
2.3.3. Aufrufgraph	23
2.4. Informationsrückgewinnung	23
2.5. Maschinelles Lernen	24
2.5.1. Überwachtes maschinelles Lernen	25
2.5.2. Künstliche neuronale Netze	27

2.6.	Sprachwissenschaftliche Grundbegriffe	31
2.6.1.	Natürliche Sprache	31
2.6.2.	Syntax	31
2.6.3.	Semantik	31
2.6.4.	Pragmatik	32
2.6.5.	Lexikalische Semantik	32
2.7.	Wissensquellen	33
2.7.1.	Ontologien	33
2.7.2.	WordNet	34
2.7.3.	Wikipedia und DBpedia	35
2.8.	Sprachverarbeitung	36
2.8.1.	Portionierung	36
2.8.2.	Satzerkennung	36
2.8.3.	Wortarterkennung	37
2.8.4.	Wortgrundformbestimmung	38
2.8.5.	Stoppwortentfernung	38
2.8.6.	Syntaktisches Zerteilen	39
2.8.7.	Erkennung semantischer Rollen	40
2.8.8.	Disambiguierung von Wortbedeutungen	42
2.8.9.	Themen	44
2.9.	Wortrepräsentationen und Sprachmodelle	46
2.9.1.	Vektorrepräsentationen	46
2.9.2.	Wort- und Dokumenteinbettungen	48
2.9.3.	(Neuronale) Sprachmodelle	51
2.10.	Wort- und Dokumentähnlichkeit	55
2.11.	Evaluationsmetriken	58
2.12.	Hypothesentests	62
3.	Verwandte Arbeiten	65
3.1.	Nachverfolgbarkeit von Anforderungen zu Quelltext	65
3.1.1.	Verfahren der Informationsrückgewinnung	65
3.1.2.	Auffassung als Optimierungsproblem	74
3.1.3.	Worteinbettungen und maschinelles Lernen	75
3.2.	Nachverfolgbarkeit von Anforderungen zu Anforderungen	78
3.3.	Nachverfolgbarkeit von Testfällen zu Quelltext	82
3.4.	Nachverfolgbarkeit von Belangen zu Versionskontrollsystem-Einbuchungen	85
3.5.	Andere Formen von Nachverfolgbarkeit	88
3.6.	Konzeptlokalisierung in Quelltext	90
3.7.	Quelltextsuche	94
II.	Automatische Nachverfolgbarkeit von Anforderungen und Quelltext	99
4.	Lösungsorientierte Analyse	101
4.1.	Nachverfolgbarkeit als Problem der Informationsrückgewinnung	102


4.2.	Semantische Lücke	103
4.3.	Semantik von Anforderungen	105
4.4.	Semantik von Quelltext	107
4.5.	Repräsentation der Semantik	111
4.6.	Semantische Ähnlichkeitsvergleiche	114
4.7.	Ebene der Abbildung	116
5.	Feingranulare Wiederherstellung von Nachverfolgbarkeitsverbindungen	119
5.1.	Repräsentation der Artefakte	121
5.1.1.	Anforderungsrepräsentation	124
5.1.2.	Quelltextrepräsentation	128
5.2.	Ähnlichkeitsbestimmung	130
5.3.	Aggregation	132
5.4.	Varianten	135
5.4.1.	Methodenkommentare	135
5.4.2.	Filterung von Vorlagenelementen	136
5.4.3.	Ausnutzen von Aufrufabhängigkeiten	137
6.	Evaluation des grundlegenden Verfahrens	139
6.1.	Datensätze	141
6.2.	Methodik	144
6.3.	Einfluss der Varianten	145
6.4.	Einfluss der Schwellenwerte	149
6.5.	Vergleich mit Referenzimplementierungen	156
6.6.	Gefährdungen der Gültigkeit	159
7.	Zusammenfassung	161
III.	Anforderungsklassifikation als Vorfilter	163
8.	Anforderungsklassifikation zur Filterung von Eingaben in FTLR	165
8.1.	Informationen aus der Anforderungsklassifikation	167
8.2.	Integration in FTLR	172
9.	Transferlernen für Anforderungsklassifikation	175
9.1.	Bestehende Ansätze zur Anforderungsklassifikation	176
9.2.	NoRBERT	179
9.3.	Ergebnisse auf dem PROMISE NFR-Datensatz	181
9.3.1.	Identifikation funktionaler und nichtfunktionaler Anforderungen	184
9.3.2.	Identifikation von funktionalen und Qualitätsaspekten	186
9.3.3.	Identifikation von Anliegen in funktionalen Anforderungen	187
9.3.4.	Auswahl geeigneter NoRBERT-Konfigurationen	189
9.4.	Klassifikationsergebnisse auf Nachverfolgbarkeitsdatensätzen	190
9.4.1.	Identifikation der Klassen für die Anforderungselementfilter	193

9.4.2.	Identifikation der relevanten Anliegen in funktionalen Anforderungen	196
9.4.3.	Zusammenfassung	199
9.5.	Gefährdungen der Gültigkeit	200
10.	Evaluation der Anforderungselementfilterung	203
10.1.	Vergleich mit ungefilterten Varianten	204
10.2.	Vergleich mit Filterung von Vorlagenelementen	211
10.3.	Kombination der Filtervarianten	213
11.	Zusammenfassung	217
IV.	Einsatz bimodaler Sprachmodelle in FTLR	219
12.	Bimodale, kontextsensitive Sprachmodelle als Repräsentationsbasis	221
12.1.	Integration in FTLR	223
12.2.	Wahl geeigneter Sprachmodelle	224
13.	Evaluation mit bimodalen Sprachmodellen	227
13.1.	Auswirkung der Übersetzung	228
13.2.	Vergleich der Einbindungsvarianten	229
13.3.	Vergleich von FTLR mit fastText und UniXcoder	231
13.4.	Gefährdungen der Gültigkeit	234
14.	Zusammenfassung	237
V.	Epilog	239
15.	Vergleich mit bestehenden Ansätzen	241
15.1.	Auswahl der Vergleichdatensätze und Verfahren	241
15.2.	Unüberwachte Wiederherstellung von Nachverfolgbarkeitsverbindungen	244
15.3.	Überwachte Wiederherstellung von Nachverfolgbarkeitsverbindungen	249
15.4.	Gefährdungen der Gültigkeit	251
15.5.	Zusammenfassung des Vergleichs	252
16.	Zusammenfassung und Ausblick	255
16.1.	Diskussion der Thesen	258
16.2.	Weiterführende Themen und Ausblick	260
16.2.1.	Optimierungen und Erweiterungen	260
16.2.2.	Möglichkeiten zur Weiterentwicklung	262
16.3.	Abschließende Bemerkungen	263
Literaturverzeichnis		265

Anhang	286
A. Etikettensätze	289
A.1. Wortarten	289
A.2. Syntaktisches Zerteilen	290
A.3. Semantische Rollen	291
B. Stoppwortlisten	293
B.1. Stoppwörter für natürliche Sprache	293
B.2. Quelltextstoppwörter	295
C. Javadoc-Etiketten	297
D. Weitere Informationen zu den Datensätzen	301
E. Weitere Evaluationsergebnisse	305
E.1. Vergleich mit Referenzimplementierungen	306
E.2. Evaluationsergebnisse mit Anforderungsfiltern	309
E.3. Auswirkungen der Übersetzung auf FTLR	313
E.4. Evaluationsergebnisse mit UniXcoder	314

Abbildungsverzeichnis

1.1.	Beispiel feingranularer Nachverfolgbarkeitsbeziehungen zwischen einer Anwendungsfallbeschreibung und verschiedenen Quelltextelementen	4
1.2.	Überblick der Teilschritte FTLRs zur automatischen Identifikation von Nachverfolgbarkeitsverbindung	7
2.1.	Schema einer Nachverfolgbarkeitsmatrix	15
2.2.	Aufrufgraph eines Beispielprogramms	23
2.3.	Schematischer Aufbau verschiedener Topologien für künstliche neuronale Netze	28
2.4.	Portionierung eines Beispielsatzes	36
2.5.	Wortartetiketten für einen Beispielsatz	37
2.6.	Syntaxbaum für einen Beispielsatz	39
2.7.	Abhängigkeitsgraph für einen Beispielsatz	40
2.8.	Semantische Rollen für einen Beispielsatz	41
2.9.	Disambiguierung der Nomen in einem Beispielsatz	44
2.10.	Verschiedene Arten von Vektorrepräsentationen eines Wortes in einer Texteingabe	46
2.11.	Schematische Repräsentation von semantischen Zusammenhängen zwischen Worteinbettungen	50
2.12.	Schematische Darstellung eines <i>BERT</i> -basierten Klassifikators unter Nutzung der Repräsentation des [CLS]-Tokens	54
2.13.	Schematische Darstellung der Wortüberführungsdistanz für ein Dokumentpaar	57
2.14.	Beispiele der Klassifikationsergebnisse für Nachverfolgbarkeitsverbindungen	59
3.1.	Beispiel eines Fehlerberichts aus dem Eclipse SWT-Projekt	90
3.2.	Beispiel eines Elements aus dem CodeSearchNet-Datensatz	94
4.1.	Beispiel einer Nachverfolgbarkeitsverbindung, deren Zusammenhang nur auf Basis der Semantik erkennbar ist	102
4.2.	Beispiel einer Anwendungsfallbeschreibung aus dem eTour-Projekt	106
5.1.	Überblick über FTLRs Teilschritte zur automatischen Identifikation von Nachverfolgbarkeitsverbindungen	120
5.2.	Überblick der Teilschritte zur Repräsentation der Artefaktelemente in FTLR durch Einbettungs-Multimengen	125
5.3.	Portionierung einer Anwendungsfallbeschreibung mit Schablone	126
5.4.	Portionierung einer Anforderung in Anforderungselemente	126
5.5.	Illustration der Vorverarbeitungsschritte für Anforderungselemente	127
5.6.	Illustration der Vorverarbeitungsschritte für Quelltextelemente	129

5.7.	Gegenüberstellung der verschiedenen Teilschritte zur Ähnlichkeitsbestimmung	131
5.8.	Schematisches Beispiel des Aggregationsschrittes von FTLR	133
5.9.	Illustration der Extraktion eines Quelltextelements mit Kommentar	135
5.10.	Beispiel für das Filtern von Vorlagenelementen in FTLR	137
6.1.	Varianz der optimierten Mehrheitsschwellenwerte	153
6.2.	Varianz der optimierten Abschlusschwellenwerte	153
8.1.	Illustration der Unterklassifizierungen von funktionalen Anforderungen aus bestehenden Arbeiten	169
8.2.	Ablauf der Integration der Anforderungselementklassifikation in FTLR	172
8.3.	Darstellung der in FTLR integrierten Anforderungselementfilter	174
9.1.	Darstellung des Einsatzes des NoRBERT -Verfahrens auf einer Beispieleingabe	180
12.1.	Darstellung der verschiedenen Repräsentationsmöglichkeiten auf Basis eines bimodalen Transformer-basierten Sprachmodells	222
13.1.	Varianz der optimierten Mehrheitsschwellenwerte mit UniXcoder	233
13.2.	Varianz der optimierten Abschlusschwellenwerte mit UniXcoder	234
15.1.	Präzision-Ausbeute-Kurven von FTLR- im Vergleich mit COMET_{MAP}	249

Tabellenverzeichnis

2.1.	Auszug möglicher Etiketten in einem Javadoc-Kommentar	22
2.2.	Auszug aus dem Etikettensatz der Penn Treebank für Wortarten	37
2.3.	Auszug des Etikettensatzes der Penn Treebank für syntaktische Kategorien .	39
2.4.	Auszug aus den Abhängigkeiten der Universal Dependencies und ihre jeweilige Bedeutung	40
2.5.	Auszug der Etiketten für semantische Rollen	41
2.6.	Mögliche Ergebnisse einer binären Klassifikation beim Vergleich mit einem Goldstandard	58
2.7.	Beispielhafte Berechnung der durchschnittlichen Präzision für eine Anfrage eines Quellartefakts	61
2.8.	Kritische Werte für die Teststatistik des Wilcoxon-Vorzeichen-Rang-Tests . .	62
3.1.	Übersicht über die am häufigsten verwendeten Datensätze zur TLR zwischen Anforderungen und Quelltext	66
3.2.	Übersicht über die am häufigsten verwendeten Datensätze zur TLR zwischen Anforderungen und Anforderungen	79
5.1.	Übersicht der konfigurierbaren Ähnlichkeitsbestimmungsverfahren	132
6.1.	Übersicht der verwendeten Datensätze	140
6.2.	Weitere Details zu den verwendeten Datensätzen	140
6.3.	Vergleich des Einflusses der Merkmalsvarianten auf das FTLR -Verfahren . . .	146
6.4.	Durchschnittliches F_1 -Maß und MAP der Varianten ohne Albergate	147
6.5.	Vergleich des Einflusses der Merkmalsvarianten auf das FTLR -Verfahren mit optimierten Schwellenwerten	148
6.6.	Vergleich der festgelegten mit der optimierten Schwellenwertkombination für FTLR- 	150
6.7.	Vergleich der festgelegten mit der optimierten Schwellenwertkombination für FTLR-	151
6.8.	Vergleich der festgelegten mit der optimierten Schwellenwertkombination für FTLR-	152
6.9.	Vergleich der Ergebnisse der Schwellenwertkombinationen	155
6.10.	Vergleich der Ergebnisse der Schwellenwertkombinationen auf den iTrust- Varianten	156
6.11.	Vergleich der Referenzimplementierungen mit FTLR	157
9.1.	Verteilung der Klassen im originalen PROMISE NFR-Datensatz	182
9.2.	Überblick über den umetikettierten PROMISE NFR-Datensatz	183

9.3.	Überblick über die Klassenverteilung der funktionalen Aspekte im umetikettierten PROMISE NFR-Datensatz	183
9.4.	Ergebnisse der Klassifikation in funktionale (F) oder nichtfunktionale (NF) Anforderungen auf dem PROMISE NFR-Datensatz	185
9.5.	Ergebnisse der binären Klassifikation der Klassen des umetikettierten PROMISE NFR-Datensatzes	187
9.6.	Binäre Klassifikation der funktionalen Aspekte im umetikettierten PROMISE NFR-Datensatz durch NoRBERT	188
9.7.	Übersicht der Klassenverteilung auf den Nachverfolgbarkeitsdatensätzen . .	190
9.8.	Ergebnisse bezüglich der Genauigkeit verschiedener NoRBERT -Konfigurationen auf den Klassen funktionale Aspekte (F) und Nutzerbezogenes (NB)	193
9.9.	Ergebnisse, bezüglich des F_1 -Maßes des Auftretens einer Klasse, verschiedener NoRBERT -Konfigurationen auf den beiden Klassen funktionale Aspekte (F) und Nutzerbezogenes (NB)	195
9.10.	Ergebnisse bezüglich der Genauigkeit verschiedener NoRBERT -Konfigurationen auf den Klassen Funktion, Verhalten und Daten	197
9.11.	Ergebnisse, bezüglich des F_1 -Maßes des Auftretens einer Klasse, verschiedener NoRBERT -Konfigurationen auf den Klassen Funktion, Verhalten und Daten .	198
9.12.	Beste NoRBERT -Konfigurationen auf den Nachverfolgbarkeitsdatensätzen . .	199
10.1.	Vergleich der Anforderungsfiltervarianten mit den ungefilterten FTLR -Ergebnissen	205
10.2.	Vergleich der Ergebnisse der unterschiedlichen Schwellenwertkombinationen auf den Anforderungsfiltervarianten	206
10.3.	Vergleich der Anforderungsfiltervarianten auf Basis des Goldstandards mit den ungefilterten FTLR -Ergebnissen	208
10.4.	Vergleich der Ergebnisse der unterschiedlichen Schwellenwertkombinationen auf den Anforderungsfiltervarianten auf Basis des Goldstandards	210
10.5.	Vergleich der Ergebnisse der Anforderungsfiltervarianten mit FTLR mit Anwendungsfallvorlagenfilter auf den unterschiedlichen Schwellenwertkombinationen	212
10.6.	Vergleich der Ergebnisse FTLRs mit dem NF-Filter in Kombination mit dem Anwendungsfallvorlagenfilter auf den unterschiedlichen Schwellenwertkombinationen	214
13.1.	Vergleich der Auswirkung einer automatischen Übersetzung auf die Leistung FTLRs	228
13.2.	Vergleich von FTLRs Leistung mit den UniXcoder-Einbindungsvarianten . .	230
13.3.	Vergleich von FTLRs Leistung mit unterschiedlichen Sprachmodellen	232
15.1.	Übersicht der verwendeten Vergleichsdatensätze bestehender Arbeiten . . .	242
15.2.	Vergleich von FTLRs Leistung mit bestehenden Ansätzen zur unüberwachten TLR zwischen Anforderungen und Quelltext	245
15.3.	Vergleich des durchschnittlichen F_1 -Maßes bzw. MAP der Ansätze	247

15.4. Vergleich von FTLRs Leistung mit bestehenden Ansätzen zur überwachten TLR zwischen Anforderungen und Quelltext	250
E.1. Vergleich der Ergebnisse der Merkmalsvarianten mit angepassten festen Schwellenwerten	305
E.2. Vergleich der ECosS-Referenzimplementierung mit FTLR	306
E.3. Vergleich der ACosS-Referenzimplementierung mit FTLR	307
E.4. Vergleich der AWMD-Referenzimplementierung mit FTLR	308
E.5. Vergleich der Anforderungsfiltervarianten mit den ungefilterten FTLR -Ergebnissen bei optimierten Schwellenwerten	309
E.6. Vergleich der Anforderungsfiltervarianten auf Basis des Goldstandards mit den ungefilterten FTLR -Ergebnissen bei optimierten Schwellenwerten	310
E.7. Vergleich der Ergebnisse der Anforderungsfiltervarianten mit FTLR mit Anwendungsfallvorlagenfilter auf optimierten Schwellenwerten	311
E.8. Vergleich der Ergebnisse der Anforderungsfiltervarianten auf Basis des Goldstandards mit FTLR mit Anwendungsfallvorlagenfilter auf optimierten Schwellenwerten	312
E.9. Vergleich der Auswirkung einer automatischen Übersetzung auf die Leistung FTLRs	313
E.10. Vergleich von FTLRs Leistung mit den UniXcoder-Einbindungsvarianten	314

Quelltextausschnittsverzeichnis

2.1. Beispiel eines Javadoc-Kommentars einer Methode	22
4.1. Beispiel einer Methode mit Kommentar und Rumpf	109

Abkürzungsverzeichnis

ACoS	Anwendung der Kosinusähnlichkeit auf Artefaktebene	158
ANN	Künstliches neuronales Netz (engl.: <i>artificial neural network</i>)	27
AP	Durchschnittliche Präzision (engl.: <i>average precision</i>)	61
AST	Abstrakter Syntaxbaum (engl.: <i>abstract syntax tree</i>)	83
AWMD	Anwendung der Wortüberführungsdistanz auf Artefaktebene	157
BERT	Bidirectional Encoder Representations from Transformers	53
BoE	Einbettungs-Multimenge (engl.: <i>bag-of-embeddings</i>)	51
BoW	Wort-Multimenge (engl.: <i>bag-of-words</i>)	47
CBOW	Kontinuierliche Wort-Multimenge (engl.: <i>continuous bag-of-words</i>)	49
CNN	Faltendes neuronales Netz (engl.: <i>convolutional neural network</i>)	92
CoEST	Center of Excellence for Software & Systems Traceability	134
CSG	Kontinuierliches Skip-Gramm (engl.: <i>continuous skip-gram</i>)	49
DAO	Datenzugriffsobjekt (engl.: <i>data access object</i>)	143
ECoS	Anwendung der Kosinusähnlichkeit auf Elementebene	158
ELMo	Embeddings from Language Models	52
ES	Frühes Anhalten (engl.: <i>early stopping</i>)	184
FFNN	Vorwärts gerichtetes neuronales Netz (engl.: <i>feed-forward neural network</i>)	28
ft	FastText	232
FTLR	Fine-grained Traceability Link Recovery	119
GAT	Graph-Aufmerksamkeitsnetz (engl.: <i>graph attention network</i>)	178
GBO	Grafische Benutzeroberfläche	143
GRU	Vergatterte rekurrente Einheit (engl.: <i>gated recurrent unit</i>)	80
HBN	Hierarchisches Bayes'sches Netz (engl.: <i>hierarchical Bayesian network</i>)	77
IR	Informationsrückgewinnung (engl.: <i>information retrieval</i>)	23

JS	Probabilistisches Jensen-Shannon-Modell (engl.: <i>probabilistic Jensen and Shannon model</i>)	70
JSP	Jakarta Server Pages	141
LDA	Latente Dirichlet Allokation (engl.: <i>latent dirichlet allocation</i>)	45
LG	Logistische Regression (engl.: <i>logistic regression</i>)	26
LSI	Latente Semantische Indizierung (engl.: <i>latent semantic indexing</i>)	45
LSTM	Langes Kurzzeitgedächtnis (engl.: <i>long short-term memory</i>)	29
MAP	Mittelwert der durchschnittlichen Präzision (engl.: <i>mean average precision</i>)	61
ML	Maschinelles Lernen (engl.: <i>machine learning</i>)	24
MRR	Mittlerer Kehrwerttrung (engl.: <i>mean reciprocal rank</i>)	92
NB	Nutzerbezogenes (engl.: <i>user-related</i>)	170
NBC	Naïve-Bayes Klassifikator (engl.: <i>Naïve Bayes classifier</i>)	26
NLP	Verarbeitung natürlicher Sprache (engl.: <i>natural language processing</i>)	36
NLTK	Natural Language Toolkit	38
NoRBERT	Non-functional and functional Requirements classification using BERT	179
OOV	Wörter außerhalb des Vokabulars (engl.: <i>out-of-vocabulary words</i>)	50
OS	Überabtasten (engl.: <i>oversampling</i>)	77
OSS	<i>Open Source Software</i>	69
PCA	Hauptkomponentenanalyse (engl.: <i>principal component analysis</i>)	72
pLSI	Probabilistische LSI (engl.: <i>latent semantic indexing</i>)	45
PoS tag	Wortartetikette (engl.: <i>part-of-speech tag</i>)	37
PoS tagging	Wortarterkennung (engl.: <i>part-of-speech tagging</i>)	37
PV-DBOW	Verteilte Wort-Multimengen-Version von Absatzvektoren (engl.: <i>Distributed Bag of Words version of Paragraph Vector</i>)	51
PV-DM	Verteiltes Absatzvektoren-Speichermodell (engl.: <i>Distributed Memory Model of Paragraph Vectors</i>)	51
RE	Anforderungstechnik (engl.: <i>requirements engineering</i>)	166
RF	Zufallswald (engl.: <i>random forest</i>)	77
RNN	Rekurrentes neuronales Netz (engl.: <i>recurrent neural network</i>)	29
RTM	Relationales Themenmodell (engl.: <i>relational topic model</i>)	70

SLOC	Quelltextzeilen (engl.: <i>source lines of code</i>)	143
SRL	Erkennung semantischer Rollen (engl.: <i>semantic role labeling</i>)	42
SVM	Stützvektormachine (engl.: <i>support vector machine</i>)	26
TF-IDF	Vorkommenshäufigkeit-Inverse Dokumenthäufigkeit (engl.: <i>term frequency-inverse document frequency</i>)	47
TL	Nachverfolgbarkeitsverbindung (engl.: <i>trace link</i>)	15
TLR	Wiederherstellung von Nachverfolgbarkeitsverbindungen (engl.: <i>traceability link recovery</i>)	18
ULMFiT	Universal Language Model Fine-tuning for Text Classification	52
UML	<i>Unified Modeling Language</i>	14
US	Unterabtasten (engl.: <i>undersampling</i>)	184
UXC	UniXcoder	230
VCS	Versionskontrollsystem (engl.: <i>version control system</i>)	85
VSM	Vektorraummodell (engl.: <i>vector space model</i>)	45
WMD	Wortüberführungsdistanz (engl.: <i>Word Movers Distance</i>)	56
WSD	Disambiguierung von Wortbedeutungen (engl.: <i>word sense disambiguation</i>)	42
WWW	Weltweites Internet (engl.: <i>world wide web</i>)	122

Teil I.

Prolog

1. Einleitung

Traceability is always there, without ever having to think about getting it there, as it is built into the engineering process; traceability has effectively „disappeared without a trace.“

(*The Grand Challenge of Traceability [Got+12a]*)

Im Laufe der Entwicklung und Wartung eines Softwaresystems entstehen verschiedenste Artefakte. Diese Artefakte, wie Anforderungen, Spezifikationen oder der Quelltext selbst, sind hierbei keinesfalls unabhängig voneinander, sondern stellen zumeist Verfeinerungen oder Umsetzungen der Inhalte eines anderen Artefakttypen dar. Der Zusammenhang, welche Teile des Quelltextes die Umsetzung welcher Anforderungen darstellen, ist dabei essenziell für ein vollständiges Verständnis des Systems. Deshalb wird für die Zertifizierung von sicherheitskritischen Systemen wie z. B. in der Luft- und Raumfahrt [RTC00; ECS09; Rie13] ein Erfassen dieser Information explizit gefordert. Auch ganz allgemein kann diese Information für das Verständnis der Zusammenhänge in einem System förderlich sein. Beispielsweise profitieren Projekte bei der Einarbeitung neuer Mitarbeiter:innen von einem schnelleren und leichteren Zugriff auf die textuelle Beschreibung in Form der Anforderungen und ihre jeweilige Umsetzung im Quelltext. Auch im Softwareentwicklungsprozess nachgelagerte Analysen wie die Änderungsauswirkungsanalyse oder das Bestimmen der Abdeckung der Anforderungen profitieren von vorhandenen Nachverfolgbarkeitsinformationen. Die Nachverfolgbarkeitsinformation erleichtern den Prozess für den Menschen oder ermöglichen es sogar, die Analyse zu automatisieren [CGZ+12]. Dennoch fehlen Nachverfolgbarkeitsinformationen in den meisten Softwareprojekten. Dies lässt sich zu großen Teilen dem hohen manuellen Aufwand und den daraus resultierenden hohen Kosten zuschreiben, die das Erstellen und Pflegen der Verbindungen erfordert. Laut den „Grand Challenges of Traceability“ von Gotel et al. [Got+12a] ist das automatische Identifizieren dieser Zusammenhänge eine der Herausforderungen, die adressiert werden müssen, um die beschriebenen Vorteile in einer Vielzahl von Projekten kosteneffizient zur Verfügung zu stellen.

Bisherige Verfahren, die Nachverfolgbarkeitsverbindungen (engl.: *trace links*, TLs) automatisch generieren, stützen sich zumeist auf Techniken der Informationsrückgewinnung (engl.: *information retrieval*, IR) [Ant+02; MM03; Get+11; Loh+13; HDO03; ZSC10; Mor+20b] oder des maschinellen Lernens (engl.: *machine learning*, ML) [GCC17; ZCS17; Mil+19]. Da die beschreibenden Teile des Quelltextes ebenso wie die Anforderungen aus Sequenzen von natürlichsprachlichen Wörtern bestehen, nutzen diese Verfahren oftmals die lexikalische Ähnlichkeit dieser Artefakte. Sie erzielen vergleichsweise geringe Präzisionen auf akzeptablen Ausbeuteniveaus [HDS06; SHC15; Ant+17]. Da diese Verfahren

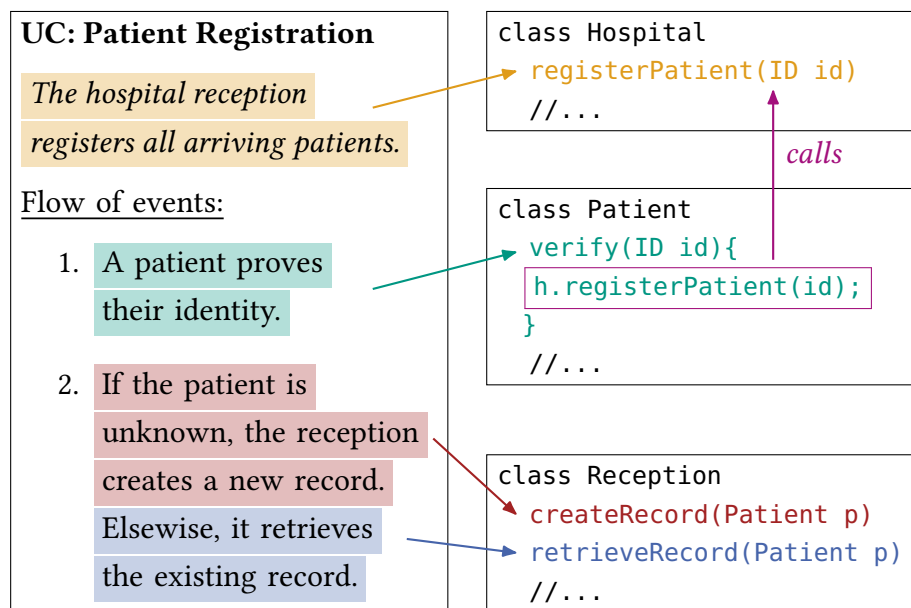


Abbildung 1.1.: Beispiel feingranularer Nachverfolgbarkeitsbeziehungen zwischen einer Anwendungsfallbeschreibung und verschiedenen Quelltextelementen (in Anlehnung an [Hey+21])

auf die syntaktischen Merkmale der Eingaben setzen, tun sie sich schwer damit, die Ähnlichkeit von Artefakten zu bestimmen, die nicht syntaktisch ähnlich sind. Anforderungen und Quelltext enthalten aber Formulierungen auf unterschiedlichen Abstraktionsniveaus und verwenden oftmals unterschiedliche Begrifflichkeiten. Ihr Zusammenhang wird also nur über ihre Semantik klar. So lässt sich der Zusammenhang zwischen der Aussage „A patient proves their identity“ in der Anwendungsfallbeschreibung und der Methode `verify(ID id)` der Klasse `Patient` in Abbildung 1.1 nur über ihre Bedeutung herleiten. Ein automatisches Verfahren muss hier erkennen, dass das Nachweisen einer Identität eine Form der Verifikation ist. Um erfolgreich TL zu bestimmen, muss die semantische Lücke (engl.: *semantic gap*) zwischen den Artefakttypen überbrückt werden [De +12].

Bisherige Ansätze, die Semantik in die Betrachtung miteinzubeziehen, setzen auf Techniken wie Themenmodellierung [AAT10] oder Worteinbettungen [GCC17; ZCS17; Che+19]. Allerdings werden diese Techniken oftmals auf Artefaktebene angewandt, anstatt eine detaillierte Aufteilung und Auflösung der Bestandteile der Anforderungen und der Quelltextdateien durchzuführen. Ein solches Vorgehen kann zu fehlerhaften Ergebnissen führen, insbesondere wenn semantisch unzusammenhängende Wörter oder zu viele Aspekte aggregiert werden. So kann es passieren, dass die zur Ähnlichkeitsbestimmung gewählte Repräsentation irgendwo zwischen den eigentlich beschriebenen Aspekten liegt. Dies würde es deutlich erschweren, die Repräsentationen mit ihren tatsächlichen Gegenständen zu verbinden. Betrachtet man die Artefakte allerdings als eine Zusammensetzung einzelner semantischer Einheiten und versucht die Ähnlichkeiten dieser feingranularen Elemente zu bestimmen, kann dies die Abbildung erleichtern. Ein solches Vorgehen bietet außerdem den Vorteil auch feingranulare Beziehungen innerhalb eines Artefakttyps, wie z. B. die Aufrufabhängigkeit zwischen den Methoden `verify` und `register` in Abbildung 1.1, zu

nutzen. Diese deutet für beide Artefakte auf eine gemeinsame Relevanz für die gegebene Anwendungsfallbeschreibung hin.

Eine weitere Möglichkeit sehr gute Ergebnisse für die Abbildung zwischen den Artefakten zu erzielen ist es, überwacht maschinelles Lernen (engl.: *supervised machine learning*) einzusetzen [MEH18; Mil+19; Lin+21]. Hierfür benötigen die Verfahren bereits vorhandene initiale TLs eines Projektes, auf denen sie ein Modell trainieren können. Doch gerade die zeitaufwendige und kostspielige Erstellung von Nachverfolgbarkeitsinformationen ist es, die das Schaffen von Nachverfolgbarkeit (engl.: *traceability*) in den meisten Fällen verhindert. Der Wert dieser Information wird als nicht ersichtlich genug wahrgenommen, um die Kosten für die Erstellung einzupreisen [Got+12a]. Da außerdem die durch das Modell erlernten Zusammenhänge eines Projektes nicht notwendigerweise auf ein neues ungesehenes Projekt übertragbar sind, können die Modelle nicht ohne Weiteres wiederverwendet werden. Eine Anwendung eines Verfahrens ohne Feinanpassung auf Projektspezifika stellt allerdings ein realistisches Szenario in der praktischen Anwendung der automatischen Wiederherstellung von Nachverfolgbarkeitsverbindungen dar. Denn hierdurch lässt sich das Verfahren auf eine Großzahl von Projekten anwenden, ohne einen weiteren Einsatz von Ressourcen zu benötigen. Es müssen insbesondere keine Expert:innen zur Verfügung stehen, die initiale TLs manuell erstellen. Deshalb stellt die Automatisierung der Erstellung von Nachverfolgbarkeitsinformationen ohne initiale TLs eine der entscheidenden Herausforderungen dar, um Nachverfolgbarkeitsinformationen weitreichend einzuführen [Got+12a].

Die Aufgabe der Wiederherstellung von Nachverfolgbarkeitsverbindungen (engl.: *traceability link recovery*, TLR) ohne bestehende TLs ist somit die am flexibelsten einsetzbare aber auch schwierigste Form der automatischen Nachverfolgbarkeit, da sie mit den geringsten Informationen auskommen muss. Sie ist aber, aufgrund ihrer flexiblen Einsetzbarkeit, auch ein Weg, Nachverfolgbarkeitsinformation in möglichst vielen Projekten nutzbar zu machen und somit das Potenzial weiterführender Analysen im Softwareentwicklungs- und wartungsprozess auszuschöpfen. In Anlehnung an unüberwachtes maschinelles Lernen wird diese Aufgabe im Rahmen dieser Arbeit als unüberwachte TLR bezeichnet.

1.1. Zielstellung und Einschränkungen

Ziel dieser Arbeit ist es, ein Verfahren zur unüberwachten TLR entwickeln, welches automatisiert TLs zwischen Anforderungen und Quelltext identifiziert. Anforderungen sollen in natürlicher Sprache vorliegen und nicht strukturell eingeschränkt sein. Die Anforderungen können Strukturen, wie beispielsweise vorgegebenen Vorlagen, folgen, müssen dies aber nicht. Der betrachtete Quelltext soll dabei in objektorientierter Programmierung vorliegen. Dies ermöglicht es auch strukturelle Informationen, wie beispielsweise Vererbungsbeziehungen zwischen Typen, als Informationsquellen zu nutzen.

Um ein automatisiertes Generieren von TLs erreichen zu können, muss ein System in der Lage sein, die durch die Artefakte beschriebenen Funktionalitäten und Eigenschaften zu

erkennen und zueinander in Beziehung zu setzen. Hierzu reicht es nicht aus, ausschließlich die syntaktischen Merkmale der Eingaben zu analysieren, sondern die Semantik der Artefakte muss geeignet repräsentiert und interpretiert werden. Eine Schwierigkeit hierbei stellt die semantische Lücke zwischen Anforderungen und Quelltext dar, welche aufgrund der unterschiedlichen Entstehungszeitpunkte und Abstraktionsniveaus besteht. Das entwickelte Verfahren soll also in der Lage sein, Artefakte basierend auf ihrer Semantik zu repräsentieren und diese semantische Repräsentation für die Abbildung zu nutzen. Um mit den potenziell vielschichtigen Aspekten innerhalb eines Artefakts umgehen zu können, soll sich das Verfahren außerdem feingranulare Beziehungen zwischen den Artefakten zunutze machen. Dadurch soll gewährleistet werden, dass nicht semantisch zusammenhängende Teile eines Artefakts nicht zu einer Verfälschung der Abbildung der gesamten Artefakte führen, wie es bei einer Aggregation aller Informationen eines Artefakts der Fall sein kann.

Um das Verfahren in möglichst vielen Anwendungsszenarien einsetzen zu können, sollen die TLs nachträglich wiederhergestellt werden. Hierdurch können die TLs auch in Projekten generiert werden, in denen während der Erstellung der Artefakte keine Nachverfolgbarkeitsinformationen gespeichert wurden. Außerdem soll das Verfahren ohne initiale TLs auskommen. Dies ermöglicht es, das Verfahren ohne manuellen Aufwand eines Experten oder einer Expertin bzw. ohne erneutes Trainieren eines Modells auf andere, potenziell unbekannte Projekte anzuwenden. Diese Randbedingungen bedeuten also, dass außer dem vollständigen Quelltext sowie allen Anforderungen keine weiteren Informationen über das Projekt zur Verfügung stehen. Insbesondere müssen also keine Informationen zur Art und Ausprägung der TLs im jeweiligen Projekt vorliegen. Außerdem soll das Verfahren auch ohne zeitliche Informationen des Erstellungsprozesses, wie beispielsweise die zeitliche Abfolge des Erstellens der Artefakte, auskommen. Diese Informationen wären zwar nützlich für die Identifikation von Nachverfolgbarkeitsinformation, stehen aber nicht in allen Projekten zur Verfügung und würden somit die Anwendbarkeit des Verfahrens einschränken.

Ziel 1: Entwicklung eines automatischen Verfahrens zur unüberwachten Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext

Unterziel 1.1: Das Verfahren soll auf Basis der semantischen Zusammenhänge zwischen den Artefakten arbeiten

Unterziel 1.2: Das Verfahren soll sich feingranulare Beziehungen der Artefakte zunutze machen

Einschränkung 1: Fokussierung auf natürlichsprachliche Anforderungen

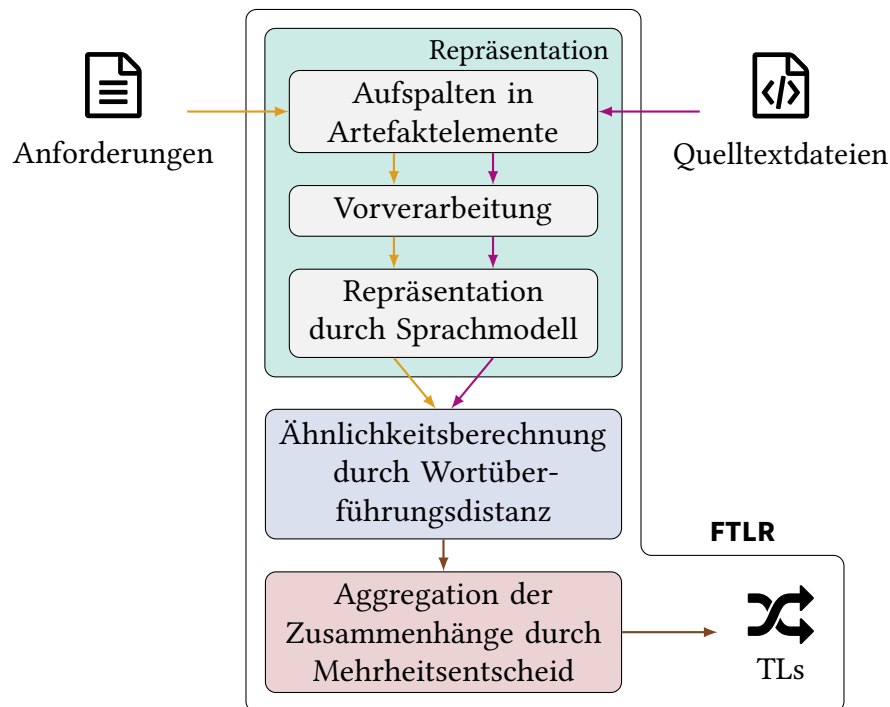


Abbildung 1.2.: Überblick der Teilschritte **FTLRs** zur automatischen Identifikation von Nachverfolgbarkeitsverbindungen (engl.: *trace links*, TLs) zwischen Anforderungen und Quelltext

Einschränkung 2: Fokussierung auf objektorientierten Quelltext

Einschränkung 3: Verzicht auf Informationen zur Art und Ausprägung der Nachverfolgbarkeitsverbindungen eines Projektes

1.2. Automatische Wiederherstellung von Nachverfolgbarkeitsverbindungen mittels feingranularer Beziehungen

In dieser Dissertation befasse ich mich mit der Frage, inwiefern ein Verfahren zur automatischen TLR von feingranularen Ähnlichkeiten profitieren kann. Hierbei fokussiere ich mich auf die Nachverfolgbarkeit von natürlichsprachlichen Anforderungen zu objektorientiertem Quelltext. Im Folgenden bezeichne ich sowohl einzelne Anforderungen als auch ganze Quelltextdateien als Artefakte und ihre Bestandteile, wie z. B. Sätze und Methoden, als Artefaktelemente. Im Gegensatz zur *grobgranularen* Abbildung zwischen den gesamten

Artefakten nenne ich die Abbildung zwischen einzelnen Artefaktelementen *feingranular*¹. Meine Beiträge unterteilen sich in ein automatisiertes Verfahren zur Bestimmung von TLs auf Basis feingranularer, worteinbettungsbasierter Ähnlichkeiten [Hey+21], einen Ansatz zur Klassifikation von Anforderungselementen [Hey+20a] und einem Verfahren zur Nutzung von bimodalen, kontextsensitiven Sprachmodelle zur automatischen TLR. Die Klassifikation unterstützt das Nachverfolgbarkeitsverfahren bei der Beschränkung auf relevante Teile, indem auf Basis ihrer Ergebnisse Anforderungselemente von der Verarbeitung ausgeschlossen werden. Das Verfahren zur Nutzung eines bimodalen, auf Quell- und natürlichsprachlichem Text vortrainiertem Sprachmodells dient der Analyse der Vor- und Nachteile dieser Form von Modellen für die TLR im Vergleich zu statischen Sprachmodelle, wie es Worteinbettungen sind.

Das Verfahren zur Bestimmung von TLs, *Fine-grained Traceability Link Recovery (FTLR)*, setzt sich aus den folgenden drei Schritten zusammen: der Repräsentation der Artefaktelemente durch Mengen von Worteinbettungen, der Berechnung von Ähnlichkeiten zwischen den Artefaktelementen mithilfe der Wortüberführungsdistanz (engl.: *Word Movers Distance*, WMD) [Kus+15] und einer Aggregation der feingranularen Zusammenhänge über einen Mehrheitsentscheid (s. Abbildung 1.2). Für die Repräsentation wird ein auf 24 Terabyte Textdaten vortrainiertes *fastText*-Worteinbettungsmodell [Gra+18] verwendet, welches Wörter bzw. Wortteile, die in ähnlichen Kontexten auftreten, auf ähnliche Vektoren abbildet. Jedes Artefaktelement wird durch die Menge der Worteinbettungsvektoren seiner, nach einer Vorverarbeitung² verbliebenen, Wörter repräsentiert. Im Gegensatz zu bestehenden Arbeiten werden die Vektoren der Wörter eines Artefaktelements zur Bestimmung der Ähnlichkeit zu einem anderen Artefaktelement nicht zu einem einzelnen Vektor aggregiert. Stattdessen bestimmt sich die Ähnlichkeit aus der minimalen kumulativen Distanz, die benötigt wird, um jeden Vektor der Vektormenge des einen Artefakts auf einen Vektor der Vektormenge des anderen Artefakts abzubilden. Diese Ähnlichkeitsberechnung wird auch als Wortüberführungsdistanz bezeichnet und erlaubt einen semantischen Ähnlichkeitsvergleich, der nicht durch Aggregation verfälscht wird [Kus+15]. Die eigentliche Bestimmung der Nachverfolgbarkeitsverbindungskandidaten erfolgt durch eine Mehrheitsentscheidung über alle Elemente eines Artefakts, um die vorherrschenden Aspekte und Zusammenhänge pro Artefakt zu bestimmen und ggf. irrelevante Zusammenhänge zu ignorieren. **FTLR** kann sich außerdem strukturelle Informationen des Quelltextes in Form von Aufrufabhängigkeiten zwischen Methoden und der Anforderungen in Form von vorhandenen Vorlagenelementen zunutze machen, um die Abbildung zu verbessern.

Für die Klassifikation der Anforderungselemente nutze ich das auf Anforderungen feinangepasste *BERT*-Sprachmodell **NoRBERT** [Hey+20a]. Dieses kann für ein Anforderungselement bestimmen, ob es ausschließlich nichtfunktionale oder nutzerorientierte Informationen enthält und somit für eine Abbildung auf Quelltext irrelevant ist. Dies ermöglicht es mir zu zeigen, dass durch automatisches Filtern dieser irrelevanten Elemente die Präzision von

¹ Die Abbildung einer gesamten Anforderung auf eine Quelltextklasse wäre somit eine *grobgranulare* Abbildung, wohingegen die Verbindung zwischen einem Anforderungssatz und einer Methode einer Quelltextklasse eine *feingranulare* Abbildung darstellen würde.

² Aufspalten der Binnenmajuskelschreibweise, Grundformbestimmung und Stoppwortentfernung

FTLR gesteigert werden kann. Außerdem konnte ich zeigen, dass durch den erstmaligen Einsatz von Transformer-basierten Sprachmodelle für die Anforderungsklassifikation, eine verbesserte Übertragbarkeit der Klassifikationsmodelle auf neue Projekte erreicht werden konnte.

Zusätzlich zur Verwendung von *fastText*-Worteinbettungen zur Repräsentation der Artefaktelemente habe ich außerdem eine Variante von **FTLR** entwickelt, welche das bimodale Sprachmodell *UniXcoder* [Guo+22a] verwendet. Dieses ist im Gegensatz zu *fastText* nicht statisch, sondern kontextsensitiv. Dies bedeutet, dass es in der Lage ist, ein Wort im Kontext seiner umgebenden Worte zu repräsentieren, anstatt unabhängig vom Kontext immer dieselbe Repräsentation für ein Wort zu verwenden. Außerdem wurde dieses Modell auf einem großen bimodalen Korpus vortrainiert, welcher aus natürlichsprachlichen Methodendokumentationen und ihren entsprechenden Methoden in unterschiedlichen Programmiersprachen besteht. Der Hintergedanke ist, dass ein derart trainiertes Modell in der Lage sein sollte, eine präzisere Repräsentation von Eingaben aus der Softwareentwicklungsdomäne zu erstellen. Da ein solches Modell allerdings auch stark von der Größe der Trainingsmenge und dessen Eignung für die jeweilige Anwendungsdomäne abhängt, dient diese Variante von **FTLR** dazu zu überprüfen, ob sich der erhöhte Aufwand (bezüglich Hardwareanforderungen und Rechenzeit) für die TLR auszahlt.

Die Beiträge meiner Arbeit unterstützen Entwickler:innen beim Nachvollziehen und Verdeutlichen der Zusammenhänge zwischen Anforderungen und Quelltext. Sie machen das oftmals implizite, lediglich von den beteiligten Personen gehaltene Wissen explizit und stellen es somit einer breiteren Gruppe von Personen zur Verfügung. Zusätzlich werden durch dieses explizite Wissen nachgelagerte Analysen wie die Änderungsauswirkungsanalyse (engl.: *change impact analysis*) oder Software-Wiederverwendbarkeitsanalyse (engl.: *software reusability analysis*) ermöglicht oder verbessert [CGZ+12]. Außerdem kann das entwickelte Klassifikationsverfahren, auch unabhängig von der Nachverfolgbarkeit, für die automatische Verarbeitung von Softwareentwicklungsartefakten eingesetzt und somit in einer Vielzahl von Analysen verwendet werden.

Beitrag 1: Ein automatisiertes Verfahren zur Bestimmung von Nachverfolgbarkeitsverbindungen auf Basis feingranularer, worteinbettungsbasierter Ähnlichkeiten (**FTLR**)

Beitrag 2: Einen Ansatz zur Klassifikation von Anforderungselementen (**NoRBERT**) und dessen Integration in **FTLR**

Beitrag 3: Einen Ansatz zur Nutzung bimodaler, kontextsensitiver Sprachmodelle zur unüberwachten TLR zwischen Anforderungen und Quelltext

1.3. Thesen

Die Beiträge dieser Arbeit zielen auf die Verbesserung der automatischen Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext ab. In den letzten drei Jahrzehnten haben sich bereits viele Arbeiten mit dieser Zielstellung befasst, doch keines erzielte die nötige Güte hinsichtlich Präzision und Ausbeute für eine vollautomatische Anwendung. Um zu überprüfen, welche Verbesserungen die Beiträge dieser Arbeit gegenüber bestehenden Verfahren erzielen und wie diese sich untereinander verhalten, werden die folgenden Thesen aufgestellt:

- T₁**: Eine unüberwachte Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext mittels feingranularer, worteinbettungsbasierter Beziehungen steigert die Leistung gegenüber bestehenden unüberwachten Verfahren.
- T_{1.1}**: Der Einbezug struktureller Informationen des Quelltexts und der Anforderungen kann die Leistung eines worteinbettungs-basierten Verfahrens steigern.
- T_{1.2}**: Der Einsatz der Wortüberführungsdistanz zur wortweisen Abbildung der Artefaktelemente verbessert die Abbildungsgüte gegenüber einfachen, aggregierten Vektorabbildungen.
- T_{1.3}**: Ein Aggregieren feingranularer Ähnlichkeiten zur Bestimmung von Nachverfolgbarkeitsverbindungen auf Artefaktebene ermöglicht eine höhere Leistung bei der Abbildung im Vergleich zur direkten Abbildung der gesamten Artefakte.
- T₂**: Durch geeignete Klassifikation der Anforderungen können irrelevante Teile der Eingabe ausgeschlossen werden und damit die Präzision eines worteinbettungs-basierten Verfahrens zur Nachverfolgbarkeit gesteigert werden.
- T₃**: Der Einsatz eines bimodalen, kontextsensitiven Sprachmodells für die Repräsentation der Artefaktelemente verbessert die Leistung eines feingranularen Verfahrens zur Nachverfolgbarkeit.

1.4. Aufbau der Arbeit

Der Aufbau dieser Arbeit gliedert sich folgendermaßen: Im nachfolgenden Kapitel werde ich zunächst auf die für das Verständnis dieser Arbeit notwendigen Grundlagen eingehen (Kapitel 2). Anschließend werden in Kapitel 3 verwandte Arbeiten aus dem Forschungsfeld der Wiederherstellung von Nachverfolgbarkeitsverbindungen und weiteren verwandten Gebieten der Softwaretechnik vorgestellt und analysiert. In Kapitel 4 und 5 wird ein grundlegendes Verfahren zur automatischen TLR zwischen Anforderungen und Quelltext entwickelt. Hierzu werden zunächst die Anforderungen an ein solches System analysiert sowie mögliche Lösungsansätze diskutiert (Kapitel 4). Auf Basis dieser Analyse wird daraufhin in Kapitel 5 das Verfahren **FTLR**, welches sich feingranulare, worteinbettungs-basierte Beziehungen zunutze macht, und seine Varianten vorgestellt. Kapitel 6 beschreibt

daraufhin die Ergebnisse der verschiedenen **FTLR**-Varianten auf Vergleichsdatensätzen für die TLR und liefert somit Erkenntnisse bezüglich der Thesen $T_{1.1}$, $T_{1.2}$ und $T_{1.3}$.

Es folgt in den nächsten Kapiteln die Beschreibung des Verfahrens zur Anforderungsklassifikation und dessen Integration in **FTLR** als Vorfilter. Zunächst werden in Kapitel 8 geeignete Klassen für diese Zielstellung identifiziert sowie Möglichkeiten der Integration analysiert. Kapitel 9 enthält daraufhin die Beschreibung des Verfahrens **NoRBERT**, welches Transferlernen (engl.: *transfer learning*) für die Anforderungsklassifikation nutzt. Das Kapitel enthält außerdem eine Evaluation der Güte des Klassifikationsverfahrens, bemessen auf Vergleichsdatensätzen für die Anforderungsklassifikation sowie den TLR-Vergleichsdatensätzen. In Kapitel 10 wird daraufhin betrachtet, welchen Effekt die Integration der Anforderungsklassifikation auf die Ergebnisse **FTLRs** hat. Dieses Kapitel liefert somit Erkenntnisse bezüglich These T_2 .

Die folgenden zwei Kapitel (Kapitel 12 und 13) beschäftigen sich mit dem Einsatz bimodaler, kontextsensitiver Sprachmodelle in **FTLR**. Hierbei werden in Kapitel 12 verschiedene Integrationsmöglichkeiten in **FTLR** analysiert und ein geeignetes Sprachmodell identifiziert. Kapitel 13 enthält daraufhin die Ergebnisse der verschiedenen Integrationsvarianten auf den TLR-Vergleichsdatensätzen sowie eine vergleichende Analyse **FTLRs** mit den unterschiedlichen Sprachmodellen. Dieser Teil der Arbeit liefert also Erkenntnisse zu These T_3 .

In Kapitel 15 werden Erkenntnisse zu These T_1 geliefert, indem die Ergebnisse **FTLRs** mit denen bestehender Arbeiten zur Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext verglichen werden. Abschließend fasst Kapitel 16 die Ergebnisse dieser Arbeit zusammen. Zunächst werden die Erkenntnisse dieser Arbeit bezüglich der aufgestellten Thesen betrachtet. Anschließend werden weiterführende Forschungsfragen und mögliche weiterführende Arbeiten diskutiert.

2. Grundlagen

The greatest risk we face in software development is that of overestimating our own knowledge.

(Jim Highsmith)

Dieses Kapitel dient der Vermittlung der zum Verständnis dieser Arbeit notwendigen Grundlagen. Da der Forschungsfokus dieser Arbeit auf der Nachverfolgbarkeit von Softwareentwicklungsartefakten liegt, wird dieses Forschungsfeld zunächst eingeführt (Abschnitt 2.1). Daraufhin werden in den Abschnitten 2.2 und 2.3 Eigenschaften der in dieser Arbeit als Eingaben betrachteten Artefakte, Anforderungen und Quelltext, beschrieben. Anschließend erklären Abschnitte 2.4 und 2.5 Grundlagen der Informationsrückgewinnung und des maschinellen Lernens, welche für das Verständnis der in dieser und verwandten Arbeiten verwendeten Verfahren relevant sind. Da ein Großteil der betrachteten Informationen in natürlicher Sprache vorliegt, werden in Abschnitt 2.6 sprachwissenschaftliche Grundbegriffe eingeführt und daraufhin in Abschnitt 2.8 Verfahren zur Verarbeitung natürlicher Sprache vorgestellt. Außerdem greifen einige Verfahren zur Sprachverarbeitung auf Wissensquellen zurück, um allgemeines bzw. sprachliches Wissen nutzbar zu machen. Abschnitt 2.7 beschreibt hierbei mögliche Wissensquellen und ihren Inhalt und Aufbau. Um natürliche Sprache für Verfahren des maschinellen Lernens nutzbar zu machen, müssen Wörter und Dokumente geeignet repräsentiert werden. Abschnitt 2.9 führt verschiedene Möglichkeiten dieser Repräsentation ein und verdeutlicht den Bezug zu maschinellen Lernverfahren, indem das Konzept von Sprachmodellen verdeutlicht wird. Viele Verfahren zur automatischen Nachverfolgbarkeit von Softwareentwicklungsartefakten reduzieren das Problem auf Ähnlichkeitsvergleiche zwischen Wörtern und Dokumenten. Daher werden in Abschnitt 2.10 verschiedene Möglichkeiten der Ähnlichkeitsbestimmung zwischen Wörtern und Dokumenten beschrieben. Abschließend geben die Abschnitte 2.11 und 2.12 einen Überblick über die üblicherweise zur Evaluation von Nachverfolgbarkeitsverfahren genutzten Metriken und Hypothesentests.

2.1. Nachverfolgbarkeit von Softwareentwicklungsartefakten

Mit Nachverfolgbarkeit (engl.: *traceability*) wird im Rahmen der Softwareentwicklung die Möglichkeit beschrieben, Informationen innerhalb von Softwareartefakten miteinander in Verbindung zu setzen und diese Beziehung nutzbar zu machen [Got+12b]. Um Nachverfolgbarkeit zu erzielen, müssen also nachverfolgbare Verbindungen zwischen den

Artefakten existieren. Diese Verbindungen werden als Nachverfolgbarkeitsverbindungen (engl.: *trace links*, TLs) bezeichnet.

Nachverfolgbarkeit von Softwareentwicklungsartefakten ist von großem Wert, da die Existenz von Nachverfolgbarkeitsinformationen (engl.: *traceability information*) viele weitere Aufgaben und Analysen im Softwareentwicklungs- und -wartungsprozess verbessert oder sogar erst ermöglicht [WvP10; BRA14]. Die wohl wichtigsten dieser Aufgaben sind die Änderungsauswirkungsanalyse, die Software-Wiederverwendbarkeitsanalyse und die Abhängigkeitsanalyse.

2.1.1. Nachverfolgbares Artefakt

Als ein nachverfolgbares Artefakt (engl.: *trace artifact*) wird in Bezug auf Nachverfolgbarkeit jedes Artefakt im Software- bzw. Systementwicklungsprozess bezeichnet, welches dem Prozess des Nachverfolgens zugänglich gemacht werden kann.

Gotel et al. [Got+12b] definieren diese nachverfolgbaren Artefakte wie folgt:

Definition 1: Nachverfolgbares Artefakt (engl.: *trace artifact*) [Got+12b]

A *traceable* unit of data (e.g., a single requirement, a cluster of requirements, a UML class, a UML class operation, a Java class or even a person). A trace artifact is one of the trace elements and is qualified as either a source artifact or as a target artifact when it participates in a trace. The size of the traceable unit of data defines the granularity of the related trace.¹

Somit kann es sich bei einem nachverfolgbaren Artefakt um eine einzelne Anforderung (oder auch nur Teile dieser), Mengen von Anforderungen oder ganze Anforderungsdokumente, aber auch z. B. Diagramme der *Unified Modeling Language* (UML) oder Quelltext und seine einzelnen Bestandteile, handeln. Zumeist wird aus Gründen der Lesbarkeit sowohl das Objekt als Ganzes als auch jedweder enthaltene nachverfolgbare Teil mit „Artefakt“ bezeichnet. Dieser Konvention schließe ich mich in dieser Arbeit an.

Nachverfolgbare Artefakte lassen sich anhand ihres Typs klassifizieren. Dieser Typ gibt an, von welcher Art und Funktion das Artefakt ist und bezieht sich zumeist auf eines der anerkannten und dokumentierten Artefakte des Software- bzw. Systementwicklungsprozesses [Got+12b]. So wären z. B. Anforderungen, Entwurfsbeschreibungen, Quelltext oder Testfälle potenzielle Artefakttypen.

¹ Zu Deutsch: Eine nachverfolgbare Dateneinheit (z. B. eine einzelne Anforderung, eine Gruppe von Anforderungen, eine UML-Klasse, eine UML-Klassenoperation, eine Java-Klasse oder sogar eine Person). Ein nachverfolgbares Artefakt ist eines der Elemente der Nachverfolgbarkeit und wird entweder als Quell- oder als Zielartefakt eingeordnet, sobald es Teil einer TL ist. Die Größe der nachvollziehbaren Dateneinheit definiert die Granularität der Verbindung.

	QA ₁	QA ₂	QA ₃	...	QA _N
ZA ₁	0	1	0	...	0
ZA ₂	0	0	0	...	1
...
ZA _N	1	1	1	...	0

Abbildung 2.1.: Schema einer Nachverfolgbarkeitsmatrix mit Quell- und Zielartefakten (QA bzw. ZA) und beispielhaften Einträgen für Nachverfolgbarkeitsverbindungen gekennzeichnet mit einer Eins

2.1.2. Nachverfolgbarkeitsverbindung

Gotel et al. [Got+12b] definieren Nachverfolgbarkeitsverbindungen (engl.: *trace links*, TLs) wie folgt:

Definition 2: Nachverfolgbarkeitsverbindung (engl.: *trace link*, TL) [Got+12b]

A specified association between a pair of artifacts, one comprising the source artifact and one comprising the target artifact.²

Auch wenn diese Notation eine Richtung der Assoziation impliziert, kann in der Praxis jede TL auch in die entgegengesetzte Richtung traversiert werden. Oftmals wird auch von der primären bzw. entgegengesetzten Richtung gesprochen.

TLs können aufgrund ihrer Struktur (Syntax) und ihrem Zweck (Semantik) charakterisiert werden [Got+12b]. Mögliche Nachverfolgbarkeitsverbindungstypen wären z. B. „implementiert“, „testet“, „verfeinert“ oder „ersetzt“. Diese Typen repräsentieren zumeist die semantische Rolle, die diese Verbindung einnimmt (vgl. Abschnitt 2.8.7). Außerdem charakterisiert man TL zusätzlich anhand der Granularität, die sie abbilden [Got+12b]. Die Granularität einer TL wird durch die Granularität des Quell- bzw. Zielartefakts bestimmt. So wäre eine Verbindung zwischen einer Quelltextmethode und einem Satz einer Anforderung feingranularer als die TL zwischen der Anforderung und der Klasse, welche diese Artefakte enthalten. Üblicherweise werden z. B. TL zwischen Anforderungen und Quelltext auf Granularität ganzer Anforderungen und Klassen angegeben. Feingranularere Verbindungen haben sich allerdings bereits als hilfreich herausgestellt [Hey+21].

Die Menge aller TLs wird häufig in Form einer Matrix repräsentiert (vgl. Abbildung 2.1). Diese sogenannte Nachverfolgbarkeitsmatrix (engl.: *traceability matrix*) besitzt je eine Spalte bzw. Zeile pro möglichen Quell- bzw. Zielartefakt und repräsentiert eine vorhandene TL zwischen einem Quell-Ziel-Paar mit einer Eins an der entsprechenden Stelle in der Matrix. Alle weiteren Einträge der Matrix entsprechen der Null.

² Zu Deutsch: Eine Verbindung zweier nachverfolgbarer Artefakte bei der ein Artefakt die Funktion des Quellartefakts und eine die des Zielartefakts einnimmt

2.1.3. Nachverfolgbarkeit

Gotel et al. definieren Nachverfolgbarkeit (engl.: *traceability*) wie folgt:

Definition 3: Nachverfolgbarkeit (engl.: *traceability*) nach Gotel et al. [Got+12b]

The potential for traces to be established and used.³

Nach dieser Definition ist Nachverfolgbarkeit ein Attribut von Artefakten. Der ISO/IEC/IE-EE 24765 Standard [ISO17] hingegen definiert Nachverfolgbarkeit folgendermaßen:

Definition 4: Nachverfolgbarkeit (engl.: *traceability*) nach ISO/IEC/IEEE 24765 Standard [ISO17]

1. degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another
2. discernible association among two or more logical entities, such as requirements, system elements, verifications, or tasks
3. degree to which each element in a software development product establishes its reason for existing⁴

In dieser Definition wird deutlich, dass der Begriff überladen ist und sowohl für die Möglichkeit bzw. den Grad der Möglichkeit steht, dass Artefakte in Verbindung gebracht werden können, als auch für die Verbindung selbst. Dies trifft vermutlich mehr auf den englischen Begriff *Traceability* zu, als auf Nachverfolgbarkeit. Insbesondere, da im Englischen der Begriff *trace*, für das Tripel aus Quell- und Zielartefakt zusammen mit der TL zwischen ihnen, manchmal mit *Traceability* synonym verwendet wird. Der dritte Punkt der Definition des Standards bezieht sich hingegen wieder mehr auf den ersten Punkt, indem Nachverfolgbarkeit den Grad bestimmt, zu dem nachvollziehbar ist, weshalb Artefakte im Softwareentwicklungsprozess existieren. Ist es also möglich, eine TL zwischen zwei Artefakten zu finden, dient diese als Indikator für den Grund ihrer Existenz. Die Standarddefinition ist also dahingehend umfassender als die Definition von Gotel et al., als dass diese nur den ersten Punkt der Standarddefinition abdeckt.

Der Begriff Nachverfolgbarkeit wird häufig in Verbindung mit dem primär untersuchten Artefakttyp benutzt. So wird häufig von Anforderungsnachverfolgbarkeit (engl.: *requirements*

³ Zu Deutsch: Die Möglichkeit, dass Nachverfolgbarkeitsverbindungen zwischen Artefakten erstellt und genutzt werden können.

⁴ Zu Deutsch: 1. der Grad, in dem eine Beziehung zwischen zwei oder mehreren Produkten des Entwicklungsprozesses hergestellt werden kann, insbesondere zwischen Produkten, die in einer Vorgänger-Nachfolger- oder Haupt-Untergeordnet-Beziehung zueinander stehen
2. die erkennbare Assoziation zwischen zwei oder mehreren logischen Einheiten, wie z. B. Anforderungen, Systemelementen, Verifikationen oder Tätigkeiten
3. der Grad, in dem jedes Element eines Softwareentwicklungsprodukts die Grundlage für seine Existenz nachweist

traceability), Softwarenachverfolgbarkeit (engl.: *software traceability*) oder Systemnachverfolgbarkeit (engl.: *systems traceability*) gesprochen. Die für diese Arbeit hauptsächlich relevante Spezialform von Nachverfolgbarkeit, die Anforderungsnachverfolgbarkeit, definieren Gotel und Finkelstein [GF94] wie folgt:

Definition 5: Anforderungsnachverfolgbarkeit (engl.: *requirements traceability*) [GF94]

The ability to describe and follow the life of a requirement in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases).⁵

Laut dieser Definition, bezieht sich also Anforderungsnachverfolgbarkeit hauptsächlich auf die Möglichkeit, Anforderungen und mit Anforderungen in Verbindung stehende Artefakte in ihrem Lebenszyklus und ihrer Beziehung zueinander zu betrachten.

Der ISO/IEC/IEEE 24765 Standard [ISO17] definiert Anforderungsnachverfolgbarkeit hingegen als eines der folgenden drei Konzepte:

Definition 6: Anforderungsnachverfolgbarkeit (engl.: *requirements traceability*) nach ISO/IEC/IEEE 24765 Standard [ISO17]

1. identification and documentation of the derivation path (upward) and allocation/flow-down path (downward) of requirements in the requirements hierarchy
2. discernible association between a requirement and related requirements, implementations, and verifications
3. traceabilities in domain and application requirements respectively and those between them⁶

Diese Definition unterscheidet sich zur Vorherigen, indem nicht von einer Vorwärts- bzw. Rückwärtsrichtung, sondern von Abwärts- bzw. Aufwärtspfaden gesprochen wird. Diese beziehen sich aber auf die gleichen Konzepte. Darüber hinaus unterscheidet diese Definition noch zwischen der Identifikation und Dokumentation der Nachverfolgbarkeit (1.) und den Beziehungen, welche diese ausmachen (2. und 3.).

⁵ Zu Deutsch: Die Fähigkeit, den Lebenszyklus einer Anforderung sowohl vorwärts als auch rückwärts zu beschreiben und nachzuvollziehen (d. h. von den Ursprüngen über die Entwicklung und Spezifikation bis hin zur anschließenden Bereitstellung und Nutzung sowie in Zeiten laufender Verfeinerung und Iteration in jeder dieser Phasen)

⁶ Zu Deutsch: 1. Identifikation und Dokumentation des Ableitungspfades (nach oben) und des Zuweisungs-/Ablaufpfades (nach unten) von Anforderungen in der Anforderungshierarchie
2. erkennbarer Zusammenhang zwischen einer Anforderung und verwandten Anforderungen, Implementierungen und Verifizierungen
3. Nachverfolgbarkeiten in Fach- und Anwendungsanforderungen bzw. zwischen ihnen

Im Zusammenhang mit Anforderungsnachverfolgbarkeit wird oftmals noch zwischen Prä- und Post-Anforderungsnachverfolgbarkeit unterschieden [Got+12b]. Ersteres bezeichnet die Möglichkeit für TLs, welche den Bezug zwischen den Anforderungen und ihrem Ursprung darstellen. So wäre die Verbindung zwischen Aussagen von Interessenvertreter:innen (engl.: *stakeholder*) und den daraus entstandenen Anforderungen an das System ein Beispiel für Prä-Anforderungsnachverfolgbarkeit. Post-Anforderungsnachverfolgbarkeit hingegen bezieht sich auf TLs, die, ausgehend von den Anforderungen, zu auf diesen basierenden Artefakten führen. So ist eine TL zwischen Anforderung und dem diese Anforderung umsetzenden Quelltext eine Form von Post-Anforderungsnachverfolgbarkeit.

2.1.4. Formen des Nachverfolgens

Um Nachverfolgbarkeit zu etablieren, müssen TLs erstellt und gewartet werden. Dies zu gewährleisten, kann auf unterschiedliche Arten erreicht werden. Hierbei wird zwischen manuellen und automatischen Vorgehen unterschieden [Got+12b]. Beim manuellen Nachverfolgen wird Nachverfolgbarkeit erreicht, indem TLs manuell durch den Menschen erstellt und auch gewartet werden. Dies kann von Hand oder mithilfe von Werkzeugunterstützung (z. B. durch Ziehen und Ablegen (engl.: *drag and drop*)) geschehen. Bei automatischen Verfahren werden die Verbindungen durch Werkzeuge vollautomatisch erzeugt und bei Änderungen aktualisiert. Wenn das Ergebnis eines automatischen Verfahrens einem Menschen zur Verifikation oder Selektion vorgelegt wird, spricht man von semi-automatischem Nachverfolgen.

Die Erstellung der TLs kann zusätzlich außerdem hinsichtlich des Erstellungszeitpunkts klassifiziert werden [Got+12b]. Werden die TLs während der Erstellung der zu verbindenden Artefakte erzeugt, spricht man von der Erfassung von Nachverfolgbarkeitsverbindungen (engl.: *traceability link capture*). Erstellt man hingegen die Verbindungen erst zu einem Zeitpunkt, zu dem bereits beide Artefakte existieren, wird dies als Wiederherstellung von Nachverfolgbarkeitsverbindungen (engl.: *traceability link recovery*, TLR) bezeichnet. Man spricht hier von Wiederherstellung, da die Information über die TLs mindestens dem Verfasser des Artefakts zu einem Zeitpunkt klar gewesen sein muss, diese Information aber nicht dokumentiert wurde oder verloren gegangen ist.

Von einer Vorwärts- bzw. Rückwärtsrichtung bzw. von Vorwärts- bzw. Rückwärtsverfolgbarkeit spricht man, wenn das Nachverfolgen den Schritten des Entwicklungsprozesses folgt (vorwärts, z. B. von Anforderungen über Entwurfsbeschreibungen zu Quelltext) bzw. dem entgegengesetzten Pfad folgt (rückwärts, z. B. von Quelltext über Entwurfsbeschreibungen zu den Anforderungen) [Got+12b].

Diese beiden Arten lassen sich außerdem der vertikalen Nachverfolgbarkeit zuordnen, welche Artefakte verbindet, die unterschiedliche Abstraktionsniveaus besitzen. Der Begriff horizontale Nachverfolgbarkeit hingegen wird benutzt, wenn Artefakte desselben Abstraktionsniveaus verbunden werden [Got+12b]. Ein Beispiel wäre das Verbinden verschiedener Versionen derselben Anforderung oder aller Anforderungen, die sich mit der Leistung des Systems befassen.

2.2. Anforderungen

Anforderungen stellen die Erwartungen an ein zu entwickelndes Produkt dar. Im Fall von Softwareanforderungen sind diese Produkte Softwaresysteme oder softwareintensive Systeme. Der ISO/IEC/IEEE Standard 24765 [ISO17] definiert Anforderungen folgendermaßen:

Definition 7: Anforderungen (engl.: *requirements*) nach ISO/IEC/IEEE 24765 Standard [ISO17]

1. statement that translates or expresses a need and its associated constraints and conditions
2. condition or capability that must be met or possessed by a system, system component, product, or service to satisfy an agreement, standard, specification, or other formally imposed documents
3. provision that contains criteria to be fulfilled
4. a condition or capability that must be present in a product, service, or result to satisfy a contract or other formally imposed specification⁷

Dieser Definition folgend beschreiben Anforderungen also, welche Fähigkeiten das System erfüllen soll bzw. welche Bedingungen und Kriterien an das System gestellt werden. Die Umsetzung der Erwartung ist allerdings nicht Teil der Anforderung. Somit befassen sich Anforderungen mit dem Problemraum und nicht dem Lösungsraum [Bal10]. Anforderungen werden oftmals in textueller, natürlichsprachlicher Form dokumentiert, damit sie auch von Interessenvertreter:innen verstanden werden können, die über kein technisches Wissen verfügen [Som12]. Es existieren aber auch Anforderungsdokumentationsvarianten, die z. B. programmiersprachenähnliche Notation oder sogar formale Konstrukte wie endliche Automaten oder Aussagenlogik verwenden. Diese werden aber nur selten verwendet [Som12]. Zusätzlich können Anforderungen auch gänzlich oder unterstützend in graphischer Notation verfasst werden, welche oftmals mit textuellen Etiketten versehen sind. Beispiele hierfür wären UML-Anwendungsfall- oder -Sequenzdiagramme.

Auch bei den natürlichsprachlich dokumentierten Anforderungen kann der Grad der vorgegebenen Struktur variieren. Zumeist wird hier zwischen unstrukturierten und strukturierten natürlichsprachlichen Anforderungen unterschieden. Als unstrukturiert zählen hierbei Anforderungen, die keiner Schablone oder vorgegebenem Wortlaut folgen und meist durch einzelne oder wenige natürlichsprachliche Sätze ausgedrückt werden. Unter strukturierten

⁷ Zu Deutsch: 1. Aussage, die einen Bedarf und die damit verbundenen Einschränkungen und Bedingungen umschreibt oder ausdrückt

2. Bedingung oder Fähigkeit, die ein System, eine Systemkomponente, ein Produkt oder ein Dienstleister erfüllen oder aufweisen muss, um eine Vereinbarung, einen Standard, eine Spezifikation oder andere formal vorgegebene Dokumente zu erfüllen

3. eine Vorgabe, die Kriterien enthält, die erfüllt werden müssen

4. eine Bedingung oder Fähigkeit, die in einem Produkt, einem Dienstleister oder einem Ergebnis vorhanden sein muss, um einen Vertrag oder eine andere formell vorgeschriebene Spezifikation zu erfüllen

natürlichsprachlichen Anforderungen versteht man Anforderungen, die zwar als natürlichsprachlicher Text formuliert sind, dabei aber einer Schablone oder einem Formular folgen. Beispiele hierfür sind unter anderem Anwendungsfallbeschreibungen (engl.: *use case descriptions*) mit festgelegten Feldern [Coc00], wie z. B. Name, Zusammenfassung, Ablaufbeschreibung und Vor- und Nachbedingungen, oder Anwender:innenerzählungen (engl.: *user stories*), die zumeist einer Satzschablone wie beispielsweise „As a [who], I want to [what] so that [why]“ folgen.

Neben der Form in der Anforderungen dokumentiert werden, werden Anforderungen oftmals in verschiedene Arten eingeteilt. Diese Klassifikation kann anhand verschiedener Gesichtspunkte geschehen. So wird oftmals das Abstraktionsniveau unterschieden und Anforderungen in Nutzer- und Systemanforderungen eingeteilt [Som12]. Oft wird hierbei auch von grobgranularen (engl.: *high-level*) und feingranularen (engl.: *low-level*) Anforderungen gesprochen. Außerdem werden Anforderungen zumeist in funktionale bzw. nichtfunktionale Anforderungen unterteilt. Hierbei stellen funktionale Anforderungen laut dem Wörterbuch des International Requirements Engineering Board (IREB) [Gli22] eine Anforderung bezüglich eines Ergebnisses oder Verhaltens, das von einer Funktion eines Systems bereitgestellt werden soll, dar. Nichtfunktionale Anforderungen hingegen umfassen entweder Qualitätsanforderungen oder Einschränkungen. Qualitätsanforderungen sind dabei Anforderungen, die sich auf Qualitätsanliegen beziehen, welche nicht durch funktionale Anforderungen abgedeckt sind. Einschränkungen bzw. Randbedingungen sind Anforderungen, die den Lösungsraum über das hinaus einschränken, was notwendig ist, um die gegebenen funktionalen Anforderungen und Qualitätsanforderungen zu erfüllen.

Diese Form der Klassifikation nimmt eine sich gegenseitig ausschließende Trennung von funktionalen und nichtfunktionalen Anforderungen an. Diese Annahme wird in der Forschungsgemeinschaft durchaus kontrovers diskutiert [Gli07; Li+14; EVF16]. Denn in der Praxis können insbesondere natürlichsprachlich dokumentierte Anforderungen sowohl funktionale als auch Qualitätsaspekte des Systems beschreiben. Aus diesem Grund schlagen verschiedene Forscher:innen eine nicht ausschließende Klassifikation vor, welche lediglich festhält, ob eine Anforderung funktionale und/oder Qualitätsaspekte beschreibt [Bro15; EVF16; Dal+19].

2.3. Quelltextinformationen

In diesem Abschnitt werden die in dieser Arbeit vorwiegend verwendeten Quelltextmerkmale im Detail vorgestellt. Auch wenn die Inhalte dieses Abschnitts zum Grundwissen eines jeden Informatikers gehören, können durch eine kurze Beschreibung Unklarheiten in den verwendeten Begrifflichkeiten und Darstellungsformen dieser Arbeit vermieden werden.

2.3.1. Bezeichner

Unter Bezeichnern versteht man die textuelle Oberflächenform, welche den Namen beispielsweise für Typen, Methoden oder Variablen darstellt. Diese Bezeichner werden im Quelltext dazu verwendet, die jeweiligen Elemente zu referenzieren. Darüber hinaus dienen sie Entwickler:innen zum Verständnis des durch sie repräsentierten Elements. Aus diesem Grund empfehlen Richtlinien [Mar08], aussagekräftige Bezeichner zu wählen.

In der Regel dürfen Bezeichner nicht Schlüsselwörtern der verwendeten Programmiersprache entsprechen und keine Leerzeichen enthalten, da diese sonst nicht von anderen strukturgebenden Teilen des Quelltextes unterschieden werden könnten. Um trotzdem eine lesbare Trennung von aus mehreren Wörtern zusammengesetzten Bezeichnern zu schaffen, werden Bezeichner häufig mit Trennsymbolen und Schreibkonventionen, wie z. B. der Binnenmajuskelschreibweise (engl.: *camel case*) oder einer Trennung mittels Unterstrichen (engl.: *snake case*), getrennt. Bei der Binnenmajuskel-Schreibweise wird ein neues Wort wie in `myIdentifizier` durch einen Großbuchstaben gekennzeichnet. Bei der Trennung durch einen Unterstrich würde sich entsprechend `my_identifizier` ergeben. Prinzipiell sind Namenskonventionen nicht festgelegt, doch bestimmte Konventionen sind de-facto Standard für bestimmte Programmiersprachen. So wird für Java-Quelltext zumeist die Binnenmajuskel-Schreibweise verwendet, wohingegen in Python für alle Bezeichner außer Klassen die Trennung mit Unterstrich dominiert.

Ein weiteres Phänomen, welches bei Bezeichnern häufig auftritt, ist der Einsatz von Abkürzungen [New+19]. Diese können zwar die Lesbarkeit des Quelltextes verringern, erhöhen aber die Schreibgeschwindigkeit. Außerdem bedingt die Eindeutigkeit der Bezeichner, dass beispielsweise mehrere Variablen, die dasselbe Objekt halten, unterschiedliche Bezeichner haben müssen. Auch dies ist ein häufiger Fall für Abkürzungen oder Auslassungen. Ein Beispiel hierfür wäre der Bezeichner `myId`, welcher das Wort *identifizier* durch *id* abkürzt.

2.3.2. Quelltextkommentare

Quelltextkommentare dienen dazu, den Zweck und oder das Verhalten eines Quelltextausschnittes zu beschreiben. Obwohl Kommentare ein Bestandteil des Quelltextes sind, tragen sie nicht zu seiner Ausführungssemantik bei, sondern dienen lediglich dem Verständnis bzw. der Dokumentation. Grundsätzlich kann zwischen Block- und Zeilenkommentaren unterschieden werden [Pep07]. Zeilenkommentare entsprechen exakt einer Zeile oder eines Teils einer Zeile des Quelltextes. Blockkommentare hingegen können sich über mehrere Zeilen erstrecken. Im Gegensatz zu den Teilen des Quelltextes, die Ausführungssemantik tragen, werden Kommentare keiner Syntaxprüfung unterzogen und können somit neben natürlichsprachlichen Texten beispielsweise auch Internetadressen, Quelltext oder Sonderzeichen enthalten. Auch Kommentare ohne jeglichen Inhalt sind möglich.

Da es im Quelltext keine Vorschriften dazu gibt, wo ein Kommentar platziert werden muss, können Kommentare prinzipiell an jeder beliebigen Stelle platziert werden. Es gibt aber

Tabelle 2.1.: Ein Auszug der möglichen Etiketten in einem Javadoc-Kommentar. Eine vollständige Liste findet sich in Anhang C

Etikette	Erklärung
@param	Beschreibung eines Parameters
@return	Beschreibung der Rückgabe einer Methode
@see oder @link	Verweis auf einen Typ, eine Methode, Attribut oder Paket.

```

1  /**
2   * Checks whether the given number is even or not.
3   *
4   * @param number the integer to check.
5   * @return true if the number is even, otherwise false.
6   */
7  public boolean isEven(int number) {
8      return number % 2 == 0;
9  }

```

Quelltextausschnitt 2.1: Beispiel eines Javadoc-Kommentars einer Methode

auch für Kommentare, die einen bestimmten Zweck erfüllen sollen, je nach Programmiersprache gewisse Konventionen. So wird für Java ein beschreibender Kommentar für einen Typ, Methode oder ein Attribut zumeist über die Definition dieses Elements geschrieben, wohingegen der beschreibende Kommentar einer Methode in Python zumeist auf dessen Signatur folgt.

Darüber hinaus existieren für viele Programmiersprachen Werkzeuge, die automatisiert aus Kommentaren Dokumentationen, z. B. im HTML-Format, erstellen. Zumeist folgen diese Dokumentationskommentare einer eigenen Syntax und definieren gewisse Etiketten zur Strukturierung des Kommentars und somit auch der resultierenden Dokumentation. Für Java ist dieses Werkzeug Javadoc [Ora]. Javadoc ermöglicht es beispielsweise, Verweise zwischen Elementen des Quelltextes zu definieren und auf bestimmte Elemente der Definitionen zu verweisen. Dies geschieht über eigene Etiketten. Einen Auszug dieser Etiketten ist in Tabelle 2.1 dargestellt. Dokumentationskommentare finden sich vor allem an Typ- und Methodendefinitionen sowie an Attributen. Grundsätzlich können jedoch auch Dokumentationskommentare an beliebigen Stellen im Quelltext platziert werden.

Quelltextausschnitt 2.1 liefert ein Beispiel für einen Dokumentationskommentar in Java. In der zweiten Zeile ist eine allgemeine Beschreibung des Zwecks der Methode angegeben. Die Zeilen 4 und 5 umfassen Beschreibungen des Eingabeparameters bzw. des Rückgabewertes der Methode. Durch die Etiketten am Anfang dieser beiden Zeilen kann erkannt werden, auf welches Element der Methodendefinition sich die Zeile bezieht.

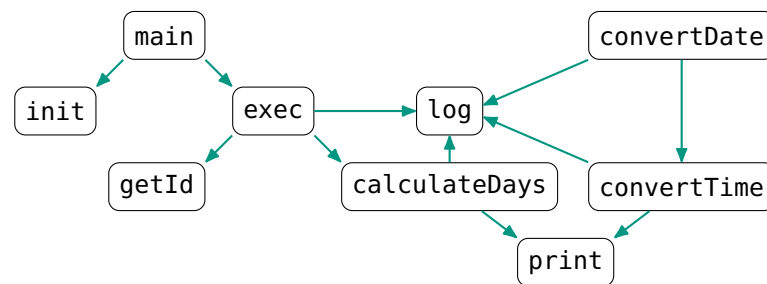


Abbildung 2.2.: Aufrufgraph eines Beispielprogramms

2.3.3. Aufrufgraph

Ein Aufrufgraph ist ein gerichteter Graph, der Aufrufabhängigkeiten zwischen Methoden abbildet [Aho+07]. Hierbei fungieren die Methoden als Knoten des Graphen und die Kanten repräsentieren die Aufrufe. Aufrufgraphen dienen der Visualisierung und Analyse von Aufrufabhängigkeiten zwischen Methoden. Abbildung 2.2 zeigt ein Beispiel für einen Aufrufgraphen. Die Richtung der Pfeile kennzeichnet hierbei, welche die aufrufende und welche die aufgerufene Methode ist. So ruft die `exec`-Methode die Methoden `getId`, `log` und `calculateDays` auf und wird selber nur von der `main`-Methode aufgerufen.

Grundsätzlich kann zwischen dynamischen und statischen Aufrufgraphen unterschieden werden. Dynamische Aufrufgraphen bilden die aufgetretenen Aufrufe eines oder mehrere Programmdurchläufe ab, wohingegen statische Aufrufgraphen alle potenziell möglichen Aufrufabhängigkeiten innerhalb eines Programms abbilden. Da viele moderne Programmiersprachen dynamische Bindung und Reflexion erlauben, ist das Erstellen eines präzisen statischen Aufrufgraphen nicht trivial lösbar, weshalb die meisten Ansätze den Aufrufgraphen überapproximieren.

2.4. Informationsrückgewinnung

Nach Manning et al. [MRS08] ist die Informationsrückgewinnung wie folgend definiert:

Definition 8: Informationsrückgewinnung (engl.: *information retrieval*, IR) [MRS08]

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).⁸

Folgt man dieser Definition so beschäftigt sich das Forschungsfeld der IR mit dem oftmals automatischen Auffinden von Dokumenten zu einem gewissen Informationsbedürfnis.

⁸ Zu Deutsch: Informationsrückgewinnung (IR) ist das Auffinden von Materialien (meistens Dokumente), die unstrukturiert vorliegen (meistens in Textform), um ein Informationsbedürfnis zu einer großen Sammlung von Daten (die meistens auf Computern gespeichert sind) zu befriedigen.

Dieses Informationsbedürfnis wird dabei meist in Form einer Anfrage (engl.: *query*) definiert. Diese ist oftmals ebenso textueller Natur. Verfahren, die der IR zugeordnet werden, bestimmen also basierend auf dieser Anfrage relevante Dokumente aus einer zum Teil großen Menge von möglichen Kandidatendokumenten. Zumeist geschieht dies, indem die Relevanz der Dokumente durch einen Ähnlichkeitsvergleich mit der Anfrage bestimmt wird. Dokumente können dabei sowohl große Textansammlungen wie Bücher, Webseiten oder Veröffentlichungen als auch kleinere Einheiten wie Sätze, Absätze oder Stichworte sein. Außerdem muss ein Dokument nicht aus natürlichsprachlichem Text bestehen, sondern kann z. B. auch in einer Programmiersprache vorliegen. Letztlich wird aber auch in diesem Fall ein textueller Ähnlichkeitsvergleich durchgeführt. IR-Verfahren zeichnen sich außerdem dadurch aus, dass sie im Gegensatz zu Verfahren des maschinellen Lernens ohne eine Anpassung auf die konkrete Aufgabe bzw. die konkreten Dokumente angewandt werden können, welches sie flexibel einsetzbar macht.

2.5. Maschinelles Lernen

Das Forschungsgebiet des maschinellen Lernen (engl.: *machine learning*, ML) beschäftigt sich mit Verfahren und Techniken (und den dahinterliegenden Theorien), welche die Lösung einer Problemstellung selbstständig basierend auf Erfahrungen erlernen, bzw. sich dieser Lösung annähern [Mit97].

Generell lassen sich die Techniken des maschinellen Lernens in die folgenden drei Kategorien einteilen [Mit97]:

- **überwachtes maschinelles Lernen:** Beim überwachten maschinellen Lernen (engl.: *supervised machine learning*) geht es um das Lernen einer Problemlösung basierend auf Musterlösungs-Ein- und Ausgabepaaren. Anhand dieser Beispiele wird ein Modell, welches die Zuordnung von Eingabe zur Ausgabe erlernt, erzeugt. Ein solches trainiertes Modell ist daraufhin in der Lage, für zukünftige Eingaben potenzielle Ausgaben vorherzusagen. Der Prozess des Trainierens eines solchen überwachten Modells wird zumeist als Trainingsphase (engl.: *training phase*) und die zum Trainieren genutzten Ein- und Ausgabepaare als Trainingsdaten (engl.: *training data*) bezeichnet. Es wird bei dieser Art von maschinellem Lernen von überwachtem maschinellen Lernen gesprochen, da Musterlösungs-Ein- und Ausgabepaare zur Verfügung stehen und somit ebenso die Art der Ausgaben bekannt ist.
- **unüberwachtes maschinelles Lernen:** Beim unüberwachten maschinellen Lernen (engl.: *unsupervised machine learning*) wird ausschließlich auf Basis der Eingabedaten gelernt. Hierzu wird in den Eingabedaten nach Strukturen bzw. Gemeinsamkeiten und Unterschieden gesucht und diese als Ausgabe repräsentiert. Man spricht hier von unüberwachtem maschinellen Lernen, da ohne zu erwartende Ausgaben trainiert wird. Die am häufigsten genutzte Form des unüberwachten maschinellen Lernens stellt das Gruppieren der Eingabedaten (engl.: *clustering*) dar. Hierbei erlernt das Modell Gruppenzugehörigkeiten basierend auf den Eigenschaften der Eingabedaten.

- **bestärkendes maschinelles Lernen:** Beim bestärkenden maschinellen Lernen (engl.: *reinforcement learning*) wird in den Lernprozess eine Rückmeldung der Umgebung miteingebunden. Hierzu werden initiale Modelle genutzt, um eine erste Vorhersage der Ausgabe zu produzieren. Diese Ausgabe wird daraufhin der Umgebung (ein weiteres Programm oder manchmal auch ein/e menschlicher Nutzer:in) vorgelegt, welches Rückmeldung wie z. B. ein einfaches richtig oder falsch, zurückspielt. Das Modell passt sich daraufhin automatisch an die gegebene Rückmeldung an.

Abgeleitet von diesen Kategorien existieren weitere Verfeinerungen. So nutzt beispielsweise halb-überwachtes maschinelles Lernen (engl.: *semi-supervised machine learning*) eine geringe Anzahl an Musterlösungs-Ein- und Ausgabepaaren, um ein erstes Modell zu trainieren, welches die potenziell große Menge an weiteren Trainingsdaten automatisch klassifiziert. Daraufhin wird das Modell auf den gesamten Daten neu trainiert.

2.5.1. Überwachtes maschinelles Lernen

Zur Kategorie des überwachten maschinellen Lernens (engl.: *supervised machine learning*) zählen Techniken und Verfahren, die, aufgrund von vorhandenen Musterlösungs-Ein- und Ausgabepaaren, ein Modell trainieren, was den Zusammenhang zwischen Eingabe und Ausgabe erlernt. Ein solches Musterlösungspaar wird als Trainingsinstanz (engl.: *training instance*) bezeichnet und die Menge aller dieser Trainingsinstanzen bildet die Trainingsdaten. Die Trainingsinstanzen werden dabei zumeist nicht in ihrer unverarbeiteten Form verwendet, sondern durch eine Vorverarbeitung in eine für statistische Verfahren geeignete Form überführt. Die Eingabe wird hierbei auf eine Menge von Merkmalen (engl.: *features*) abgebildet und diese durch einen Merkmalsvektor (engl.: *feature vector*) repräsentiert. So könnte beispielsweise ein Wort eines Satzes durch die Merkmale Satzposition, Wortart, umgebende Worte und das Wort selbst repräsentiert werden. Die Ausgabe wird meist entweder durch eine konkrete Klasse oder einen kontinuierlichen numerischen Wert repräsentiert, je nachdem, ob es sich bei der Problemstellung um ein Klassifikations- oder ein Regressionsproblem handelt. Unabhängig von der Art des Problems versuchen überwachte Lernverfahren Muster bzw. Strukturen in den Merkmalen der Eingabedaten zu finden, die es ihnen erlauben, die Ausgabe vorherzusagen und diese durch Modellierung explizit zu machen.

Klassifikationsprobleme werden anhand der Anzahl möglicher Ausgaben kategorisiert. Soll zwischen zwei Klassen bzw. der Zugehörigkeit zu einer Klasse unterschieden werden, spricht man von einem binären Klassifikationsproblem (engl.: *binary classification*). Kommen hingegen mehrere Klassen als mögliche Ausgaben infrage, wird dies als Mehrklassen-Klassifikation (engl.: *multiclass classification*) bezeichnet. Zusätzlich kann ein Klassifikationsproblem auch bezüglich der Menge an gleichzeitig vergebenen Klassen pro Instanz kategorisiert werden. Wenn mehr als eine Klasse pro Instanz zugelassen ist, spricht man von einer Mehrfachetiketten-Klassifikation (engl.: *multi-label classification*).

Einen großen Einfluss auf die Qualität des erlernten Modells hat die Menge an zur Verfügung stehenden Trainingsdaten. Je nach verwendeten Verfahren und Eindeutigkeit der Merkmale für den vorherzusagenden Zusammenhang wird eine erhebliche Menge an Trainingsinstanzen benötigt. In den allermeisten Fällen gilt, je mehr unterschiedliche Trainingsinstanzen zur Verfügung stehen, desto besser kann das Verfahren das Modell feinstjustieren und auf unterschiedliche Eingaben generalisieren. Da allerdings zumeist die Musterlösungsausgaben händisch generiert werden müssen, stehen nicht für alle Problemstellungen ausreichend Trainingsdaten zur Verfügung, welches die Anwendbarkeit von überwachten Lernverfahren limitiert.

Ein weiterer wichtiger Faktor ist die Verteilung der Klassen über die Trainingsinstanzen. Dominiert eine Klasse die Menge der Trainingsinstanzen stark, kann es sein, dass das Lernverfahren sich an die Eingaben überanpasst (engl.: *overfitting*). Dies bedeutet, dass das Verfahren die Klassenzugehörigkeit „auswendig“ lernt und somit ein schlechter generalisierendes Modell erlernt wird. Daher möchte man idealerweise eine möglichst gleichverteilte Aufteilung der Klassen über die Trainingsinstanzen erzeugen.

Aufgrund des Mangels an Trainingsdaten ist dies aber nicht immer möglich, weshalb oftmals Verfahren zur Mitigierung dieses Effektes angewandt werden. Eine Möglichkeit, im Fall einer stark unausgewogenen Klassenverteilung einer Überanpassung vorzubeugen, stellen Abtaststrategien dar. Die einfachsten Abtaststrategien für binäre Klassifikationsprobleme sind das Über- bzw. Unterabtasten (engl.: *over-/undersampling*). Bei der Überabtastung wird die in der Minderheit vorliegende Klassenpopulation wiederholt dem Trainingsdatensatz hinzugefügt, bis diese der Menge der dominanten Klasseninstanzen entspricht. Bei der Unterabtastung werden hingegen aus der Menge der dominanten Klasseninstanzen (zumeist zufällig) Instanzen entfernt, bis die Population der Größe der Minderheit entspricht. Somit reduziert die Unterabtastung die Größe der dominierenden Klassenpopulation und belässt die Minderheit unverändert, wohingegen die Überabtastung die Größe der Minderheitspopulation vergrößert und die Instanzen der dominierenden Klasse in den Trainingsdaten beibehält.

Je nach Art des Problems und den vorhandenen Ressourcen eignen sich unterschiedliche Verfahren zur Bestimmung eines Modells. In den letzten Jahren haben Verfahren basierend auf künstlichen neuronalen Netzen in vielen Problemstellungen überzeugende Fortschritte erreicht. Allerdings benötigen diese eine sehr große Menge an Trainingsdaten, weshalb für manche Anwendungsfälle noch immer klassischere Verfahren wie ein Naïve-Bayes Klassifikator (engl.: *Naïve Bayes classifier*, NBC), eine Stützvektormachine (engl.: *support vector machine*, SVM) oder logistische Regression (engl.: *logistic regression*, LG) verwendet werden.

Naïve-Bayes Klassifikator Beim Naïven-Bayes Klassifikator basiert die Zugehörigkeit zu einer Klasse auf einem einfachen probabilistischen Modell. Hierbei wird angenommen, dass die Wahrscheinlichkeit P für eine Klasse k nur von den Merkmalen der Eingabe abhängt. Mithilfe des Satzes von Bayes lässt sich das Modell daraufhin wie folgt definieren:

$$P(k|\vec{e}) = \frac{P(k) \cdot P(\vec{e}|k)}{P(\vec{e})} \quad (2.1)$$

Die Eingabe wird durch einen Merkmalsvektor \vec{e} repräsentiert. Um die Berechnung der Wahrscheinlichkeiten zu vereinfachen, wird „naiv“ angenommen, dass die einzelnen Merkmale unabhängig voneinander sind und der Zähler, welcher nicht von k abhängt, durch einen konstanten Faktor s ersetzt. Hieraus ergibt sich die folgende Formel:

$$P(k|\vec{e}) = \frac{1}{s} \cdot P(k) \prod_{i=1}^n P(e_i|k) \quad (2.2)$$

Stützvektormachine Verfahren die eine Stützvektormachine (engl.: *support vector machine*, SVM) nutzen, versuchen die Trainingsinstanzen mittels einer Hyperebene in zwei Klassen zu trennen. Es wird also aus den Trainingsdaten eine Hyperebene bestimmt, die die Trainingsinstanzen bezüglich ihrer Klassenzugehörigkeit möglichst gut trennt. Gut bedeutet in diesem Fall die bestmögliche Trennung und einen maximalen Abstand zu allen Merkmalsvektoren. Beschrieben wird die resultierende Hyperebene durch die namensgebenden Stützvektoren. Diese Methode nutzt keine Wahrscheinlichkeitsverteilungen und ist somit kein probabilistisches Modell. SVMs können nur lineare Klassifikationsprobleme lösen und sind auf binäre Klassifikationen beschränkt. Durch eine Projektion in höherdimensionale Räume können aber auch nicht-lineare Probleme gelöst werden.

Logistische Regression Die logistische Regression (engl.: *logistic regression*, LG) klassifiziert Eingaben basierend auf einer Regressionskurve. Diese wird aus den Trainingsdaten gebildet, indem die Parameter einer logistischen Funktion bestimmt werden. Wie bei der Stützvektormachine dient die resultierende Kurve als Trennung der Klassen bezüglich ihrer Merkmale. Logistische Regression kann für binäre wie auch Mehrklassen-Klassifikation genutzt werden und erzielt in vielen Anwendungsfällen, z. B. in der Computerlinguistik, von den klassischen Verfahren die besten Ergebnisse. Dies lässt sich unter anderem damit begründen, dass im Gegensatz zu einem Naïve-Bayes Klassifikator nicht angenommen wird, dass die Merkmale unabhängig voneinander sind. Allerdings benötigt ein auf logistischer Regression basierender Klassifikator deutlich höheren Trainingsaufwand als beispielsweise ein Naïve-Bayes Klassifikator.

2.5.2. Künstliche neuronale Netze

Als künstliches neuronales Netz (engl.: *artificial neural network*, ANN) werden überwachte maschinelle Lernverfahren bezeichnet, die auf Basis von miteinander vernetzten Neuronen arbeiten [Mit97; Bis06]. Ein solches Neuron ist eine Verarbeitungseinheit, welche eine Menge von Eingaben erhält, diese gewichtet, zusammenfasst (meist durch eine Summe) und einer nicht-linearen Funktion übergibt, um eine Ausgabe zu erzeugen. Diese nicht-lineare

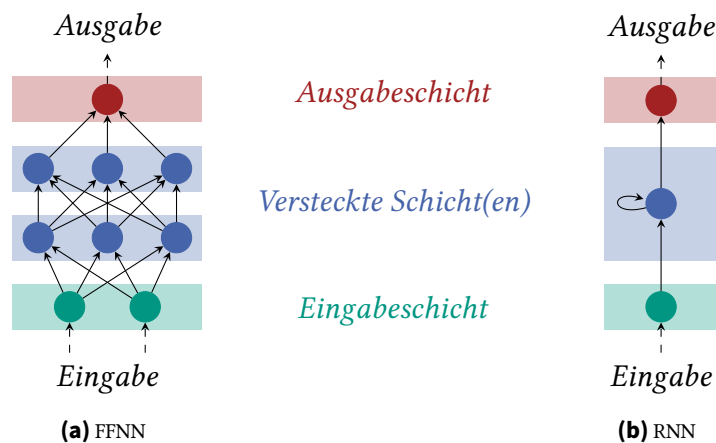


Abbildung 2.3.: Schematischer Aufbau verschiedener Topologien für künstliche neuronale Netze

Funktion wird häufig auch Aktivierungsfunktion (engl.: *activation function*) genannt. Die am häufigsten eingesetzten Aktivierungsfunktionen sind die Sigmoidfunktion und die Softmax-Funktion. Verbindungen zwischen Neuronen sind gerichtet. In neuronalen Netzen werden Neuronen üblicherweise in Schichten angeordnet, sodass nur Neuronen unterschiedlicher (meist aufeinanderfolgender) Schichten verknüpft werden.

Künstliche neuronale Netze werden zumeist dadurch trainiert, dass die Gewichte der Verbindungen sowie ggf. weitere anpassbare (Schwellen-)Werte, auf Basis der Trainingsdaten automatisch angepasst werden [Mit97]. Beginnend mit zufällig zugewiesenen Gewichten wird für die Trainingsdatenmenge anhand einer Fehlerfunktion der Abstand zur richtigen Lösung bestimmt. Ausgehend von diesem wird mit Hilfe des sogenannten Rückpropagierungsverfahrens (engl.: *backpropagation*) die Gewichtungen aller Verbindungen ausgehend von der letzten Schicht angepasst. Hierzu wird das Gradientenabstiegsverfahren zur Minimierung des Fehlers verwendet.

2.5.2.1. Vorwärts gerichtete neuronale Netze

Die einfachste Form des Aufbaus (auch Topologie genannt) eines künstlichen neuronalen Netzes stellen vorwärts gerichtete neuronale Netze (engl.: *feed-forward neural networks*, FFNNs) dar. Diese geben Neuronenausgaben nur in Verarbeitungsrichtung (vorwärts) an weitere Neuronen weiter. Prinzipiell können FFNNs aus beliebig vielen Schichten bestehen, welche üblicherweise vollständig miteinander verknüpft sind; also alle Neuronen der vorangehenden Schicht werden mit allen Neuronen der darauffolgenden Schicht verbunden. Im Falle eines einschichtigen FFNNs besteht das Netz nur aus einer sogenannten Ausgabeschicht, welche basierend auf der Eingabe die Ausgabe des Modells generiert. Ab zwei Schichten, wird zusätzlich zur Ausgabeschicht die erste Schicht des Netzes als Eingabeschicht bezeichnet. Diese erhält die Eingabe, zumeist gegeben als Merkmalsvektor, und reicht diese an die nächste Schicht weiter. Sollte das Netz weitere Schichten zwischen Ein- und Ausgabeschicht besitzen, werden diese Zwischenschichten auch versteckte Schichten (engl.: *hidden layer*) genannt, da von außen nicht mit ihnen interagiert werden

kann. Abbildung 2.3a zeigt eine mögliche Topologie eines FFNNs mit zwei versteckten Schichten zu je drei Neuronen.

2.5.2.2. Rekurrente neuronale Netze

Im Gegensatz zu FFNNs können in rekurrenten neuronalen Netzen (engl.: *recurrent neural network*) Informationen, sprich Neuronenausgaben, auch an Neuronen derselben oder sogar vorhergehenden Schichten weitergereicht werden. Dies ermöglicht es dem Netz, auch auf Informationen vorangehender Berechnungen zurückzugreifen und somit auch auf Reihenfolgeabhängigkeiten einzugehen. Abbildung 2.3b zeigt die einfachste Form eines RNN mit einem Neuron in der versteckten Schicht, welches als Eingabe sowohl die Informationen aus der Eingabeschicht als auch das letzte Ergebnis seiner eigenen Berechnung erhält. Einem RNN können Eingaben in einer Art Sequenz nacheinander präsentiert werden. Dies ermöglicht es der Berechnung in Zeitschritt $x + 1$ auf das Ergebnis von Zeitschritt x zurückzugreifen. Dies ist ein Grund, warum RNNs gerade in der Verarbeitung von natürlicher Sprache häufig eingesetzt werden, da eine Texteingabe als eine Sequenz von Wörtern aufgefasst werden kann.

Eine spezielle Form eines RNN ist ein RNN mit langem Kurzzeitgedächtnis (engl.: *long short-term memory*, LSTM) [HS97]. Diese versprechen aufgrund spezieller, als Kette angeordneter Zellen, einen besseren Umgang mit Langzeitabhängigkeiten. Diese speziellen Zellen reichen explizit eine Zellzustand (engl.: *cell state*) genannte Information von einer zur nächsten Zelle weiter. Jede Zelle kann entscheiden, welche Informationen hinzugefügt, entfernt oder unverändert weitergereicht werden. Dies geschieht über sogenannte Tore (engl.: *gates*), speziell verknüpfte Neuronen der Zelle. Jede Zelle hat hierbei drei Eingaben: Die aktuelle Eingabe von außen (z. B. das aktuelle Wort), der versteckte Zustand des vorherigen Zeitschritts dieser Zelle und der Zellzustand der vorangehenden Zelle. Zuerst entscheidet das Tor des Vergessens (engl.: *forget gate*), auf Basis der äußeren Eingabe und des versteckten Zustands des vorherigen Zeitschritts, welche Informationen des Zellzustandes vergessen werden sollen. Daraufhin entscheidet das Eingangstor (engl.: *input gate*), welche Teile des Zellzustands auf Basis der Eingabe aktualisiert werden sollen. Abschließend wird auf Basis des neuen Zellzustands, der Eingabe und dem versteckten Zustand mittels des Ausgabeters (engl.: *output gate*) die Ausgabe der Zelle generiert und zusammen mit dem Zellzustand weitergereicht. Beim Trainieren eines LSTMs müssen also die Gewichte der speziellen Verknüpfungen für Eingabe, Ausgabe und Vergessen erlernt werden.

Soll neben der Eingabe auch die Ausgabe eines RNNs eine Sequenz darstellen, wird häufig die Enkodierer-Dekodierer-Architektur (engl.: *encoder-decoder* oder *seq2seq*) verwendet. Hierbei wird die Aufgabe des Repräsentierens der Eingabe, dem Enkodieren, von der Aufgabe des Erzeugens der Ausgabe, dem Dekodieren, getrennt. Der Enkodierer besteht aus einer Menge von rekurrenten Einheiten die jeweils ein Element der Eingabesequenz verarbeiten und diese Information an die jeweils nächste Einheit weitergeben. Die versteckte Ausgabe der letzten rekurrenten Einheit wird daraufhin als Eingabe für die erste Einheit des Dekodierers verwendet. Der Dekodierer besteht wiederum aus einer Menge von rekurrenten

Einheiten, welche auf Basis des versteckten Ausgabe der jeweils vorhergehenden Einheit eine Ausgabe, z. B. ein Wort, produzieren. Ein Vorteil dieses Vorgehens liegt darin, dass Sequenzen unterschiedlicher Länge aufeinander abgebildet werden können und potenziell der Enkodierer bzw. Dekodierer je nach Aufgabe ausgetauscht werden kann. Daher werden Enkodierer-Dekodierer häufig in der maschinellen Übersetzung angewandt.

Hyperparameter Neben der Topologie des Netzes spielen außerdem weitere Eigenschaften beim Entwurf eines neuronalen Netzes eine große Rolle. So müssen je nach Art des Netzes bestimmte Hyperparameter genannte Eigenschaften gewählt werden. Im Folgenden wird eine Auswahl der wichtigsten Hyperparameter gegeben [Mit97; GBC16]:

- **Anzahl der Neuronen einer Schicht:** Beim Entwurf eines neuronalen Netzes muss für jede versteckte Schicht die Anzahl der enthaltenen Neuronen definiert werden. Für Eingabe- und Ausgabeschicht wird die Anzahl der Neuronen von der Eingabe bzw. der Art der gewünschten Ausgabe bestimmt. So hat beispielsweise eine Ausgabeschicht zur binären Klassifikation nur ein Neuron.
- **Epochenanzahl:** Aufgrund der Beschaffenheit des Rückpropagierungsverfahrens reicht für das Trainieren eines ANNs ein einfaches Präsentieren aller Trainingsdaten meist nicht aus, um ein gutes Modell zu erhalten. Deshalb werden die Trainingsdaten dem Netz oftmals mehrfach zum Zweck des Trainierens präsentiert. Die Epochenanzahl (engl.: *epoch number*) gibt hierbei an, wieviele dieser Durchläufe, Epochen genannt, das Netz (maximal) zum Trainieren nutzt. Generell gilt, dass eine höhere Epochenanzahl eine bessere Feinanpassung der Verbindungsgewichte erlaubt, aber auch das Risiko einer Überanpassung des Modells birgt.
- **Stapelgröße:** Die Stapelgröße (engl.: *batch size*) gibt die Anzahl der Trainingsinstanzen an, welche dem Netz präsentiert werden, bevor eine Anpassung der Gewichte erfolgt. Für eine Epoche werden die Trainingsdaten in N Stapel der gewählten Größe unterteilt und somit N mal die Gewichtung angepasst. Oftmals ergeben kleinere Stapelgrößen besser angepasste Modelle; sie führen aber auch zu einem deutlich höheren Trainingsaufwand.
- **Lernrate:** Mit Lernrate (engl.: *learning rate*) wird ein Parameter des Gradientenabstiegs bezeichnet. Sie bestimmt die Schrittweite pro Iteration mit der sich dem Minimum der Verlustfunktion angenähert wird. Wird die Lernrate zu hoch gewählt, kann es passieren, dass das Verfahren über Minima springt und damit suboptimale Ergebnisse erzielt oder gar nicht konvergiert. Wählt man die Lernrate allerdings zu niedrig, konvergiert das Verfahren langsamer und kann in lokalen Minima feststecken bleiben.

2.6. Sprachwissenschaftliche Grundbegriffe

Ein Großteil der in dieser Arbeit betrachteten Informationen und Dokumente liegt in natürlicher Sprache vor oder umfasst Elemente in natürlicher Sprache, wie z. B. die Bezeichner des Quelltextes. Da sich außerdem die meisten Verfahren zur Wiederherstellung von Nachverfolgbarkeitsverbindungen auf die natürlichsprachlichen Eigenschaften und Zusammenhänge der nachverfolgbaren Artefakte beziehen, werden in diesem Abschnitt wichtige Grundbegriffe aus den Sprachwissenschaften eingeführt. Diese sind notwendig, um ein einheitliches Verständnis von Eigenschaften natürlicher Sprache zu erhalten und somit über diese sprechen und diese analysieren zu können.

2.6.1. Natürliche Sprache

In der Sprachwissenschaft bezeichnet natürliche Sprache die geäußerte Form der Kommunikation des Menschen (textuell oder anderweitig), welche basierend auf einer historischen Entwicklung einer Verständigungsweise entstanden ist [Bus02]. Ihr gegenüber stehen künstliche Sprachen, welche gezielt für einen Verwendungszweck erschaffen wurden und somit nicht diese Form der Entwicklung erfahren haben. Programmiersprachen wären Beispiele für künstliche Sprachen. Im Gegensatz zu künstlichen Sprachen existieren in natürlichen Sprachen zumeist lexikalische und strukturelle Mehrdeutigkeiten, welche dazu führen, dass die Interpretation ihrer Ausdrücke einer gewissen Vagheit unterliegt [Bus02]. Natürliche Sprachen besitzen ein Regelwerk, die sogenannte Grammatik, welches vorgibt, welche Zeichenfolgen zugelassen sind (Syntax) und was deren Bedeutung ist (Semantik).

2.6.2. Syntax

Die Syntax einer natürlichen Sprache, definiert das Regelsystem zum Bau von Texteinheiten, wie Wörtern und Sätzen. Sie definiert, in welcher Form Zeichen zu Wörtern und diese zu übergeordneten Einheiten wie Phrasen-, Teilsätzen oder ganzen Sätzen zusammengesetzt werden dürfen [Bus02]. Außerdem bildet die Syntax zusammen mit der Semantik und Pragmatik einen der Teilbereiche der Zeichentheorie, der Semiotik [Mor38]. Sie beschreibt die Anordnung und Beziehung von Zeichen untereinander [Bus02].

2.6.3. Semantik

In der Semiotik bezeichnet Semantik die Lehre von der Beziehung zwischen Zeichen und ihrer Bedeutung. In der Sprachwissenschaft allgemein bezieht sich der Begriff allerdings auf die Lehre von der wörtlichen Bedeutung von sprachlichen Ausdrücken [Bus02]. Die Semantik beschäftigt sich also mit denjenigen Beziehungen zwischen natürlichsprachlichen Einheiten (Zeichen, Wörter, Phrasen usw.), welche die Bedeutung ausmachen. Dies umfasst die Bedeutung eines Wortes im Satzkontext, die Bedeutung einer Phrase und die Bedeutung

eines gesamten Satzes. So kann das Wort *Bank* je nach Satzkontext sowohl die Bedeutung Sitzgelegenheit, als auch Geldinstitut einnehmen. Die Semantik beschäftigt sich weiterhin damit, wie die Bedeutung komplexer Ausdrücke aus den Bedeutungen der sie bildenden Einheiten zusammengesetzt werden kann, also wie z. B. die Bedeutung einer Phrase aus den Teilbedeutungen der enthaltenen Elemente bestimmt werden kann.

2.6.4. Pragmatik

Den Zweck und die Bedeutung einer Äußerung im jeweiligen Äußerungskontext untersucht die Pragmatik. Sie beschäftigt sich mit Zusammenhängen, die sich nur aus dem Kontext eines Diskurses oder eines zusammenhängenden Dokumentes ergeben. So kann der Satz „*You have to read this book!*“ sowohl als Befehl als auch als Empfehlung gemeint sein, je nachdem, ob er von einem Lehrer an einen Schüler oder durch einen Freund geäußert wurde.

Die drei Ebenen Syntax, Semantik und Pragmatik hängen voneinander ab [Oll72]. Ohne ein Wissen über syntaktische Regeln ist eine semantische Interpretation von Phrasen und Sätzen nicht möglich. Ebenso wird eine Interpretation der Äußerungsabsicht erst ermöglicht, sobald die Bedeutung der Sätze bestimmt ist.

2.6.5. Lexikalische Semantik

Mit lexikalischer Semantik oder auch Wortsemantik wird innerhalb der Semantik das Feld der Bedeutungslehre einzelner lexikalischer Zeichen bezeichnet [Bus02]. Hierbei stehen die verschiedenen Bedeutungen von Wörtern und ihre bedeutungsgebundenen Beziehungen im Fokus. Wörter werden hierbei durch *Lexeme* repräsentiert. Ein *Lexem* stellt eine Bedeutungseinheit dar und besteht aus einer Paarung eines Wortes, unabhängig von der konkreten Form bzw. syntaktischen Funktion, mit seiner Bedeutung [Bus02]. Als Repräsentant eines Lexems fungiert das *Lemma*. Das Lemma ist die Grundform des Wortes und auch die Form, welche man in einem Lexikon oder sonstigem Nachschlagewerk finden würde (Zitierform) [Bus02]. So wäre das Lemma von *singe* die Grundform *singen* und (ich) *singe*, (du) *singst* und (er) *singt* teilen sich die gleiche Bedeutung und somit das gleiche Lexem.

Außerdem beschäftigt sich die lexikalische Semantik mit Beziehungen zwischen den Bedeutungen natürlichsprachlicher Ausdrücke. Diese sogenannten *semantischen Relationen* setzen die Bedeutungen der Ausdrücke in Beziehung zueinander und ordnen damit den Ausdrücken semantische Zusammenhänge zu. Im Folgenden wird ein kurzer Überblick über die für diese Arbeit relevanten semantischen Relationen gegeben.

Synonymie Zwei sprachliche Ausdrücke werden als *synonym* bezeichnet, wenn sie dieselbe Bedeutung tragen. Synonyme können hierbei aus einfachen Variationsmöglichkeiten in der Sprache resultieren, wie z. B. bei „erhalten und bekommen“, oder aber ihren Ursprung in regionalen (Pilz und Schwammerl), soziodialektalen (Geld und Moneten), stilistischen (Raum und Gemach) oder fachsprachlichen (bedeutungsgleich und synonym) Gegebenheiten haben [Bus02]. In der Sprachwissenschaft wird dabei zwischen perfekten und nicht-perfekten *Synonymen* unterschieden. Können zwei sprachliche Ausdrücke in jedem Kontext sinnerhaltend gegeneinander ausgetauscht werden, gelten sie als perfekte Synonyme.

Hyponymie und Hyperonymie Hyponymie bezeichnet die inhaltliche Spezialisierung eines Ausdrucks. Somit ist z. B. Wal *hyponym* zu Säugetier, weil die Bedeutung von Wal spezifischer ist als die von Säugetier, aber jeder Wal ein Säugetier ist. Die umgekehrte Richtung, also die inhaltliche Generalisierung, wird als *Hyperonym* bezeichnet. Somit lassen sich mittels dieser semantischen Relationen hierarchieähnliche Gliederungen von Ausdrücken aufbauen [Bus02].

Meronymie und Holonymie Die Meronymie ist eine weitere hierarchiebildende semantische Relation [Bus02]. Sie repräsentiert eine Teil-Ganzes-Beziehung. Hierbei ist ein Ausdruck ein *Meronym* eines anderen, falls seine Bedeutung ein Teil der Bedeutung des anderen Ausdrucks ist [Bus02]. So ist beispielsweise Reifen ein Meronym von Auto. Umgekehrt ist hierbei Auto ein *Holonym* von Reifen, weil es die Entität darstellt, welche den Reifen beinhaltet. Im Allgemeinen sind Meronymie und Holonymie asymmetrisch und nicht transitiv (Das Haus hat eine Tür. Die Tür hat einen Griff. Aber das Haus hat keinen Griff).

2.7. Wissensquellen

Im Rahmen dieser Arbeit sind zwei Arten von mit dem Computer verarbeitbaren Wissensquellen relevant: Quellen die sprachliches Wissen wie z. B. semantische Relationen beinhalten (WordNet usw.) und Quellen die Weltwissen zur Verfügung stellen (Wikipedia bzw. DBpedia).

2.7.1. Ontologien

Mit dem Begriff Ontologie wird in der elektronischen Datenverarbeitung eine Form der Wissensrepräsentation bezeichnet. Der Begriff lehnt sich dabei an den gleichnamigen Begriff für einen Fachbereich der Philosophie an, welcher sich mit allem Seienden beschäftigt. Die Verbindung besteht darin, dass konkret oder abstrakt existierende Dinge,

genannt Entitäten, sowie ihre strukturellen und semantischen Beziehungen untereinander betrachtet werden. Gruber [Gru93] definiert Ontologien im Kontext der Informatik folgendermaßen:

Definition 9: Ontologie (engl.: *ontology*) nach Gruber [Gru93]

An ontology is an explicit specification of a shared conceptualization.⁹

Ontologien stellen also Wissen zu einer Domäne explizit bereit. In einer Ontologie werden einzelne Wissens-elemente, Individuen genannt, unter gemeinsamen Konzepten (auch als Klassen bezeichnet) zusammengefasst. Diese Konzepte können wiederum unter anderen Konzepten zusammengefasst werden, sodass eine hierarchische Struktur entsteht. Diesen Prozess bezeichnet man als Konzeptbildung (engl.: *conceptualization*). Zusätzlich zu den einzelnen Wissens-elementen und Konzepten können in Ontologien Relationen zwischen den Elementen abgebildet werden, die über die hierarchische Struktur der Konzeptbildung hinausgehen. Relationen können uni- oder bidirektional sein. Außerdem können Konzepten und Individuen durch den Einsatz von Datenrelationen auch Attribute und Werte zugewiesen werden. Die Kernelemente einer Ontologie sind also Konzepte, Individuen und Relationen.

2.7.2. WordNet

Bei WordNet [Mil95; FM98] handelt es sich um eine lexikalische Datenbank der englischen Sprache. Sie enthält Informationen über semantische Relationen zwischen Bedeutungen und baut aus diesen Beziehungen ein hierarchisches lexikalisches Netz auf. Grundeinheit der Datenbank bilden sogenannte *Synsets*. Ein Synset besteht aus einer Menge von Synonymen, die eine Bedeutung teilen und stellt somit eine Bedeutungseinheit dar. Außerdem besitzt jedes Synset eine Beschreibung in Textform, der sogenannten *Glosse*. Ein Begriff (repräsentiert durch sein Lemma) kann Teil mehrerer Synsets sein, wenn er mehrere Bedeutungen besitzt. Beispielweise ist der englische Begriff *bass* Teil der folgenden Synsets:

[bass, bass voice, basso]

– Glosse: *the lowest adult male singing voice*

[bass, sea bass]

– Glosse: *the lean flesh of a saltwater fish of the family Serranidae*

Dies bedeutet der Begriff *bass* teilt mindestens die beiden Bedeutungen: tiefste männliche Gesangsstimme und fettarmes Fleisch eines Salzwasserfisches aus der Familie der Serranidae (sowie auch des Fisches selbst).

⁹ Zu Deutsch: Eine Ontologie ist die explizite Spezifikation gemeinsamer Konzeptbildung.

WordNet unterscheidet bei der Einteilung der Lemmata in Synsets zwischen den vier Wortarten Substantiv, Verb, Adjektiv und Adverb. Neben den Synonymen umfasst WordNet Informationen zu weiteren semantischen Relationen zwischen den Bedeutungen von Substantiven und Verben. Am wichtigsten sind hierbei wohl die hierarchiebildenden Relationen. So werden Hyperonym- oder Hyponym-Beziehungen sowie Teil-Ganzes Beziehungen (Meronymie und Holonymie) zwischen den Synsets beschrieben, welche jeweils eine Form von Hierarchie der Bedeutungen abbilden.

WordNet umfasst in der aus dem Jahre 2011 stammenden aktuellsten Version 3.1 117791 *Synsets*, davon 82192 für Nomen, 13789 für Verben, 18185 für Adjektive und 3625 für Adverbien¹⁰.

2.7.3. Wikipedia und DBpedia

Bei der Wikipedia handelt es sich um eine offene Online-Enzyklopädie der *Wikimedia Foundation Inc.*, welche durch ihre Nutzer:innen gepflegt und erweitert wird. Grundelement der Wikipedia sind die Artikel (oder Inhaltsseiten), welche sich jeweils einem Thema widmen und Informationen zu diesem Thema zusammentragen. Hierbei erhalten verschiedene Bedeutungen eines Begriffs zumeist eigene Artikel. So existieren die Artikel *Bass (voice type)* und *Bass (fish)*, korrespondierend zu den beiden Synsets aus Abschnitt 2.7.2. Die Artikel verweisen aufeinander, indem Begriffe, die in der Beschreibung eines Artikels auftreten, mittels eines Verweises auf ihren eigenen Artikel versehen sind. Außerdem sind die Artikel mit Kategorien versehen, welche sie einem Themenbereich zuordnen. Diese Kategorien sind wiederum in Über- und Unterkategorien eingeordnet und bilden somit eine inhaltliche Systematik und Begriffshierarchie.

Durch die kontinuierliche Erweiterung und die Möglichkeit eines jeden zur Wikipedia beizutragen, ist sie heute die umfangreichste Enzyklopädie. Sie umfasst aktuell über 6,6 Millionen englischsprachliche Inhaltsseiten¹¹. Wegen ihres großen Umfangs und der relativen Aktualität der Artikel wird die Wikipedia für viele Aufgaben der Sprachverarbeitung als Daten- und Wissensbasis verwendet.

Das DBpedia-Projekt hat zum Ziel, die in der Wikipedia enthaltenen Informationen in eine elektronisch verarbeitbare, strukturierte Form von Wissensgraphen umzuwandeln [Leh+15]. Hierzu werden teils automatisch, teils durch Nutzerbeteiligung die Artikel und ihre Relationen untereinander als Faktenwissen abgebildet. Dies ermöglicht es, algorithmische Abfragen der Relationen durchzuführen und somit z. B. die Hierarchie der Kategorien nutzbar zu machen. Zusätzlich wurden mithilfe von Nutzerbeteiligung weitere Relationen und Einordnungen der durch die Artikel bestimmten Konzepte durchgeführt. Die dem Wissensgraph zugrundeliegende Ontologie umfasst in der aktuellen Version über 4,8 Millionen

¹⁰ Informationen aus der WordNet 3.1-Variante des Natural Language Toolkits (NLTK) <https://www.nltk.org/>, zuletzt besucht am 26.04.2023.

¹¹ <https://en.wikipedia.org/wiki/Special:Statistics?action=raw>, zuletzt besucht am 26.04.2023.

The image shows the sentence "The system shall return the current time and date." where each word and the period are enclosed in a light blue rounded rectangular box, illustrating the process of tokenization.

Abbildung 2.4.: Portionierung des Beispielsatzes „The system shall return the current time and date.“

Individuen in 768 Klassen und 3000 Eigenschaften¹² Diese Klassen bilden zusammen eine Hierarchie ähnlich zu der von WordNet, welche aber gewollt flacher ausgelegt ist, um sie besser visualisierbar und navigierbar zu machen [Leh+15]. So umfasst sie Beziehungen wie *subClassOf*, *domain* oder *member*.

2.8. Sprachverarbeitung

Mit Sprachverarbeitung bzw. Verarbeitung natürlicher Sprache (engl.: *natural language processing*, NLP) oder auch Rechnerlinguistik wird die Lehre von der Aufstellung und Anwendung von Theorien und Modellen natürlicher Sprache bezeichnet, die auf Computersystemen ausgeführt werden können [JM09]. Ziel der Modelle und Theorien ist es, natürliche Sprache zu analysieren und für den Computer erfass- und verarbeitbar zu machen. Im Mittelpunkt steht die Untersuchung der Eigenschaften natürlichsprachlicher Einheiten, wie z. B. Wörtern, Phrasen oder Sätzen. Typischerweise werden Sprachanalysen definiert, die einzelne Eigenschaften der Einheiten (z. B. Wortarten oder Satzstrukturen) untersuchen. Durch das Ausführen dieser Analysen durch den Computer erlangt dieser Zugriff auf Informationen über den untersuchten natürlichsprachlichen Text. Viele der Sprachanalysen bauen aufeinander auf oder profitieren zumindest von den Ergebnissen der anderen Analysen, sodass sich häufig eine Art Fließbandstruktur als Prozess ergibt.

2.8.1. Portionierung

Die erste Aufgabe der Verarbeitung natürlichsprachlicher Dokumente stellt zumeist die Portionierung (engl.: *tokenization*) dar [JM09]. Ziel dieses Schrittes ist es, die Eingabe in syntaktische Abschnitte einzuteilen. Diese Abschnitte werden als *Token* bezeichnet. Sie stellen die kleinste zu verarbeitende Einheit der folgenden Analysen dar. In Textdokumenten werden üblicherweise die einzelnen Wörter und Satzzeichen als Token aufgefasst. In Abbildung 2.4 ist die Portionierung des Satzes „*The system shall return the current time and date*“ dargestellt. Jeder Block stellt hierbei ein Token dar.

2.8.2. Satzerkennung

Aufbauend auf einer Portionierung der Eingabe werden natürlichsprachliche Dokumente oftmals in ihre struktur- und sinngebenden Elemente unterteilt [JM09]. Die einfachste Form dieser Unterteilung ist das Erkennen von Absätzen und Sätzen. Eine Satzerkennung

¹² <https://www.dbpedia.org/resources/ontology/>, zuletzt besucht am 26.04.2023.

Tabelle 2.2.: Auszug aus dem Etikettensatz der Penn Treebank [MMS93] für Wortarten

Etikette	Bedeutung
DT	Determinativ
NN	Substantiv, Singular
MD	Modalverb
VB	Verb, Infinitiv
JJ	Adjektiv
CC	Koordinierende Konjunktion

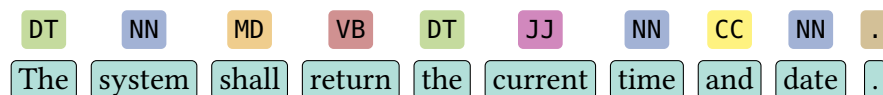


Abbildung 2.5.: Wortartetiketten für den Beispielsatz „The system shall return the current time and date.“: Die Bedeutung der jeweiligen Etiketten kann Tabelle 2.2 entnommen werden

erfolgt typischerweise anhand von Regelsystemen auf Basis von Satzzeichen und Abfolgen von Token. Eine Schwierigkeit dieser Aufgabe stellen oftmals Einschübe, Sonderzeichen in Token oder Zitate dar.

2.8.3. Wortarterkennung

Die Wortarten der Token einer textuellen Eingabe bilden die Grundform syntaktischer Informationen. Sie liefern ein grundlegendes Verständnis der Funktion der Wörter bzw. Token eines Satzes. Die Wortarten werden mittels sogenannter Wortartetiketten (engl.: *part-of-speech tags*, PoS tags) repräsentiert. Um eine einheitliche Darstellung zu ermöglichen, haben sich verschiedene Etikettensätze (engl.: *tagsets*) herausgebildet. Der meist verwendete dieser Sätze ist das Penn Treebank Tagset [TMS03]. Die eigentliche Aufgabe der Zuordnung der Wortarten zu den jeweiligen Grundeinheiten einer textuellen Eingabe wird in der Sprachverarbeitung mit Wortarterkennung (engl.: *part-of-speech tagging*, PoS tagging) bezeichnet.

In Abbildung 2.5 ist der Satz „*The system shall return the current time and date.*“ mit den entsprechenden Wortartetiketten des Penn Treebank Tagset versehen. Die Bedeutungen der einzelnen Etiketten wird in Tabelle 2.2 erläutert. Hierbei lässt sich z. B. aus der Wortartetikette *NN* von *system* ablesen, dass es sich um ein Substantiv im Singular handelt, was eine nützliche Information für ein Verständnis der beschriebenen Gegebenheiten ist. Eine vollständige Übersicht über die enthaltenen Etiketten und ihre jeweiligen Bedeutungen ist in Anhang A.1 gegeben.

2.8.4. Wortgrundformbestimmung

Viele Analysen und Verfahren, die auf natürlichsprachlichen Texten arbeiten, nutzen die Wortgrundform der auftretenden Wörter. Dadurch sind sie unabhängiger von der konkreten Wortform im Text (wie z. B. Beugung) und es werden dieselben Wörter einfacher identifiziert und somit generischer analysiert. Typischerweise wird zwischen zwei Herangehensweisen zur Bestimmung der Wortgrundform unterschieden: die Wortstammbildung (engl.: *stemming*) und die Lemmatisierung (engl.: *lemmatization*). Erstere bildet ein Wort auf seinen Wortstamm ab, indem meist über einfache Heuristiken die Wortendung abgeschnitten wird [Por80]. Dieses Vorgehen kann zu unerwünschten Verallgemeinerungen führen. So würden die beiden Wörter *university* und *universe* mittels des Porter-Stemmer-Algorithmus [Por80] beide auf den Wortstamm *univers* reduziert und somit von einer nachfolgenden Analyse als gleich angesehen, obwohl sie nicht semantisch gleich oder ähnlich sind. Verfahren die eine Lemmatisierung durchführen bestimmen hingegen für jedes Wort das zugehörige Lemma (vgl. Abschnitt 2.6.5) [JM09]. Hierfür verwenden sie die Wortart des Wortes und den aktuellen Kontext der Äußerung. Eine triviale Umsetzung eines solchen Verfahrens wäre ein einfaches Nachschlagen des Wortes im Lexikon. Doch die meisten heutigen Verfahren setzen auf maschinelles Lernen, um auch Randfälle abdecken zu können. Eine Lemmatisierung würde z. B. das englische Verb *was* auf sein Lemma (*to*) *be* oder das Adjektiv *better* auf *good* abbilden.

2.8.5. Stoppwortentfernung

Als Stoppwort werden in der Sprachverarbeitung Wörter bezeichnet, die vor der Verarbeitung eines natürlichsprachlichen Textes entfernt werden. Zumeist sind dies Wörter die keinen oder nur einen sehr geringen Beitrag zur Semantik eines Textes haben. Prinzipiell können aber beliebige Wörter als Stoppwörter fungieren. So kann z. B. bei der Verarbeitung von Quelltextkommentaren ein Filtern von programmiersprachenspezifischen Begriffen angewandt werden. Stoppwörter werden meist in Stoppwortlisten geführt und durch einen Vergleich der Wörter im Text (oder ihrer Lemmata) identifiziert und entfernt. Standard-Stoppwortlisten für die englische Sprache enthalten z. B. Wörter wie Konjugationen, Artikel und Präpositionen, die nur eine strukturierende Funktion im Satz innehaben [JM09]. In Anhang B.1 ist die Standard-Stoppwortliste des Natural Language Toolkit (NLTK)¹³ für die Englische Sprache abgebildet. Generell sollte man bei Verwendung von Standard-Stoppwortlisten darauf achten, dass tatsächlich alle gefilterten Wörter für die momentane Anwendung keine Relevanz haben. So enthalten manche Stoppwortlisten für die englische Sprache auch die Verben *be* und *have*, welche z. B. für die Bestimmung von semantischen Relationen oder der Struktur des Satzes durchaus relevant sein können.

Tabelle 2.3.: Auszug des Etikettensatzes der Penn Treebank [TMS03] für syntaktische Kategorien. Eine vollständige Liste befindet sich in Anhang A.2

Etikette	Bedeutung
ROOT	Wurzel
S	Einfacher Deklarativsatz
NP	Nominalphrase
VP	Verbalphrase

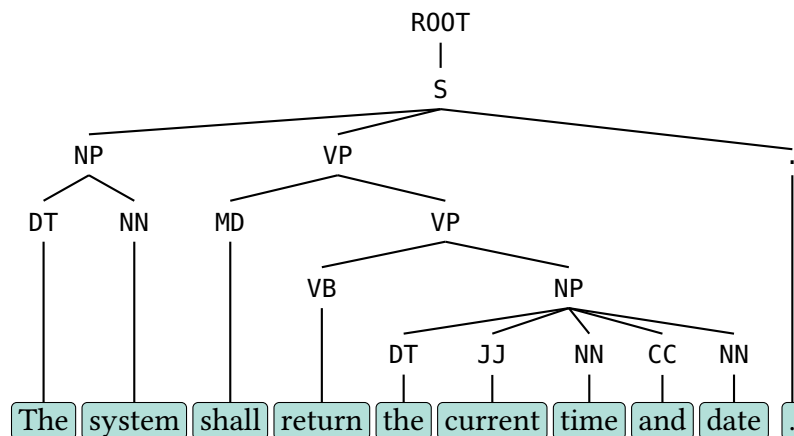


Abbildung 2.6.: Syntaxbaum für den Beispielsatz „The system shall return the current time and date.“: Die Bedeutungen der Etiketten, die syntaktische Kategorien beschreiben, können Tabelle 2.3 entnommen werden

2.8.6. Syntaktisches Zerteilen

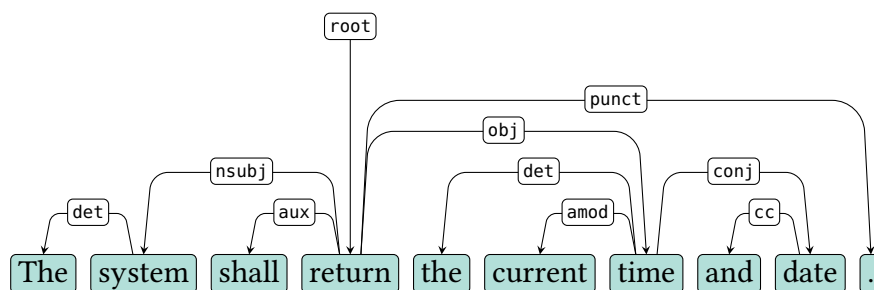
Aufbauend auf den Token und ihren Wortarten wird zumeist eine weitere Form von syntaktischer Information über den zu verarbeitenden Text gewonnen. Beim syntaktischen Zerteilen (engl.: *(syntactic) parsing*) wird die Satzsyntax analysiert und Informationen über die enthaltenen Phrasen und ihre syntaktischen Zusammenhänge generiert. Generell unterscheidet man zwischen zwei Arten von syntaktischem Zerteilen, dem Zerteilen mit einem Konstituenzzerteiler (engl.: *constituency parser*) bzw. einem Dependenzerteiler (engl.: *dependency parser*). Ersterer erzeugt Syntaxbäume (engl.: *parse trees*), welche die hierarchische Struktur des Satzes und seiner konstituierenden Phrasen in Baumstruktur abbildet. Zweiterer generiert Abhängigkeitsgraphen (engl.: *dependency graphs*), die Zusammenhänge der Token des Satzes abbilden, wie z. B., dass ein Nomen das direkte Subjekt eines Verbs darstellt und einen Artikel besitzt, der ihm vorangestellt ist. Da die Abhängigkeitsgraphen Zusammenhänge darstellen, können sie auch Teile der Semantik des Satzes abbilden. Beide Formen bieten unterschiedliche Vor- und Nachteile und werden deshalb meist separat angewandt und individuell genutzt.

Abbildung 2.6 zeigt einen Syntaxbaum für das bereits in Abschnitt 2.8.1 verwendete Beispiel. In Tabelle 2.3 sind alle im Beispiel auftretenden Etiketten (des Penn Treebank-

¹³ <https://www.nltk.org/>, zuletzt besucht am 26.04.2023.

Tabelle 2.4.: Auszug aus den Abhängigkeiten der Universal Dependencies [dMar+21] und ihre jeweilige Bedeutung

Etikette	Bedeutung
root	Wurzel
nsubj	Nominales Subject
det	Determinativ (Artikel)
aux	Hilfsverb
obj	Direktes Objekt
amod	Adjektivischer Modifikator
conj	Mit Konjunktion verbundene Elemente
cc	Koordinierende Konjunktion
punct	Interpunktion

**Abbildung 2.7.:** Abhängigkeitsgraph für den Beispielsatz „The system shall return the current time and date.“: Die Bedeutungen der Etiketten können Tabelle 2.4 entnommen werden

Etikettensatzes) für syntaktische Kategorien aufgelistet. Das Beispiel zeigt, dass der Satz aus einer Nominalphrase und einer geschachtelten Verbalphrase besteht.

Abbildung 2.7 zeigt hingegen den Abhängigkeitsgraph für das Beispiel unter Verwendung der Abhängigkeitsetiketten der Universal Dependencies [dMar+21]. Tabelle 2.4 enthält die zu sehenden Abhängigkeiten mit ihrer Bedeutung. Aus diesem Beispiel wird klar, dass der Abhängigkeitsgraph in der Lage ist, ausgehend von der Wurzel (meist dem dominanten Prädikat) Ketten von Zusammenhängen zwischen diesem Prädikat und den Teilen des Satzes zu bilden. So lässt sich direkt ablesen, dass *system* das Subjekt und *time and date* das Objekt des Satzes darstellt, welches über den Modifikator *current* näher beschrieben wird.

2.8.7. Erkennung semantischer Rollen

Eine weitere wichtige Aufgabe für die Analyse natürlicher Sprache stellt das Verständnis von durch Prädikate beschriebenen Handlungs-, Ereignis-, oder Zustandsbeschreibungen und den an ihnen teilnehmenden Argumenten dar. Da die Zusammenhänge zwischen einem Prädikat und seinen Argumenten in einem Satz durch unterschiedliche syntaktische

Tabelle 2.5.: Auszug der Etiketten für semantische Rollen des *CoNLL-2004/5*-Etikettensatzes [CM04; CM05]: Die vollständige Liste befindet sich in Anhang A.3.

Etikette	Bedeutung
V	Verb (Prädikat)
A0	Handelnder (Agens)
A1	Thema (behandeltes Objekt)
AM-MOD	Modalverb

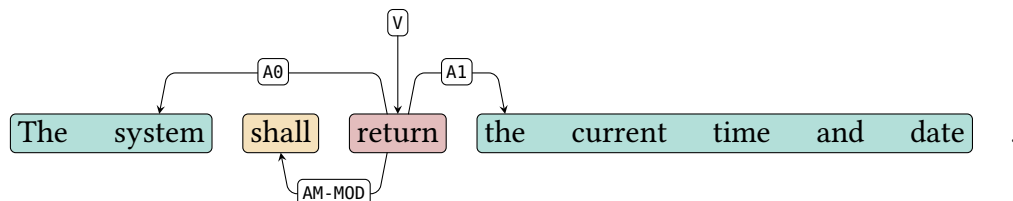


Abbildung 2.8.: Semantische Rollen für den Beispielsatz „The system shall return the current time and date.“: Die Bedeutungen der Etiketten können Tabelle 2.5 entnommen werden

Konstruktionen abgebildet werden können, benötigt man eine Abstraktion dieser Konstruktionen. Das hierfür verwendete Konstrukt sind *semantische Rollen*. Hiermit werden die Rollen bezeichnet, welche Phrasen(-teile) einnehmen können, wenn sie in Verbindung mit einem Prädikat interpretiert werden; also die abstrakten Rollen der Argumente eines Prädikats, die sie innerhalb einer Handlungs-, Ereignis- oder Zustandsbeschreibung einnehmen können. Semantische Rollen können auf unterschiedlichen Abstraktionsniveaus dargestellt werden. So kann dasselbe Argument durch eine sehr spezifische semantische Rolle wie z. B. *PLAYER*, dem Spieler, eine abstrakte Rolle wie *AGENT*, also dem Handelnden, bis hin zu einer überabstrakten Darstellung wie *PROTO-AGENT* repräsentiert werden. Unter einem *PROTO-AGENT* werden hierbei alle Rollen zusammengefasst, denen mehr Eigenschaften, die einem Handelnden zuzuordnen sind, zugeordnet werden können als Eigenschaften, die einem behandelten Subjekt/Objekt zuzuordnen sind [Dow91].

Der meistverwendete Etikettensatz für semantische Rollen, wurde im Rahmen der Vergleichsaufgaben der *Conference on Computational Natural Language Learning (CoNLL)* der Jahre 2004 und 2005 erstellt (*CoNLL-2004 Shared Task* [CM04] und *CoNLL-2005 Shared Task* [CM05]). Dieser stellt eine Mischform zwischen spezifischen und abstrakten Rollen dar. Jedes Prädikat erhält die Etikette *V* sowie alle Subjekte und Objekte des Prädikats *A**-Etiketten, wobei * für eine Nummer zwischen Null und Fünf steht¹⁴. Die *A**-Etiketten beziehen sich hierbei auf die für das jeweilige Prädikat spezifischen Rollen aus der PropBank-Datenbank [PGK05]. Da die Argumentnummern in der PropBank-Datenbank einer gewissen Struktur folgen, entspricht die *A0*-Rolle zumeist dem Handelnden (engl.: *agent*) und die *A1*-Rolle dem behandelten Objekt oder Thema (engl.: *patient* bzw. *theme*). Zusätzlich umfasst der Etikettensatz Rollen für modifizierende Argumente wie z. B. *AM-TMP* für einen Zeitausdruck oder *AM-CAU* für die Ursache. Diese Etiketten beginnen

¹⁴ Darüber hinaus gibt es außerdem die Rolle *AA*, um mehr als fünf Objekte erfassen zu können.

immer mit *AM*-. Außerdem besteht die Möglichkeit, Referenzen auf andere Rollen (*R*-*, wobei * die referenzierte Rolle darstellt) zu vergeben.

In Abbildung 2.8 ist der Satz „*The system shall return the current time and date.*“ mit den entsprechenden Etiketten versehen. Eine semantische Rolle wird dabei als gerichtete Kante dargestellt, welche ausgehend vom betrachteten Prädikat zu derjenigen Phrase führt, welche die Rolle einnimmt. Tabelle 2.5 bietet einen Auszug aus dem CoNLL-Etikettensatz mit den verwendeten Argumenten und einer kurzen Beschreibung ihrer Bedeutung (ohne die spezifischen Rollen aus PropBank). Somit existiert in diesem Beispiel lediglich das Prädikat *return*, welches als *A0*-Argument (also dem Handelnden) das System besitzt und als dessen behandeltes Objekt das aktuelle Datum und Zeit fungiert (*A1*-Rolle). *AM-MOD* gibt an, dass *shall* als Modalverb zu *return* fungiert.

In der Sprachverarbeitung wird die Aufgabe der Zuordnung der semantischen Rollen zu den Argumenten eines Prädikats als Erkennung semantischer Rollen (engl.: *semantic role labeling*, SRL) bezeichnet. Als Eingabe nutzen entsprechende Verfahren zumeist zusätzlich zu den Token die Wortarten, sowie Informationen über die vorhandenen Phrasen [JM09]. Moderne Verfahren setzen dabei auf Techniken des maschinellen Lernens und vortrainierte Sprachmodelle und erreichen F_1 -Maß-Werte von bis zu 90 % auf dem *CoNLL-2005* Datensatz¹⁵.

2.8.8. Disambiguierung von Wortbedeutungen

Gegenüber künstlichen bzw. formalen Sprachen sind natürliche Sprachen mehrdeutig [JM09]. Diese Mehrdeutigkeit äußert sich zumeist sowohl in ihrer Syntax als auch in der Semantik und Pragmatik (vgl. Abschnitte 2.6.2 bis 2.6.4). Gerade Mehrdeutigkeiten in Letzteren stellen für die Verarbeitung natürlicher Sprache eine große Herausforderung dar. So kann der Satz „*I will go to the bank.*“ sowohl bedeuten, dass jemand zu einem Geldinstitut geht oder aber auch, dass das Ziel eher eine Böschung oder Ufer ist. Je nach nachgelagerter Analyse kann dies einen immensen Einfluss auf die Interpretation haben. Will man z. B. Ähnlichkeiten zwischen Sätzen bestimmen und als Vergleichssatz wird der Satz „*My account is blocked.*“ genutzt, dann würde Interpretation von *bank* als Geldinstitut eine deutlich höhere Ähnlichkeit ergeben, als wenn *bank* als Ufer interpretiert wurde. Deshalb beschäftigt sich die Sprachverarbeitung mit Verfahren zur Bestimmung bzw. Disambiguierung von Wortbedeutungen (engl.: *word sense disambiguation*, WSD).

Die Bedeutung eines Wortes hängt hierbei immer vom Kontext ab, in dem es geäußert wird. Dies umfasst sowohl die umgebenden Wörter, Sätze usw. als auch die Äußerungssituation. Firth [Fir57] beschreibt dieses Phänomen mit der Aussage:

„*You shall know a word by the company it keeps!*“¹⁶

¹⁵ <https://paperswithcode.com/sota/semantic-role-labeling-on-conll-2005>, zuletzt besucht am 26.04.2023.

¹⁶ Zu Deutsch: Du sollst ein Wort anhand seiner Gesellschaft kennen/verstehen!

Wir Menschen sind in der Lage, selbst für uns unbekannte Wörter eine Bedeutung zu ermitteln, sofern wir Beispiele für seine Verwendung in einem Kontext erhalten. Lin [Lin98] nutzt, um dies zu verdeutlichen, das folgende leicht modifizierte Beispiel aus „*A Computational Analysis of Meaning*“ [Nid15] von Nida:

A bottle of *tezgüino* is on the table.
 Everybody likes *tezgüino*.
Tezgüino makes you drunk.
 We make *tezgüino* out of corn.

Obwohl *tezgüino* ein erfundenes Wort darstellt, kann ein Mensch aus dem Kontext schließen, dass es sich hierbei voraussichtlich um ein alkoholisches Getränk auf Maisbasis handelt.

Bestehende WSD-Ansätze setzen ebenso auf den Kontext, um die korrekte Bedeutung für ein Wort zu bestimmen. In dieser Aufgabe werden ebenso Etiketten, welche die jeweilige Bedeutung repräsentieren, eingesetzt. Allerdings ist es für Bedeutungen deutlich schwieriger, einen vollständigen Satz an Etiketten anzugeben, da auch offene Wortklassen betrachtet werden und somit potenziell die Menge der Wörter nicht bestimmt werden kann. Selbst wenn man ein einzelnes Wort betrachtet, kann die Menge aller Bedeutungen schwierig zu bestimmen sein. Deshalb wird bei der WSD häufig auf externe Wissensquellen als Basis der Etiketten zurückgegriffen. Das häufigste Beispiel hierfür ist sicherlich die Verwendung von *Synsets* aus WordNet [Mil95; FM98] (s. Abschnitt 2.7.2). Da externe Wissensquellen potenziell mit der Zeit wachsen und sich an die Sprache anpassen können, erweitert sich damit auch der potenzielle Etikettensatz. Allerdings enthalten nicht alle Wissensquellen für alle Wortarten Einträge, die sich als Etiketten nutzen lassen. So bietet die Wikipedia zwar Artikel zu vielen auch teils sehr speziellen Bedeutungen von Nomen, aber nahezu keine Einträge für Adjektive oder Verben. WordNet hingegen enthält *Synsets* für Adjektive oder Verben, umfasst dafür aber eine deutlich kleinere Menge an Bedeutungen für Nomen, die überhaupt ein *Synset* haben.

Aufgrund der Größe der Etikettensätze und dem vergleichsweise seltenen Auftreten der einzelnen Bedeutungen handelt es sich bei der Disambiguierung von Wortbedeutungen um eine der schwierigsten Aufgaben der Sprachverarbeitung. Den Stand der Technik bilden auch hier Verfahren des überwachten maschinellen Lernens¹⁷. Allerdings können diese nur mit Etiketten und somit Bedeutungen umgehen, die sie während des Trainings in ausreichender Menge gesehen haben. Dies limitiert ihre Anwendbarkeit in spezifischen Kontexten und Domänen, für die nur wenig bis keine Trainingsdaten existieren. Aus diesem Grund haben auch wissensbasierte Ansätze immer noch ihre Berechtigung. Diese Ansätze nutzen die relationale Struktur von Wissensquellen wie WordNet oder Wikipedia aus, um über diese Relationen die richtige Bedeutung aus den Beziehungen zu den Kontextwörtern abzuleiten [ALS14; MRN14; WWF20].

¹⁷ Siehe hierzu <https://paperswithcode.com/task/word-sense-disambiguation>, zuletzt besucht am 26.04.2023.

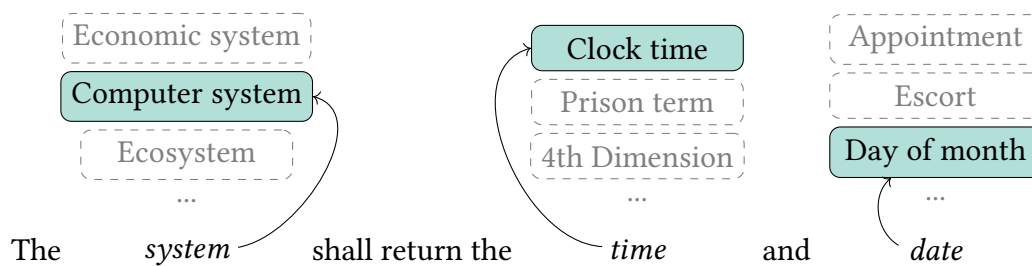


Abbildung 2.9.: Disambiguierung der Nomen in dem Beispielsatz „The system shall return the time and date“: Die gewählte Bedeutung ist grün hervorgehoben

Abbildung 2.9 zeigt die Disambiguierung der Nomen im Beispielsatz „*The system shall return the current time and date*“. Die möglichen Etiketten sind lose angelehnt an existierende WordNet-Synsets, werden allerdings durch einen sprechenden Namen, anstatt der Synset-ID repräsentiert.

2.8.9. Themen

Eine weitere Form einer semantischen Analyse stellt die Bestimmung des vorherrschenden Themas einer natürlichsprachlichen Eingabe dar. In der Sprachverarbeitung werden diese Themen zumeist durch einzelne Schlagwörter, eine Menge von Wörtern oder kurzen Phrasen dargestellt. Ziel ist es, mit der jeweiligen Repräsentation den Inhalt der Eingabe möglichst zutreffend zu beschreiben. Damit stellen diese Themen zumeist eine inhaltliche Verdichtung bzw. Zusammenfassung der Semantik der Eingabe dar.

Es wird zwischen einer Themenextraktion (engl.: *topic extraction*), Themenmodellierung (engl.: *topic modeling*) und Themenetikettierung (engl.: *topic labeling*) unterschieden. Bei der Themenextraktion werden Wörter aus der Eingabe selbst als Repräsentanten eines Themas ausgewählt. Diese Wörter sind zumeist häufige oder besonders relevante Wörter. Verfahren der Themenextraktion repräsentieren Themen meist als Wortmengen, da ein einzelnes, die Eingabe vollends beschreibendes, Wort nicht unbedingt Bestandteil der Eingabe sein muss. Bei der Themenmodellierung wird das Dokument in thematisch zusammenhängende Abschnitte unterteilt. Aufgabe ist es also, Abschnitte (ggf. auch unzusammenhängende) zu identifizieren, die einem gemeinsamen Thema zuzuordnen sind. Themenextraktion und Themenmodellierung werden in der Literatur häufig zusammengefasst und als ein Schritt beschrieben (unter der Bezeichnung Themenmodellierung). Die Themenetikettierung weist letztlich Abschnitten eines Dokuments (potenziell basierend auf Themenmodellierungen) ein Thema zu, welches durch Themenetiketten repräsentiert wird. Hierbei müssen, im Gegensatz zur Themenextraktion, die als Etiketten verwendeten Beschreibungen nicht unbedingt aus Wörtern der Eingabe bestehen. So können z. B. Themenetiketten aus externen Wissensquellen wie Wikipedia oder Ähnlichem stammen. So könnten für den Satz „*The system shall return the current time and date.*“ beispielsweise die Themenetiketten *Function* und *Time* gewählt werden.

2.8.9.1. (Probabilistische) Latente Semantische Indizierung

Latente Semantische Indizierung (engl.: *latent semantic indexing*, LSI) [Dee+90] oder auch manchmal Latente Semantische Analyse (engl.: *latent semantic analysis*) genannt, nutzt Singulärwertzerlegung (engl.: *singular-value decomposition*) [GR71], um assoziative Beziehungen zu modellieren. LSI erweitert Vektorraummodelle (engl.: *vector space models*, VSMs), indem es annimmt, dass zugrundeliegende, latente Strukturen in den Mustern der Wordbenutzung über Dokumente hinweg existieren [Dum94] und diese über statistische Verfahren vorhergesagt werden können. Hierbei wird eine die Dokumente repräsentierende Term-Dokument-Matrix in k orthogonale Faktoren (meist 100 bis 300) zerlegt, welche es ermöglichen, die Originalmatrix zu approximieren. LSI repräsentiert also Dokumente bzw. Eingaben anstatt als Menge ihrer Wörter durch einen Vektor kontinuierlicher Werte für jede der k Dimensionen. Da die Menge der Dimensionen meist deutlich niedriger ist als die Menge der Wörter, erzeugt diese Kompression Vektoren für Eingaben, die auch dann ähnlich sind, wenn sie nicht exakt die gleichen Wörter enthalten [Dum94].

Probabilistische LSI (engl.: *latent semantic indexing*, pLSI) [Hof99] unterbaut LSI mit einem statistischen Modell. pLSI modelliert jedes Wort eines Dokuments als eine Stichprobe einer Mischverteilung, bei der jede Komponente multinomiale Zufallsvariablen sind, welche als Repräsentation eines Themas angesehen werden können [BNJ03]. Somit wird ein Dokument auf eine Wahrscheinlichkeitsverteilung über einer festen Menge von Themen reduziert.

2.8.9.2. Latente Dirichlet Allokation

Das meistverwendete Verfahren zur Themenmodellierung ist die Latente Dirichlet Allokation (engl.: *latent dirichlet allocation*, LDA) [BNJ03]. Sie verbessert pLSI, indem sie ein vollständig generatives Bayes'sches Modell verwendet und besser mit den Überanpassungsproblemen von pLSI umgehen kann. LDA definiert ein Thema als eine statistische Verteilung (Dirichlet-Verteilung) über einem festen Vokabular. Das Vokabular wird durch die Menge der Wörter in den Eingaben definiert.

LDA lässt sich am einfachsten über den generativen Prozess erklären, den es als Grundlage für die Entstehung der Dokumente annimmt. Es wird angenommen, dass die Themen bereits bekannt sind. Die Wörter jedes Dokuments in der Eingabemenge werden daraufhin durch folgenden zweischrittigen Prozess generiert: Wähle zunächst eine zufällige Verteilung der Themen. Für jedes Wort im Dokument wähle erst ein zufälliges Thema und dann ein zufälliges Wort aus der Verteilung der Wörter des gewählten Themas. Somit bilden die Verteilungen der Themen die Wahrscheinlichkeiten der Wörter in einem Dokument ab. LDA versucht nachträglich diese Verteilungen zu bestimmen. Es bestimmt also die Wahrscheinlichkeit, dass ein Thema in einem Dokument auftritt darüber, welche Wörter in den Dokumenten auftreten. Charakteristisch für LDA ist hierbei, dass alle als Eingabe verwendeten Dokumente dieselben Themen teilen, aber jedes Dokument eine andere Verteilung dieser Themen aufweist [Ble12].

The system shall return the time and date . The date is formatted ...		
...
and: 0	and: 1	and: 1
be: 0	be: 0	be: 1
date: 0	date: 2	date: 2
format: 0	format: 0	format: 1
return: 0	return: 1	return: 1
shall: 0	shall: 1	shall: 1
system: 0	system: 1	system: 1
the: 0	the: 2	the: 3
time: 1	time: 1	time: 1
...
(a) 1-aus-n-Kodierung	(b) Wort-Multimenge (Kontextfenster 5 Wörter)	(c) Wort-Multimenge (gesamtes Dokument)

Abbildung 2.10.: Verschiedene Arten von Vektorrepräsentationen eines Wortes (*time*) in einer Texteingabe

2.9. Wortrepräsentationen und Sprachmodelle

In der Sprachverarbeitung existieren verschiedene Formen der Repräsentation von Wörtern und Texteingaben. Zunächst lassen sich Wörter oder auch ganze Texte als aufeinanderfolgende Menge von Zeichen repräsentieren. Diese Form eignet sich gut für zeichenbasierte Vergleiche und Regeln, aber weniger für die Verarbeitung durch maschinelle Lernverfahren, welche einen Merkmalsvektor als Eingabe erwarten. Daher werden oftmals Vektorrepräsentationen genutzt. Diese bieten außerdem den Vorteil Ähnlichkeiten über Vektordistanzmaße zu bestimmen. In den letzten 10 Jahren kamen zu den einfachen Vektorrepräsentationen außerdem Sprachmodelle hinzu, welche eine Vektorrepräsentation von Wörtern oder Texten aus großen Mengen von Textdaten erlernen und somit eine bessere semantische Repräsentation der Eingaben versprechen.

2.9.1. Vektorrepräsentationen

Ziel von Vektorrepräsentationen von textuellen Eingaben ist es, eine einheitliche Form der Repräsentation bereitzustellen, die durch maschinelle Lernverfahren oder Vektorraumanalysen genutzt werden kann. Die wohl einfachste Form dieser Repräsentation stellt die sogenannte 1-aus-n-Kodierung (engl.: *one-hot encoding*) dar [Tha17]. Diese spannt einen Vektorraum über einem Vokabular auf, indem jedes Wort des Vokabulars einer Dimension des Vektorraums entspricht. Somit kann ein Wort durch den Vektor repräsentiert werden, der eine Eins in der entsprechenden Dimension und sonst nur Nullen enthält. So wäre die Repräsentation des Wortes *time* mittels einer 1-aus-n-Kodierung wie in Abbildung 2.10a dargestellt. Ein großer Nachteil dieser Repräsentation besteht darin, dass die entstehenden Vektoren sehr dünn besetzt sind und der Kontext eines Wortes nicht weiter betrachtet wird.

Eine Form auch den Kontext des zu repräsentierenden Wortes miteinzubeziehen, sind Vektoren auf Basis von Wort-Multimengen (engl.: *bag-of-words*, BoWs) [JM09]. Diese werden ebenfalls auf einem dem Vokabular entsprechendem Vektorraum definiert. Jedoch anstatt nur für das zu repräsentierende Wort selbst einen Eintrag im Vektor ungleich Null zu vergeben, werden nun für alle Wörter in einem vorher bestimmten Kontext Vektoreinträge gesetzt. Hierzu wird zumeist mittels eines festgelegten Kontextfensters (engl.: *sliding window*) um das zu repräsentierende Wort gearbeitet. Abbildung 2.10b zeigt die Repräsentation des Wortes *time* innerhalb eines Dokuments bestehend aus „*The system shall return the current time and date. The date should be formatted ...*“, wenn ein Kontextfenster von 5 Token in beide Richtungen verwendet wird. Wird die gesamte Eingabe bzw. das gesamte Dokument als Kontext gewählt, lässt sich BoW auch zur Repräsentation von Dokumenten nutzen (vgl. Abbildung 2.10c). Normalerweise werden bei BoW-Vektoren mehrfache Auftreten desselben Wortes im Kontext durch ein Zählen der Häufigkeit repräsentiert (vgl. Abbildungen 2.10b und 2.10c). Je nach Anwendungsfall kann aber auch ein binäres Kodieren oder eine Angabe der relativen Häufigkeit sinnvoll sein.

Ein Nachteil der BoW-Repräsentation für ganze Dokumente ist, dass die Wortreihenfolge ignoriert wird. Hierdurch kann die Semantik der Aussage schlechter abgebildet und Zusammenhänge, wie z. B. zusammengesetzte Nomen, nicht erfasst werden. Um diesem entgegenzuwirken, kann BoW in Kombination mit N-Grammen verwendet werden. Ein N-Gramm (engl.: *n-gram*) ist eine Sequenz bestehend aus N aufeinanderfolgenden Token [JM09]. So kann der Satz „*The system shall repeat the sound*“ in die 3-Gramme (auch Trigramme genannt) „*The system shall*“, „*system shall repeat*“, „*shall repeat the*“ und „*repeat the sound*“ unterteilt werden. Anstatt nun das Auftreten einzelner Wörter in einem Dokument zu zählen und als Vektorrepräsentation zu nutzen, kann das Auftreten der N-Gramme gezählt und die in einem Korpus auftretenden N-Gramme als Dimensionen verwendet werden. Man spricht in diesem Fall auch von einer N-Gramm-Multimenge (engl.: *bag-of-n-grams*).

Ein weiteres Problem von BoW besteht darin, dass hochfrequente Wörter in einem Dokument die Repräsentation dominieren können, selbst aber wenig Beitrag zur Semantik liefern. Eine Möglichkeit hiermit umzugehen, ist ein Entfernen von Stoppwörtern (vgl. Abschnitt 2.8.5). Doch häufig gibt es über die Menge an Stoppwörtern hinaus, für einen bestimmten Anwendungsfall bzw. eine bestimmte Menge an Texteingaben weitere irrelevante Wörter. Ein Verfahren, welches versucht diese Wörter zu bestimmen bzw. diesen eine geringere Gewichtung zu geben, ist die Vorkommenshäufigkeit-Inverse Dokumenthäufigkeit (engl.: *term frequency-inverse document frequency*, TF-IDF). TF-IDF basiert auf der Hypothese, dass Wörter, die häufig im aktuellen Kontext aber selten in einer Textsammlung auftreten, einen höheren Beitrag zur Semantik des Kontextes leisten als Wörter, die über alle Dokumente hinweg häufig auftreten. Es werden also Wörter, die spezifischer für einzelne Dokumente sind, höher gewichtet als generische und damit wenig diskriminative Wörter für alle Dokumente. Hierzu wird zunächst die Vorkommenshäufigkeit

(engl.: *term frequency*, TF) [Luh57] eines Wortes in der aktuellen Eingabe bzw. dem zu repräsentierenden Dokument berechnet:

$$\begin{aligned} \text{TF}(t, d) &= \#(t, d) \\ \text{TF}_{\text{rel}}(t, d) &= \frac{\#(t, d)}{\max_{t' \in d} \#(t', d)} \end{aligned} \quad (2.3)$$

Hierbei gibt $\#(t, D)$ die Anzahl der Vorkommen von Term t in Dokument d an. Diese kann mittels des Auftretens des am häufigsten vorkommenden Terms in d normalisiert werden. In diesem Fall spricht man von der relativen Auftretenshäufigkeit (TF_{rel}). Anschließend wird die inverse Dokumenthäufigkeit (engl.: *inverse document frequency*, IDF) [Spa72] berechnet, welche die Spezifität eines Terms für die Gesamtmenge der betrachteten Dokumente D angibt:

$$\text{IDF}(t) = \log \frac{N}{\sum_{d:t \in D} 1} \quad (2.4)$$

Hierbei gibt N die Gesamtanzahl an Dokumenten an und $d:t$ ist ein Dokument, das Term t enthält. Dieser Wert ist umso höher, je weniger Dokumente den Term t beinhalten. Als Dokument sollte hierbei eine Texteingabe gelten, die später einzeln repräsentiert oder analysiert werden soll. Dies kann also ein Satz eines Textes sein, sofern verschiedene Sätze verglichen werden sollen, oder aber auch ganze Textdokumente, wenn diese in Bezug auf eine Sammlung von Textdokumenten interpretiert werden sollen. Im Kontext von Anforderungsnachverfolgbarkeit wären z. B. einzelne Anforderungen eine sinnvolle Wahl für ein Dokument, da diese abgebildet und voneinander abgegrenzt werden sollen. Letztlich wird dann die TF-IDF-Gewichtung eines Wortes bzw. Terms t innerhalb eines Dokuments/Kontexts d über eine Gewichtung der Vorkommenshäufigkeit mit der inversen Dokumenthäufigkeit bestimmt:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \cdot \text{IDF}(t) \quad (2.5)$$

2.9.2. Wort- und Dokumenteinbettungen

Auch BoW-basierte Vektorrepräsentationen resultieren oftmals in dünn besetzten Vektoren und sind somit für Lernverfahren weniger geeignet. Ebenso können diese dünn besetzten Vektoren schlechter Ähnlichkeiten zwischen Wörtern (wie z. B. Synonyme) abbilden. Eine Alternative stellen deshalb Worteinbettungen (engl.: *word embeddings*) dar. Auch Worteinbettungen sind Vektorrepräsentationen und basieren auf den Kontexten in denen Wörter auftreten. Sie folgen also ebenso dem Konzept von Firth [Fir57] „*You shall know a word by the company it keeps*“. Allerdings verwenden sie eine festgelegte von der Größe des Vokabulars unabhängige Anzahl an Dimensionen (meist 100 bis 300), welche dicht besetzt sind. Um dies zu erreichen und trotzdem den Kontext der Wörter zu repräsentieren, setzen Worteinbettungen auf die Eigenschaft von neuronalen Netzen,

in ihren versteckten Schichten eine Art von Repräsentation der gesehenen Eingaben zu lernen (vgl. Abschnitt 2.5.2). Hierzu wird ein FFNN auf einem geeignet großen Textkorpus mit einer Kontextvorhersage-Pseudoaufgabe trainiert. Die Worteinbettung eines Wortes (repräsentiert durch eine 1-aus-n-Kodierung) ergibt sich dann als die Kantengewichte der Neuronen der versteckten Schicht für diese Eingabe. Somit entspricht die Dimension der Einbettung der Anzahl an Neuronen der versteckten Schicht des neuronalen Netzes. Diese Form von Einbettungen sind statisch. Dies bedeutet, dass sie nach Abschluss des Trainings für jedes trainierte Wort genau eine Repräsentation bereitstellen. Sie können ohne eine Verwendung des dahinterliegenden neuronalen Netzes verwendet werden, indem die jeweiligen Einbettungen über einem Vokabular abgespeichert werden. Da diese Einbettungen statisch sind, hat ein Wort allerdings unabhängig von seiner Bedeutung im Kontext immer die gleiche Repräsentation. Tritt also eine Bedeutung und somit ein bestimmter Kontext eines Wortes besonders häufig in den Trainingsdaten auf, wird die erlernte Einbettung dieses Wortes eher dazu tendieren, diese Bedeutung zu repräsentieren und somit seltenere Bedeutungen und ihre Ähnlichkeiten zu anderen Wörtern schlechter abbilden.

Die bestehenden Verfahren zur Generierung von Worteinbettungen unterscheiden sich in der Pseudoaufgabe, welche zum Trainieren des neuronalen Netzes verwendet wird. All diese Pseudoaufgaben haben allerdings gemeinsam, dass die Musterlösung für die Vorhersage bereits aus dem Text extrahiert werden kann. Es handelt sich somit um selbstüberwachtes maschinelles Lernen (engl.: *self-supervised learning*), welches die Möglichkeit bietet, große Mengen von Trainingsdaten zu nutzen.

word2vec Den Durchbruch von Worteinbettungen in der Sprachverarbeitung stellt das 2013 vorgestellte Softwarepaket *word2vec* [Mik+13b; Mik+13a] von Mikolov et al. dar. Es umfasst zwei Methoden zum effizienten Trainieren von Worteinbettungen: Kontinuierliche Wort-Multimenge (engl.: *continuous bag-of-words*, CBOW) und Kontinuierliches Skip-Gramm (engl.: *continuous skip-gram*, CSG). Bei CBOW wird als Pseudoaufgabe die Vorhersage eines maskierten Wortes basierend auf einem Kontextfenster genutzt. CSG hingegen nutzt als Klassifikationsaufgabe die Vorhersage der umgebenden Wörter basierend auf dem aktuell betrachteten Wort. Es wird also für beide Verfahren der Text des Trainingskorpus mittels eines Kontextfensters durchlaufen. Im Fall von CBOW wird nun das Wort in der Mitte des Fensters als Musterlösung hergenommen, welche durch Eingabe der weiteren Wörter im Kontextfenster vorhergesagt werden muss. Bei CSG hingegen dient nur das Wort in der Mitte des Fensters als Eingabe und die weiteren Wörter des Kontextfensters müssen vorhergesagt werden. In beiden Fällen wird ein binärer Klassifikator trainiert, welcher für die aus dem Text extrahierten Musterlösungen (Positivbeispiele) und durch zufälliges Sampling aus dem Korpus generierten Negativbeispiele entscheiden muss, ob diese im Kontext sind.

Mikolov et al. konnten in ihrer Arbeit „*Efficient Estimation of Word Representations in Vector Space*“ [Mik+13a] außerdem zeigen, dass die durch ihre Methodik erzeugten Worteinbettungen gewisse semantische Zusammenhänge der Wörter erhalten. Zunächst einmal haben Einbettungen von Wörtern, die in den gleichen oder ähnlichen Kontexten auftreten

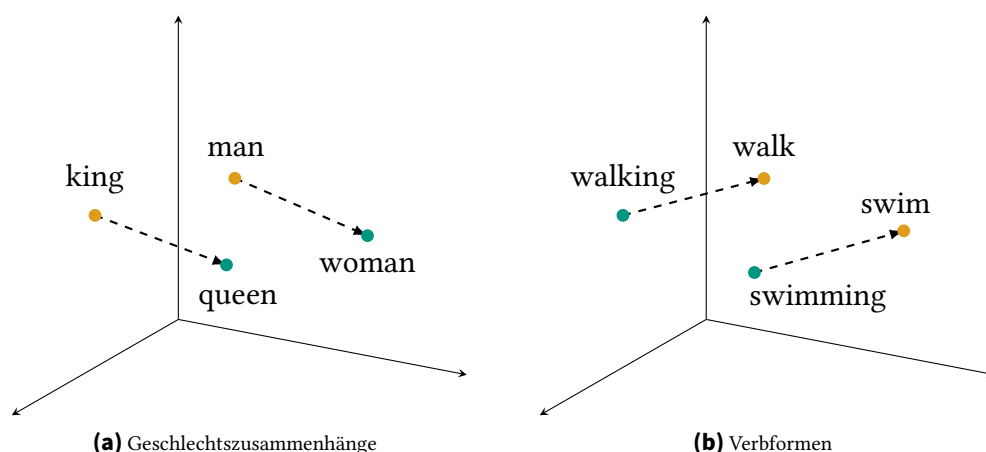


Abbildung 2.11.: Schematische Repräsentation von semantischen Zusammenhängen zwischen Worteinbettungen

eine geringe Vektordistanz zueinander. Sie konnten aber auch zeigen, dass Einbettungen von Wortpaaren, die in einem gleichen Zusammenhang stehen (z. B. Länder zu ihren Hauptstädten oder weibliche und männliche Form) sich ebenso ähnlich zueinander im Vektorraum verhalten. So konnte z. B. für ein *word2vec*-Modell gezeigt werden, dass der Einbettungsvektor von *king* subtrahiert mit dem Einbettungsvektor von *man* addiert mit dem Einbettungsvektor von *woman* einen Vektor ergibt, der am nächsten zur Vektorrepräsentation von *queen* liegt (vgl. Abbildung 2.11).

fastText *fastText* [Boj+17] von Bojanowski et al. stellt eine Weiterentwicklung der CSG-Methodik von *word2vec* dar. Ein Nachteil des ursprünglichen Ansatzes ist, dass er nicht mit Wörtern umgehen kann, die nicht während des Trainings gesehen wurden. Um auch solche Wörter außerhalb des Vokabulars (engl.: *out-of-vocabulary words*, OOV) abbilden zu können, zerlegt *fastText* Wörter in ihre Zeichen-N-Gramme und lernt zusätzlich zur Einbettung des gesamten Wortes auch Einbettungen für diese N-Gramme. Dies ermöglicht es einem unbekanntem Wort eine Einbettung zuzuweisen, die auf den in diesem Wort enthaltenen N-Grammen basiert. Der Gedanke, welcher hier dahinter steht, ist, dass viele Wörter gemeinsame Prä- oder Suffixe besitzen und Wörter die sich N-Gramme teilen in vielen Fällen auch verwandte Wörter darstellen. So würde *fastText* das Wort *prefix* beispielsweise durch die Summe der Vektorrepräsentationen seiner Zeichen-Trigramme <pr, pre, ref, efi, fix, ix> und der Repräsentation des gesamten Wortes <prefix> repräsentieren. Die Symbole < und > werden hierbei als Sonderzeichen eingefügt, um den Wortanfang und das Wortende zu kennzeichnen. *fastText* nutzt standardmäßig alle Zeichen-N-Gramme der Größen $N = 3$ bis 6.

doc2vec *word2vec* und *fastText* sind in der Lage, Vektorrepräsentationen für Wörter zu generieren. Möchte man allerdings mehr als einzelne Wörter repräsentieren (z. B. Sätze, Absätze oder ganze Dokumente), so müssen die Einbettungen der einzelnen Wörter in geeigneter Art aggregiert werden. Ein einfaches Mitteln der Vektoren würde allerdings

zu Informationsverlusten führen, da der resultierende Vektor potenziell nah zu Vektoren liegt, die nichts mehr mit der Semantik der Eingabe zutun hat (z. B. wenn mehrere sehr unterschiedliche Kontexte in der Eingabe vorherrschen). Eine weitere Möglichkeit wäre es, die Einbettungen in einer Art BoW zu nutzen, einer Einbettungs-Multimenge (engl.: *bag-of-embeddings*, BoE). Diese ließe sich aber nicht ohne Weiteres als Eingabe in ein maschinelles Lernverfahren nutzen. Außerdem haben beide Verfahren den Nachteil, dass sie die Information über die Reihenfolge der Wörter im Satz verlieren und somit beispielsweise die Phrasen „Gehen wir essen“ und „Wir gehen essen“ zur gleichen Repräsentation führen, obwohl sie eine andere Semantik haben.

Um diese Nachteile zu umgehen, nutzt *doc2vec* [LM14] ein analoges Vorgehen zu *word2vec*, mit dem Fokus sogenannte Absatzeinbettungen zu lernen. Auch *doc2vec* besitzt zwei Methodiken: verteiltes Absatzvektoren-Speichermodell (engl.: *Distributed Memory Model of Paragraph Vectors*, PV-DM) und verteilte Wort-Multimengen-Version von Absatzvektoren (engl.: *Distributed Bag of Words version of Paragraph Vector*, PV-DBOW). Die Grundidee ist, zusätzlich zu den Wörtern, außerdem einen eindeutigen Absatzindikator, in der jeweiligen Pseudoaufgabe zu nutzen, welcher über alle Kontexte der zu repräsentierenden Eingabe, bzw. des zu repräsentierenden Absatzes, gleich bleibt. Wählt man z. B. Sätze als zu repräsentierende Eingabe, würde jeder Satz im Trainingskorpus einen eindeutigen Absatzindikator bekommen und jeder Trainingskontext, der aus diesem Satz generiert wird, erhält außerdem diesen Indikator als Eingabe. Andere Sätze im Korpus erhalten dann entsprechend andere Indikatoren. Im Fall von PV-DM wird dieser Absatzindikator mit den Wörtern der Eingabe konkateniert und die Aufgabe ist analog zu CBOW von *word2vec* die Vorhersage eines Wortes basierend auf den Kontextwörtern (+ Absatzindikator). Somit stellen letztlich die Kantengewichte der Neuronen der versteckten Schicht bei Eingabe des jeweiligen Absatzindikators die Repräsentation dieses „Absatzes“ dar. Bei PV-DBOW müssen die Wörter einer zu repräsentierenden Eingabe bzw. Absatzes nur mittels der Eingabe des Absatzindikators vorhergesagt werden (analog zu CSG in *word2vec*).

2.9.3. (Neuronale) Sprachmodelle

Statistische Modelle die Sequenzen von Wörtern Wahrscheinlichkeiten zuweisen werden in der Sprachverarbeitung als Sprachmodelle bezeichnet [JM09]. Sprachmodelle können also gegeben eine Sequenz von Wörtern vorhersagen, welches Wort das wahrscheinlichste nächste Wort darstellt. Die einfachste Ausprägung eines solchen Modells ist das N-Gramm-Modell, welches basierend auf den Auftretenshäufigkeiten der N-Gramme in einem Korpus, deren Wahrscheinlichkeiten bestimmt. Auch die in Abschnitt 2.9.2 vorgestellten Verfahren zur Generierung von Worteinbettungen können als eine Ausprägung eines Sprachmodells verstanden werden. Die verwendeten Pseudoaufgaben und das auf ihnen trainierte neuronale Netz bestimmt ebenso Wahrscheinlichkeiten für Abfolgen von Wörtern; in diesem Fall vor allem hinsichtlich des Zusammenhangs zwischen Kontextwörtern (z. B. aus einem Kontextfenster) und einem betrachteten Wort. Sie ignorieren hierbei allerdings die Reihenfolge der Wörter im Kontext, wohingegen N-Gramme eine stricte Reihenfolge abbilden.

In den letzten Jahren wird allerdings, wenn von Sprachmodellen gesprochen wird, zumeist von kontextsensitiven neuronalen Sprachmodellen gesprochen. Statische Worteinbettungs-Modelle, wie *word2vec* oder *fastText*, mitteln die Wahrscheinlichkeiten aller Bedeutungen eines Wortes (vgl. Abschnitt 2.9.2). Hierbei werden also alle Auftretenswahrscheinlichkeiten von Kontextwörtern miteinbezogen, die in einem beliebigen Kontext mit dem vorhergesagten Wort auftraten. Hat das betrachtete Wort allerdings verschiedene Bedeutungen so können sich die potenziellen Kontextwörter und somit die validen Sequenzen, in denen das Wort auftritt, von Bedeutung zu Bedeutung stark unterscheiden. Beispielsweise ist die Wahrscheinlichkeit für das Wort *setzen* im Kontext des Wortes *Bank* deutlich höher, wenn bekannt ist, dass es sich um die Bedeutung *Sitzgelegenheit* handelt und nicht um das *Finanzinstitut*. Um diesen Nachteil von statischen Worteinbettungs-Modellen zu verbessern, wurden kontextsensitive bzw. dynamische Modelle eingeführt, welche eine Einbettung eines Wortes basierend auf dem aktuellen Kontext bestimmen.

ELMo Das erste Beispiel für ein solches kontextsensitives Sprachmodell stellt *Embeddings from Language Models (ELMo)* [Pet+18] dar. *ELMo* nutzt ein bidirektionales Sprachmodell, welches aus einem vorwärts- und einem rückwärts-gerichteten RNN besteht. Genauer kommen hier LSTMs (vgl. Abschnitt 2.5.2.2) zum Einsatz, welche gerade mit längeren Eingaben besser umgehen können. Die beiden LSTMs bekommen als Eingabe eine Sequenz von Wörtern, die das vorherzusagende Wort umgeben, und durch eine statische Tokeneinbettung repräsentiert werden. Diese verarbeiten sie vorwärts bzw. rückwärts in L Schichten. Durch das bidirektionale Vorgehen kann das RNN auch Informationen aus dem Verarbeitungsschritt des jeweils vorhergehenden bzw. nachfolgenden Wortes nutzen. Um die letztlichen, kontextualisierten Einbettungen zu erhalten, konkateniert *ELMo* zuerst die Gewichte der jeweiligen versteckten Schichten des Vorwärts- und Rückwärts-LSTMs zu je einem Vektor, um diese anschließend durch eine gewichtete Summe zu vereinen. *ELMo* erreicht durch den Einsatz mehrerer LSTM-Schichten somit eine kontextualisierte Repräsentation des Wortes, basierend auf dem Kontext der Wörter, in dem es in der Eingabe auftritt. Die resultierende Einbettung entspricht also der Repräsentation des Wortes gegeben die Wörter in seiner aktuellen Umgebung.

Transferlernen Ein weiterer zentraler Schritt für die Anwendung von Sprachmodellen ist der Einsatz von Transferlernen. Mit *Universal Language Model Fine-tuning for Text Classification (ULMFiT)* [HR18] stellen Howard und Ruder einen Prozess vor, um mehr als nur die resultierenden Einbettungen aus dem Training eines Sprachmodells nutzbar zu machen. Sie überführen das Konzept des Transferlernens aus der Computervision auf Sprachmodelle und ermöglichen es damit, basierend auf einem vortrainierten Modell eine Feinanpassung (engl.: *fine-tuning*) auf eine spezielle Aufgabe durchzuführen, ohne das gesamte Modell von Neuem trainieren zu müssen. Dies ermöglicht es, dass Sprachmodell auf einer großen Datenmenge und verwandten Aufgabe vorzutrainieren (engl.: *pre-training*) und für viele Aufgaben wiederzuverwenden. Für eine neue Aufgabe kann dann das vortrainierte Modell mit weniger Trainingsdaten feinangepasst werden. Der Gedankengang hinter diesem Konzept ist, dass viele Aufgaben der Sprachverarbeitung auf gemeinsamen

Zusammenhängen der Sprache aufbauen. Werden solche generellen (bzw. für eine Gruppe von Aufgaben relevanten) Zusammenhänge bereits durch das Sprachmodell während des Vortrainierens gelernt, kann die Feinanpassung bereits auf diesen aufbauen und sich somit ganz auf die eigentliche Aufgabe konzentrieren. Außerdem lässt sich somit ein vortrainiertes Sprachmodell für unterschiedliche Problemstellungen wiederverwenden. Die Feinanpassung lässt sich aufteilen in eine Feinanpassung des Sprachmodells und ein Trainieren des eigentlichen Klassifikators für das gegebene Problem. Ersteres kann über verschiedene Methodiken des Zurückspiels von Ergebnissen an das neuronale Netz geschehen. Letzteres geschieht zumeist durch das Anhängen einer weiteren neuronalen Schicht (meist vorwärts gerichtet) an das bestehende Sprachmodell. Prinzipiell kann die Feinanpassung auch ohne den Schritt der Anpassung des Sprachmodells durchgeführt werden.

Durch die Einführung der Transformer-Architektur [Vas+17] für Enkodierer-Dekodierer neuronale Netze, welche einen noch besseren Umgang mit Langzeitabhängigkeiten zwischen Eingaben als LSTMs haben, entstanden neue Formen von Sprachmodellen. Transformer-Architekturen bestehen aus gestapelten sogenannten Transformer-Blöcken, welche mehrschichtige Netze mit linearen Schichten und einen „*self-attention*“ genannten Mechanismus kombinieren [JM22]. Die *self-attention*-Schichten erlauben es dem Netz aus beliebig großen Kontexten direkt zu entscheiden, welche Informationen extrahiert werden sollen. Sie müssen diese also nicht erst durch dazwischenliegende rekurrente Verbindungen leiten, wie es bei einem RNN der Fall ist [JM22]. Das Netz kann also einfacher aus weit zurückliegenden Wörtern Informationen ziehen, sofern es lernt, dass diese relevant sind. Eine weitere Neuerung, welche Vaswani et al. mit dem Transformer einführen, ist die sogenannte „*multi-headed attention*“, welche im Gegensatz zu bestehenden *Attention*-Modellen und Mechanismen dem Modell dabei hilft, sich auf mehrere verschiedene Positionen zu fokussieren und damit dem Problem entgegenwirkt, dass das momentan betrachtete Wort andere Wörter dominiert. Radford et al. [Rad+18] setzen Transformer erstmals für Sprachmodelle ein, die feinangepasst werden können. Hierbei nutzen sie anstatt der Enkodierer und Dekodierer nur die Dekodierer, inklusive der *Attention*-Schichten. Da die Transformer-Dekodierer bereits so ausgelegt sind, dass sie zukünftige Token maskieren, sind sie einfach für Sprachmodellierung, also die Vorhersage des nächsten Wortes, nutzbar. Durch geeignete Eingabetransformationen lässt sich dieses Modell für unterschiedliche Problemstellungen, wie Klassifikation, semantische Implikation, Ähnlichkeitsberechnung oder Mehrfachauswahl nutzen.

BERT Das von Verfahren von Radford et al. ermöglicht es zwar, ein feinanzpassbares vortrainiertes Sprachmodell basierend auf Transformern zu trainieren, allerdings verarbeitet es die Eingabe nur in Vorwärtsrichtung und weist damit Schwächen auf, die bereits in *ELMo* durch ein bidirektionales Vorgehen adressiert wurden. Das erste bidirektionale Transformer-basierte Sprachmodell stellt *Bidirectional Encoder Representations from Transformers (BERT)* [Dev+19] dar. *BERT* nutzt anstatt den Dekodierern der Transformer-Architektur, die Enkodierer, welche sowohl die vorherigen als auch die folgenden Token miteinbeziehen. Da ein solches Vorgehen in einem mehrschichtigen Netz dazu führen

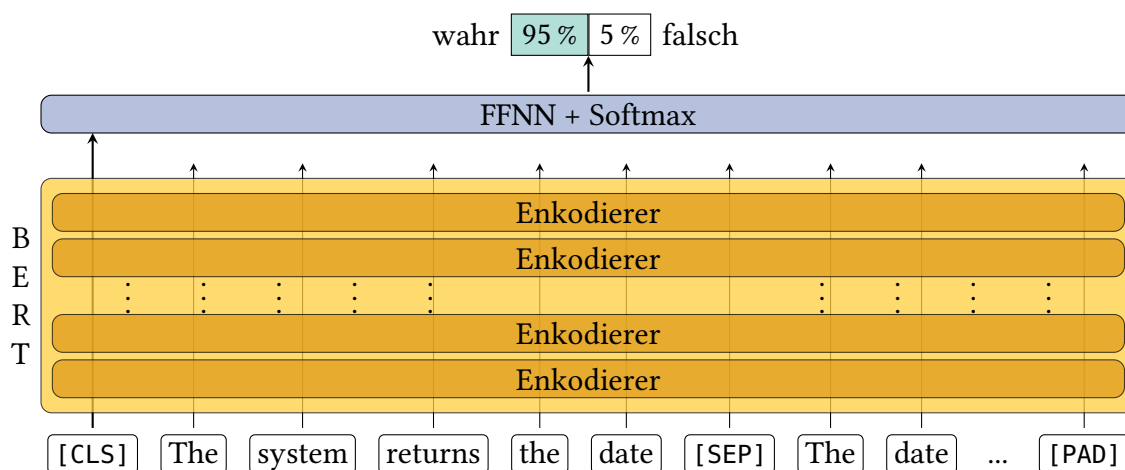


Abbildung 2.12.: Schematische Darstellung eines *BERT*-basierten Klassifikators unter Nutzung der Repräsentation des [CLS]-Tokens

würde, dass jedes Wort sich indirekt in der Berechnung selbst sieht und somit die Vorhersageaufgabe trivialisiert, nutzt *BERT* das Konzept des maskierten Sprachmodellierens (engl.: *masked language modeling*). Hierzu maskiert *BERT* während des Vortrainierens zufällig 15 % der Wörter der Eingabe und lässt das Modell die jeweils maskierten Wörter vorhersagen, indem die Ausgabe an der Stelle des maskierten Wortes als Eingabe für die Vorhersage genutzt wird. Würde also in Abbildung 2.12 das Wort *date* maskiert, würde dieses durch das Token [MASK] ersetzt und anstatt der Ausgabe für das [CLS]-Token, die Ausgabe für die 6. Stelle (die Stelle des maskierten Wortes) in das FFNN gegeben und dieses darauf trainiert, *date* vorherzusagen.

Zusätzlich nutzt *BERT* eine weitere Pseudoaufgabe zum Vortrainieren: Das Bestimmen der Wahrscheinlichkeit, dass Satz B auf Satz A folgt. Hierzu führt *BERT* die mit [SEP] dargestellten Separator-Token zwischen den Sätzen ein und nutzt für die Bestimmung der Wahrscheinlichkeit die Ausgabe für das [CLS]-Token, welches vor jede Eingabe gehängt wird (s. Abbildung 2.12). Die Idee hinter diesem [CLS]-Token lässt sich mit der Idee des Absatzindikators von *doc2vec* vergleichen: Es soll die Semantik der gesamten Eingabe repräsentieren. Im Gegensatz zu *doc2vec* ist *BERTs* Ausgabe für das [CLS]-Token allerdings kontextualisiert und somit spezifisch für die konkrete Eingabe. Aufgrund der gleichzeitigen Betrachtung des gesamten Kontextes arbeitet *BERT* mit einer festen Eingabelänge. Sollte eine Eingabe kürzer als diese Länge sein, werden die restlichen Token mit [PAD]-Token aufgefüllt. Im Vortraining der Originalversion ist diese maximale Eingabelänge 128 für 90 % der Trainingsdaten und nochmals 10 % mit einer maximalen Eingabelänge von 512 Token.

Devlin et al. präsentieren zwei Modelle von *BERT*. *BERT_{BASE}* besteht aus zwölf Enkodierern mit 768 versteckten Einheiten und zwölf *Attention*-Köpfen. Damit besitzt das *BERT_{BASE}*-Modell 110M Parameter, die trainiert werden müssen. *BERT_{LARGE}* nutzt 24 Enkodierer mit 1024 versteckten Einheiten und 16 *Attention*-Köpfen, was zu 340M Parametern führt. Vortrainierte Versionen dieser beiden Modelle sind frei verfügbar und wurden auf der englischen Wikipedia und dem BooksCorpus [Zhu+15] trainiert. Die gesamte Trainings-

menge umfasst 3,3 Milliarden Wörter. Zum Veröffentlichungszeitpunkt konnte *BERT* in elf verschiedenen Sprachverarbeitungsaufgaben den Stand-der-Technik teils deutlich verbessern.

Möchte man *BERT* nun beispielsweise für die Klassifikation oder eine vergleichbare Problemstellung fein anpassen, wird zumeist wie in Abbildung 2.12 vorgegangen. Die Eingabe wird tokenisiert und mit dem [CLS]- und ggf. [PAD]-Token versehen. Je nach Aufgabe kann desweiteren mit dem [SEP]-Token eine semantische Trennung zwischen zwei Einheiten eingeführt werden. Daraufhin wird die Eingabe in das vortrainierte *BERT*-Modell gegeben und die Ausgabe für das [CLS]-Token, welche eine Art Vereinigung aller anderen Token darstellt, in ein als der eigentliche Klassifikator fungierendes weiteres Netz gegeben. Dieses besteht zumeist aus einem FFNN mit einer *Softmax*-Schicht zur Erzeugung einer Wahrscheinlichkeitsverteilung über K mögliche Ausgabekategorien. In Abbildung 2.12 ist beispielhaft eine binäre Klassifikation dargestellt. Es existieren also nur $K = 1$ Ausgabekategorien, wie es z. B. bei der Aufgabe der Bewertung, ob ein Satz als möglicher Folgesatz eines anderen Satzes infrage kommt, der Fall wäre.

2.10. Wort- und Dokumentähnlichkeit

Viele Aufgabenstellungen in der Sprachverarbeitung für die Softwaretechnik lassen sich auf Ähnlichkeitsvergleiche zwischen Artefakten zurückführen. So wird beispielsweise die Anforderungsnachverfolgbarkeit (vgl. Abschnitt 2.1.3) häufig auf einen Ähnlichkeitsvergleich zwischen der Anforderung und dem Zielartefakt zurückgeführt. Hierbei werden die Artefakte als Textdokumente aufgefasst und versucht zu bestimmen, welche Quelle zu welchen Zielartefakten ausreichend ähnlich sind, um ein Kandidat für eine TL zu sein. Da es sich hierbei um Textdokumente handelt, lässt sich dies letztlich auf Ähnlichkeiten zwischen Wörtern und Dokumenten zurückführen.

Üblicherweise werden diese Ähnlichkeiten auf Basis der in Abschnitt 2.9 vorgestellten Vektorrepräsentationen bestimmt. Die dahinterliegende Annahme ist, dass die Vektorraumdistancen der kontextbezogenen Repräsentation eine Annäherung an die semantische Distanz zwischen den Wörtern/Dokumenten darstellen. Um Ähnlichkeiten zwischen Wörtern zu bestimmen, lassen sich so beispielsweise der euklidische Abstand oder der Kosinus nutzen.

Euklidischer Abstand Der euklidische Abstand zweier Vektorrepräsentationen u und v (z. B. BoW-Vektoren oder Worteinbettungen) kann wie folgt berechnet werden:

$$d_{Euklid}(u, v) = \|u - v\|_2 \quad (2.6)$$

Je kleiner dieser Abstand, umso ähnlicher sind sich die Vektoren und potenziell auch die durch sie repräsentierten Wörter. Ein Nachteil des euklidischen Abstands für die Ähnlichkeitsberechnung ist, dass die Länge der Vektoren einen großen Beitrag zur resultierenden

Distanz haben kann [MRS08]. So können zwei BoW-Vektoren auf sehr ähnlichen Kontexten, sprich denselben Wörtern, aufbauen, aber einer der beiden deutlich länger als der andere sein, da einzelne Wörter mehrfach auftreten. Dies würde dazu führen, dass die beiden Vektoren eine höhere Distanz und somit eine niedrigere Ähnlichkeit erhielten, obwohl sie (nahezu) das gleiche Vokabular umfassen.

Kosinusähnlichkeit Um diesen Effekt zu kompensieren, wird oftmals anstatt des einfachen euklidischen Abstands, der Kosinus verwendet. Diese als Kosinusähnlichkeit bezeichnete Ähnlichkeitsmetrik berechnet den Kosinus des Winkels ϕ zwischen den Vektoren:

$$\text{cosSim}(u, v) = \frac{u \cdot v}{\|u\|_2 \|v\|_2} = \cos \phi \quad (2.7)$$

Sie entspricht dem Skalarprodukt der Vektoren normiert auf die Länge 1. Da die Ähnlichkeit auf dem Winkel zwischen den Vektoren basiert, ist sie unabhängig von der Länge der Vektoren.

Auch die Ähnlichkeit zwischen Dokumenten kann mittels des euklidischen Abstands oder der Kosinusähnlichkeit bestimmt werden. Hierzu wird jedes Dokument durch einen einzigen Vektor repräsentiert, welcher beispielsweise dem BoW-Vektor oder einer Absatzeinbettung entspricht. Eine weitere Variante zur Erstellung eines solchen Vektors stellt ein Aggregieren der einzelnen Vektoren der im Dokument enthaltenen Wörter dar. Ein Verfahren zur Aggregation wäre z. B. das Mitteln der Wortvektoren. Die eigentliche Ähnlichkeit der beiden Dokumente lässt sich daraufhin über den euklidischen Abstand bzw. die Kosinusähnlichkeit der aggregierten Vektoren bestimmen. Repräsentiert man ein Dokument als eine Einbettungs-Multimenge, also eine Menge von Vektorrepräsentationen der im Dokument enthaltenen Wörter, so kann eine Dokumentähnlichkeit auch als die maximale/minimale Ähnlichkeit zweier Wörter (Wort 1 aus Dokument 1 und Wort 2 aus Dokument 2) bestimmt werden. Allerdings tendieren diese Vorgehen dazu, die Zusammenhänge zwischen den Dokumenten zu stark zu vereinfachen bzw. auf einzelne Aspekte zu reduzieren.

Wortüberführungsdistanz Die Wortüberführungsdistanz (engl.: *Word Movers Distance*, WMD) [Kus+15] stellt eine weitere auf Einbettungs-Multimengen basierende Ähnlichkeitsmetrik für Dokumente dar. Sie lässt sich als ein Spezialfall der *Earth Mover's Distance* [RTG98] auffassen. Die Ähnlichkeit zweier Dokumente wird hierbei durch die minimale kumulative Distanz ihrer Einbettungs-Multimengen bestimmt. WMD fasst hierzu die euklidische Distanz eines Paares von Worteinbettungen als deren semantische Ähnlichkeit auf. Es setzt also auf die Ähnlichkeitsannahmen von Vektorrepräsentationen und insbesondere auf Worteinbettungen. Zunächst wird für jedes Wortpaar bestehend aus einem Wort aus Dokument A und einem Wort aus Dokument B ihre euklidische Distanz bestimmt. Daraufhin wird für jedes Wort aus Dokument A, dasjenige Wort aus Dokument B gesucht, zu welchem es den geringsten Abstand hat; welches also die geringste Distanz zur Überführung ineinander benötigt. Die Wortüberführungsdistanz ergibt sich dann aus

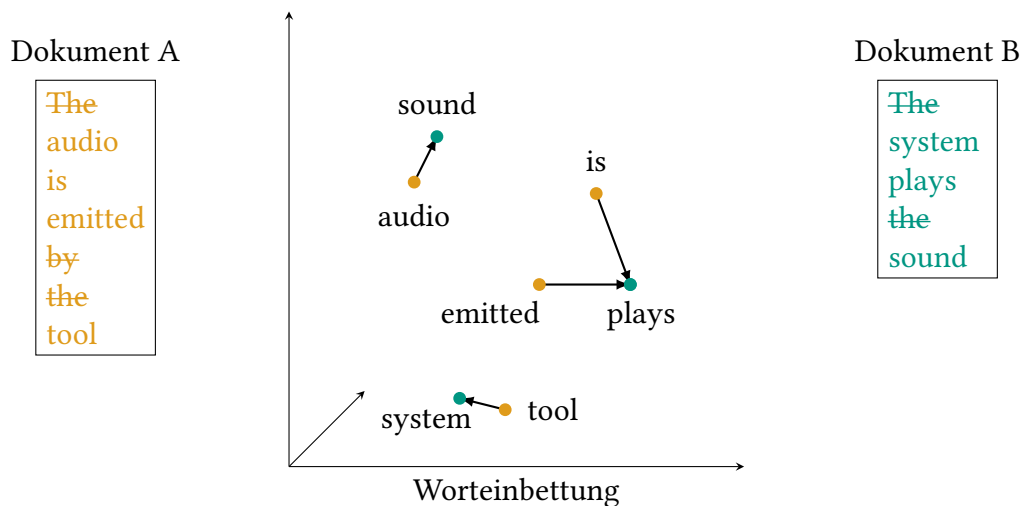


Abbildung 2.13.: Schematische Darstellung der Wortüberferungsdistanz für ein Dokumentenpaar. Nicht-Stoppwörter werden durch ihre Worteinbettungsvektoren repräsentiert (Punkte) und Pfeile verdeutlichen die minimalen Überführungen auf Wörter aus dem anderen Dokument, basierend auf der euklidischen Distanz zwischen ihren Worteinbettungen (angelehnt an Abbildung aus [Kus+15])

der Summe der Distanzen dieser minimalen Überführungen, also den minimalen Überführungskosten. Es wird also jedes Wort aus Dokument A auf das nächstgelegene/ähnlichste Wort aus Dokument B abgebildet und die Dokumentähnlichkeit sind die kumulativen Abbildungskosten. Außerdem werden einzelne Abbildungen der Wörter durch die normierten Auftretenshäufigkeiten im Dokument gewichtet. Dies folgt dem Gedankengang, dass häufige Wörter relevanter sind. Um mit dieser Gewichtung nicht auch unwichtige aber häufige Wörter miteinzubeziehen, wird die Wortüberferungsdistanz immer in Kombination mit einer Stopwortentfernung (vgl. Abschnitt 2.8.5) eingesetzt.

Abbildung 2.13 zeigt eine schematische Darstellung der minimalen Überführungen für ein Dokumentenpaar bestehend aus den Sätzen „*The audio is emitted by the tool*“ und „*The system plays the sound*“. Obwohl die Dokumente aus vollkommen unterschiedlichen Wörtern bestehen, kann mittels der semantischen Eigenschaften der verwendeten Worteinbettungsvektoren eine Ähnlichkeit der Dokumente festgestellt werden.

Jaccard-Koeffizient Eine weitere Möglichkeit Ähnlichkeiten zwischen Dokumenten zu bemessen, stellen mengentheoretische Ansätze dar. Am häufigsten verwendet wird hierbei der Jaccard-Koeffizient [Jac12]. Dieser beschreibt die Ähnlichkeit zweier Mengen (A und B) über das Verhältnis zwischen ihrer Schnittmenge und ihrer Vereinigung:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (2.8)$$

In der Anwendung für Dokumentähnlichkeit werden diese Mengen zumeist durch binäre BoW-Vektoren dargestellt und bemessen somit, welcher Anteil der insgesamt in den Dokumenten verwendeten Wörter in beiden Dokumenten auftauchen.

Tabelle 2.6.: Mögliche Ergebnisse einer binären Klassifikation beim Vergleich mit einem Goldstandard

		Vorhersage	
		Positiv	Negativ
Tatsächlich	Positiv	t_p	f_n
	Negativ	f_p	t_n

2.11. Evaluationsmetriken

Die Qualität eines Verfahrens wird zumeist anhand der Güte bestimmt, mit der es die zu erwartende Ausgabe bzw. das vorgegebene Ziel erzeugt bzw. erreicht. Handelt es sich dabei um eine erwartete Ausgabe, kann das Problem auf eine Klassifikationsaufgabe zurückgeführt und entsprechend evaluiert werden. Hierzu werden zumeist Musterlösungen, sogenannte Goldstandards, hergenommen, welche die erwartete Ausgabe repräsentieren und mit denen die von einem Verfahren erzeugte Ausgabe verglichen wird. Im Fall von TLR (s. Abschnitt 2.1) wäre diese Erwartungshaltung beispielsweise die korrekte Menge an TLs für die vorhandenen nachverfolgbaren Artefakte. Der Goldstandard würde also ausschließlich aus den zu erwartenden TLs bestehen und die Klassifikationsaufgabe wäre es, für alle potenziellen Quell- und Zielartefaktpaare zu bestimmen, ob eine TL zwischen ihnen besteht.

Bei einer solchen Klassifikationsaufgabe und somit dem Vergleich zwischen Goldstandard und erzeugter Ausgabe können die in Tabelle 2.6 dargestellten vier Fälle auftreten. Mit richtig positiv (engl.: *true positive*, t_p) wird eine positive Klassifikation des Verfahrens bezeichnet, welche der erwarteten positiven Klassifikation des Goldstandards entspricht. Das Verfahren hat also eine zu treffende Klassifikation auch tatsächlich richtig klassifiziert. Eine positive Klassifikation in der TLR wäre die Aussage, dass zwischen einem konkreten Quell- und Zielartefaktpaar eine TL besteht. Mit richtig negativ (engl.: *true negative*, t_n) wird eine klassifizierte Einheit bezeichnet, die laut Goldstandard nicht positiv klassifiziert werden sollte und auch durch das Verfahren nicht positiv klassifiziert wurde. Im Fall der TLR also ein Quell-/Zielartefaktpaar, für das korrekterweise keine TL identifiziert wurde. Falsch positiv (engl.: *false positive*, f_p) hingegen sind diejenigen Fälle, in denen das Verfahren eine Zuordnung zu einer Klasse durchführt, welche aber laut Goldstandard nicht vorgesehen war (z. B. TLs zwischen nicht zusammenhängenden Artefakten). Mit falsch negativ (engl.: *false negative*, f_n) werden dann diejenigen Fälle bezeichnet, in denen der Goldstandard eine Zuordnung zu einer Klasse erwartet hätte, das Verfahren diese aber nicht vorhergesagt hat. In der TLR wären dies nicht identifizierte TLs. Abbildung 2.14 stellt diese vier Fälle für ein Verfahren zur Generierung von TLs graphisch dar. Grüne TLs kennzeichnen im Goldstandard erwartete TLs, schwarze die vom Verfahren ausgegebenen. Kann das Problem nicht auf ein binäres Klassifikationsproblem zurückgeführt werden, weil mehr als eine Klasse vorhergesagt werden soll (z. B. bei Wortarterkennung) müssen die Fälle pro Klasse ermittelt werden.

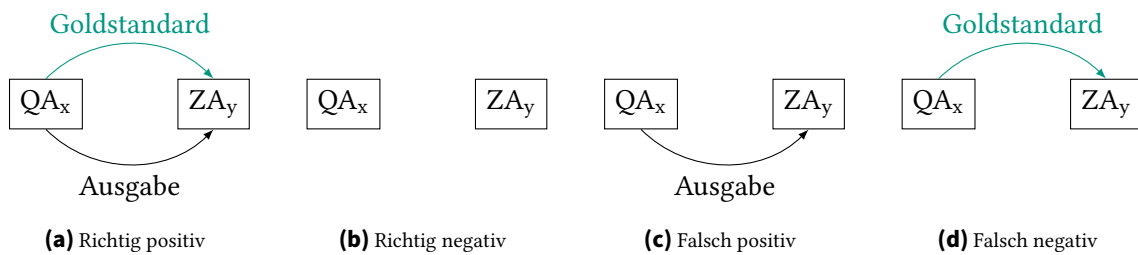


Abbildung 2.14.: Beispiele der Klassifikationsergebnisfälle für Nachverfolgbarkeitsverbindungen zwischen einem Quell- und einem Zielartefakt (QA bzw. ZA)

Auf Basis dieser Fälle lassen sich nun Metriken zur Bewertung der Güte der Klassifikationsergebnisse definieren. Präzision (engl.: *precision*), Ausbeute (engl.: *recall*), F_β -Maß und Genauigkeit (engl.: *accuracy*) sind die standardmäßig für Klassifikationsergebnisse verwendeten Metriken. Diese Metriken müssen ebenso bei einem Mehrklassenproblem pro Klasse bestimmt werden, können aber potenziell gemittelt werden.

Präzision Die Präzision beschreibt, wie gut ein Klassifikator darin ist, ausschließlich korrekte Vorhersagen zu produzieren. Sie gibt also den Anteil der richtigen Klassifikationen an allen von einem Klassifikator einer Klasse zugeordneten Einheiten an. Die Berechnung der Präzision setzt sich also aus den richtig positiv und falsch positiv Klassifizierten zusammen:

$$\text{Präzision} = \frac{t_p}{t_p + f_p} \quad (2.9)$$

Eine hohe Präzision bedeutet demnach, dass ein Klassifikator nur wenige Fehlklassifikationen produziert.

Ausbeute Die Ausbeute beschreibt die Fähigkeit des Klassifikators, alle erwarteten Klassifikationen zu produzieren. Sie gibt also den Anteil der vom Klassifikator richtig klassifizierten Einheiten an den insgesamt erwarteten Zuordnungen an. Somit betrachtet die Berechnung der Ausbeute ebenso die richtig positiv Klassifizierten, setzt sie aber ins Verhältnis zu den im Goldstandard vorhandenen Einheiten (diese entsprechen den richtig positiv zusammen mit den falsch negativ Klassifizierten, da diese eigentlich ebenso hätten positiv klassifiziert werden sollen):

$$\text{Ausbeute} = \frac{t_p}{t_p + f_n} \quad (2.10)$$

Eine hohe Ausbeute gibt demnach an, dass ein Klassifikator einen großen Anteil der erwarteten Zuordnungen zu einer Klasse auch produziert.

F-Maße Das Ziel eines Klassifikators sollte es sein, das erwartete Ergebnis, definiert durch den Goldstandard, bestmöglich abzubilden. Somit sollten sowohl Präzision als auch Ausbeute möglichst hoch sein. Um nun bemessen zu können, inwieweit ein Klassifikator dieses Ziel erreicht, wird oftmals das F_β -Maß angewandt. Dieses kombiniert Präzision und Ausbeute durch eine mit β angegebene Gewichtung miteinander:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Präzision} \cdot \text{Ausbeute}}{(\beta^2 \cdot \text{Präzision}) + \text{Ausbeute}} \quad (2.11)$$

Die am häufigsten verwendete Variante des F_β -Maß es ist das F_1 -Maß. Dieses entspricht dem harmonischen Mittel zwischen Präzision und Ausbeute:

$$F_1 = 2 \cdot \frac{\text{Präzision} \cdot \text{Ausbeute}}{\text{Präzision} + \text{Ausbeute}} \quad (2.12)$$

Wählt man für β Werte kleiner als 1, so gewichtet man die Präzision höher. Ist β hingegen größer als 1, so erhöht man den Anteil der Ausbeute am Endergebnis. So gewichten das $F_{0,5}$ -Maß bzw. das F_2 -Maß jeweils entweder die Präzision oder die Ausbeute doppelt so hoch wie die jeweils andere Metrik (vgl. Gleichung (2.13) bzw. Gleichung (2.14)).

$$F_{0,5} = 1,25 \cdot \frac{\text{Präzision} \cdot \text{Ausbeute}}{(0,25 \cdot \text{Präzision}) + \text{Ausbeute}} \quad (2.13)$$

$$F_2 = 5 \cdot \frac{\text{Präzision} \cdot \text{Ausbeute}}{(4 \cdot \text{Präzision}) + \text{Ausbeute}} \quad (2.14)$$

Genauigkeit Die Genauigkeit beschreibt die Güte eines Klassifikators allgemein, indem sie alle vier Fälle miteinbezieht. Das heißt, dass auch die richtig negativ klassifizierten Fälle miteinbezogen werden:

$$\text{Genauigkeit} = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \quad (2.15)$$

Ein Nachteil dieser Metrik liegt darin, dass in Klassifikationsproblemen mit einer verhältnismäßig großen Menge von nicht zuzuordnenden Einheiten (wie beispielsweise bei der TLR) die Metrik von den richtig negativen Klassifikationsergebnissen dominiert werden kann. In diesen Fällen wird die Güte des Verfahrens überschätzt. Deshalb wird in solchen Klassifikationsproblemen meist das F-Maß bevorzugt.

Metriken für Nachverfolgbarkeit Da bisherige Verfahren zur TLR noch weit von einem F_1 -Maß entfernt sind, das nötig wäre, um eine vollständige Automatisierung zu ermöglichen, liefern die meisten Verfahren anstatt nur der einzelnen für ein Quellartefakt relevanten TLs eine geordnete Liste aller möglicher Verbindungen. Diese Listen werden daraufhin Expert:innen vorgelegt, welche die tatsächlichen TLs bestimmen müssen. Um diesen die

Tabelle 2.7.: Beispielhafte Berechnung der durchschnittlichen Präzision (engl.: *average precision*) für eine Anfrage eines Quellartefakts mit vier möglichen Zielartefakten. Nur ZA₁ und ZA₃ werden laut Goldstandard erwartet

r	Verbindungsliste	t _p	f _p	Präzision(r)	Erwartet(r)
1	ZA ₁	1	0	1	1
2	ZA ₄	1	1	1/2	0
3	ZA ₂	1	2	1/3	0
4	ZA ₃	2	2	2/4	1

$$\left. \begin{array}{l} \\ \\ \\ \end{array} \right\} AP = \frac{1 \cdot 1 + 0 \cdot 1/2 + 0 \cdot 1/3 + 1 \cdot 2/4}{2} = 0,75$$

Arbeit der Auswahl der richtigen TLs zu erleichtern, ist es wichtig, dass die richtigen Ergebnisse möglichst weit oben in der geordneten Liste auftreten.

Die Qualität dieser Listen wird dabei durch den Mittelwert der durchschnittlichen Präzision (engl.: *mean average precision*, MAP) bestimmt. Hierzu werden die durchschnittlichen Präzisionen (engl.: *average precisions*, APs) der Listen für jedes Quellartefakt gemittelt. Die AP einer Ausgabeliste wird mittels der folgenden Formel bestimmt [MRS08]:

$$AP = \frac{\sum_{r=1}^{|\text{erhalten}|} (\text{Präzision}(r) \cdot \text{Erwartet}(r))}{|\text{erwarteteVerbindungen}|} \quad (2.16)$$

Wobei $|\text{erhalten}|$ die Anzahl der zurückerhaltenen TLs angibt. r identifiziert den Rang eines Elements in der geordneten Liste und mit $\text{Präzision}(r)$ wird die Präzision der Liste angegeben, wenn diese nach diesem Rang abgeschnitten wird. $\text{Erwartet}(r)$ ist eine binäre Funktion die angibt, ob die Nachverfolgbarkeitsverbindung an Rang r korrekt (1) oder inkorrekt (0) ist. Mit $|\text{erwarteteVerbindungen}|$ wird die Gesamtanzahl der laut Goldstandard für dieses Quellartefakt erwarteten TLs angegeben.

Tabelle 2.7 zeigt die Berechnung der AP für ein Quellartefakt und eine von einem Verfahren generierte Verbindungsliste. Der Goldstandard sieht in diesem Fall nur ZA₁ und ZA₃ als korrekte TLs vor. Daher setzt sich die durchschnittliche Präzision dieser Anfrage aus den Präzisionen der Verbindungsliste zusammen, wenn diese jeweils an den Rängen abgeschnitten wurde, wo die korrekten/erwarteten Zielartefakte in der Liste auftreten (Rang 1 und 4). Normiert wird daraufhin mit Zwei, da die vom Goldstandard erwarteten Verbindungen und somit die in die Berechnung einfließenden Faktoren Zwei sind.

Der Mittelwert der durchschnittlichen Präzision wird dann letztlich als arithmetisches Mittel über die durchschnittlichen Präzisionen aller Quellartefakte (QA) berechnet:

$$MAP = \frac{\sum_{n=1}^{|\text{QA}|} AP(n)}{|\text{QA}|} \quad (2.17)$$

Tabelle 2.8.: Kritische Werte für die Teststatistik W des Wilcoxon-Vorzeichen-Rang-Tests, welche unterschritten werden müssen, um die Nullhypothese abzulehnen

α	n						
	4	5	6	7	8	9	10
0,05	—	0	2	3	5	8	10

2.12. Hypothesentests

Statistische Hypothesentests, oder auch Signifikanztests genannt, prüfen die Gültigkeit einer Hypothese [Fah+16]. Da die Gültigkeit einer Hypothese statistisch nicht bewiesen werden kann, arbeiten sie mit der Gegenhypothese. Aufgrund von Messungen zeigen Hypothesentests, dass die Gegenhypothese unwahrscheinlich ist, um damit Schlüsse über die potenzielle Korrektheit der eigentlichen Hypothese zu ermöglichen. Hierbei wird oftmals als Nullhypothese H_0 die Gegenhypothese der eigentlichen Forschungshypothese angenommen und die zu überprüfende Forschungshypothese stellt die Alternativhypothese H_A dar. Ist die Alternativhypothese gerichtet, geht es also z. B. um eine Veränderung im positiven Sinne, so wird ein einseitiger, ansonsten ein zweiseitiger Test durchgeführt. Die statistischen Hypothesentests bestimmen eine Prüfgröße und bestimmen aufgrund dieser Prüfgröße, ob ein festgelegter kritischer Wert über- bzw. unterschritten wird. Wird dieser kritische Wert unterschritten, kann die Nullhypothese verworfen werden. Dieser kritische Wert bestimmt sich aus dem festgelegten Signifikanzniveau α , welches in der Softwaretechnik meist auf $\alpha = 0,05$ festgelegt wird. Zumeist wird der p -Wert bestimmt, welcher die Wahrscheinlichkeit darstellt, dass die aktuelle Beobachtung oder eine noch extremere gemacht wird, unter der Voraussetzung, dass die Nullhypothese gilt. Sofern dieser p -Wert unter dem Signifikanzniveau liegt, so handelt es sich um ein signifikantes Ergebnis und die Testentscheidung fällt für die Alternativhypothese, die eigentliche Forschungshypothese, aus.

Wilcoxon-Vorzeichen-Rang-Test Der gepaarte Wilcoxon-Vorzeichen-Rang-Test [Wil92] ist ein nicht-parametrisierter statistischer Hypothesentest für zwei abhängige („gepaarte“) ordinal skalierte Stichproben. Er betrachtet das Vorzeichen und den Rang der Differenz der Messwerte eines Stichprobenpaares. Es wird also nicht mit den absoluten Differenzen gerechnet, sondern bestimmt ob eine positive oder negative Differenz vorliegt und in welcher Ordnung die Differenzen zueinander stehen. Somit handelt es sich bei diesem Test um einen verteilungsunabhängigen Test.

Zur Bestimmung der Teststatistik W werden zunächst die Differenzen aller Stichprobenpaare bestimmt. Der Rang eines Paares wird dadurch bestimmt, dass die Paare unabhängig vom Vorzeichen von der kleinsten absoluten Differenz aufsteigend nummeriert werden. Das Stichprobenpaar mit der kleinsten absoluten Differenz enthält den Rang 1, das Paar mit der nächstgrößeren absoluten Differenz den Rang 2 usw. Daraufhin werden die Ränge der Stichprobenpaare mit positiven bzw. negativen Differenzen separat summiert. Hieraus

ergibt sich die Summe der positiven Ränge T_+ und die Summe der negativen Ränge T_- . Die Teststatistik W bildet sich aus der kleineren der beiden Summen:

$$W = \min(T_+, T_-) \quad (2.18)$$

Sollte ein Paar eine Differenz von Null aufweisen, wird dieses aus der Rangbildung ausgeschlossen. Bei Gleichständen der Differenzen mehrerer Paare wird als Rang der Durchschnitt der potenziellen Ränge der Paarungen verwendet. Haben beispielsweise vier Paare eine Differenz von 1 und dies ist der niedrigste Messwert, so würden diese potenziell die Ränge 1 bis 4 erhalten. Hierdurch wird ihr Rang auf $((1 + 2 + 3 + 4)/4) = 2,5$ gesetzt.

Für kleine Stichproben mit einer Stichprobenanzahl $n < 20$ muss die Signifikanz mittels der in Tabelle 2.8 dargestellten Werte bemessen werden, da hier der kritische Wert nicht asymptotisch normalverteilt ist. Ist also der Wert des Teststatistik W unterhalb des für die Stichprobenanzahl gegebenen Wertes, wird die Nullhypothese abgelehnt. Auf Basis der möglichen Verteilung der W -Werte bei einer gegebenen Anzahl an Stichproben kann dann die Wahrscheinlichkeit p für dieses Ereignis bestimmt werden. Ist p unterhalb des Signifikanzniveaus α so ist das Ergebnis statistisch signifikant.

3. Verwandte Arbeiten

In diesem Kapitel möchte ich für die Aufgabenstellung der Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext relevante Arbeiten vorstellen. Dies umfasst zum einen Arbeiten, die sich mit demselben Problem befassen (Abschnitt 3.1). Bei diesen werde ich auf die Stärken und Schwächen eingehen und herausarbeiten, was diesen Verfahren zu einer Anwendung in der Praxis fehlt. Zum anderen umfasst dieses Kapitel aber auch Arbeiten, die sich zwar mit Nachverfolgbarkeit befassen, aber zwischen anderen Artefakttypen abbilden (Abschnitte 3.2 bis 3.5). Da diese Arbeiten dieselbe Aufgabe auf anderen, aber teils verwandten Eingaben durchführen, lassen sich hier potenzielle Herangehensweisen für die eigentliche Aufgabenstellung dieser Arbeit ableiten. Abschließend werde ich in den Abschnitten 3.6 und 3.7 einen Überblick über Arbeiten zur Konzeptlokalisierung und Quelltextsuche geben. Diese Aufgaben sind zwar keine Form der TLR, stellen aber weitere Aufgaben der Softwaretechnik dar, welche sich mit der Abbildung von natürlichsprachlichen Dokumenten auf Quelltext befassen. Die als Quellartefakt betrachteten Dokumente unterscheiden sich vom Abstraktionsniveau und den verwendeten Begrifflichkeiten zwar deutlich von Anforderungen und die zu überbrückende semantische Lücke zwischen den Dokumenten ist geringer als bei Anforderungen und Quelltext. Trotzdem könnte die Ähnlichkeit der Aufgabenstellung dazu führen, dass hier interessante Erkenntnisse für die TLR gewonnen werden können.

3.1. Nachverfolgbarkeit von Anforderungen zu Quelltext

Die automatische Generierung von Nachverfolgbarkeitsinformationen zwischen Softwareartefakten ist im Fokus der Forschung seit den 1990er Jahren. Den ersten großen technologischen Durchbruch erzielte hierbei der Einsatz von Informationsrückgewinnung (engl.: *information retrieval*, IR) (s. Abschnitt 2.4). Die aufeinander abzubildenden Artefakte werden als natürlichsprachliche Texte aufgefasst. Kandidaten für TLs werden über die textuelle Ähnlichkeit der Artefakte identifiziert. Zur Repräsentation der Artefakte und Berechnung der Ähnlichkeiten wurden über die Jahre verschiedene IR-Methoden auf das Problem angewandt.

3.1.1. Verfahren der Informationsrückgewinnung

Antoniol et al. setzen in ihrer Arbeit „*Recovering Traceability Links between Code and Documentation*“ [Ant+02] auf probabilistische und Vektorraummodelle (engl.: *vector space*

Tabelle 3.1.: Übersicht über die am häufigsten verwendeten Datensätze zur TLR zwischen Anforderungen und Quelltext. Bei mehreren verschiedenen Versionen desselben Datensatzes wurde die Anzahl an TLs dem Bezeichner angehängt, um diese eindeutig zu identifizieren. Quellartefakttypen umfassen Handbucheinträge (HB), klassische Anforderungen (R) und Anwendungsfälle (UC). KB steht für eine textuelle Beschreibung von Quelltextklassen, wie sie anstatt des tatsächlichen Quelltextes in einigen Datensätzen als Zielartefakt verwendet werden. Mit n. V. werden nicht (mehr) online verfügbare Datensätze gekennzeichnet

Datensatz	Quelltyp	Sprache		Artefaktanzahl			Verfügbarkeit
		Natürl.	Prog.	Quelle	Ziel	TL	
LEDA	HB	EN	C++	88	208	98	n. V.
Albergate ₅₈	UC	IT	Java	16	60	58	n. V.
Albergate ₅₃	UC	IT	Java	17	55	53	[CoE]
EasyClinic ₉₃	UC	EN/IT	KB	30	47	93	[CoE]
EasyClinic ₃₇₃	UC	EN/IT	KB	30	47	373	n. V.
IBS	R	EN	KB	164	71	420	n. V.
EBT ₁₃₅	R	EN	Java	41	52	135	n. V.
EBT ₁₀₈₆	R	EN	Java	41	50	1086	n. V.
EBT ₉₈	R	EN	Java	41	50	98	[CoE]
LC	R	EN	KB	34	25	91	n. V.
SE450	R	EN	Java	46	475	1252	n. V.
eTour ₃₆₆	UC	EN/IT	Java	58	174	366	n. V.
eTour ₃₈₅	UC	EN/IT	Java	58	116	385	[TEF11]
eTour ₃₀₈	UC	EN/IT	Java	58	116	308	[CoE]
iTrust ₃₁₄	UC	EN	Java/JSP	k. A.	k. A.	314	n. V.
iTrust ₅₃₅	UC	EN	Java/JSP	131	332	535	n. V.
iTrust ₃₉₉	UC	EN	Java/JSP	131	367	399	[CoE]
iTrust ₅₈	UC	EN	Java/JSP	k. A.	k. A.	58	n. V.
SMOS	UC	IT/EN	Java	67	100	1044	[CoE]
eAnci	UC	IT	Java	139	55	567	[CoE]
Dronology	R	EN	Java	30	455	2985	n. V.
LibEST	R	EN	C	52	14	204	[Mor+20a]

models, VSMs), um Verbindungen zwischen Softwaredokumentation und Quelltext herzustellen. Hierbei werden die Artefakte zunächst vorverarbeitet, indem sowohl für die Dokumentation als auch den Quelltext, die Groß- und Kleinschreibung vereinheitlicht, Stoppwörter entfernt und eine morphologische Analyse eingesetzt wird, um den Text zu normalisieren (vgl. Abschnitt 2.8). Aus dem Quelltext werden zuvor alle Bezeichner extrahiert und ggf. in ihre Bestandteile aufgeteilt (AmountDue → amount und due). Zur Abbildung der Artefakte setzen Antoniol et al. zwei unterschiedliche Verfahren ein. Ein probabilistisches Modell, welches für jedes Quelltextartefakt Wahrscheinlichkeiten dafür bestimmt, dass ein Dokumentationsartefakt relevant für diesen Quelltext ist. Ein VSM fasst die Artefakte als Vektoren auf und bestimmt Kandidaten über die Kosinusähnlichkeit ihrer Vektoren (vgl. Abschnitt 2.10). Das probabilistische Modell setzt hierbei auf einen Markov-Prozess mit Unigram-Wahrscheinlichkeiten basierend auf der Häufigkeit der Wörter im

Artefakt. Für das VSM wird ein Artefakt durch einen Vektor repräsentiert. Dieser hat die Dimension des Vokabulars der Artefakte; also der Anzahl der einzigartigen Wörter im Datensatz. Ein Vektor eines Artefakts wird dann beschrieben durch die TF-IDF-Bewertung seiner Wörter (vgl. Abschnitt 2.9.1). Ein Eintrag ist also gleich Null, wenn das korrespondierende Wort nicht im Artefakt auftritt und ansonsten gleich der Worthäufigkeit des Wortes im Artefakt gewichtet mit der inversen Dokumenthäufigkeit des Wortes.

In einer abschließenden Fallstudie auf Version 3.4 der LEDA-Bibliothek und ihrer Dokumentation in englischer Sprache erreichte das VSM für eine geringere Zahl an Kandidaten (Zwölf) eine Ausbeute von 100 % bei einer Präzision von 3,92 %. Auf dem Albergate-System und seinen Anforderungen in Italienisch zeigten beide Verfahren eine ähnliche Qualität (F_1 -Maß von bis zu 49,15 %) und einem Erreichen von 100 % Ausbeute bei sechs Kandidaten mit einer Präzision von 13,80 %. Nähere Informationen zu diesen und weiteren häufig verwendeten Datensätzen finden sich in Tabelle 3.1¹. Hier wurde Albergate mit 58 Nachverfolgbarkeitsverbindungen verwendet (Albergate₅₈). Das erreichte F_1 -Maß und die niedrige Präzision bei hoher Ausbeute sind hierbei zwar ein erster Durchbruch, aber noch weit von einer Qualität entfernt, bei der von einer Automatisierung gesprochen werden kann. Entwickler:innen müssten pro Artefakt mindestens 12 Kandidaten analysieren, um die passenden korrespondierenden Artefakte zu erhalten.

Marcus und Maletic nutzen in ihrer Arbeit „*Recovering Documentation-to-source-code Traceability Links Using Latent Semantic Indexing*“ [MM03] hingegen Latente Semantische Indizierung (engl.: *latent semantic indexing*, LSI), zur Repräsentation der Semantik in Dokumentation und Quelltext. LSI basiert auf VSM, ist allerdings in der Lage, durch positive Korrelation zwischen ähnlichen Begriffen auch mit Synonymen besser umgehen zu können. Diese Korrelation hängt allerdings stark von den Formulierungen im Trainingskorpus, also in diesem Fall den Formulierungen in den Artefakten ab. LSI ist nur in der Lage, die häufigsten Beziehungen zwischen zusammen auftretenden Begriffen durch eine Projektion mithilfe von Singulärwertzerlegung [GR71] in einen Unterraum abzubilden (vgl. Abschnitt 2.8.9.1). Um ein Korpus zum Trainieren des LSI-Modells zu erstellen, wurden die Dokumentation und der Quelltext in Teile aufgeteilt, welche die Dokumente zur Abbildung darstellen. Marcus und Maletic entschieden sich hierbei dafür, dass der Quelltext einer Quelltextdatei ein Dokument darstellt, wohingegen für die Dokumentation die Gliederung in Abschnitte durch die Autoren der Dokumentation verwendet wird. Die Vorverarbeitung des Quelltextes entspricht hierbei weitestgehend dem Vorgehen von Antoniol et al. [Ant+02], wobei zusätzlich auch noch die Kommentare betrachtet werden. Die eigentliche Abbildung der Artefakte geschieht wiederum durch den Einsatz von Kosinusähnlichkeit zwischen den Vektoren der Artefakte, welche durch das LSI-Modell generiert werden. Der/Die Nutzer:in hat daraufhin die Möglichkeit, durch Setzen eines Schwellenwerts zu bestimmen, bis zu welcher Ähnlichkeit zwei Artefakte als Kandidaten für eine TL angesehen werden. Als zweite Variante nutzen die Autoren die Anzahl an auszugebenden Kandidaten pro Eingabeartefakt, wie es auch schon in der Arbeit von Antoniol et al. [Ant+02] vorgeschlagen wurde.

Abschließend vergleichen die Autoren ihr Verfahren mit dem von Antoniol et al. auf der

¹ Information zur Domäne der Datensätze finden sich in Anhang D.

LEDA-Bibliothek und dem Albergate₅₈-Datensatz. Für LEDA übertrifft das LSI-Verfahren die Vergleichsverfahren sowohl in Präzision als auch Ausbeute für jeglichen Schwellenwert. Außerdem wird eine Ausbeute von 100 % bereits bei 11 Kandidaten erreicht (Präzision von 11,79 %). Für den Albergate-Datensatz sind die Ergebnisse vergleichbar; allerdings wird hier 100 % Ausbeute bereits bei der gleichen Anzahl an Kandidaten (Sechs) mit einem höheren Präzisionswert von 16,38 % erreicht.

Den Einsatz von Themenmodellierung (engl.: *topic modeling*) zur Generierung von Nachverfolgbarkeitsinformationen schlagen Asuncion et al. in ihrer Arbeit „*Software Traceability with Topic Modeling*“ [AAT10] vor. Im Gegensatz zu den vorhergehenden Arbeiten fokussieren sich die Autoren auf eine Generierung von Nachverfolgbarkeitsinformationen während der Erstellung der Artefakte (engl.: *traceability link capture*) und nicht der TLR. Asuncion et al. nutzen hierbei Latente Dirichlet Allokation (engl.: *latent dirichlet allocation*, LDA) zur Themenmodellierung (vgl. Abschnitt 2.8.9.2), um, basierend auf dem Thema der aktuell bearbeiteten Architektur-Komponente, Entwickler:innen relevante Artefakte (Spezifikationen, Wiki-Seiten usw.) vorzuschlagen. Außerdem kann das Themenmodell durch Entwickler:innen dazu verwendet werden, die Menge der verfügbaren Artefakte zu filtern und zu kategorisieren.

Zur Evaluation haben die Autoren eine Fallstudie mit dem ArchStudio 4-System durchgeführt. Dabei lassen sie das Themenmodell sieben Themen über alle Artefakte hinweg lernen. Die vorgestellte Unterstützung zeigte sich als hilfreich für die Entwickler:innen des Projekts. In einer zweiten Evaluation bemessen Asuncion et al. die Möglichkeit ihr Themenmodell zu nutzen, um eine Wiederherstellung von Nachverfolgbarkeitsverbindungen durchzuführen. Hierzu verwenden sie LDA im VSM und vergleichen dessen Ergebnisse mit LSI auf dem EasyClinic₉₃-Datensatz (s. Tabelle 3.1). LDA erzielte hierbei durchweg bessere Ergebnisse als LSI.

3.1.1.1. Einsatz von externen Wissensquellen

Um mit dem Problem der semantischen Lücke zwischen den Wörtern im Quellartefakt und den Oberflächenformen im Zielartefakt umzugehen, wurden über die Jahre verschiedenste Techniken zur Integration von externen Wissensquellen in den IR-Prozess vorgestellt. Hayashi et al. nutzen in ihrer Arbeit „*Sentence-to-Code Traceability Recovery with Domain Ontologies*“ [HYS10] Domänenontologien, um domänenspezifisches Wissen und semantische Beziehungen für die Generierung von Nachverfolgbarkeitsinformationen zwischen Dokumentation und Quelltext nutzen zu können. Diese Domänenontologien werden händisch erstellt und enthalten Informationen über existierende Domänenkonzepte und wie diese miteinander in Beziehung stehen (z. B., dass in einem Zeichenprogramm Ovale gezeichnet werden können). Um diese Informationen nutzen zu können, extrahieren sie aus dem Quelltext (Java) den Aufrufgraphen und die enthaltenen Bezeichner. Die statische Analyse des Quelltextes liefert dabei Informationen über die Methoden und Methodenaufrufe. Die Bezeichner und Typen im Quelltext werden anhand ihrer Wichtigkeit und Häufigkeit ihres Auftretens bewertet; Klassen- oder Methodennamendefinitionen sind wichtiger als Referenzen auf Attribute und Methoden. Die Wörter aus Ein- und Ausgabe

werden außerdem mittels Stoppwortentfernung gefiltert und durch Wortstammbildung und Synonymauflösung normalisiert. Die Generierung der Nachverfolgbarkeitsverbindungskandidaten geschieht daraufhin, indem Teilgraphen des Aufrufgraphen gesucht werden, welche die in der Eingabe beschriebene Funktionalität abdecken. Diese Teilgraphen sollen eine möglichst große Abdeckung der Zusammenhänge der beschriebenen Funktionalität aus dem Satz und den zugehörigen Informationen aus der Domänenontologie aufweisen. Die Abbildung der Konzepte zwischen Satz, Ontologie und Quelltextelementen geschieht hierbei auf Basis von gleichen Wörtern.

Hayashi et al. bemessen die Qualität ihres Werkzeugs auf dem *Open Source Software* (OSS)-Projekt JDraw mit einer Domänenontologie mit 38 Konzepten und 45 Relationen. Für sechs ausgewählte Dokumentationssätze erreicht das Werkzeug mithilfe der Ontologie eine deutlich höhere Ausbeute (im Durchschnitt 32 Prozentpunkte). Allerdings sinkt die erreichte Präzision in vier der Fälle.

Eine weitere Wissensquelle integrieren Zou et al. in ihrer Arbeit „*Improving Automated Requirements Trace Retrieval*“ [ZSC10]. Sie nutzen den Inhalt des Projekt-Glossars, um bestimmten Termen in den Artefakten eine höhere Relevanz zuzuordnen. Hierzu wird ein probabilistisches Netzwerk eingesetzt, welches die Relevanz eines Artefaktpaars über eine bedingte Wahrscheinlichkeit berechnet. Diese bedingte Wahrscheinlichkeit ist als Funktion über die Häufigkeit eines Auftretens derselben Terme in beiden Artefakten definiert. Die Autoren nutzen dabei drei Erweiterungen des TF-IDF-Maßes. Zunächst erhöhen sie die Relevanz von Artefaktpaaren, die zwei oder mehr verschiedene Terme teilen. Zusätzlich sorgen gemeinsame Phrasen in einem Artefaktpaar für eine höhere Relevanz dieser Paarung. Als dritte Erweiterung wird das Projekt-Glossar genutzt, indem Artefaktpaare mit gemeinsam auftretenden Termen oder Phrasen, die sich im Projekt-Glossar befinden, als relevanter eingestuft werden.

Die Qualität des gewählten Ansatzes wurde auf fünf Datensätzen mit unterschiedlichen Zielartefakttypen evaluiert. Die Datensätze IBS, EBT₁₃₅ und LC bilden von Anforderungen auf UML-Klassenbeschreibungen ab (s. Tabelle 3.1). Der Datensatz SE450 mit Studentenprojekten hat als Zielartefakt Java-Klassen und der CM1-Datensatz verknüpft grob- mit feingranularen Anforderungen (s. Tabelle 3.2). Nur die Datensätze IBS und SE450 enthalten ein Projekt-Glossar. Die Ergebnisse zeigen, dass die drei Erweiterungen die Qualität des Verfahrens deutlich steigern konnten, allerdings der Effekt der einzelnen Erweiterungen von Projekt zu Projekt variierte. Der Einsatz des Glossars zeigte im IBS-Datensatz zwar eine Verbesserung, im SE450-Datensatz erzeugte dieser hingegen eine Verschlechterung.

Neben der detaillierteren Betrachtung der Zusammenhänge im Quelltext bietet auch eine tiefergehende Betrachtung der Semantik in den Anforderungen Potenzial, um die semantische Lücke zwischen den Artefakten zu schließen. Mahmoud und Niu untersuchen dazu in ihrer Arbeit „*On the Role of Semantics in Automated Requirements Tracing*“ [MN15] verschiedene Möglichkeiten, semantische Informationen in die Verarbeitung der IR-Methoden zu integrieren. Sie vergleichen hierzu den Einsatz von VSMs mit Thesaurus-Informationen und VSMs mit Wortartinformationen mit einem VSM ohne semantische Informationen und LSI- und LDA-Modellen. Für die VSMs mit Thesaurus-Informationen nutzen die Autoren WordNet (vgl. Abschnitt 2.7.2) und einen manuell erstellten domänenspezifischen Thesaurus, um Synonym-Beziehungen einfließen zu lassen. Die Synonym-Information

wird dann über einen Synonym-Koeffizienten in die Ähnlichkeitsberechnung einbezogen, wie es auch schon von Hayes et al. in ihrer Arbeit „*Advancing Candidate Link Generation for Requirements Tracing*“ [HDS06] vorgeschlagen wurde. Die wortartbasierten Modelle setzen die Einordnung der Terme in Wortarten ein, um nur bestimmte Wortarten für die Ähnlichkeitsberechnung zu nutzen. Mahmoud und Niu betrachten hierbei jeweils ein Modell, das nur Nomen bzw. Verben einbezieht. Zusätzlich bewerten die Autoren die Effektivität zweier Methoden zur Integration von Informationen über lexikalische Relationen zwischen den Konzepten in den Artefakten. Zum einen verwenden sie die Explizite Semantische Analyse (engl.: *explicit semantic analysis*), welche ein Textfragment durch einen hochdimensionalen gewichteten Vektor aus Konzepten aus Wikipedia repräsentiert. Hierzu wird für jeden Term im Textfragment seine Relevanz zu allen Artikeln aus der Wikipedia bestimmt und mit TF-IDF gewichtet. Der abschließende Vektor bildet sich dann aus der Summe der Vektoren für die einzelnen Terme. Dieser Vektor kann daraufhin mittels Kosinusähnlichkeit zur Bestimmung von TLs genutzt werden. Zum anderen setzen Mahmoud und Niu die Normalisierte Google Distanz (engl.: *normalized Google distance*) ein. Diese bestimmt die Ähnlichkeit zweier Terme über Anfragen an Googles Suchmaschine. Die Ähnlichkeit entspricht dem Anteil der Dokumente bei einer Suchanfrage, die beide Terme enthalten, an allen Dokumenten, die einen der beiden Terme enthalten. Sie bestimmen die Ähnlichkeit aller Terme in den Artefakten und speichern diese in einem Thesaurus, sodass wiederum das VSM-Verfahren für Thesauri genutzt werden kann.

Mahmoud und Niu vergleichen die Verfahren auf den Datensätzen iTrust₃₁₄ und eTour₃₈₅ (Anforderungen zu Quelltext, s. Tabelle 3.1) und CM1 (Anforderungen zu Anforderungen, s. Tabelle 3.2). Im Vergleich der VSMS übertraf der domänenspezifische Thesaurus den WordNet-basierten und die latenten Modelle, blieb allerdings für niedrigere Schwellenwerte hinter der Präzision des VSM zurück. Die wortartbasierten Modelle erreichen eine deutlich höhere maximale Präzision als das normale VSM, erreichen aber nur deutlich geringere maximale Ausbeutewerte auf den Anforderungen-zu-Quelltext-Datensätzen. Explizite Semantische Analyse erzielt bessere Ergebnisse als Normalisierte Google Distanz, übertrifft aber erst bei sehr niedrigen Schwellenwerten die Präzision des VSMS, erzielt allerdings insgesamt höhere Ausbeute Werte. Die Autoren konnten also feststellen, dass Wortarten-Filterung die höchste Präzision liefert, aber in dieser Form zu restriktiv ist, um eine akzeptable Ausbeute zu gewährleisten. Domänenspezifische Thesauri mit Synonym-Beziehungen ermöglichen hingegen eine höhere Ausbeute als alle anderen Verfahren bei höheren Präzisionswerten als Verfahren, die auf anderen lexikalischen Relationen basieren.

3.1.1.2. Kombinerende Ansätze

Die Vorteile verschiedener IR-Techniken mittels Kombination zu nutzen, schlagen Gethers et al. in ihrer Arbeit „*On Integrating Orthogonal Information Retrieval Methods to Improve Traceability Recovery*“ [Get+11] vor. Sie kombinieren VSMS mit probabilistischen Jensen-Shannon-Modellen (engl.: *probabilistic Jensen and Shannon models*, JSs) und relationalen Themenmodellen (engl.: *relational topic models*, RTMs). Die RTMs sind dabei eine Kombination von LDAs generativem Prozess und logistischer Regression. RTMs erlauben es

dadurch, Themenverteilungen über eine Term-Dokument-Matrix vorherzusagen. Diese können dann wiederum ähnlich zu vektorbasierten Verfahren als Vergleichsmaß zwischen Artefakten genutzt werden. Sie kombinieren immer zwei der drei Modelle mittels gewichteter affiner Transformation.

Gethers et al. evaluieren ihr Verfahren auf den Datensätzen eTour₃₆₆, EasyClinic₉₃, SMOS und eAnci (s. Tabelle 3.1). Es zeigt sich, dass sich eine Kombination des Themenmodells mit einer der anderen Techniken durchweg positiv auf die Qualität der generierten Kandidaten auswirkt. Sie konnten mit einer Kombination von JS und RTM eine durchschnittliche Präzision von über 60 % auf EasyClinic und 35 % auf eTour erreichen.

Auch Lohar et al. kombinieren in ihrer Arbeit „*Improving Trace Accuracy Through Data-driven Configuration and Composition of Tracing Features*“ [Loh+13] unterschiedliche Techniken zur Generierung von TLs. Allerdings kombinieren sie nicht die Ergebnisse verschiedener Abbildungsverfahren, sondern versuchen anhand initialer TLs die ideale Kombination aus Vorverarbeitung, Lexikonaufbau und Ähnlichkeitsberechnung zu bestimmen. Hierzu verwenden die Autoren einen genetischen Algorithmus, welcher den Merkmalsraum der möglichen Konfigurationen basierend auf den initialen Musterlösungen exploriert. Für die Vorverarbeitung können die Schritte Auftrennen von zusammengesetzten Bezeichnern, Wortstambildung, Stoppwortentfernung und Akronymexpansion jeweils sowohl für das Quell- als auch Zielartefakt genutzt werden. Als Möglichkeiten zum Lexikonaufbau wird entweder TF-IDF auf den Projektdokumenten oder auf dem American National Corpus [IS04] genutzt. Für die Auswahl des Kernalgorithmus, der Ähnlichkeitsberechnung, stehen LSI mit unterschiedlicher Anzahl an Themen und VSMS mit unterschiedlichen Ähnlichkeitsmaßen (Kosinus, Jaccard und Dice) zur Verfügung. Zur Evaluation des Verfahrens wurden 6 Projekte herangezogen; zwei Industrieprojekte aus der Verkehrsdomäne, welche zwischen Anforderungen und Entwurfsbeschreibungen abbilden und nicht öffentlich verfügbar sind, einem Datensatz für TLs zwischen Anforderungen und Quelltext (iTrust₅₃₅, s. Tabelle 3.1), einem Datensatz mit TLs zwischen Anforderungen und Testfallbeschreibungen (EasyClinic mit 63 erwarteten TLs) sowie zwei Datensätze mit TLs zwischen Anforderungen und Anforderungen (CCHIT und CM1, s. Tabelle 3.2). Die Ergebnisse zeigen, dass die besten Konfigurationen von Projekt zu Projekt mit Werten für die durchschnittliche Präzision von 83,5 % für EasyClinic bis hin zu 28,5 % für das größere der beiden Industrieprojekte variieren. Zusätzlich untersuchen Lohar et al. die beste Konfiguration über alle Projekte hinweg. Keine der Konfigurationen lieferte über alle Projekte ein gutes Ergebnis, doch die beste Konfiguration für iTrust ergab den besten durchschnittlichen Präzisionswert von 42,3 % über alle Projekte. Außerdem konnten die Autoren zeigen, dass auch bei unterschiedlichen Paarungen von Quell- und Zielartefakttyp im gleichen Projekt unterschiedliche Konfigurationen die besten Ergebnisse lieferten. Die Konfiguration hängt also nicht nur vom Projekt, sondern auch von den Artefakttypen ab.

Rodriguez et al. kombinieren in ihrer Arbeit „*Leveraging Intermediate Artifacts to Improve Automated Trace Link Retrieval*“ [RCF21] Verfahren zur direkten Abbildung zwischen Artefakten mit Verfahren, die dazwischenliegende Artefakttypen nutzen. Sie machen sich also zunutze, dass, falls Artefakte existieren, die auf einem Verfolgbarkeitspfad zwischen dem Quell- und dem Zielartefakt liegen (z. B. Entwurfsbeschreibungen, bei einem Pfad

Anforderungen→Entwurfsbeschreibungen→Quelltext), diese relevante Informationen für potenzielle TLs zwischen den Artefakten enthalten können. Für die Abbildung zwischen den Artefakten nutzen die Autoren dabei VSM und LSI. Die direkten Abbildungen eines Pfades von Quell- über Zwischen- zu Zielartefakt aggregieren sie, indem die Ähnlichkeiten multipliziert werden. Da potenziell mehrere Artefakte als Zwischenartefakt für ein Paar von Quell- und Zielartefakt fungieren können, werden diese über eine von drei Möglichkeiten aggregiert. Es wird entweder die maximale Ähnlichkeit, die Summe der Ähnlichkeiten oder eine mit Hauptkomponentenanalyse (engl.: *principal component analysis*, PCA) gewichtete Summe verwendet.

In einer Evaluation auf dem proprietären Softwaresystem TrainController und den Datensätzen Dronology, EasyClinic₃₇₃ und EBT₁₀₈₆ (s. Tabelle 3.1) zeigte die Kombination einer direkten Abbildung zwischen Quell- und Zielartefakt mit den aggregierten transitiven Ähnlichkeiten über eine einfache Summe, die besten Ergebnisse. Diese erzielte MAP-Werte von 39,9 % für Anforderung-zu-Quelltext-Verbindungen auf Dronology und bis zu 83,9 % auf EBT.

Gao et al. verbessern in ihrer Arbeit „Using Consensual Biterms from Text Structures of Requirements and Code to Improve IR-Based Traceability Recovery“ [Gao+23] IR-basierte TLR, indem sie gemeinsam auftretende Wortpaare in Anforderungen und Quelltext identifizieren. Ihr Ansatz TAROT umfasst vier Schritte. Zunächst werden gemeinsam auftretende Wortpaare in Anforderungen und Quelltext identifiziert. Ein gemeinsam auftretendes Wortpaar in einer Anforderung wird hierbei über zwei Token der Wortarten Verb, Nomen oder Adjektiv mit direkter Abhängigkeit im Abhängigkeitsgraphen eines Satzes (s. Abschnitt 2.8.6) definiert. Im Quelltext werden gemeinsam auftretende Wortpaare in Bezeichnern durch ein Auftrennen der Binnenmajuskel- oder Unterstrichschreibweise und anschließendem Bilden von aufeinander folgenden Wortpaaren identifiziert. Für Quelltextkommentare wird wie bei Anforderungen verfahren. Die gemeinsam auftretenden Wortpaare aus dem Quelltext werden mittels der identifizierten Wortpaare der Anforderungen gefiltert und nur diejenigen beibehalten, welche auch in den Anforderungen auftreten. Bei diesem Prozess wird die Reihenfolge der Wortpaare ignoriert und Grundformen der Wörter gebildet, sodass `sendEmail` und `emailSent` zu demselben Wortpaar (`send`, `email`) führt. Im zweiten Schritt von TAROT werden die Artefakttexte mittels der identifizierten Wortpaare angereichert. Hierzu werden die Wortpaare gewichtet mit ihrer Auftretenshäufigkeit der Vektorrepräsentation der Artefakte hinzugefügt. Im dritten Schritt werden nach einer Vorverarbeitung (Stoppwortentfernung, Kleinschreibung und Wortstambildung) mittels TF-IDF-Gewichtung und VSM, LSI oder JS Kandidatenverbindungen erzeugt. Abschließend werden die IR-Ergebnisse mit globalen und lokalen Gewichten auf Basis der Menge der geteilten Wortpaare der Artefakte angepasst.

In einer Evaluation auf einer neueren Version des iTrust- und Gantt-Datensatzes sowie den OSS-Projekten Maven, Pig, Infinispan, Drools, Derby, Seam2 und Groovy zeigt, dass TAROT eine signifikante Verbesserung im F_1 -Maß gegenüber Varianten ohne Identifikation gemeinsamer Wortpaare erzielt. Hierbei wurden für die 7 OSS-Projekte, die eigentlich keine Anforderungen enthalten, Belangbeschreibungen mittels Heuristiken [Gao+22] zusammengefasst, um anforderungsähnliche Eingaben zu generieren. Die so entstandenen

„Anforderungen“ sind also deutlich implementierungsnäher als klassische Anforderungen und enthalten oftmals bereits konkrete Verweise auf die Quelltextartefakte.

IR-basierte Ansätze erreichten einen großen Schritt in Richtung automatische Generierung von TLs, doch sie erreichen immer noch nur eine relativ geringe Präzision. Die meisten basieren auf den Wahrscheinlichkeiten und Ähnlichkeiten der Wörter und Token und haben somit Schwierigkeiten Ähnlichkeiten zwischen Artefakten zu bestimmen, die nicht textuell oder syntaktisch verwandt sind. Keiner der Ansätze ist in der Lage, die Semantik der natürlichsprachlichen Artefakte zu verstehen und sich zunutze zu machen.

3.1.1.3. Einbezug von Aufruf- und Datenabhängigkeiten im Quelltext

Eine andere Herangehensweise an dieses Problem stellt die Betrachtung von strukturellen Informationen und Abhängigkeiten des Quelltextes dar. Die Idee hier ist es, dass über die Verbindungen im Quelltext weitere relevante Artefakte für eine Suchanfrage gefunden werden können. Panichella et al. untersuchen in ihrer Arbeit „*When and How Using Structural Information to Improve IR-Based Traceability Recovery*“ [Pan+13], ob strukturelle Informationen die textuellen Informationen ergänzen können. Hierzu vergeben Sie für strukturell verbundene Zielartefakte einen adaptiven Bonus. Sie berechnen zunächst die textuelle Ähnlichkeit mittels JS- oder VSM-Modellen. Der adaptive Bonus ergibt sich aus der textuellen Ähnlichkeit, welche mit einem Bonusfaktor proportional zum Median der Varianz der Ähnlichkeitswerte gewichtet wird. Als strukturelle Verbindungen nutzen Panichella et al. Methodenaufrufe und Vererbungsbeziehungen.

Sie vergleichen das Ergebnis dieses Verfahrens mit dem alleinigen Anwenden der IR-Techniken auf den Datensätzen EasyClinic₉₃, eTour₃₆₆ und SMOS (s. Tabelle 3.1). Hierbei nutzen sie zwei Varianten ihres Verfahrens: Zunächst eine Variante, die den Bonus auf alle Kandidatenverbindungen anwendet und eine, die den Bonus nur dann anwendet, wenn ein/e Entwickler:in die ursprüngliche Kandidatenverbindung als korrekt markiert hat. Die Ergebnisse zeigen, dass die strukturellen Informationen die Qualität der Ergebnisse verbessert. Dies trifft allerdings zum Teil erst durch die Filterung durch Entwickler:innen zu. Dies deutet darauf hin, dass strukturelle Informationen nur in gewissen Situationen hilfreich sind und diese zu Erkennen, die eigentliche Herausforderung darstellt.

In der Arbeit „*Can Method Data Dependencies Support the Assessment of Traceability Between Requirements and Source Code?*“ [Kua+15] untersuchen Kuang et al. zusätzlich zu strukturellen Informationen außerdem Datenabhängigkeiten zwischen Methoden. Sie beschränken sich dabei auf Datenabhängigkeiten, die sich zur Laufzeit im Applikationsspeicher befinden. Um diese Abhängigkeiten zu finden, müssen sowohl direkte Datenabhängigkeiten als auch Abhängigkeiten über transitive Aufrufketten durch Aliase und Zeiger betrachtet werden. Hierzu verwenden Kuang et al. dynamische Quelltextanalysen und speichern sowohl die Datenabhängigkeiten als auch die Aufrufabhängigkeiten der Methoden in einem Graph. Diese Information nutzen die Autoren, um bestehende TLs zwischen Anforderungen und Quelltext zu überprüfen und gegebenenfalls zu erweitern bzw. zu verbessern. Zu diesem Zwecke verwenden sie zwei Klassifikatoren. Einen Basis-Klassifikator, welcher TLs basierend auf den Nachbarschaften einer Methode (Daten- bzw. Aufrufabhängigkeit) und deren

Zuweisungen zu einer Anforderung bestimmt. Besitzen mehr als 50 % der Nachbarmethoden eine TL zu einer Anforderung, wird davon ausgegangen, dass die aktuell betrachtete Methode ebenso wahrscheinlich eine Verbindung zu dieser Anforderung besitzt. Der zweite Klassifikator nutzt zusätzlich die inverse Datentyphäufigkeit, um Nachbarknoten, die ausschließlich über häufig geteilte Datentypen verbunden sind, aus der Klassifikation auszuschließen. Diese könnten zu Fehlvorhersagen führen. Die an die inverse Dokumenthäufigkeit von TF-IDF angelehnte Gewichtung fasst Datenabhängigkeiten als Dokumente und die entsprechenden Datentypen als Wörter auf. Dies bedeutet häufig auftretende Datentypen bekommen ein geringeres Gewicht.

Kuang et al. evaluieren ihr Verfahren auf fünf Softwaresystemen, VideoOnDemand (VoD), Chess, Gantt, JHotDraw und iTrust. Dabei wurden 91 Anforderungen zufällig gezogen und die eigentlich auf Klassen bezogenen TLs wurden jeweils auf alle Methoden der in den Klassen enthaltenen Methoden abgebildet. Die Ergebnisse zeigen, dass Datenabhängigkeiten komplementäre Informationen zu den Aufrufabhängigkeiten liefern und eine Kombination beider Informationen zu verbesserter Qualität der gefundenen TLs führt.

Ein etwas anderes Ziel verfolgen Hammoudi et al. in ihrer Arbeit „*TraceRefiner*“ [Ham+21]. Sie nutzen die Struktur des Quelltextes, um auf Basis von vorhandenen (grobgranularen) Anforderung-zu-Klassen-TLs feingranularere Anforderung-zu-Methoden-Verbindungen abzuleiten. Eine solche Abbildung auf feingranulare Elemente des Quelltextes ist für einen Analysten hilfreicher als die grobe Information der implementierenden Klasse [Kon+11]. Allerdings ist diese Aufgabe auch schwieriger, weshalb die Autoren sich dazu entschieden haben, auf vorhandene grobgranulare TLs zurückzugreifen. Ausgehend von den Informationen der Klassenverbindungen leiten sie entlang der Aufrufabhängigkeiten der Methoden, deren potenzielle Verbindungen ab. Das Vorgehen folgt der Annahme, dass Methoden, deren aufgerufene/aufrufende Methoden alle eine Verbindung zu einer Anforderung haben, auch mit hoher Wahrscheinlichkeit mit dieser Anforderung in Verbindung stehen. Da initial keine Methodenverbindungen existieren, wird diese Annahme zunächst dahingehend aufgeweicht, dass die Klassen der aufgerufenen/aufrufenden Methoden eine Verbindung zu dieser Anforderung haben.

In einer Evaluation auf den vier OSS-Projekten Chess, Gantt, iTrust und JHotDraw konnte das Verfahren eine Präzision von 61 % bei einer Ausbeute von 79 % erreichen.

3.1.2. Auffassung als Optimierungsproblem

Einen ganz anderen Ansatz für die automatische Generierung von TLs stellt das Auffassen des Problems als multikriterielles Optimierungsproblem dar. Hierbei ist das Ziel, Mengen an Artefaktpaaren zu finden, die Pareto-optimal bezüglich mehrerer Zielfunktionen sind. Eine dieser Pareto-optimalen Lösungen stellt daraufhin die zu erstellenden TLs dar. Rodriguez und Carver nutzen in ihrer Arbeit „*Multi-Objective Information Retrieval-Based NSGA-II Optimization for Requirements Traceability Recovery*“ [RC20] den genetischen NSGA-II-Algorithmus [Deb+02]. Der Algorithmus generiert basierend auf einer zufällig gebildeten initialen Population von Lösungsmengen optimierte Populationen, indem genetische Operatoren angewandt werden. Hierbei werden Lösungen ausgewählt und in ihnen

einzelne Artefaktpaare mutiert bzw. vertauscht. Rodriguez und Carver nutzen hierzu eine Wahrscheinlichkeit von 30 % für die Mutation und 50 % für ein Vertauschen ausgewählter Artefaktpaare. Als Zielfunktionen kommen zwei auf Ähnlichkeiten basierende Funktionen zum Einsatz. Zunächst eine Ähnlichkeit berechnet mittels des Jaccard-Koeffizienten (s. Abschnitt 2.10), welcher die Ähnlichkeit zweier Artefakte anhand des Anteils der überlappenden Wörter im Verhältnis zu der Gesamtmenge der Wörter in den Artefakten darstellt. Die zweite Zielfunktion basiert auf der Kosinusähnlichkeit der mit TF-IDF gewichteten Wortvektoren der Artefakte. Die Menge an Artefaktpaaren, die durch den Algorithmus generiert werden, wählen die Autoren anhand der Annahme, dass eine Anforderung maximal mit drei Klassen in Verbindung steht (dreifache Klassenanzahl).

Rodriguez und Carver evaluieren ihr Verfahren auf den Datensätzen EBT₉₈, Albergate₅₃ und eTour₃₀₈. Die Ergebnisse zeigen, dass dieser Ansatz zwar leicht niedrigere Präzisionen als IR-basierte Ansätze erzielt, allerdings in der Lage ist, deutlich höhere Ausbeuten (bis zu 77 %) zu erreichen. Im Durchschnitt über 10 Durchläufe erzielen sie auf eTour ein F_1 -Maß von 71,63 % bei einer Präzision von 83 % und einer Ausbeute von 63,4 %. Die Schwankungen über die verschiedenen Durchläufe bleiben dabei trotz zufälliger initialer Populationen im Rahmen von 24 %, sind aber zumeist nah an den durchschnittlichen Werten.

3.1.3. Worteinbettungen und maschinelles Lernen

Eine andere Herangehensweise an das Problem der semantischen Ähnlichkeit stellt die Nutzung von Worteinbettungen dar. Zhao et al. verwenden in ihrer Arbeit „*An Improved Approach to Traceability Recovery Based on Word Embeddings*“ [ZCS17] Worteinbettungen zur Repräsentation der Artefakte im Vektorraum. Sie nutzen ein auf Ranglernen (engl.: *learning to rank*) basierendes Verfahren für die Generierung von TLs. Als Worteinbettungen verwenden die Autoren für ihr WQI genanntes Verfahren ein selbst trainiertes CBOW-*word2vec*-Modell mit einer Fenstergröße von 5 und einer Dimension von 200 (vgl. Abschnitt 2.9.2). Dieses wurde auf 264M Wikipedia-Artikeln trainiert, welche das Wort *Software* in Kombination mit Wörtern aus einer softwarespezifischen Termliste enthielten. Als Vorverarbeitung nutzen Zhao et al. Stoppwortentfernung, Auftrennen von zusammengesetzten Bezeichnern und Kleinschreibung aller Wörter. Die Ähnlichkeit zweier Artefakte wird daraufhin über den Durchschnitt der maximalen Ähnlichkeiten der Wörter aus dem Quellartefakt mit den Wörtern des Zielartefakts bestimmt. Die Ähnlichkeiten jedes Wortes mit einem anderen Wort wird mittels der Kosinusähnlichkeit ihrer Worteinbettungsvektoren bestimmt. Falls es sich bei einem der Wörter um ein seltenes Wort in den Artefakten handelt (ermittelt mittels inverser Dokumenthäufigkeit), erhält die Ähnlichkeit einen Bonus. Anschließend wird eine Rang-basierte SVM auf den Merkmalen semantische Ähnlichkeit, Jaccard-Koeffizient, IDF-Summe, Anzahl an Schlüsselwörtern und Länge trainiert, um die Präzision des Verfahrens zu verbessern.

Zhao et al. evaluieren ihren Ansatz auf den Datensätzen CM1, GANNT, eTour₃₀₈, iTrust₃₉₉ und EasyClinic₉₃. Die Ergebnisse zeigen, dass das vorgestellte Verfahren ein als Vergleich herangezogenes LSI-Modell in allen Fällen außer EasyClinic (Anwendungsfall→Quelltext) im durchschnittlichen F_1 -Maß übertrifft. Das erzielte durchschnittliche F_1 -Maß variiert

von 14 % auf eTour bis hinzu 63 % auf EasyClinic. Außerdem konnten die Autoren zeigen, dass ihr Verfahren besser als ein rein *word2vec*-basiertes Verfahren TLs vorhersagt. Der Effekt von Ranglernen konnte als durchweg positiv bemessen werden, mit einer durchschnittlichen Verbesserung von 15,9 % im MAP.

Einen Nachteil vorangegangener Arbeiten, das Ignorieren der Reihenfolge der Wörter, versuchen Chen et al. durch die Betrachtung von sequenzieller Semantik auszugleichen. Die Autoren verbinden in ihrer Arbeit „*Enhancing Unsupervised Requirements Traceability with Sequential Semantics*“ [Che+19] distanzbewusste Sequenzmuster mit Dokumenteinbettungen. Die Sequenzmuster repräsentieren eine Abfolge von Token, die im Datensatz über einer festgelegten Häufigkeit auftreten. Hierbei wird die Distanz der Muster dadurch miteinbezogen, dass keine Abfolgen betrachtet werden, die eine größere Lücke als 2 Token aufweisen. Die Dokumenteinbettungen werden daraufhin mittels des PV-DBOW-Algorithmus von *doc2vec* (vgl. Abschnitt 2.9.2) auf der Konkatenation der Token der Anforderungen mit den in ihr enthaltenen Sequenzmustern trainiert. Da die Sequenzmuster potenziell redundante Informationen zu den Token der Anforderung enthalten, nutzen Chen et al. PCA. Eine TL zwischen zwei Artefakten wird schließlich dadurch bestimmt, dass die Ähnlichkeit ihrer Dokumenteinbettungen über 70 % der maximalen Ähnlichkeit zwischen den Artefakten liegt.

In einer Evaluation auf den Datensätzen CM1, Gantt, eTour₃₀₈, iTrust₃₉₉ und EasyClinic₉₃ übertrifft das Verfahren das WQI-Verfahren von Zhao et al. in allen bis auf einem Anwendungsfall (EasyClinic: Anwendungsfall → Testfall). Allerdings sind die Verbesserungen teilweise gering, da das Verfahren zwar eine höhere Präzision erreicht, aber in der Ausbeute zumeist geringfügig schlechtere Ergebnisse erzielt.

Eine weitere Form des Einsatzes von Worteinbettungen für das TLR präsentieren Zhang et al. in ihrer Arbeit „*Recovering Semantic Traceability between Requirements and Source Code Using Feature Representation Techniques*“ [Zha+21]. Die Autoren kombinieren *word2vec*-Worteinbettungen mit dem *multi-headed self-attention*-Mechanismus aus der Transformer-Architektur (vgl. Abschnitt 2.9.3). Hierdurch können sie die statischen Worteinbettungen dazu nutzen, kontextsensitive Texteinbettungen zu generieren, welche die eigentliche Semantik der Sätze besser abbilden. Hierzu werden die durch die *self-attention* gewichteten *word2vec*-Worteinbettungen mittels einer linearen Transformation zu einem „semantischen“ Vektor der Texteingabe transformiert. Diese Texteinbettungen vergleichen Zhang et al. daraufhin mittels Kosinusähnlichkeit. Den Quelltext unterteilen sie in Kommentare und sonstigen Quelltext und berechnen die Ähnlichkeit zwischen Anforderung und Kommentaren und Anforderung und sonstigem Quelltext einzeln pro Artefaktpaar. Die eigentlichen TLs ermitteln sie dann basierend auf einer gewichteten Summe der beiden Ähnlichkeiten. Aus der Menge der möglichen Nachverfolgbarkeitsverbindungskandidaten wählen die Autoren daraufhin diejenigen N mit der höchsten Ähnlichkeit aus.

In einer Evaluation auf den Datensätzen iTrust₃₉₉ und eTour₃₀₈ erzielte das Verfahren beim Verwenden der besten 10 % der Verbindungen ein durchschnittliches F_1 -Maß von 41,9 %.

In den letzten Jahren wurde auch überwacht maschinelles Lernen als Ansatz für das Generieren von TLs untersucht. Einen Nachteil dieses Ansatzes, das Vorhandensein von

Musterlösungsverbindungen für einen Großteil der Verbindungen in einem Projekt, versucht die Arbeit „*Tracing with Less Data*“ [Mil+19] von Mills et al. mittels aktivem Lernen (engl.: *active learning*) anzugehen. Ihr Ansatz ALCATRAL nutzt eine Kombination aus aktivem Lernen und einer binären Klassifikation, um TLs vorherzusagen. Hierzu wird ein Paar von Artefakten zunächst durch einen Vektor aus IR-, Textqualität- und Artefaktbasierten Merkmalen repräsentiert. IR-basierte Merkmale beschreiben die Ähnlichkeit der beiden Artefakte anhand verschiedener Ähnlichkeitsmetriken (VSM, LSI, LDA usw.). Textqualitätbasierte Merkmale beschreiben die Qualität des Quellartefakts bzw., ob das Artefakt generell schwer zu verbinden ist. Artefaktbasierte Merkmale liefern generelle Informationen über die Artefakte, wie z. B. die Anzahl der einzigartigen Terme oder den Überlapp der Terme der beiden Artefakte. Um die Anzahl von 131 Merkmalen zu reduzieren und eventuelle Redundanzen zu vermeiden, nutzt ALCATRAL Merkmalsauswahl mittels Pearson-Korrelation. Da die Gesamtmenge der möglichen Verbindungen generell deutlich mehr inkorrekte als korrekte Verbindungen enthält, nutzen Mills et al. Überabtasten (engl.: *oversampling*, OS) der unterrepräsentierten Klasse mittels SMOTE. Basierend auf einer initialen Menge von Musterlösungsverbindungen wird daraufhin ein Klassifikator trainiert, welcher vorhersagt, ob eine potenzielle TL valide oder invalide ist. Als Klassifikator nutzen Mills et al. einen Zufallswald (engl.: *random forest*, RF). Um die Menge an zusätzlich zu den initialen Musterlösungsverbindungen zu generierenden Verbindungen gering zu halten, nutzt ALCATRAL aktives Lernen. Es werden diejenigen Verbindungen Expert:innen zur Klassifikation vorgelegt, bei denen sich das aktuelle Modell am unsichersten ist. Daraufhin wird der Klassifikator auf dem bisherigen Trainingsdatensatz, ergänzt um die neu klassifizierten Verbindungen, trainiert. Dieses Vorgehen kann wiederholt werden bis eine festgelegte Menge an Trainingsdaten erzeugt wurde.

In einer Evaluation auf den Datensätzen eAnci, EasyClinic₉₃, eTour₃₆₆, iTrust₅₈, MODIS und SMOS konnten die Autoren zeigen, dass ALCATRAL bereits ab einer Trainingsmenge von 40 % der TLs vergleichbare oder bessere Ergebnisse als bisherige überwachte Lernverfahren (TRAIL von Mills et al. [MEH18]) mit 90 % der Trainingsdaten erreicht. Das Verfahren erzielt zwar den Stand der Technik in überwachter Generierung von TLs (F_1 -Maß von 61,1 % auf eTour und 67,7 % auf EasyClinic), kann aber weiterhin nur in Projekten eingesetzt werden, in denen eine nicht zu vernachlässigende Menge an Musterlösungsverbindungen bereits vorhanden ist. Außerdem muss für aktives Lernen Expert:innen zur Verfügung stehen, welchen die unsicheren Verbindungen vorgelegt werden können.

Ein Problem bei der Nutzung von Ähnlichkeitsmetriken zur Bestimmung von TLs stellt die Wahl eines geeigneten Schwellenwertes dar. Dieser hängt von der verwendeten Methodik und dem Projekt ab und kann zumeist nur mit bereits vorhandenen initialen Musterlösungsverbindungen bestimmt werden. Um trotzdem in Situationen ohne vorhandene Musterlösungsverbindungen die Stärken verschiedener IR- und ML-Techniken nutzen zu können, setzen Moran et al. ein Hierarchisches Bayes'sches Netz (engl.: *hierarchical Bayesian network*, HBN) ein. Hierbei fassen sie in ihrer Arbeit „*Improving the Effectiveness of Traceability Link Recovery Using Hierarchical Bayesian Networks*“ [Mor+20b] das Generieren von TLs als Bayes'sches Inferenzproblem auf, welches bestimmt, ob eine Verbindung zwischen zwei Artefakten als wahrscheinlich gilt. Sie verwenden in ihrem Verfahren COMET ein HBN mit drei Stufen: Zunächst werden zehn IR- bzw. ML-basierte textuelle

Ähnlichkeitsberechnungen kombiniert. In Stufe Zwei integrieren die Autoren Entwickler:innenrückmeldung zu in der ersten Stufe unsicheren Entscheidungen (sofern dies möglich ist). Abschließend werden in Stufe Drei Informationen zu transitiven Verbindungen zwischen weiteren Artefakttypen in die aktuelle TLR integriert. Für die Betrachtung eines vollautomatischen Verfahrens spielen hierbei nur die Stufen Eins und Drei eine Rolle. Die Kombination der Ähnlichkeitsberechnungen in Stufe Eins erfolgt, indem zunächst die Ähnlichkeitswerte der Berechnung mittels einer Sigmoidfunktion normalisiert werden, welche auf den Median der Verteilung aller Ähnlichkeitswerte zwischen zwei Artefakten im Datensatz zentriert ist. Daraufhin wird eine logistische Regression auf die normalisierten Ähnlichkeitswerte angewandt, um die Parameter einer Beta-Verteilung zu bestimmen, welche letztlich das Wahrscheinlichkeitsmodell, abgebildet durch eine diskrete Bernoulli-Verteilung, informiert. Die Informationen aus Stufe Zwei und Drei informieren ebenso das Wahrscheinlichkeitsmodell. Die eigentliche Entscheidung, ob zwei Artefakte eine TL erhalten sollen, wird daraufhin über das Bestimmen der A-posteriori-Wahrscheinlichkeit mittels des Bayes Theorems bestimmt. Da die Berechnung dieser Wahrscheinlichkeit nicht trivial ist nutzen Moran et al. die Approximationsverfahren Maximum-a-posteriori-Schätzung (MAP), Markov-Ketten-Monte-Carlo-Verfahren mit No-U-Turn Abtastung (NUTS) und Variationsinferenz (VI).

In ihrer Evaluation konnten die Autoren zeigen, dass ihr Verfahren auf den Datensätzen eTour₃₀₈, EBT₉₈, SMOS, iTrust₃₉₉ und LibEST (s. Tabelle 3.1) auf Stufe Eins bereits nah an die durchschnittliche Präzision desjenigen Verfahrens aus ihrer Kombination herankommt, welches jeweils die ideale Wahl gewesen wäre. Stufe Drei erzielt nur geringfügig bis keine Verbesserung. Stufe Zwei allerdings erzielt den manuell erkauften und zu erwartenden positiven Effekt.

Die Verfahren zur automatischen Generierung von TLs haben sich über die Jahre deutlich verbessert. Allerdings benötigen sie meist entweder eine Menge an Musterlösungsverbindungen, welche in der Praxis selten vorliegt, oder sie erreichen entweder eine hohe Präzision oder eine hohe Ausbeute, aber selten beides. Dies verhindert eine vollständige Automatisierung des Prozesses, welche F_1 -Maße von über 80 % benötigen würde [HDS06]. Außerdem besteht ein Mangel an großen Musterlösungsdatensätzen, was die Einsatzmöglichkeiten von komplexen ML-Verfahren beschränkt.

3.2. Nachverfolgbarkeit von Anforderungen zu Anforderungen

Neben den in Abschnitt 3.1 bereits vorgestellten Verfahren, die neben der vertikalen TLR zwischen Anforderungen und Quelltext auch die Nachverfolgbarkeit zwischen unterschiedlicher Formen von Anforderungen betrachten [ZSC10; MN15; Loh+13; RCF21; ZCS17; Che+19; MEH18; Mil+19], existieren auch Ansätze, die sich rein auf diese Form beschränken. Da diese Form einen Artefakttyp mit der Zielsetzung dieser Arbeit teilt, bieten die Verfahren einen Einblick in mögliche Verarbeitungsschritte für das Verständnis von Anforderungsartefakten.

Tabelle 3.2.: Übersicht über die am häufigsten verwendeten Datensätze zur TLR zwischen Anforderungen und Anforderungen. Artefakttypen umfassen klassische grobgranulare Anforderungen (R), feingranulare Entwurfsbeschreibungen (EB), Komponentenbeschreibungen (KB) und regulatorische Vorschriften (RV). Informationen zu den abgedeckten Domänen können in Anhang D eingesehen werden

Datensatz	Sprache	Artefakttyp		Artefaktanzahl			Verfügbarkeit
		Quelle	Ziel	Quelle	Ziel	TL	
MODIS	EN	R	EB	19	50	41	[SM05]
CM1	EN	R	EB	235	220	361	[CoE]
InfusionPump	EN	R	KB	126	21	131	[CoE]
Gantt	EN	R	EB	17	69	68	[CoE]
WARC	EN	R	EB	63	89	136	[CoE]
CCHIT	EN	R	RV	116	1064	587	[CoE]

Hayes et al. kombinieren in ihrer Arbeit „*Improving Requirements Tracing via Information Retrieval*“ [HDO03] IR-basierte Verfahren mit einem Thesaurus, um besser mit unterschiedlichen Ausdrücken für dasselbe Konzept umgehen zu können. Konkret nutzen sie TF-IDF in Kombination mit einem einfachen Thesaurus. Dieser Thesaurus besteht aus Tripeln, welche die gefühlte Ähnlichkeit zweier Ausdrücke angeben. Diese Einträge werden manuell von Analyst:innen aufgrund von Schlüsselwörtern oder -phrasen bestimmt. Die Ähnlichkeiten dieser Thesauruseinträge fließen in die Kosinusähnlichkeit zur Bestimmung von möglichen TLs ein. Als Vorverarbeitung setzen Hayes et al. Stoppwortentfernung und Wortstambildung ein.

Sie evaluieren ihr Verfahren auf dem MODIS-Datensatz, welcher übergeordnete mit untergeordneten Anforderungen in Verbindung setzt. Weitere Informationen zu diesem und anderen häufig verwendeten Datensätzen zur TLR zwischen Anforderungen und Anforderungen finden sich in Tabelle 3.2. Ihr Verfahren konnte eine Präzision von 40.7 % und eine Ausbeute von 85.4 % erreichen.

Assawamekin et al. nutzen in ihrer Arbeit „*Ontology-Based Multiperspective Requirements Traceability Framework*“ [ASP10] Ontologien als Repräsentation der beschriebenen Informationen in Anforderungen. Auf Basis dieser Repräsentation nutzen sie Techniken des Ontologieabgleichs (engl.: *ontology matching*), um automatisch TLs zu generieren. Zunächst werden mittels eines Zerteilers (vgl. Abschnitt 2.8.6) die syntaktischen Abhängigkeiten in den Anforderungen bestimmt. Basierend auf diesen Abhängigkeiten werden zunächst Ausdrücke, die wenig Informationen tragen (z. B. Artikel oder Hilfsverben), herausgefiltert, die Ausdrücke lemmatisiert und zusammengesetzte Nomen verbunden. In den übrigen Abhängigkeiten werden regelbasiert neun Typen von Zusammenhängen identifiziert (u.a. Attribut-, Eigenschafts- und Generalisierung/Spezialisierungs-Beziehungen) und in Prädikatform gespeichert. Diese Zusammenhänge werden daraufhin in einer Ontologie gespeichert, welche einer Basisontologieform folgt (s. Abschnitt 2.7.1). Um nun TLs zwischen zwei Anforderungsdokumenten zu identifizieren, werden Korrespondenzen in den jeweiligen Ontologien gesucht. Mittels semantischer Relationen aus WordNet und Stringähnlichkeiten werden auch nicht direkte Übereinstimmungen zugelassen. Die ei-

gentlichen TLs ergeben sich daraufhin aus einer Überdeckungsanalyse, welche bestimmt, ob zwei Konzeptmengen vollständig, einschließlich oder partiell übereinstimmen.

In einer Evaluation auf drei Ausprägungen eines Krankenhausinformationssystems und zwei Ausprägungen eines Hochschulinformationssystems konnte das Verfahren eine durchschnittliche Präzision von 87 % und eine durchschnittliche Ausbeute von 77 % für vollständig überdeckte Teile der Anforderungen erzielen.

Schlutter und Vogelsang verwenden in ihrer Arbeit „*Improving Trace Link Recovery Using Semantic Relation Graphs and Spreading Activation*“ [SV21] ebenfalls Abhängigkeitsgraphen und kombinieren diese mit der Erkennung semantischer Rollen (s. Abschnitt 2.8.7), um einen gemeinsamen semantischen Graph der Quell- und Zielartefakte zu erstellen. Der semantische Graph repliziert die Phrasenstruktur der Eingabe (vgl. Abschnitt 2.8.6) und erweitert sie um Argumentstrukturen aus der Erkennung semantischer Rollen, indem Kanten zwischen den entsprechenden Verbphrasen und den die Argumente repräsentierenden Phrasen gezogen werden. Zusätzlich werden diejenigen Teile des Graphen zusammengeführt, welche dieselben Strukturen überdecken. So werden in mehreren Anforderungen wiederkehrende Wörter, Nominalphrasen oder Verben durch denselben Knoten repräsentiert. Auch Pronomen, die dieselbe Entität darstellen wie eine Nominalphrase, also welche korreferent sind, werden verbunden. Ihr Ansatz [SV20; SV21] ermittelt basierend auf diesem Graph mögliche TLs durch den Einsatz von Aktivierungsausbreitung (engl.: *spreading activation*). Ausgehend von den Knoten eines Quellartefakts werden dafür die mit diesen verbundenen Knoten aktiviert und dies in mehreren Iterationen pulsartig wiederholt, um im Graph wichtige Knoten zu identifizieren. Die Kandidatenliste möglicher TLs bestimmt sich daraufhin durch eine mit TF-IDF gewichtete Aggregation der Aktivierungswerte der Wörter in den möglichen Zielartefakten.

In einer Evaluation auf den Datensätzen InfusionPump, CCHIT, Gantt, CM1 und WARC (s. Tabelle 3.2) mit grobgranularen Anforderungen als Quell- und feingranularen Anforderungen als Zielartefakt konnten die Autoren MAP-Werte von bis zu 50 % bei einer Ausbeute von 90 % erzielen.

Guo et al. nutzen in ihrer Arbeit „*Semantically Enhanced Software Traceability Using Deep Learning Techniques*“ [GCC17] RNNs in Kombination mit Worteinbettungen, um basierend auf einer Menge von initialen TLs weitere mögliche Nachverfolgbarkeitsverbindungskandidaten vorherzusagen. Hierzu wird zunächst ein Worteinbettungsmodell mittels des CSG-Vorgehens von *word2vec* auf einem großen Korpus von Domänentexten trainiert. Dieses Modell wird dazu verwendet, die Wörter der Quell- bzw. Zielartefakte zu repräsentieren. Die so repräsentierten Wörter werden daraufhin in ein RNN gegeben, um einen einzelnen semantischen Vektor pro Artefakt zu erhalten. Diese semantischen Vektoren dienen als Eingabe für eine weitere Schicht, welche basierend auf dem Winkel und der Distanz zwischen den beiden Vektoren bestimmt, ob ein potenzieller Nachverfolgbarkeitsverbindungskandidat vorliegt. Die Autoren vergleichen verschiedene Arten von RNNs wie vergatterte rekurrente Einheiten (engl.: *gated recurrent units*, GRUs), LSTMs und ihre bidirektionalen Varianten. Um die Parameter des Netzes zu trainieren, werden die initialen TLs verwendet.

In einer Evaluation auf einem Industriedatensatz aus der Zugsteuerungsdomäne mit 1650 Anforderungen als Quell- und 466 als Zielartefakte erzielte das BI-GRU mit einer Trai-

ningsmenge von 45 % der Daten das beste Ergebnis mit MAP-Werten von 59,8 %. Dies liegt 41 % bzw. 32 % über den Werten, die ein VSM bzw. LSI auf demselben Problem und denselben Daten erzielt. Werden sogar 80 % der Daten als Training eingesetzt, erzielt das Verfahren MAP-Werte von bis zu 83,4 %. Allerdings ist durch das Verwenden von mehr Daten für das Training auch der Testdatensatz geschrumpft, was dazu führt, dass das auf 45 % der Daten trainierte Modell auf diesem kleineren Testdatensatz ebenfalls MAP-Werte von bis zu 80 % erreicht.

Wang et al. nutzen ebenfalls ANNs für die Nachverfolgbarkeit von Anforderungen zu Anforderungen. In ihrer Arbeit „*Enhancing Automated Requirements Traceability by Resolving Polysemy*“ [Wan+18] setzen sie ein FFNN dazu ein, um korreferente Entitäten zu bestimmen. Hierbei werden diejenigen Terme identifiziert, die dieselbe Echtwelt-Entität beschreiben. Hiermit wollen sie das Polysemieproblem (ein Wort, das mehrere Bedeutungen hat) bei IR-basierter TLR verringern. Die Idee ist, dass durch die Identifikation von Korreferenzen zwischen Termen, bestehende Mehrdeutigkeiten besser aufgelöst werden können. Das FFNN bestimmt hierbei innerhalb einer Anforderung diejenigen Terme, die am meisten verwandt miteinander sind. Auf Basis dieser Terme werden über die Anforderungen hinweg mittels eines Gruppen-Paar Rangmodells (engl.: *cluster-pair rank model*) Gruppen korreferenter Terme bestimmt. Diese korreferenten Terme werden daraufhin in einer Anforderung-Term-Matrix als ein Term zusammengefasst.

Hiermit konnten die Autoren in einer Studie auf acht Projekten zeigen, dass VSM- und LSI-basierte TLR signifikant verbessert werden konnte. Auf dem CM1-Datensatz erzielt das verbesserte LSI-Verfahren ein F_2 -Maß von 69 % im Gegensatz zu den 60 % des LSI-Verfahrens ohne Bestimmung korreferenter Terme.

Lin et al. nutzen in ihrer Arbeit „*Enhancing Automated Software Traceability by Transfer Learning from Open-World Data*“ [Lin+22] Transferlernen, um vortrainierte Sprachmodelle auf das Problem der TLR anzupassen. Ihr Ansatz NLTrace soll hierdurch mit einer geringeren Menge an Trainingsdaten (vorhandenen TLs) auskommen können und somit das Kaltstartproblem vieler ML-basierter Ansätze für die TLR verringern. Die Autoren untersuchen dabei drei verschiedene Strategien des Transferlernens. Zum einen trainieren sie ein bestehendes bert-base-uncased-Modell [Dev+19] auf einem Datensatz aus Kommentaren, Belangen, Anfragen zur Übernahme von Quelltextänderungen (engl.: *pull requests*) und Einbuchungen von GitHub-Projekten aus den Jahren 2016 bis 2021² weiter, um ein generelles Softwaretechnik-Sprachmodell zu generieren. In der zweiten Strategie wird das bert-base-uncased-Modell für jedes zu evaluierende Projekt mit einem kleinen domänenspezifischen Korpus feinangepasst. Dieser Korpus wird automatisch mittels Google-Suche der relevantesten Terme im Projekt aufgebaut. In der dritten Strategie wird das bert-base-uncased-Modell auf der verwandten Aufgabe der Vorhersage von Verbindungen zwischen Belangen und Einbuchungen aus dem GitHub-Datensatz feinangepasst. Die so entstandenen Modelle vergleichen sie mit unangepassten Sprachmodelle und klassischen IR-basierten TLR-Verfahren auf drei verschiedenen Aufgaben. Diese sind die Vervollständigung bestehender TLs, die Identifikation von TL für neu hinzugefügte Arte-

² Der Datensatz umfasst 2,1 TB an Daten.

fakte und der Generierung von TLs ohne vorhandene initiale TLs.

In einer Evaluation auf den Projekten CCHIT, PTC, Dronology und CM1 konnte das Modell mit der Feinanpassung auf der verwandten Aufgabe der Vorhersage von Verbindungen zwischen Belangen und Einbuchungen auf allen Aufgaben das beste Ergebnis erzielen. Auf der Vervollständigungsaufgabe erzielt das Modell eine Verbesserung von 10,2 % im F_2 -Maß und 13,84 % im MAP gegenüber dem ursprünglichen *BERT*-Modell. Bei der Identifikationsaufgabe auf neuen Artefakten ist die Verbesserung mit 27,7 % im F_2 -Maß und 29,9 % im MAP sogar noch größer. Lediglich auf der Aufgabe der Generierung ohne vorhandene TLs schneidet das klassische VSM-Verfahren im MAP um 7 % besser ab. Wird dem Modell allerdings Zugriff auf 10 vorhandene TLs gegeben, so kann es auch hier eine Verbesserung von 28,6 % im F_2 -Maß und 11,4 % im MAP erzielen. Allerdings erzielt es dabei auf CM1 ein F_2 -Maß von 60,4 % und einen MAP von 56,2 % und damit schlechtere Ergebnisse als vorherige Arbeiten. Selbiges gilt für CCHIT. Die anderen Datensätze wurden in bestehenden Arbeiten nicht verwendet.

Die Herangehensweisen für Nachverfolgbarkeit von Anforderungen zu Anforderungen ähneln denjenigen für Anforderung zu Quelltext (s. Abschnitt 3.1). Eine Ausnahme stellen hierbei Sprachmodelle wie *BERT* dar, welche zum Teil überzeugende Ergebnisse auf dieser Aufgabe erzielen. Ein Grund warum diese bisher nicht auf Anforderungen zu Quelltext angewandt wurden, ist die Art der Eingaben mit denen diese Sprachmodelle arbeiten. Diese sind auf großen Mengen von natürlichsprachlichen Sätzen vortrainiert. Die Informationen des Quelltextes liegen aber oftmals nicht in Form von syntaktisch vollständigen Sätzen vor. Daher ist es fraglich, ob diese Art von Modellen tatsächlich ohne Weiteres auf die oftmals nur aus einzelnen Termen bestehenden Informationen im Quelltext angewandt werden können. Auch andere linguistische Informationen, wie beispielsweise die semantischen Rollen, Abhängigkeitsgraphen oder Korreferenzen lassen sich für Quelltext nicht oder nur schwierig in der Form bestimmen, wie dies für natürlichsprachlichen Text möglich ist, wie er in Anforderungen vorliegt. Somit bestätigen die hier vorgestellten Arbeiten entweder die Erkenntnisse aus Abschnitt 3.1 oder werfen die Frage auf, ob und wie Sprachmodelle für die TLR von Anforderungen zu Quelltext eingesetzt werden können.

3.3. Nachverfolgbarkeit von Testfällen zu Quelltext

Die vorangehend beschriebenen Aufgaben der TLR beschäftigen sich beide mit Nachverfolgbarkeit von Anforderungen. Aber auch die Nachverfolgbarkeit von Quelltext zu weiteren Artefakttypen kann interessante Informationen für Verarbeitungsschritte und Herangehensweisen bezüglich des Quelltextes liefern. Die in der Literatur am häufigsten betrachtete Form der vertikalen Nachverfolgbarkeit von Quelltext, welche nicht auf Anforderungen abbildet, ist die Nachverfolgbarkeit von Testfällen zu denjenigen Quelltextteilen, die diese testen.

Csuvik et al. setzen dabei in ihrer Arbeit „*Source Code Level Word Embeddings in Aiding Semantic Test-to-Code Traceability*“ [CKV19] Dokumenteinbettungen für die Identifikation

von TLs zwischen JUnit-Testfällen und Java-Klassen ein. Hierzu vergleichen sie die Ergebnisse dreier Strategien, ein *doc2vec*-Modell (s. Abschnitt 2.9.2) zu trainieren. Die erste Variante trainiert das Modell auf dem Quelltext selbst. Hierbei wird der Quelltext als strukturierter Text aufgefasst und anhand von Klammern, Interpunktion und Binnenmajuskeln in Token aufgetrennt. Die zweite Variante basiert auf dem abstrakter Syntaxbaum (engl.: *abstract syntax tree*, AST). Hierbei werden die Token durch eine Tiefensuche auf dem AST bestimmt, indem für jeden besuchten Knoten sein korrespondierender Typ ausgegeben wird. Die dritte Variante trainiert das Modell nur auf den Blattknoten des AST. Hierbei ist ein Satz die Menge der Blattknoten eines Unterbaums. Konstanten werden mit Platzhaltern ersetzt und Binnenmajuskelschreibweise aufgelöst. Diese *doc2vec*-Repräsentationen trainieren die Autoren auf den OSS-Projekten Apache Commons Lang, Apache Commons Math, JFreeChart und Mondrian. Hierbei unterscheiden sie zwischen einem Training auf jedem Projekt einzeln, einem Training auf allen Projekten zusammen und einem Training auf allen Projekten mit zusätzlicher JavaDoc-Dokumentation (s. Abschnitt 2.3.2). Die eigentliche Abbildung zwischen Testfall und Java-Klassen geschieht daraufhin mittels einer Kosinusähnlichkeitsbestimmung der jeweiligen *doc2vec*-Repräsentationen.

Die Ergebnisse auf den vier Projekten zeigen, dass das Modell, welches auf den Blattknoten und jedem Projekt einzeln trainiert wurde, die höchste Präzision erzielt. Das Verfahren erzielt dabei eine deutlich höhere Präzision als LSI. Eine Hinzunahme von JavaDoc-Dokumentation erhöhte die Präzision nur geringfügig. Leider treffen die Autoren keine Aussage über die Ausbeute des Ansatzes und es bleibt somit unklar, ob die höhere Präzision auf Kosten einer niedrigeren Ausbeute erzielt wurde.

In ihrer Arbeit „*Large Scale Evaluation of Natural Language Processing Based Test-to-Code Traceability Approaches*“ [KCV21] erweitern Kicsi et al. den *doc2vec*-basierten Ansatz von Csuvik et al. [CKV19] um das Ausnutzen von Namenskonventionen und eine Betrachtung von Aufrufabhängigkeiten. Die Aufrufabhängigkeiten werden genutzt, um Quelltextklassen auszuschließen, die keine Methode enthalten, welche im Rahmen des Testfallrumpfes aufgerufen wird. Verwendet ein Projekt Namenskonventionen für die Testfallbenennung, so werden diese dazu genutzt, TLs zu identifizieren. Heißt die Testklasse beispielsweise `SomeClassTest` und es existiert eine Klasse `SomeClass` so wird diese als möglicher Nachverfolgbarkeitsverbindungskandidat angesehen. Zusätzlich nutzen Kicsi et al. einen Ensembleansatz, um die ähnlichsten Klassen pro Testfall zu finden. Hierbei wird die Ähnlichkeitsliste des *doc2vec*-Modells gefiltert, indem diejenigen Klassen entfernt werden, die sich nicht in den Top-N-Ergebnissen der Ähnlichkeitslisten eines LSI- und eines TF-IDF-Ansatzes befinden.

Die Evaluation auf den Projekten ArgoUML, Apache Commons Lang, Apache Commons Math, Gson, JFreeChart, Joda-Time, Mondrian und PMD zeigt, dass die besten Ergebnisse hinsichtlich Präzision ohne Ensembleansatz erzielt werden. Hierbei schneidet das *doc2vec*-Modell mit Aufrufabhängigkeiten und Ausnutzen von Namenskonventionen am besten ab und erzielt auf den Projekten Commons Lang, Gson, JFreeChart und Joda-Time eine durchschnittliche Präzision von 76,6 % auf den Top-5-Ergebnissen. In dieser Evaluation schneidet das *doc2vec*-Modell, welches den gesamten Quelltext als strukturierten Text auffasst, am besten ab. Leider bemessen die Autoren auch hier nicht die Ausbeute ihres Verfahrens und argumentieren damit, dass diese gleich der Präzision sei. Da sie aber

Präzision der Top-N-Ergebnisse bemessen, ist auch hier die Ausbeute relevant für eine Einordnung der Güte ihrer Ergebnisse.

White und Krinke vergleichen in ihrer Arbeit „*TCTracer*“ [WK22] verschiedene Techniken und ihre Kombinationen für eine feingranulare Abbildung zwischen Testfall und getesteter Methode. Außerdem überprüfen sie, ob eine Anwendung der Techniken auf Klassenebene (Abbildung Testfallklasse auf Quelltextklasse) durch die Informationen der Abbildung auf Methodenebene und andersherum verbessert werden kann. Als Techniken setzen sie Namenskonventionen, längste gemeinsame Untersequenz (engl.: *longest common subsequence*), Levenshtein-Distanz [Lev66], den letzten Aufruf vor einer Zusicherung (engl.: *assertion*), TF-IDF und die Fehlerlokalisierungstechnik Tarantula [JHS02] ein.

In ihrer Evaluation auf den fünf OSS-Projekten Apache Ant, Commons IO, Commons Lang, JFreeChart und Gson erzielt auf Methodenebene längste gemeinsame Untersequenz die besten Einzelergebnisse hinsichtlich MAP und F_1 -Maß. Auf Klassenebene erzielt neben längste gemeinsame Untersequenz auch die Levenshtein-Distanz beste Einzelergebnisse. Eine Aggregation der Methodenebene auf Klassenebene führt zu schlechteren Ergebnissen, genauso wie eine Verbesserung der Klassenebene durch die Informationen aus der Methodenebene. Ein Hauptgrund ist die stark gestiegene Anzahl an falsch positiven TLs. Um die Kombination der Techniken zu evaluieren, setzen die Autoren drei Kombinationsstrategien ein: ein einfaches Mitteln der Ergebnisse aller Techniken, eine Gewichtung anhand der erzielten Präzision der Technik in alleiniger Anwendung und ein FFNN trainiert auf den Ergebnissen der Einzelanwendung. Sowohl auf Methoden- als auch Klassenebene erzielen die ersten beiden Strategien bessere Ergebnisse als die Einzelanwendung der besten Technik. Wobei auf Methodenebene das einfache Mitteln mit einem MAP von 85 % und einem durchschnittlichen F_1 -Maß von 81 % die besten Ergebnisse erzielt. Auf Klassenebene erzielen beide Strategien ein durchschnittliches F_1 -Maß von 90 % und MAP von ca. 94 %. Die FFNN-basierte Gewichtungsstrategie konnte keine Verbesserung erzielen.

Insgesamt zeigen die Ergebnisse der TLR zwischen Quelltext und Testfällen, dass auf dieser Aufgabe bereits eine hohe Präzision und auch Ausbeute durch einfache Stringähnlichkeiten und das Ausnutzen von Aufrufabhängigkeiten erzielt werden können. Gerade die bereits sehr guten Ergebnisse mit Namenskonventionen zeigen, dass diese beiden Artefakttypen syntaktisch sehr nah beieinander liegen und zumeist dieselben Bezeichner und Begrifflichkeiten verwenden oder sogar explizit aufeinander verweisen. Hieraus Erkenntnisse zur TLR zwischen Anforderungen und Quelltext abzuleiten, ist somit schwierig, da keine syntaktische und semantische Lücke zwischen den Artefakten besteht und diese zumeist in derselben Phase der Entwicklung entstehen. Anforderungen und Quelltext hingegen weisen eine deutlich größere Lücke zwischen den verwendeten Begriffen und dem beschriebenen Abstraktionsniveau auf und benötigen somit Ansätze, die in der Lage sind, diese Lücke zu schließen. Die Erkenntnis der Arbeit von Kicsi et al., dass *doc2vec* in der Lage ist, die Ähnlichkeit zwischen Quelltextartefakten präziser zu bestimmen als LSI oder TF-IDF, unterstützt allerdings die Erkenntnisse der worteinbettungsbasierten Verfahren zur TLR zwischen Anforderungen und Quelltext (s. Abschnitt 3.1.3).

3.4. Nachverfolgbarkeit von Belangen zu Versionskontrollsystem-Einbuchungen

Eine weitere, gerade in den letzten Jahren vermehrt untersuchte Form der TLR ist die Nachverfolgbarkeit von Belangen (engl.: *issues*) und Einbuchungen eines Versionskontrollsystems (engl.: *version control system*, VCS). Heutige VCSs enthalten zwar eingebaute Funktionalitäten, um die Belang-IDs, auf die sich eine Einbuchung bezieht, manuell anzugeben, oftmals enthalten Einbuchungen aber trotzdem keinen Bezug zu einer spezifischen Belang-ID oder umfassen fehlerhafte Zuordnungen [Rat+18]. Um unvollständige oder fehlerhafte Zuordnungen in einem Projekt zu vermeiden, beschäftigen sich einige Arbeiten mit der automatischen Identifikation dieser Zuordnung.

Rath et al. nutzen in ihrer Arbeit „*Traceability in the Wild*“ [Rat+18] verschiedene prozess- und textbezogene Merkmale, um einen Klassifikator zur Identifikation von fehlenden Belang-IDs in Einbuchungsnachrichten zu trainieren. Als prozessbezogene Merkmale nutzt ihr Klassifikator die Information des Einbuchenden und des dem Belang Zugeordneten. Außerdem werden die zeitliche Beziehung zwischen dem Anlegen des Belangs bzw. dem Beseitigen des Belangs und der Einbuchung, die letzte vorangehende/nachfolgende Einbuchung, die dem Belang zugeordnet ist sowie die Anzahl an Belangen und TLs, die zum Zeitpunkt der Einbuchung existieren als prozessbezogene Merkmale verwendet. Als textbezogene Merkmale nutzt das Verfahren textuelle Ähnlichkeiten basierend auf VSMs mit N-Grammen. Hierbei wird jedes Dokument, entweder Einbuchungsnachricht, Belangbeschreibung oder Quelltextdatei, als Vektor seiner enthaltenen N-Gramme aufgefasst und mit TF-IDF gewichtet. Es werden dabei N-Gramme der Größe 1 bis 4 genutzt. Zum einen wird die textuelle Ähnlichkeit einer Einbuchungsnachricht zur Belangbeschreibung und zum anderen die textuelle Ähnlichkeit der Belangbeschreibung zur ähnlichsten eingebuchten Quelltextdatei betrachtet. Als Klassifikator nutzen die Autoren einen RF.

In einer Evaluation auf den sechs OSS-Projekten Maven, Derby, Infinispan, Groovy, Drools und Pig mit insgesamt über 14000 Belangen und über 23000 Einbuchungen konnte der Klassifikator im Durchschnitt ein $F_{0,5}$ -Maß von 76,6 % erreichen. Hierbei wurde evaluiert, wie gut das Verfahren die bereits vorhandenen Zuordnungen vorhersagen konnte. Um auch eine Aussage über fehlende Zuordnungen treffen zu können, haben Rath et al. eine zufällig gezogene Anzahl der durch den Klassifikator neu gefundenen TLs (die zuvor als falsch positiv angesehen wurden) der drei Projekte Derby, Drools und Maven vier Forscher:innen zur Bewertung vorgelegt. Von diesen 240 Nachverfolgbarkeitsverbindungskandidaten waren 30 tatsächliche TLs.

Auch die Arbeit von Lin et al. betrachtet die Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Belangen und Einbuchungen. Ihre Arbeit „*Traceability Transformed*“ [Lin+21] setzt hierzu Transferlernen eines *BERT*-Sprachmodells ein. Der Ansatz T-BERT baut dabei auf dem *CodeBERT*-Modell [Fen+20] auf, welches auf dem *CodeSearchNet*-Datensatz [Hus+20] vortrainiert wurde. Dieses Sprachmodell wurde auf einer Menge von 2.3M Paaren von natürlicher Sprache, in Form von Methodendokumentation, und zugehörigem Quelltext sowie 6.4M Quelltextauschnitten von Methoden ohne Dokumentation vortrainiert.

T-BERT sieht einen dreischrittigen Prozess vor. Zunächst wird das *CodeBERT*-Modell trainiert. Darauf folgt ein Zwischentraining auf der Aufgabe für eine gegebene Methode zu bestimmen, ob eine Methodendokumentation diese beschreibt. Der Gedanke hierbei ist, dass dieses Zwischentraining eine verwandte Aufgabe zur TLR darstellt und deshalb die große Menge an verfügbaren Trainingsdaten aus dem CodeSearchNet-Datensatz dazu genutzt werden kann, das Modell weiter auf die Aufgabe der TLR vorzubereiten. Abschließend wird das Modell dann auf die eigentliche Aufgabe der Wiederherstellung von Nachverfolgbarkeitsverbindungen feinangepasst. Hierbei werden 824342 Belang-Einbuchungs-Paare aus den drei OSS-Python-Projekten Pgcli, Flask und Keras verwendet. T-BERT sieht drei unterschiedliche Architekturen für den Klassifikator vor. Die TWIN-Architektur nutzt zwei einzelne *BERT*-Modelle, um natürlichsprachliche und programmiersprachliche Artefakte separat zu repräsentieren. Diese Repräsentationen werden daraufhin in einem Vektor zusammengefasst und als Eingabe in den Klassifikator genutzt. Bei der SINGLE-Architektur wird ein einzelnes *BERT*-Modell genutzt, indem die natürlichsprachlichen und Quelltextartefakte mit speziellen Token markiert und dann konkateniert an das Modell übergeben werden. Die daraus resultierende Repräsentation wird als Eingabe für den Klassifikator genutzt. Die SIAMESE-Architektur ist eine Mischform der beiden anderen Architekturen. Hier wird ein einzelnes *BERT*-Modell verwendet, aber beide Eingabetypen separat in das Modell eingegeben und die Ausgaben wiederum wie bei der TWIN-Architektur verarbeitet.

In einer Evaluation auf einem nicht in der Feinanpassung gesehenen Teil der Belang-Einbuchungs-Paare aus Pgcli, Flask und Keras zeigte, dass sich eine Anwendung des Zwischentrainingsschrittes positiv auf die Leistung auswirkt. Von den drei Architekturen erzielt die SINGLE-Architektur auf allen Projekten die beste Leistung im F_1 -Maß, F_2 -Maß und MAP. Hierbei erzielt das Modell deutlich bessere Ergebnisse als ein VSM-, LSI- oder LDA-basierter Ansatz. Mit einem durchschnittlichen F_1 -Maß von 86,2 % und einem MAP von 92,3 % sind diese Ergebnisse gegenüber dem besten Vergleichsverfahren (VSM) mit einem durchschnittlichen F_1 -Maß von 47,2 % und MAP von 58,3 % vielversprechend.

Lin et al. untersuchen in ihrer Arbeit „*Information Retrieval versus Deep Learning Approaches for Generating Traceability Links in Bilingual Projects*“ [LLC21] die Leistung verschiedener IR- und Sprachmodell-basierter Ansätze auf bilingualen Projekten. Hierzu haben sie einen Datensatz aus 14 Chinesischen, einem Koreanischen, einem Japanischen und einem Deutschen OSS-Projekt mit zugehörigen Belang-Einbuchungs-Paaren erstellt. Ein Kriterium bei der Auswahl der Projekte war dabei, dass diese hauptsächlich englische Sprache umfassen, aber mindestens 1 % des verwendeten Vokabulars in der Herkunftssprache des Projektes ist. Somit variieren die Anteile an nicht englischer Sprache von 1 bis 31 %. Als Verfahren vergleichen die Autoren VSM, LSI und LDA mit und ohne automatischer Übersetzung der Fremdsprachenanteile.

Hierbei konnte gezeigt werden, dass eine Übersetzung als Vorverarbeitungsschritt die Genauigkeit der Verfahren signifikant erhöhen konnte und das VSM-basierte Verfahren mit einer durchschnittlichen Präzision von 68 % am besten abschneidet. Als weitere Betrachtung untersuchen Lin et al. den Einsatz von Worteinbettungen. Hierzu nutzen sie vortrainierte *fastText*-Modelle für Englisch und bilinguale *fastText*-Modelle. Hierbei zeigte ein englisches *fastText*-Modell mit Übersetzung als Vorverarbeitungsschritt die

beste durchschnittliche Präzision. Bilinguale und englische Modelle auf unübersetzten Datensätzen zeigen signifikant schlechtere Ergebnisse als dieser Ansatz und auch das beste VSM-Verfahren.

Als letzten Ansatz untersuchen die Autoren den Einsatz eines mehrsprachigen *BERT*-Modells. Hierbei nutzen sie die SIAMESE-Architektur aus ihrem T-BERT-Ansatz. Sie verwenden allerdings lediglich die Architektur wieder, um einen Klassifikator auf Basis eines mehrsprachigen DistilBERT-Modells [San+20] feinanzupassen. Zur Evaluation fassen Lin et al. die 14 chinesischen Projekte zu einem großen Datensatz zusammen. Das Modell wird dann auf $\frac{8}{10}$ des kombinierten Datensatzes trainiert und auf jeweils $\frac{1}{10}$ (462 TLs) validiert und getestet. Auf dieser Aufteilung erzielt das feinangepasste mehrsprachige DistilBERT-Modell eine relative Verbesserung von 23,9 % im F_1 -Maß und 27,8 % in der durchschnittlichen Präzision gegenüber dem besten worteinbettungsbasierten Verfahren. Beide Werte liegen aber mit ca. 35 % nicht besonders hoch. Die hier deutlich schlechteren Ergebnisse des Worteinbettungsverfahrens lassen sich auf den deutlich größeren Suchraum des kombinierten Datensatzes zurückführen, welcher für ein IR-basiertes Verfahren hinderlich ist. Da ein Umfang von 14 Projekten allerdings auch kein realistisches Szenario für die TLR darstellt, sind die Ergebnisse nicht verwunderlich. Wird hingegen jedes der chinesischen Projekte in Trainings-, Validierungs- und Testmenge aufgeteilt und ein gemeinsames Modell auf den Trainings- und Validierungsmengen trainiert und dann pro Projekt auf der Testmenge getestet, so erzielt das worteinbettungsbasierte Verfahren auf 10 von 14 Projekten ein besseres F_1 -Maß und eine bessere durchschnittliche Präzision. Es fällt auf, dass das IR-basierte Worteinbettungsverfahren insbesondere auf kleineren Projekten deutlich besser abschneidet und das aufwändigere feinangepasste mehrsprachige DistilBERT-Modell nur auf den drei größten Projekten bessere Ergebnisse erzielt.

Die Ergebnisse auf dieser Form der Nachverfolgbarkeit zeigen, dass mit einer großen Menge an Trainingsdaten feinangepasste Sprachmodelle vielversprechende Ergebnisse erzielen können. Leider wurden die meisten dieser Arbeiten nur mit einfachen VSM-, LSI- oder LDA-basierten Verfahren verglichen und der einzige Vergleich mit den ansonsten erfolgreichen worteinbettungsbasierten Verfahren zeigt, dass diese insbesondere auf kleineren Projekten bessere Ergebnisse erzielen. Außerdem sind Belangbeschreibungen vom verwendeten Abstraktionsniveau und den gewählten Begrifflichkeiten deutlich näher an denjenigen des Quelltextes und den dazugehörigen Einbuchungen, als es andere Formen von Anforderungen sind. Dies liegt daran, dass diese zumeist gleichzeitig oder zumindest mit Kenntnis über die entwickelten Quelltextartefakte entstehen und somit direkt auf diese verwiesen werden kann. Dadurch ist eine Übertragbarkeit der Erkenntnisse dieser Form der Nachverfolgbarkeit nicht unbedingt auf das Überbrücken der semantischen Lücke zwischen bspw. Anwendungsfallbeschreibungen und Quelltext übertragbar. Eine sehr spannende Erkenntnis, die trotzdem aus diesen Arbeiten gezogen werden kann, ist, dass für mehrsprachige Projekte eine automatische Übersetzung ins Englische positive Ergebnisse sowohl für VSM- und LSI-basierte wie auch worteinbettungsbasierte Verfahren erzielt. Da es nicht selten vorkommt, dass eine nicht unerhebliche Menge an Fremdsprachenbegriffen in Softwareentwicklungsprojekten auftreten, die weitestgehend auf Englisch verfasst sind [LLC21], ist dies eine wertvolle Erkenntnis für die Anwendung in der Praxis

und dieser Effekt könnte sich auch auf die TLR von Anforderungen zu Quelltext übertragen lassen.

3.5. Andere Formen von Nachverfolgbarkeit

Neben den bisher besprochenen Formen von Nachverfolgbarkeit, welche sich mindestens einen Artefakttyp mit der TLR von Anforderungen zu Quelltext teilen, können auch Arbeiten mit ähnlichen Artefakttypen relevant sein. Als erstes solches Beispiel soll die TLR von natürlichsprachlicher Softwarearchitekturdokumentation (engl.: *software architecture documentation*) zu formalen Architekturmodellen betrachtet werden. Hierbei stellt die natürlichsprachliche Softwarearchitekturdokumentation einen zu den Anforderungen verwandten Artefakttyp dar. Da sich allerdings sowohl bei dem formalen Architekturmodell als auch bei der natürlichsprachlichen Softwarearchitekturdokumentation um Dokumentation der Architektur handelt, sind diese auf einem weitaus ähnlicheren Abstraktionsniveau verfasst, als es bei Anforderungen und Quelltext der Fall ist. So werden normalerweise die gleichen Begriffe und Benennungen der Elemente in beiden Artefakttypen verwendet und können damit einfacher aufeinander abgebildet werden. Dafür sind die Informationen, die im Modell zur Verfügung geringer als diejenigen, die aus dem Quelltext extrahiert werden können, da beispielsweise keine Kommentare oder Implementierungen zur Verfügung stehen. Somit ist zwar die semantische Lücke zwischen den Artefakten deutlich kleiner, wird dann allerdings doch ein Element unterschiedlich benannt, so kann, mit der geringeren Informationslage auf Modellseite, eine Wiederherstellung einer Nachverfolgbarkeitsverbindung erschwert sein.

Keim et al. stellen in ihrer Arbeit „*Trace Link Recovery for Software Architecture Documentation*“ [Kei+21] die Rahmenarchitektur SWATTR vor, welche TLs zwischen textueller informeller Softwarearchitekturdokumentation und formalen Architekturmodellen, wie dem Palladio Component Model (PCM), identifiziert. Hierzu sieht SWATTR drei Phasen vor. Zunächst wird der Text bzw. das Modell extrahiert. Der Text wird dabei einer Vorverarbeitung unterzogen in der Grundformen bestimmt, Wortarten erkannt und Abhängigkeitsgraphen erstellt werden. Auf Basis dieser Informationen wird der Text daraufhin mittels verschiedener Heuristiken nach Erwähnungen von Namen und/oder Typen durchsucht. So deuten Singular-Nomen im Text auf Namen oder Typen und Plural-Nomen auf Typen hin oder es werden auf Basis der Abhängigkeiten Subjekte und Objekte bestimmt und diese je nach Typ der Abhängigkeit als Typ oder Namen klassifiziert. Zusätzlich werden Terme im Text anhand ihrer Wortähnlichkeit gruppiert, sodass z. B. *database* und *datastore* als diesselbe Entität angesehen wird. In der Extraktion des Modells werden Modellelemente mit ihrem Namen, Typ und eindeutigen ID extrahiert. In der zweiten Phase von SWATTR, der Identifikation von Elementen, werden diejenigen Elemente aus den Erwähnungen ausgewählt, die im Architekturmodell auftauchen sollten. Hierbei werden aufeinanderfolgende Erwähnungen von Name und Typ bzw. Typ und Name gesucht und verbunden oder auf Basis von erkannten zusammengesetzten Termen identifiziert. In der dritten Phase werden die Elemente aus dem Text mit denen im Modell verbunden und

dadurch die eigentlichen TLs identifiziert. Hierbei werden verschiedene Ähnlichkeitsvergleiche eingesetzt, um eine Gesamtkonfidenz zu bestimmen, dass eine textuelles Element mit zugehörigem Typ zu einem Modellelement und seinem Typ passt. Übersteigt diese Konfidenz einen gewissen Wert wird diese Paarung als TL identifiziert.

In einer Evaluation auf den drei Fallstudien Mediastore, Teammates und TeaStore konnte SWATTR ein F_1 -Maß von bis zu 53 %, 78 % bzw. 73 % erreichen. In einem Vergleich mit nachimplementierten Versionen der TLR-Ansätze von Rodriguez und Carver [RC20] und Zhang et al. [Zha+21] (s. Abschnitte 3.1.1.2 und 3.1.3) erzielt SWATTR auf Teammates und TeaStore bessere Ergebnisse als beide Ansätze. Auf Mediastore erzielt der worteinbettungs-basierte Ansatz von Zhang et al. ein höheres F_1 -Maß von 62 %. Der Ansatz von Rodriguez und Carver schneidet mit einem durchschnittlichen F_1 -Maß von 13 % deutlich schlechter ab, als es für den Verlust an Informationen auf Modellseite gegenüber der in ihrem Papier berichteten Anwendung auf Quelltext zu erwarten gewesen wäre. Da ihr Verfahren auf Jaccard- und Kosinusähnlichkeit setzt, welche ohne Weiteres auch auf Modellseite angewandt werden können, wäre hier kein so deutlicher Einbruch der Ergebnisse zu erwarten gewesen.

Die folgende Arbeit wurde ausgewählt, da sie eine ähnliche Herangehensweise wie diese Arbeit nutzt, feingranulare Quelltextinformationen. Sie betrachtet allerdings die TLR von Beschreibungen von Dateneinschränkungen (engl.: *data constraints*) auf Methoden des Quelltextes. Diese Dateneinschränkungen beschreiben, welche Werte der Daten erlaubt und erwartet werden, wie z. B. „*The Federal Reserve limits withdrawals or outgoing transfers from a savings or money market account to no more than six such transactions per statement period*“. Sie stellen also eine spezielle Form von Anforderungen dar, die meist bereits eine gewisse Art der Umsetzung vorgibt und somit implementierungsnäher ist als allgemeine Anforderungen.

Florez et al. stellen in ihrer Arbeit „*Retrieving Data Constraint Implementations Using Fine-Grained Code Patterns*“ [Flo+22] für diese Aufgabe die Rahmenarchitektur Lasso vor. Lasso nutzt dabei eine Kombination von IR-basierter TLR und einem auf Einschränkungsmustern (engl.: *constraint implementation pattern*, CIP) basierendem Ansatz. Diese CIPs sind eine Menge von 13 häufig genutzten Mustern bei der Umsetzung von Dateneinschränkungen, wie beispielsweise binärer Vergleich, if-Ketten oder Null-Checks. Durch einen syntaxbasierten Ansatzes mit kontextfreie Grammatiken (engl.: *context-free grammars*) auf dem AST werden diese Muster im Quelltext identifiziert. Die TLR-Komponente nutzt entweder Apache Lucene [MHG10] mit BM25-Ähnlichkeit, Lucene mit VSM und TF-IDF oder LSI. Lasso benötigt als Eingabe den Typ, die Operanden und den Kontext einer jeden Einschränkung. Diese müssen manuell dem System übergeben werden. Insbesondere der Kontext, welcher den Absatz im Textdokument darstellt, in dem die Dateneinschränkung beschrieben wird, muss hierbei manuell identifiziert werden. Dieser Kontext wird ebenso von der TLR-Komponente als Eingabe genutzt. Da auch die Operanden und der Typ manuell aus dem Text extrahiert werden, handelt es sich hier also nicht um eine automatische, sondern eine händische Identifikation der relevanten Teile der Eingabe.

Zur Evaluation werden 136 Dateneinschränkungen aus sieben OSS-Java-Projekten herangezogen. Dabei wurden die Bedienungsanleitungen als textuelle Eingaben verwendet.

bug report 50151:

description: *changed the RCP Browser example to create its ToolBar using SWT.FLAT /SWT.RIGHT - the ToolBar has 4 ToolItems: Back (image only), Forward (image only), Stop (text only) and Refresh (text only) - when it comes up, the Back and Forward items have a fair bit of blank space after the arrow images (see attached screen shot). From scanning MSDN, it seems like this is platform behaviour on Windows, but it's almost certainly not what an app wants. I think that many RCP apps will want this style, with tool items having mixed icons and text, with no extra blank spaces (compare with IE).*

summary: *Using SWT.RIGHT on a ToolBar leaves blank space.*

Abbildung 3.1.: Beispiel eines Fehlerberichts aus dem Eclipse SWT-Projekt

Als Goldstandard wurde manuell für jede Dateneinschränkung genau ein Beispiel im Quelltext gesucht. Die besten Ergebnisse erzielt Lasso hierbei mit der BM25-Ähnlichkeit sowie VSMs, die nahezu dieselben Ergebnisse erreichen. Lasso liefert als Ausgabe pro Einschränkung eine geordnete Liste aller Methoden und der Rang in dieser Liste gibt die Wahrscheinlichkeit an, dass die Methode die Implementierung dieser Einschränkung darstellt. Der durchschnittliche Rang der erwarteten Methoden in diesen geordneten Listen lag bei ca. 13. In 39 % der Fälle befindet sich die korrekte Methode sogar auf Platz 1 der geordneten Liste.

Die Ergebnisse dieser Arbeit zeigen, dass mit manuellen Eingaben und syntaxbasierter Abbildung von Mustern die implementierenden Methoden bestimmter Beschreibungen in Anforderungen identifiziert werden können. Insbesondere der durchschnittliche Rang der Methoden von 13 zeigt, dass ein semi-automatischer Einsatz möglich wäre, aber auch, dass selbst mit diesem spezialisierten Ansatz die erzielte Güte nicht für eine Automatisierung ausreicht. Die Informationen, die aus einer solchen feingranularen Betrachtung gezogen werden können, ließen sich allerdings potenziell trotzdem für das allgemeine Wiederherstellen von TLs zwischen Anforderungen und Quelltext nutzen, sofern ein Weg gefunden würde, die manuelle Eingabe der Informationen zu den Dateneinschränkungen zu automatisieren.

3.6. Konzeptlokalisierung in Quelltext

Über das Forschungsfeld der Nachverfolgbarkeit von Softwareartefakten hinaus gibt es noch weitere relevante Forschungsgebiete, deren verwendete Techniken und Erkenntnisse auf die in dieser Arbeit behandelte Aufgabe der automatischen Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext übertragen werden können. Eines dieser Forschungsgebiete ist die Lokalisierung von Konzepten in Quelltext. Insbesondere die Aufgabe der Lokalisierung von Fehlern (engl.: *bug localization*) stellt hierbei ein sehr aktives Forschungsfeld dar, welches viele Parallelen zur Nachverfolgbarkeit aufweist. Die Aufgabe ist hierbei das Bestimmen der relevanten Quelltextabschnitte für eine gegebene textuelle Fehlerbeschreibung einer Software (wie z. B. in Abbildung 3.1). Diese Fehlerbeschreibungen bestehen zumeist aus einem Titel, einer

Zusammenfassung, verschiedenen Formen einer detaillierteren Beschreibung, wie z. B. Fehlermeldungen oder Ausgaben des Ausführungstapels (engl.: *stack traces*) sowie weiterer Felder, wie der Version oder Komponente [PMB19]. Damit sind diese Beschreibungen bezüglich der verwendeten Begrifflichkeiten und der beschriebenen Semantik deutlich näher an der tatsächlichen Implementierung der Software und enthalten meist bereits konkrete Verweise auf den Teil der Software, in dem der Fehler auftritt. Zusätzlich werden oftmals dieselben bzw. ähnliche Begriffe für die Komponenten und sichtbaren Teile der Software benutzt wie im Quelltext, da diese Informationen zum Verfassungszeitpunkt der Beschreibungen bereits vorhanden sind. Ist sogar die Ausgabe des Stapels vorhanden, so können sogar einzelne Klassen abgelesen werden. Somit unterscheidet sich die Aufgabe in der zu überbrückenden semantische Lücke und der Unterschiedlichkeit der verwendeten Begriffe von der TLR zwischen Anforderungen und Quelltext. Die Aufgabe, eine natürlichsprachliche Beschreibung einer Funktionalität auf ihre Entsprechung im Quelltext abzubilden, teilen sich die Aufgaben allerdings. Auch in dieser Aufgabe wurde lange Zeit auf IR-Verfahren gesetzt, welche analog zur Wiederherstellung von Nachverfolgbarkeitsverbindungen in den letzten Jahren vermehrt entweder mit Worteinbettungen eingesetzt oder durch maschinelles Lernen mit Sprachmodellen ersetzt wurden [AK20; LHL22; LWC22].

Akbar und Kak vergleichen in ihrer Arbeit „*A Large-Scale Comparative Evaluation of IR-Based Tools for Bug Localization*“ [AK20] acht IR-basierte Verfahren zur Fehlerlokalisierung auf dem ebenfalls in dieser Arbeit neu eingeführten großen Vergleichsdatensatz Bugzbook. Bugzbook umfasst 29 Projekte mit insgesamt über 20000 Fehlerberichten. In diesem Vergleich zeigt sich, dass ein TF-IDF-basiertes Verfahren mit einem MAP von 34,6 % signifikant besser abschneidet als ein LDA-basiertes Verfahren [SAK17]. Das Markov-Zufallsfeld-basierte Verfahren [SAK17], welches die Wahrscheinlichkeitsverteilung für nacheinander auftretende Terme nutzt, ist wiederum mit einem durchschnittlichen MAP von 37 % signifikant besser sowohl als das TF-IDF-Verfahren als auch die Vergleichsverfahren BugLocator [ZZL12], BLU_iR [Sah+13] und PWSM [AK19]. Lediglich das SCOR-Verfahren [AK19], welches die Term-zu-Term-Abhängigkeiten des Markov-Zufallsfeld-basierten Verfahrens mit *word2vec*-Einbettungen kombiniert, erzielt mit einem durchschnittlichen MAP von 39,8 %, nochmals signifikant bessere Ergebnisse. Die Einbettungen werden dabei auf einem Teil der Projekte des Datensatzes mittels des CSG-Verfahrens von *word2vec* trainiert. Die Autoren zeigen außerdem, dass SCOR mit *word2vec*-, *fastText*- und *GloVe*-Einbettungen nahezu dieselben Ergebnisse erzielt. Ein Vergrößern der Dimension der Einbettungen von 500 auf 1000 bzw. 1500 erzielt nur vernachlässigbare Verbesserungen. Ein zusätzliches Verwenden von TF-IDF in Kombination mit SCOR erhöht den durchschnittlichen MAP nochmals auf 40,9 %.

Zhang et al. stellen in ihrer Arbeit „*Exploiting Code Knowledge Graph for Bug Localization via Bi-directional Attention*“ [Zha+20] ein auf maschinellem Lernen basierendes Verfahren für Fehlerlokalisierung vor, welches insbesondere die Struktur des Quelltextes miteinbezieht. Ihr Verfahren KGBugLocator nutzt Wissensgraph-Einbettungen auf einem Quelltextwissensgraph. Dieser Quelltextwissensgraph wird aus dem AST des Quelltextes erstellt, indem Klassen, Methoden, Parameter, Variablen und Attribute als Knoten und die Beziehungen zwischen diesen als Kanten modelliert werden. Zusätzlich wird die Fehlerbe-

schreibung durch ein LSTM und der Quelltext durch ein faltendes neuronales Netz (engl.: *convolutional neural network*, CNN) enkodiert, bei denen jeweils *word2vec*-Einbettungen als Eingabe genutzt werden. Anschließend wird ein mittels Schlüsselwörter überwachter bidirektionaler Aufmerksamkeitsmechanismus verwendet, um bestimmte Merkmale der Eingaben auszuwählen. Hierbei werden Eingabezeilen, welche Token enthalten, die sowohl in der Fehlerbeschreibung als auch im Quelltext auftreten, als Schlüsselzeilen markiert. Letztlich wird ein vollvermaschtes FFNN auf Basis der derart enkodierten Eingaben und der Wissensgraph-Einbettung genutzt, um eine Ähnlichkeitbewertung der Eingaben zu erhalten.

Die Autoren evaluieren ihr Verfahren auf den vier OSS-Java-Projekten AspectJ, SWT, JDT und Tomcat, indem sie eine zehnfache Kreuzvalidierung durchführen. Ihre Ergebnisse zeigen, dass im Vergleich zu dem IR-basierten Verfahren BugLocator [ZZL12] und den vier auf maschinellem Lernen basierenden Verfahren DNNLOC [Lam+17], DeepLocator [Xia+17], NP-CNN [HLZ16] und CAST [Lia+19] das Verfahren KGBugLocator sowohl in Genauigkeit, MAP als auch mittlerem Kehrwert (engl.: *mean reciprocal rank*, MRR) auf allen vier Projekten die besten Ergebnisse erzielt. Mit einem durchschnittlichen MAP von 45,3 % erzielt KGBugLocator eine Verbesserung von 12 Prozentpunkten gegenüber dem IR-Verfahren BugLocator und 6 Prozentpunkten gegenüber dem besten auf maschinellem Lernen basierenden Verfahren CAST. Wichtig ist aber auch hier für die Einschätzung der Ergebnisse, wie auch schon bei der TLR, dass die auf maschinellem Lernen basierenden Verfahren einen vorhandenen Goldstandard an Verbindungen benötigen (s. Abschnitt 3.1.3) und in dieser Evaluation sogar in jedem Durchlauf der Kreuzvalidierung 90 % der gesamten Daten als Training benutzt werden.

Liang et al. nutzen in ihrer Arbeit „*Modeling Function-Level Interactions for File-Level Bug Localization*“ [LHL22] semantische Merkmale auf Methodenebene für die Fehlerlokalisierung mittels eines Sprachmodells. Ihre Rahmenarchitektur FLIM nutzt hierbei als vortrainiertes Sprachmodell *CodeBERT* [Fen+20]. Dieses passen sie auf einem Trainingsdatensatz auf die Aufgabe der Fehlerlokalisierung an. Dieser besteht aus Tupeln von Fehlerberichts-zusammenfassung bzw. Fehlerberichtbeschreibung und fehlerhafter Methode, welche sie aus sechs OSS-Java-Projekten sowie dem CodeSearchNet-Datensatz [Hus+20] beziehen. Die Ergebnisse eines Fehlerbericht-Methodentupels des feinangepassten *CodeBERT*-Modells werden daraufhin mittels Kosinusähnlichkeit verglichen. Diese Ähnlichkeit wird dann zusammen mit 19 weiteren Merkmalen kombiniert und einem Ranglernen-Modell übergeben, dessen Ausgabe dann als Relevanz der beiden Eingabeartefakte angesehen wird. Diese Merkmale umfassen unter anderem lexikalische Ähnlichkeiten der Oberflächenformen, Klassen-, Methoden- oder Variablennamen innerhalb eines Artefakts, lexikalische Ähnlichkeiten der Fehlerberichts-zusammenfassung bzw. Fehlerberichtbeschreibung mit den Elementen des Quelltextes und projektspezifische Merkmale wie Fehlerhäufigkeit oder Zeit seit dem letzten berichteten Fehler.

In einer Evaluation mittels Kreuzvalidierung auf den sechs OSS-Java-Projekten AspectJ, SWT, JDT, Tomcat, EclipseUI und BIRT erzielt FLIM auf allen Projekten die besten Ergebnisse in MAP und MRR im Vergleich mit dem IR-Verfahren BugLocator [ZZL12], den Ranglernen-basierten Verfahren YBL [YBL16] und AdaptiveBL [Fej+22] und den auf maschinellem Lernen basierenden Verfahren LS-CNN [HL17], TRANP-CNN [Huo+21] und

DreamLoc [Qi+22]. Mit einem durchschnittlichen MAP von 41,2 % erzielt FLIM eine Verbesserung von zwei Prozentpunkten gegenüber dem besten Vergleichsverfahren AdaptiveBL, welches nur auf die 19 Merkmale ohne das Sprachmodell setzt. Um die Anwendung in einem realistischeren Szenario zu evaluieren, untersuchen die Autoren außerdem wie ihr Verfahren sich in einer Anwendung verhält, wenn das Sprachmodell nur auf Daten der nicht gerade evaluierten Projekte feinangepasst wurde. Hierbei erzielt FLIM einen durchschnittlichen MAP von 40,2 %.

Luo et al. nutzen in ihrer Arbeit „*Improving Bug Localization with Effective Contrastive Learning Representation*“ [LWC22] gegensätzliches maschinelles Lernen (engl.: *contrastive learning*) in Kombination mit einem vortrainierten Sprachmodell. Ihr Verfahren CoLoc umfasst zwei Vortrainingsphasen und eine Feinanpassung an die eigentliche Aufgabe. In der ersten Phase wird das *CodeBERT*-Modell [Fen+20] auf einem Korpus von Fehlerberichten aus den Projekten Mozilla, Eclipse, Netbeans und GCC weitertrainiert. Daraufhin erfolgt das gegensätzliche Vortraining auf den fünf Fehlerlokalisierungsdatensätzen AspectJ, SWT, JDT, Tomcat und EclipseUI. Hierbei wird die Zuordnung eines Fehlerberichts zu seinen fehlerhaften Dateien als positive Beispiele verwendet und alle anderen Zuordnungen desselben Fehlerberichts gelten als negative Beispiele im selben Stapel. Als Lernziel wird dabei die Darstellung der Eingaben aus dem weitertrainierten *CodeBERT*-Modell so optimiert, dass sich positive Beispiele möglichst ähnlich und negative Beispiele gleichzeitig möglichst weit entfernt sind. In der dritten Phase wird CoLoc dann auf die Fehlerlokalisierung feinangepasst, indem eine siamesische Architektur mit geteilten Parametern verwendet wird (vgl. Lin et al. [Lin+21] in Abschnitt 3.4). Es wird also dasselbe CoLoc-Modell für die Repräsentation der Fehlerberichte und der Quelltextdateien benutzt. Auf Basis einer Trainingsmenge und der Kosinusähnlichkeit als Bewertungsfunktion wird dann die Ausgabeschicht angepasst.

In einer Evaluation auf den fünf OSS-Java-Projekten AspectJ, SWT, JDT, Tomcat und EclipseUI erzielt CoLoc bessere Ergebnisse in Genauigkeit, MAP und MRR als die sechs Vergleichsarbeiten BugLocator [ZZL12], DNNLOC [Lam+17], DeepLocator [Xia+17], NP-CNN [HLZ16], CAST [Lia+19] und DeepLoc [Xia+19]. Mit einem durchschnittlichen MAP von 51,1 % erzielt CoLoc ein um 6 Prozentpunkte besseres Ergebnis als das zweitbeste Verfahren DeepLoc. In einem zweiten Experiment vergleichen die Autoren CoLocs Leistung mit der von nicht angepassten *BERT*-, *RoBERTa*- und *CodeBERT*-Modellen. Hierbei erzielt CoLoc auf allen fünf Projekten die besten Ergebnisse mit einer Verbesserung von 3 Prozentpunkten gegenüber dem zweitbesten Modell, dem nicht angepassten *CodeBERT*. Dies zeigt, dass ein Weitertrainieren eines vortrainierten Sprachmodells und das Nutzen des gegensätzlichen Lernens eine Verbesserung der Leistung auf der Aufgabe der Fehlerlokalisierung erzielen kann. Hierzu müssen aber entsprechende Musterlösungsdaten vorhanden sein. Übertragbarkeit auf neue Projekte wurde leider nicht evaluiert.

Die Ergebnisse dieser Arbeiten zur Fehlerlokalisierung zeigen, dass auch hier die besten Ergebnisse mit überwachtem maschinellen Lernen auf Basis von Sprachmodellen erzielt werden. Da diese Verfahren allerdings wie auch in der Anwendung für die TLR vorhandene Musterlösungszuordnungen benötigen, können diese nicht ohne Weiteres auf neue Projekte angewandt werden. Da außerdem die meisten Arbeiten nicht evaluieren, wie sich die Leistung ihrer Verfahren verändern, wenn sie keine Trainingsdaten

Natürlichsprachliche Anfrage:

Parse a memory string in the format supported by Java (e.g. 1g, 200m) and return the value in MiB

Quelltextausschnitt:

```
def _parse_memory(s):
    units = { 'g': 1024, 'm': 1, 't': 1 << 20, 'k': 1.0 / 1024}
    if s[-1].lower() not in units:
        raise ValueError("invalid_format:_" + s)
    return int(float(s[:-1]) * units[s[-1].lower()])
```

Abbildung 3.2.: Beispiel eines Paares aus natürlichsprachlicher Dokumentation und Quelltext aus dem CodeSearchNet-Datensatz [Hus+20; Fen+20]

aus dem evaluierten Projekt gesehen haben, lässt sich die Leistung in einem realistischeren Einsatzszenario in der Praxis, wo das Verfahren auf ein neues Projekt angewandt wird, nicht beurteilen. Lediglich Liang et al. [LHL22] bemessen diese Leistung und erzielen hierbei vielversprechende Ergebnisse. Ein Großteil dieses Erfolgs ist dabei dem verwendeten CodeSearchNet-Datensatz (auf dem auch *CodeBERT* vortrainiert ist) zuzuschreiben, der bereits ohne Feinanpassung eines Modells vielversprechende Ergebnisse bei der Fehlerlokalisierung aufweist. Dies deutet darauf hin, dass die Trainingsbeispiele von Methodendokumentation und Methodenrumpf bereits eine große Ähnlichkeit zu den Fehlerberichten und Quelltextartefakten aufweisen. Ob diese Leistung auch auf die Artefakte der TLR zwischen Anforderungen und Quelltext, welche eine größere semantische Lücke aufweisen, übertragbar ist, ist hier ungewiss. Betrachtet man die Ergebnisse der Verfahren ohne überwachtes Lernen, welche ohne Weiteres auf ungesehene Projekte angewandt werden können, so zeigt sich, dass hier Worteinbettungen die besten Ergebnisse erzielen. Diese Erkenntnis unterstützt also die in dieser Arbeit zu untersuchende Hypothese, dass worteinbettungsbasierte TLR eine höhere Leistung als bestehende unüberwachte Verfahren bietet.

3.7. Quelltextsuche

Eine weiteres verwandtes Forschungsfeld zur TLR zwischen Anforderungen und Quelltext stellt die semantische Quelltextsuche (engl.: *semantic code search*) dar. Bei dieser Aufgabe geht es um die automatische Identifikation relevanter Quelltextteile zu einer natürlichsprachlichen Anfrage. Es handelt sich also hierbei ebenfalls um eine Aufgabe aus der Informationsrückgewinnung, die als Eingabe natürlichsprachlichen Text und als Ziel der Abfrage eine Menge von Quelltextdokumenten umfasst. Man könnte auch soweit gehen die TLR zwischen Anforderungen und Quelltextdokumenten als eine Art Quelltextsuche zu bezeichnen. Oftmals wird dieser Begriff aber spezifischer für die Aufgabe verwendet, einer Beschreibung einer konkreten Funktionalität, wie z. B. *Read a text file*, mögliche konkrete Umsetzungen in einer Programmiersprache zuzuweisen. Somit spielt auch, wie

in der TLR, in dieser Aufgabe das Überbrücken der semantische Lücke zwischen natürlich-sprachlichem Text und Quelltext eine wichtige Rolle. Allerdings ist die zu überbrückende Lücke meist weniger groß. Die Anfragen beziehen sich in dieser Aufgabe auf Umsetzungen einer Funktionalität einer Methode, wie z. B. *output to html file*, und nicht, wie es bei Anforderungen der Fall sein kann, ganzer Komponenten oder Module. Die Anfragen umfassen also kleine enger umrissene Funktionalitäten. Außerdem gilt auch hier, wie schon für die Fehlerlokalisierung, dass die verwendeten Begriffe und Beschreibungen deutlich näher an der Implementierung sind und ggf. sogar direkt die behandelten Objekte und Klassen benennen. Dies kann die Abbildung deutlich erleichtern.

Der CodeSearchNet-Datensatz [Hus+20] wurde als Vergleichsdatensatz für die Evaluation von Ansätzen zur Quelltextsuche entwickelt. Er umfasst 2.3M Paare von natürlicher Sprache, in Form von Methodendokumentation, und zugehörigem Methodenquelltext sowie 6.4M Quelltextauschnitten von Methoden ohne Dokumentation. Ein Beispiel für ein solches Paar ist in Abbildung 3.2 dargestellt. Diese wurden aus OSS-Projekten auf der Plattform GitHub gesammelt. Die Projekte umfassen die Programmiersprachen Python, Java, JavaScript, PHP, Go und Ruby und wurden hinsichtlich ihrer tatsächlichen Verwendung gefiltert. Da die derart ermittelten Paare nicht notwendigerweise eine Quelltextsuchaufgabe darstellen, haben Husain et al. die CodeSearchNet-Challenge präsentiert, welche aus 99 häufigen Bing-Anfragen bezüglich Implementierungen besteht und Musterlösungen zu relevanten Quelltextabschnitte im CodeSearchNet-Datensatz umfasst. Hierbei wurden Expert:innen mögliche Kandidaten für Quelltextabschnitte vorgelegt, welche diese dann nach ihrer Relevanz für die Anfrage bemessen sollten. Die Kandidaten wurden durch ein Ensemble von ANN-basierten Ansätzen und ElasticSearch³ generiert, weshalb nicht der gesamte Datensatz, sondern nur ein Teil annotiert wurde. Die meisten Zuordnungen erhielten dabei Quelltextabschnitte aus Python und Java.

Husain et al. evaluieren vier Ansätze auf zwei Aufgaben. Es wird auf dem gesamten Datensatz ein gemeinsames Einbettungsmodell trainiert, welches eine Sequenz von Eingabetoken durch einen einzelnen Vektor repräsentiert. Das Ziel ist es hierbei, dass die Vektoren der natürlichsprachlichen Dokumentation im Vektorraum nahe zu dem beschriebenen Quelltextausschnitt sind. Als Varianten für das Bestimmen dieser Einbettungen nutzen die Autoren entweder neuronale BoWs, bidirektionale RNNs, 1D-CNNs oder *self-attention*-Modells (s. Abschnitte 2.5 und 2.9.3). Die erste Aufgabe bemisst den MRR für die Abbildung einer Methodendokumentation auf die entsprechende Methode für 999 Beispiele. Hierbei schneidet das *self-attention*-Modell mit einem durchschnittlichen MRR von 0,70 am besten ab. Auf der eigentlichen CodeSearchNet-Challenge erzielt das neuronale BoW-Modell mit einem durchschnittlichen MRR von 0,57 bzw. 0,34 die besten Ergebnisse. Die beiden Werte ergeben sich, wenn entweder nur innerhalb der annotierten Beispiele bzw. im gesamten Korpus gesucht wird. Auf Java, PHP und Python hingegen erzielt eine Kombination aus *self-attention*-Modell für die natürlichsprachliche Seite und neuronalem BoW für die Quelltextseite das beste Ergebnis.

³ <https://www.elastic.co/de/elasticsearch/>, zuletzt besucht am 26.04.2023.

Feng et al. stellen das bimodale Sprachmodell *CodeBERT* [Fen+20] vor und evaluieren dieses ebenso auf dem CodeSearchNet-Korpus. *CodeBERT* nutzt als Modellarchitektur dieselbe wie das Transformer-basierte Sprachmodell *RoBERTa* [Liu+19] mit einer absoluten Anzahl an Modellparametern von 125M. Als Eingabe erhält das Sprachmodell ein natürlichsprachliches Eingabesegment sowie ein Quelltexteingabesegment, welche durch ein spezielles Separatortoken getrennt werden. Die Autoren trainieren das Sprachmodell auf dem CodeSearchNet-Korpus mit zwei Trainingszielen. Das erste Trainingsziel ist das maskeierte Sprachmodellieren, welches auch *BERT* und *RoBERTa* verwenden (s. Abschnitte 2.5 und 2.9.3), auf den bimodalen Datenpunkten des Korpus. Das zweite Trainingsziel stellt die Detektion ersetzter Token dar. Hierbei werden mittels zweier N-Gramm-Sprachmodelle zufällig einzelne Token der Eingabe ersetzt und die Aufgabe des Modells ist es, zu identifizieren, ob ein Token ursprünglich in der Eingabe war oder nicht. Hierbei wird das Modell sowohl auf den bimodalen wie auch den unimodalen Daten trainiert.

Für die Evaluation auf dem CodeSearchNet-Datensatz passen Feng et al. das mehrsprachige *CodeBERT*-Modell für die jeweilige Programmiersprache an. Sie nutzen dabei die Ausgabe des [CLS]-Tokens als Eingabeeinbettung und geben diese in einen binären Klassifikator (*softmax*-Schicht) für die Aufgabe der Identifikation der korrekten Methode. Mit diesem Vorgehen vergleichen sie die Leistung *CodeBERTs* mit der eines neuronalen BoWs, CNNs, bidirektionalem RNNs, *self-attention*-Modells und *RoBERTa* auf den 999 Beispielen, die auch Husain et al. [Hus+20] in ihrer Evaluation nutzen. Hierbei erzielt *CodeBERT* mit einem durchschnittlichen MRR von 0,76 über alle Programmiersprachen hinweg die besten Ergebnisse.

Guo et al. ergänzen in ihrer Arbeit „*GraphCodeBERT*“ [Guo+22b] die Idee von *CodeBERT*, indem der Quelltext nicht mehr nur als Sequenz von Token aufgefasst wird, sondern auch seine Datenflüsse miteinbezieht. Zusätzlich zu den beiden Eingabesegmenten *CodeBERT*'s, wird nun ein weiteres Eingabesegment an die Eingabe angehängt, welches wiederum durch ein Separatortoken abgetrennt wird. Dieses Eingabesegment repräsentiert die Kanten des Datenflussgraphen des Quelltextausschnittes. Zusätzlich zum weiterhin verwendeten maskeierten Sprachmodellieren werden jetzt zwei neue Trainingsziele betrachtet. Zum einen müssen nun maskierte Datenflusskanten vorhergesagt werden. Hierbei werden 20 % der Knoten des Datenflussgraphen ausgewählt und es müssen alle direkten Kanten zwischen diesen Knoten vorhergesagt werden. Zum anderen wird die Aufgabe der Knotenzuordnung trainiert. Hierbei müssen für jeweils wiederum 20 % der Datenflussknoten ihre Entsprechungen im Quelltextsegment identifiziert werden.

Auch *GraphCodeBERT* wird auf dem CodeSearchNet-Datensatz vortrainiert und für die Aufgabe der Quelltextsuche evaluiert. Allerdings filtern Guo et al. für ihre Evaluation Anfragen mit schlechter Qualität anhand von manuellen Regeln und führen das Experiment anstatt nur auf 999 Beispielen auf dem gesamten Korpus aus. Im Vergleich mit denselben Verfahren wie in der Arbeit zu *CodeBERT* erzielt *GraphCodeBERT* eine signifikante Verbesserung mit einem durchschnittlichen MRR von 0,71. *CodeBERT* erzielt in dieser Aufgabe mit einem MRR von 0,69 das zweitbeste Ergebnis.

Wang et al. trainieren ihr Modell *SynCoBERT* [Wan+21a] auf zwei weiteren neuen Trainingszielen und nutzen kontrastives Lernen. Anstatt des Datenflusseingabesegments von *GraphCodeBERT* nutzen sie den sequenzialisierten AST als drittes Eingabesegment. Zu-

sätzlich zum maskierten Sprachmodellieren nutzen die Autoren daraufhin die beiden Aufgaben Bezeichneridentifikation und AST-Kantenvorhersage während des Vortrainings. Bei der Bezeichneridentifikation muss das Modell vorhersagen, ob ein Token des Quelltextsegments einen Bezeichner darstellt oder nicht. In der AST-Kantenvorhersage müssen einzelne maskierte Kanten des AST-Eingabesegments vorhergesagt werden.

Mit diesen Trainingszielen wird auch *SynCoBERT* auf dem CodeSearchNet-Datensatz trainiert und evaluiert. Bei der Evaluation der Quelltextsuche wird dasselbe Vorgehen wie in der Arbeit zu *GraphCodeBERT* verwendet. Hierbei konnten Wang et al. zeigen, dass *SynCoBERT* mit einem durchschnittlichen MRR von 0,74 nochmals eine Verbesserung gegenüber *GraphCodeBERT* erzielen konnte.

Auch Guo et al. nutzen für ihr bimodales Sprachmodell „*UniXcoder*“ [Guo+22a] den sequenzialisierten AST als drittes Eingabesegment. Anstatt der beiden neuen Trainingsziele von *SynCoBERT* nutzt *UniXcoder* allerdings die drei Aufgaben unidirektionales Sprachmodellieren (engl.: *unidirectional language modelling*), Entrauschen (engl.: *denoising objective*) und Repräsentationslernen eines Quelltextausschnitts (engl.: *code fragment representation learning*). Ersteres dient vor allem dem Training des Dekodierers, indem die jeweils nächsten Token vorhergesagt werden müssen. Beim Entrauschen werden zufällige Bereiche von Token unterschiedlicher Länge maskiert, welche daraufhin generiert werden müssen. Beim Repräsentationslernen ist das Ziel, eine semantische Einbettung eines Quelltextausschnitts zu erlernen. Hierbei wird *UniXcoder* dazu genutzt, einen AST mittels einer Bündelungsschicht auf einen einzelnen Eingabevektor abzubilden. Mittels dieses Eingabevektors soll das Modell daraufhin den eigentlichen Methodenkommentar generieren. Um einen Einsatz *UniXcoders* auch ohne AST-Sequenzierung zu ermöglichen, lernt diese Aufgabe implizit das Wissen des AST, indem zufällig 50 % der Nichtterminalsymbole des AST im Vortraining weggelassen werden. Während der Feinanpassung benötigt das Modell dann nur noch die Blätter des AST als Eingabe.

Auch *UniXcoder* wird wie auch schon die anderen Arbeiten zur Quelltextsuche auf dem CodeSearchNet-Datensatz vortrainiert. Auf diesem erzielt *UniXcoder* mit einem MRR von 0,744 nochmals eine Verbesserung gegenüber dem bisherigen besten Verfahren *SynCoBERT* [Wan+21a]. Auch auf den Aufgaben der Quelltextklonererkennung, Quelltextvollständigkeit und der Suche semantisch ähnlicher Quelltextabschnitte zeigt *UniXcoder* bessere Ergebnisse als bestehende Verfahren.

Der Stand der Technik in der Quelltextsuche zeigt, wie auch schon bei der Fehlerlokalisierung (s. Abschnitt 3.6), dass auf großen Korpora vortrainierte und auf die spezielle Aufgabe feinangepasste bimodale Sprachmodelle die besten Ergebnisse erzielen. Da es sich bei der Quelltextsuche um eine der TLR zwischen Anforderungen und Quelltext sehr ähnliche Aufgabe handelt, könnten diese Modelle auch für diese Arbeit relevant sein. Allerdings wurde die Leistung dieser Modelle bisher kaum in dem für die Praxis realistischeren Szenario einer Anwendung auf im Training ungesene Projekte bemessen. Zusätzlich ist unklar, welche Leistung die Modelle ohne eine vorherige Feinanpassung aufweisen, für die es zunächst Musterlösungszuweisungen benötigt. Da in dieser Arbeit die unüberwachte TLR untersucht wird, müsste also untersucht werden, wie sich diese Modelle ohne diese Feinanpassung verhalten. Zumindest für das *UniXcoder*-Modell zeigte

eine Untersuchung auf der Aufgabe der Suche semantisch ähnlicher Quelltextabschnitte, dass auch ohne Feinanpassung vielversprechende Ergebnisse erzielt werden konnten.

Teil II.

Automatische Nachverfolgbarkeit von Anforderungen und Quelltext

4. Lösungsorientierte Analyse der Wiederherstellung von Nachverfolgbarkeit zwischen Anforderungen und Quelltext

A major challenge in the recovery of traceability links between software artifacts is the fact that these artifacts are in different formats and at different abstraction levels.

(De Lucia et al. [De +12])

Trotz des großen Nutzens, den Nachverfolgbarkeitsinformationen für die Entwicklung und Wartung von Softwareprojekten bieten und den Forschungsanstrengungen auf dem Gebiet der Automatisierung seit den 1990er Jahren fehlen TLs gänzlich in den meisten Projekten oder sind unvollständig und fehlerbehaftet [Got+12a]. Insbesondere bei der Abbildung von Anforderungen auf Quelltext erzielen bisherige Verfahren nicht die nötige Güte, um eine Anwendung in der Praxis zu ermöglichen [Ant+17]. Gerade diese Information ermöglicht aber erst, das volle Potenzial von Analysen wie einer weitreichenden Software-Wiederverwendbarkeitsanalyse oder einer Validierung von Anforderungen auszuschöpfen. In Situationen, wo bereits eine Menge von TLs vorhanden ist, erzielen ML-basierte Verfahren bereits gute Ergebnisse (s. Abschnitt 3.1.3). Doch in den meisten Fällen hindert gerade die zeitaufwendige und kostspielige Erstellung von Nachverfolgbarkeitsinformationen, das Schaffen von Nachverfolgbarkeit. Deshalb bleibt die Automatisierung der unüberwachten Erstellung von Nachverfolgbarkeitsinformationen zwischen Anforderungen und Quelltext eine der entscheidenden Herausforderungen, um Nachverfolgbarkeitsinformationen weitreichend einzuführen [Got+12a]. Um sich diesem Ziel anzunähern, möchte ich in diesem Kapitel die Hintergedanken und Ansatzpunkte verdeutlichen, welche zu dem im folgenden Kapitel 5 vorgestellten Verfahren zur unüberwachten TLR geführt haben. Hierzu führe ich eine lösungsorientierte Analyse des Problems sowie bisheriger und möglicher Lösungsansätze durch und leite daraus mögliche Herangehensweisen ab.

Bisherige Verfahren zur automatisierten Generierung von TLs zwischen Anforderungen und Quelltext fassen beide Artefakttypen als textuelle Dokumente auf. Dies liegt vor allem daran, dass eine Interpretation von Quelltext, die sich mit der Semantik der in natürlicher Sprache vorliegenden Anforderungen vergleichen lässt, bisher nur über die natürlich-sprachlichen Elemente geglückt ist. Auch wenn maschinelles Lernen große Fortschritte in der Interpretation von Quelltext erzielt, sind diese Ergebnisse noch fern von einer allgemeingültigen Extraktion der Semantik. Es existieren Verfahren, die vielversprechende

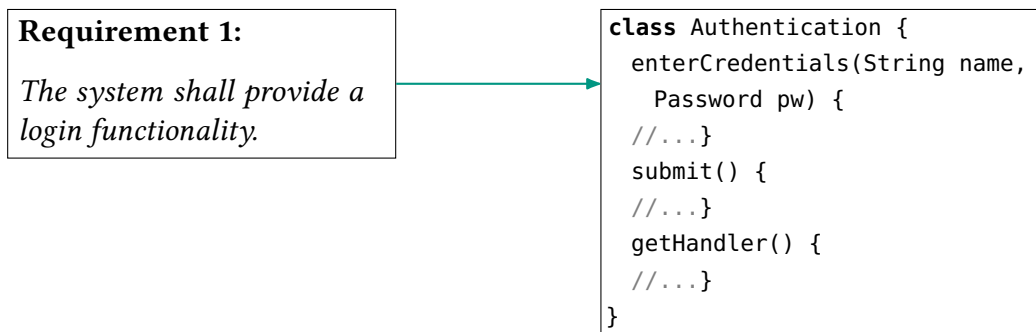


Abbildung 4.1.: Beispiel einer Nachverfolgbarkeitsverbindung, deren Zusammenhang nur auf Basis der Semantik erkennbar ist

Ergebnisse im semantischen Vergleich von Quelltextelementen erzielen, wie z. B. in der TLR zwischen Quelltext und Testfällen (s. Abschnitt 3.3) oder der Konzeptlokalisierung (s. Abschnitt 3.6). Diese eignen sich aber nur begrenzt für einen Vergleich mit den auf einem anderen Abstraktionsniveau verfassten Anforderungen. Ein Ableiten der Semantik einzelner Instruktionen und ein Interpretieren von zusammengesetzten Programmkonstrukten, wie Schleifen und Bedingungen, führt zumeist zu sehr implementierungsnahen Beschreibungen. Das Abstrahieren auf die dahinterliegenden Konzepte ist schwierig oder wird wiederum auf natürlichsprachlichen Text abgebildet, wie es bei automatisierter Quelltextzusammenfassung der Fall ist [Ahm+20; Haq+20; Wan+20; LBM21; Shi+22]. Deshalb fokussiere ich mich (wie auch alle bisherigen Ansätze) im Rahmen dieser Arbeit auf eine Interpretation der Teile des Quelltextes als natürlichsprachliche Elemente, um diese in Bezug zu den natürlichsprachlichen Anforderungen zu bringen. Dies bedeutet aber auch, dass sich der Lösungsansatz dieser Arbeit auf folgende Annahme stützt:

Annahme 1: Die Interpretation und Repräsentation der Artefakte auf Basis ihrer natürlichsprachlichen Elemente ermöglicht eine Wiederherstellung von Nachverfolgbarkeitsverbindungen.

4.1. Nachverfolgbarkeit als Problem der Informationsrückgewinnung

Betrachtet man also beide Artefakttypen, Anforderungen und Quelltext, als natürlichsprachliche Dokumente, so lässt sich das Generieren von Nachverfolgbarkeitsinformation als ein Problem der Informationsrückgewinnung (s. Abschnitt 2.4) auffassen. In der Literatur wird hierbei das Rückgewinnen des Zusammenhangs zwischen den Artefakten auf einen Ähnlichkeitsvergleich der Dokumente zurückgeführt. Es werden für jedes Quellartefakt diejenigen Artefakte aus der Menge aller Zielartefakte bestimmt, die diesem am ähnlichsten sind, und diese als Kandidaten für TLs angenommen. Dieses Vorgehen basiert

auf der Annahme, dass eine Ähnlichkeit zwischen einer Anforderung und einem Teil des Quelltextes ein guter Schätzer für das Bestehen einer Nachverfolgbarkeitsbeziehung zwischen den Artefakten ist. Der entscheidende Faktor ist folglich die Art und Güte dieser Ähnlichkeit und wie diese bestimmt oder angenähert werden kann. Ansätze, die rein textuelle oder syntaktische Vergleiche der Dokumente durchführen, erzielen nicht die nötige Güte für eine Automatisierung (s. Abschnitt 3.1.1).

Textuelle und syntaktische Ähnlichkeit bildet Zusammenhänge zwischen unterschiedlich gearteten, zu unterschiedlichen Zeitpunkten und ggf. von unterschiedlichen Personen erstellten Artefakten nicht ausreichend ab. Anforderungen und Quelltext enthalten Formulierungen auf unterschiedlichen Abstraktionsniveaus mit ggf. unterschiedlichen Begriffen und Ausdrücken. Dies stellt eine der größten Herausforderungen für die Wiederherstellung von TLs dar [De +12]. Erst eine Betrachtung der Semantik lässt einen Schluss über ihren Zusammenhang zu. Ein Beispiel für einen solchen Fall ist in Abbildung 4.1 dargestellt. Die Anforderung beschreibt das Erfordernis einer *login*-Funktionalität, wohingegen im Quelltext nur von einer Klasse *Authentication* die Rede ist, welche sich aus einer Eingabe der Anmeldeinformationen (`enterCredentials`) und einem Absenden (`submit`) zusammensetzt. Auf rein syntaktischer Ebene lässt sich der Zusammenhang zwischen den beiden Artefakten nicht identifizieren, denn es werden weder dieselben Begriffe noch ähnliche syntaktische Strukturen verwendet. Ohne einen semantischen Ähnlichkeitsvergleich lässt sich diese semantische Lücke zwischen den Artefakten nicht überbrücken und somit die Nachverfolgbarkeitsinformation nicht zurückgewinnen.

4.2. Semantische Lücke

In der Literatur wird die Differenz in Abstraktionsniveau, Formulierungen und genutzten Begriffen zwischen Artefakttypen als semantische Lücke (engl.: *semantic gap*) bezeichnet [Cle+07a; GCC17; Lin+21]. Diese begründet sich zum Teil auf den unterschiedlichen Entstehungszeitpunkten und ggf. unterschiedlichen Personen, die die Artefakte anfertigen. Zum größeren Teil resultiert diese Lücke allerdings aus der entlang des Softwareentwicklungsprozesses vollzogenen Verfeinerung und dem damit notwendigen höheren Detailgrad der Beschreibungen. Gerade im Fall des Zusammenhangs zwischen Anforderungen und ihrer Umsetzung im Quelltext ist diese Lücke besonders präsent. Denn die Umsetzung einer grobgranularen Anforderung erfordert im Quelltext meist das Zusammenspiel mehr als eines Artefakts (Klasse). Zusätzlich führt die Abbildung der Funktionalität auf verschiedene Klassen und insbesondere mehrere Methoden zu einer Verteilung der zur Interpretation nötigen Informationen. Durch objektorientierte Programmierrichtlinien wird dieser Effekt noch verschärft, indem Funktionalitäten weiter in Hilfsmethoden und -klassen differenziert werden, um die Lesbarkeit zu fördern. Dies hilft zwar dem Verständnis der einzelnen Programmteile, macht es aber auch komplexer, die für die Semantik notwendigen Informationen von denjenigen zu trennen, die nur für die Implementierung relevant sind, wie beispielsweise verwendete Datenstrukturen. So helfen die beiden Methoden `enterCredentials` und `submit` beim Verständnis der Funktionalität der Klasse

Authentication in Abbildung 4.1, die Methode `getHandler` hingegen stellt ein Implementierungsdetail dar.

Ein weiterer Aspekt der semantischen Lücke resultiert aus der Form der natürlichsprachlichen Elemente im Quelltext. In heutigen häufig verwendeten objektorientierten Programmiersprachen, wie z. B. Java, C# oder C++, setzt sich Quelltext aus einer Mischung von wohldefinierten Sprachkonstrukten der Programmiersprache (wie `if`, `for` oder `class`) und natürlichsprachlichen Bezeichnern für Variablen, Attribute, Methoden, Parameter und Klassen zusammen. Die Sprachkonstrukte tragen zwar Semantik bezüglich der Ausführung des Quelltextes, stellen aber im Bezug auf die Anforderungen zumeist nur Implementierungsdetails dar, welche einzeln betrachtet einen nur geringen bis keinen Mehrwert für die Abbildung auf Anforderungen liefern. Außerdem treten immer wieder dieselben Sprachkonstrukte in verschiedenen Quelltextartefakten auf. Erst eine semantische Interpretation der verschiedenen Kombinationen der Sprachkonstrukte liefert eine Differenzierbarkeit. Die Bezeichner hingegen tragen, durch ihre potenziell beschreibenden natürlichsprachlichen Namen, Informationen zur Semantik des Quelltextabschnittes, in dem sie auftreten. Im Verhältnis zu den textuellen Beschreibungen der Anforderungen sind diese allerdings meist deutlich kürzer gefasst. Üblicherweise stellen z. B. Methodennamen stichpunktartige Beschreibungen der Funktionalität der Methode dar. Außerdem werden Abkürzungen verwendet, um Schreibarbeit zu sparen oder mehrere Bezeichner für ähnliche Daten oder Funktionalitäten nutzen zu können. Die Art der Beschreibung erschwert also zusätzlich eine Abbildung zwischen Anforderungen und Quelltext.

Eine Ausnahme können hier Quelltextkommentare darstellen. Diese sind oftmals als natürlichsprachlicher Text, bestehend aus vollständigen Sätzen, verfasst. Ist dies der Fall, stellen sie von ihrer Form und Semantik die den Anforderungen naheliegendste Form von Inhaltselementen des Quelltextes dar. Allerdings bleibt auch bei Kommentaren der Aspekt der unterschiedlichen Abstraktionsniveaus. Auch sie können zu implementierungsnah sein oder eine nicht vollständige Sicht auf die für die Abbildung zu Anforderungen notwendige Semantik des Quelltextes bieten. Zusätzlich sind Kommentare im Quelltext oftmals nicht vorhanden, von schlechter Qualität, oder inkonsistent mit der sich im Entwicklungsprozess ändernden Funktionalität [Wen+19].

Für die automatisierte Wiederherstellung von TLs zwischen Anforderungen und Quelltext stellt die semantische Lücke eine der größten, wenn nicht sogar die größte Herausforderung dar [De +12]. Sie erschwert die Identifikation von Zusammenhängen zwischen den beiden Artefakttypen dadurch, dass zum einen textuelle/syntaktische Vergleiche nicht ausreichen und zum anderen semantische Vergleiche auf demselben Abstraktionsniveau schwieriger zu erreichen sind. Hieraus lässt sich als Ziel für ein Verfahren zur automatisierten Wiederherstellung von TLs zwischen Anforderungen und Quelltext ein Überbrücken der semantischen Lücke ableiten. Bisherige Verfahren, die syntaktische Ähnlichkeitsvergleiche einsetzen (s. Abschnitt 3.1.1) konnten dieses Ziel nicht ausreichend erfüllen. Auch auf Einbettungen basierende Verfahren, wie WQI [ZCS17], das Verfahren von Chen et al. [Che+19], welches auf sequenzielle Semantik setzt, oder das Verfahren von Zhang et al. [Zha+21], die somit bereits auf eine Art von semantischem Vergleich setzen, erzielen nicht die für eine Automatisierung nötigen Ergebnisse. Dies deutet darauf hin, dass sie die

nötige semantische Ähnlichkeit für das Bestimmen des Zusammenhangs nicht ausreichend repräsentieren konnten. Es bleibt also zu untersuchen, welche Form der Repräsentation der Artefakte und welche Art des Ähnlichkeitsvergleichs besser mit der semantischen Lücke umgehen kann.

Ziel 2: Überbrücken der semantischen Lücke

Unterziel 2.1: Abbilden des Zusammenhangs zwischen den Artefakten über semantischen Ähnlichkeitsvergleich

4.3. Semantik von Anforderungen

Anforderungen können in unterschiedlichen Formen festgehalten werden. Dies kann von formal definierten, aussagenlogisch überprüfbaren Beschreibungen bis hin zu uneingeschränkten natürlichsprachlichen Äußerungen reichen. Im Rahmen dieser Arbeit fokussiere ich mich auf natürlichsprachlich beschriebene Anforderungen unterschiedlicher Ausprägungen (s. Einschränkung 1). Diese Form von Anforderungen ist die weitaus verbreitetste in Projekten in der Praxis [KNL14; MFN04], weshalb eine Verbesserung der Nachverfolgbarkeit für diese das größte Potenzial für einen tatsächlichen Nutzen in der Industrie bietet. Unter dieser Kategorie fasse ich sowohl uneingeschränkte natürlichsprachliche Beschreibungen als auch auf natürlicher Sprache basierende, einem gewissen Muster folgende Formen von Anforderungen, wie beispielsweise Anwendungsfallbeschreibungen oder Anwender:innenerzählungen, zusammen. Durch diese breite Definition von natürlichsprachlichen Anforderungen muss ein Verfahren also in der Lage sein, mit den inhärenten Eigenschaften uneingeschränkter natürlicher Sprache, wie Mehrdeutigkeiten, Auslassungen oder Ungenauigkeiten umgehen zu können.

Anforderung 1: Umgang mit uneingeschränkter natürlicher Sprache

Für die freiste sprachliche Form, uneingeschränkter natürlichsprachlicher Anforderungen, entspricht die Semantik dieser Anforderungen der Semantik, wie sie für natürliche Sprache definiert ist (s. Abschnitt 2.6.3). Das heißt, die Semantik einer Anforderung wird aus den Zusammenhängen zwischen den Bedeutungen ihrer Einheiten und dem Kontext, in dem sie geäußert wird, gebildet. So wird die Semantik von Anforderungen, die aus mehreren Sätzen bestehen, maßgeblich aus der Bedeutung der einzelnen Sätze und den Zusammenhängen zwischen diesen Sätzen gebildet. Die Bedeutung der Sätze wiederum ergibt sich aus den Bedeutungen der in ihnen enthaltenen Phrasen und deren Zusammenhänge und letztlich aus den Bedeutungen der Wörter in den Phrasen. Außerdem müssen die Bedeutungen im Kontext der Domäne des beschriebenen Systems und den weiteren Anforderungen interpretiert werden. So hat beispielsweise die Aussage „*the system creates a new record*“ eine andere Semantik, je nachdem, ob sie Teil einer Anforderung ist, die ein System zur

Use case name:	AdvancedSearch
Description:	The tourist is searching for a site using the potential offered by the Advanced Search.
Participating Actor:	Initialized by Tourist
Entry conditions:	The Tourist has successfully authenticated to the system.
Flow of events:	<ol style="list-style-type: none">1. Enable the advanced search feature from your personal area.2. View the advanced search form.3. Fill in the form of advanced search and submit.4. Gets the position of relying on the tourist event of the use location and process the request.
Exit conditions:	The system displays a list of results. Interruption of the connection to the server ETOUR.
Quality requirements:	The system requires the transaction in more than 15 seconds.

Abbildung 4.2.: Beispiel einer Anwendungsfallbeschreibung aus dem eTour-Projekt [CoE]

Musikaufnahme oder ein System zur Patientenverwaltung beschreibt. Die Semantik einer Anforderung wird also maßgeblich durch die Bedeutungen der in ihr enthaltenen textuellen Elemente, dem Kontext der Domäne und den weiteren Anforderungen für das System bestimmt.

Doch nicht alle Teile einer Anforderung müssen gleich bedeutungstragend sein. Gerade Anforderungen, die einem gewissen Schema folgen, wie z. B. das weitverbreitete Voranstellen von „*the system should/shall ...*“, enthalten oftmals Teile, die wenig zur Differenzierung der Bedeutung dieser Anforderung im Vergleich zu den anderen Anforderungen beitragen. So kann dieser vorangestellte Teil für die Interpretation der Semantik beispielsweise weggelassen werden. Gleiches gilt für Anwender:innenerzählungen, die zumeist einem Muster wie beispielsweise „*As a [who], I want to [what] so that [why]*“ folgen. Diese enthalten also zum einen Teile, die sich in allen Anwender:innenerzählung wiederholen und zum anderen sind nicht alle Teile dieser Muster relevant für eine Zuordnung zum Quelltext. Zumeist wird der Nutzer:innentyp (*who*) nicht explizit im Quelltext modelliert. Falls er modelliert wurde, ist er außerdem für einen Großteil der TLs nicht das Ziel, weshalb er meist Rauschen darstellt. Selbes kann für die Begründung (*why*) gelten, welche zwar den Hintergedanken der Anwender:innenerzählung für Entwickler:innen verdeutlichen kann, aber wie z. B. in „*As a user, I can indicate folders not to backup so that my backup drive isn't filled up with things I don't need saved.*“ keine Informationen zur Funktionalität beinhaltet und somit wenig hilfreich für die Abbildung auf Quelltext ist. Dadurch bleibt vor allem das Nutzerziel (*what*) für eine Interpretation der Semantik von Anwender:innenerzählungen, die hilfreich für die TLR ist.

Im Fall von Anwendungsfallbeschreibungen spielen außerdem die häufig verwendeten Beschriftungen eine entscheidende Rolle für die Interpretation des auf sie folgenden Textes, aber die Beschriftungen selbst gleichen sich für alle Anwendungsfallbeschreibungen. So macht die Beschriftung *Entry conditions* im Beispiel in Abbildung 4.2 klar, dass es sich bei dem Satz „*The Tourist has successfully authenticated to the system*“, um eine Vorbedingung

für das Auftreten des Anwendungsfalles handelt. Allerdings kann bei der Repräsentation der Semantik des Anwendungsfalles die Beschriftung selbst weggelassen werden, da sie in jeder Anwendungsfallbeschreibung vorhanden ist. Diese Form von Vorlagenelementen mit Beschriftungen sind gerade für Anwendungsfallbeschreibungen sehr häufig. Es gibt eine Vielzahl solcher Vorlagen, allerdings wiederholen sich gewisse Elemente, wie z. B. eine Form von Namen, eine Beschreibung oder Kurzzusammenfassung, die Akteure, Vor- und Nachbedingungen und die zentrale Ablaufbeschreibung (vgl. [Coc00; Bal10; JHH13]). Auch hier gilt wieder, unter dem Gesichtspunkt der TLR tragen diese einzelnen Elemente nicht gleichermaßen zum für eine Abbildung nötigen Verständnis der Anwendungsfallbeschreibung bei. So beschreibt beispielsweise das Element *Quality requirements* Qualitätsmerkmale des Systems, die sich so typischerweise nicht explizit im Quelltext des Systems wiederfinden, sondern vor allem beim Entwurf des Gesamtsystems in Betracht gezogen werden¹. Hier gilt Ähnliches wie auch schon bei den Anwender:innenerzählungen für die Akteure/Nutzer. Die Vor- und Nachbedingungen sind ein etwas anderer Fall. Diese lassen sich zwar zum Teil auf Funktionalitäten des beschriebenen Systems abbilden, setzen aber vielmehr den Rahmen für den Anwendungsfall und beschreiben somit eigentlich nicht die Funktionalität des Anwendungsfalles selbst. Deshalb werden die zu den Bedingungen korrespondierenden Quelltextelemente zumeist nicht über TLs mit dem Anwendungsfall verbunden.

Für die erfolgreiche TLR zwischen Anforderungen und Quelltext ist also das Bestimmen der tatsächlich semantisch relevanten Teile der Anforderungen wichtig. Hierzu können zum einen Informationen über die verwendeten Muster oder Vorlagen helfen, zum anderen kann auch eine Klassifikation von Anforderungen beispielsweise in funktionale oder nichtfunktionale Teile relevant sein.

Ziel 3: Ableiten einer geeigneten Repräsentation der Semantik von Anforderungen

Unterziel 3.1: Erkennen von für die Nachverfolgbarkeit zu Quelltext relevanten Teilen der Anforderungen

4.4. Semantik von Quelltext

Ähnlich wie bei den Anforderungen bildet sich auch die Semantik von Quelltextelementen aus den Zusammenhängen der Bedeutungen der Teile eines Quelltextelements und dem Kontext, in dem das Quelltextelement steht. Als Grundeinheit für die Nachverfolgbarkeit wird weitestgehend ein Quelltextartefakt (eine Datei) herangezogen. Dies beinhaltet für

¹ Dies gilt für jegliche Art von nichtfunktionalen Anforderungen. Auch wenn in der Literatur debattiert wird, ob nichtfunktionale Anforderungen nicht doch zumindest indirekt zu konkreten Funktionalitäten des Systems führen [Gli07; Li+14; EVF16], so sind sie doch in Bezug auf Nachverfolgbarkeit selbst für den Menschen meist nicht konkret abbildbar.

objektorientierte Systeme zumeist eine Klasse oder Schnittstelle, kann aber im Fall von existierenden inneren Klassen auch mehrere Typen enthalten. Eine Klasse bzw. Schnittstelle steht dabei im Kontext der Komponente/dem Modul/dem Paket, in dem sie definiert ist und sollte nach Möglichkeit einen bestimmten Zweck² erfüllen. Dieser Zweck setzt sich dabei aus den Eigenschaften der Klasse bzw. ihrer nach außen sichtbaren Schnittstelle zusammen. Nur Elemente, die nach außen sichtbar sind, können auch von außen verwendet werden und damit den Zweck der Klasse beschreiben. Nach dem Geheimnisprinzips (engl.: *information hiding principle*) [PCW85] sollte eine Entwurfsentscheidung hinter einer wohldefinierten Schnittstelle verborgen werden und somit eine Änderung dieser Entscheidung nicht zur Änderung der Schnittstelle führen. Dies impliziert, dass Teile eines Quelltextartefakts, welche nicht nach außen sichtbar sind, implementierungsspezifisch und somit eher nicht semantiktragend sind. Dies gilt nicht in allen Fällen und auch das Prinzip lässt sich nicht für jede Implementierung erfüllen, aber als Heuristik kann dies für eine Auswahl der relevanten Elemente verwendet werden. Auf Basis dieser Beobachtungen und der weitreichenden Akzeptanz des Geheimnisprinzips gehe ich im Rahmen dieser Arbeit von folgender Annahme aus:

Annahme 2: Nach außen sichtbare Elemente eines Quelltextartefaktes tragen einen größeren Beitrag zur Semantik bei als interne Elemente.

Betrachtet man nun die nach außen sichtbaren Elemente einer Klasse/Schnittstelle, so setzt sich deren Semantik wiederum aus den sie bildenden Elementen im Kontext der Klasse zusammen.

Attribute Attribute werden durch ihren Bezeichner, Typ und ggf. einen Kommentar beschrieben. Zumeist werden Attribute allerdings nicht nach außen sichtbar gemacht oder nur durch öffentlich sichtbare Lese- und Schreibmethoden abgebildet. Im ersten Fall, die Attribute werden nicht nach außen gegeben, kann angenommen werden, dass sie nur für die interne Verwendung und somit implementierungsspezifisch sind. Im zweiten Fall, tragen die Attribute zwar zum Zweck der Klasse bei, aber die Informationen, die aus dem Bezeichner und dem Typ des Attributes gezogen werden können, sind redundant in der Definition und ggf. dem Kommentar der Lese- bzw. Schreibmethoden enthalten. Aus diesem Grund kann die Semantik der Attribute auf die Semantik der zu ihnen gehörenden Lese- bzw. Schreibmethoden reduziert werden.

Methoden Eine Methode setzt sich aus der Methodensignatur, also dem Methodennamen und ggf. Parameternamen und -typen, dem Methodenrumpf und ggf. einem Methodenkommentar zusammen. Der Methodenrumpf wiederum besteht aus Anweisungen und ggf. Kommentaren und diese Anweisungen können wieder aus Bezeichnern, Typen usw.

² Beispielsweise das Bereitstellen einer gewissen Funktionalität oder das Halten von Daten einer bestimmten Entität. Es gibt allerdings auch Zwecke, die sich über mehrere Klassen verteilen, sogenannte Querschnittsanliegen (engl.: *cross-cutting concerns*), wie beispielsweise das Protokollieren (engl.: *logging*).


```
1  /**
2   * Retrieves the names of persons in this class.
3   *
4   * @return a List of names of persons in this class
5   */
6  public List<Name> retrievePersonsNames() {
7      List<Name> result = new ArrayList();
8      // iterate over data structure
9      for(Person person : persons) {
10         result.add(person.getName());
11     }
12     return result;
13 }
```

Quelltextausschnitt 4.1: Beispiel einer Methode mit Kommentar und Rumpf

bestehen. Ähnlich wie bei strukturierten Anforderungen tragen auch im Fall von Methoden nicht alle Elemente gleichermaßen zur Beschreibung der Semantik der Methode und damit der Semantik der sie beinhaltenden Klasse bei. Der Methodename beispielsweise sollte eine kurze aber verständliche Beschreibung des Zwecks der Methode darstellen und somit idealerweise beschreibend für die Semantik der Methode sein. Ebenso tragen die Parameternamen Informationen über die Eingaben der Methode und somit potenziell den behandelten Objekten. Die Bezeichner der Parametertypen können ebenso hilfreich für das Verständnis des Zwecks der Methode sein, können aber auch aus Standarddatentypen wie Listen oder primitiven Datentypen bestehen, welche zu implementierungsspezifisch sein können.

Noch schwieriger zu interpretieren, ist der beschreibende Beitrag einer Instruktion im Methodenrumpf zur Semantik der Methode. Denn eine Instruktion im Methodenrumpf beschreibt maximal einen Teilschritt zur Erfüllung des Zwecks der Methode. Außerdem stellt eine Instruktion in vielen Fällen nur ein implementierungsspezifisches Detail dar, welches nur schwer (selbst im Zusammenspiel mit den anderen Instruktionen) auf ein für die Nachverfolgbarkeit hilfreiches Abstraktionsniveau zurückgeführt werden kann. Selbiges gilt für Kommentare im Rumpf einer Methode. Diese können zwar hilfreiche Informationen für das Verständnis des Quelltextes beinhalten, sind aber oftmals implementierungsspezifische Beschreibungen der auf sie folgenden Instruktionen. So hilft zwar der Kommentar `//iterate over data structure` der Methode `retrievePersonsNames` in Quelltextausschnitt 4.1 Entwickler:innen beim Verständnis der internen Funktionsweise der Methode, aber weniger bei der Repräsentation des Zwecks der Methode.

Anders ist dies beim Kommentar der Methode, sofern dieser vorhanden ist. In den meisten Programmiersprachen ist es Konvention, dass ein Kommentar direkt über einer Methodendefinition eine Beschreibung der Funktionalität der Methode enthält. Dieser kann als normaler Zeilen- oder Blockkommentar vorliegen oder aber auch einer Vorgabe zur automatischen Erzeugung einer Dokumentation, wie beispielsweise Javadoc für Java, folgen (s. Abschnitt 2.3.2). In allen Fällen stellt der Methodenkommentar eine in natürlicher Sprache

verfasste Beschreibung der Semantik der Methode dar, welche potenziell über die Informationen im Methodennamen hinausgeht. Im Fall von Dokumentationskommentaren kann außerdem die Struktur genutzt werden, um redundante oder implementierungsspezifische Informationen zu filtern. So enthält der Javadoc-Kommentar in Quelltextausschnitt 4.1 die allgemeine Beschreibung und die explizite Beschreibung des Rückgabewertes, welche in diesem Fall redundant sind. In anderen Fällen, wie z. B. einer Beschreibung von Ausnahmen oder primitiven Datentypen als Parametern, können die Informationen zu implementierungsnah für eine Ableitung der Semantik auf ein für die Nachverfolgbarkeit geeignetes Abstraktionsniveau sein. Da Methodenkommentare im Gegensatz zu Methodensignaturen nicht verpflichtend für die Beschreibung einer Methode sind, kann nicht davon ausgegangen werden, dass in allen Fällen Methodenkommentare vorliegen. Deshalb können Methodenkommentare nur als optionale Informationsquelle für die Beschreibung der Semantik einer Methode herangezogen werden.

Eine Information aus dem Methodenrumpf, die sich bereits in anderen Arbeiten zur TLR als vorteilhaft erwiesen hat, sind Aufrufabhängigkeiten zwischen Methoden (s. Abschnitt 3.1.1.3). Gemeint ist hiermit das Einbeziehen von Informationen über die von einer Methode aufgerufenen anderen Methoden und Methoden, welche die aktuell betrachtete Methode aufrufen. Allerdings zeigen die Ergebnisse von Panichella et al. [Pan+13] auch, dass diese nicht in allen Fällen zu verbesserten Ergebnissen führten. Eine mögliche Erklärung wäre hier wieder, dass Teile der Methoden zu implementierungsspezifisch sind und somit wenig Aussagekraft bezüglich der Semantik der aktuell betrachteten Methode bieten, wohingegen andere durchaus hilfreiche Informationen bereitstellen. So ist der Aufruf des Konstruktors `new ArrayList()` in Quelltextausschnitt 4.1 lediglich ein Implementierungsdetail der verwendeten Art von Liste, welches unter Betrachtung des Geheimnisprinzips keine Relevanz für den Zweck der Methode haben sollte. Der Aufruf von `person.getName()` hingegen kann bei der Interpretation der Semantik der Methode helfen, da er den Zusammenhang zwischen der Methode `retrievePersonsNames()` und dem Datentyp `Person` sowie dessen *Getter*-Methode für das Attribut `name` darstellt. Die Herausforderung ist also zu bestimmen, in welchen Fällen das Einbeziehen von Aufrufabhängigkeiten Sinn ergibt.

Klasse/Schnittstelle Auch die Klassen bzw. Schnittstellendefinition trägt in den meisten objektorientierten Programmiersprachen zur nach außen sichtbaren Schnittstelle eines Quelltextartefaktes bei. So beschreibt der Name der Klasse bzw. Schnittstelle, welche Form von Objekten dieser Typ repräsentiert und schafft somit den Kontext, in dem die Semantik der Methoden steht. Dies ist insbesondere entscheidend, falls mehrere Klassen gleich oder ähnlich benannte Methoden enthalten, welche nur durch den Bezug zur Klasse unterschieden werden können. So ist beispielsweise die Semantik einer Methode `getAddress()` eine andere, je nachdem, ob sie Teil einer Klasse mit dem Namen `Person` oder `Website` ist³. Allerdings muss der Klassenname nicht zwangsläufig eine Objektbeziehung vorgeben. Handelt es sich bei der Klasse lediglich um eine Sammlung von Funktionen,

³ In ersterem Fall bezieht sich die Methode auf die Anschrift der Person. Im zweiten Fall ist die gemeinte Adresse ein Verweis auf die Internetadresse der Webseite.

eine sogenannte *Utility*-Klasse, so beschreibt der Name meist vielmehr, welche Arten von Funktionalitäten in dieser Klasse zusammengefasst werden. In beiden Fällen trägt der Name allerdings zur Semantik des Artefakts bei und sollte somit nicht ignoriert werden.

Zusätzlich kann die Definition einer Klasse oder Schnittstelle in objektorientierten Sprachen Informationen zu implementierten Schnittstellen oder erweiterten Klassen beinhalten und somit Informationen zum Kontext, in den sie eingebettet ist, geben. So kann für die Klasse `Bag` aus ihrer Klassensignatur `class Bag<E> implements Collection<T>` abgeleitet werden, dass es sich um eine Datensammlungsklasse handelt, da sie die `Collection`-Schnittstelle implementiert. Selbiges gilt für Modul- bzw. Paketinformationen. Ebenso wie Methoden können Klassen- bzw. Schnittstellendefinition beschreibende Kommentare besitzen. Diese sollten gleichermaßen die Semantik des beschriebenen Typs in natürlicher Sprache wiedergeben und spielen somit eine Rolle für die TLR. Diese können, wie auch schon die Methodenkommentare, in Zeilen-, Block- oder Dokumentationsform vorliegen. Auch hier gilt wieder, dass nicht alle Teile die Semantik des Typs beschreiben. Im Fall von Javadoc wird beispielsweise häufig durch `@author` der Autor der Klasse angegeben, welcher zwar relevant für Entwickler:innen sein kann, aber für eine Zuordnung zu Anforderungen zumeist keine Rolle spielt⁴. Auch die Klassen- bzw. Schnittstellenkommentare sind optional und liegen somit nicht immer vor, weshalb sie nur als zusätzliche Information für die Repräsentation der Semantik eines Quelltextartefaktes genutzt werden können.

Zusammenfassend lässt sich auch für die Semantik eines Quelltextartefaktes sagen, dass nicht alle Teile gleichermaßen relevant sind und somit auch hier die größte Herausforderung darin besteht, die relevanten Teile auszuwählen und aus ihnen eine geeignete Repräsentation abzuleiten.

Ziel 4: Ableiten einer geeigneten Repräsentation der Semantik von Quelltextartefakten

Unterziel 4.1: Identifikation der für die Nachverfolgbarkeit zu Anforderungen relevanten Teile eines Quelltextartefaktes

4.5. Repräsentation der Semantik

Ausgehend von den für die Nachverfolgbarkeit relevanten Elementen der Anforderungen und des Quelltextes müssen diese in eine Repräsentation überführt werden, die es ermöglicht semantische Ähnlichkeiten zu bestimmen. Bestehende Verfahren setzen hierzu weitestgehend auf eine Repräsentation der Artefakte als Vektoren im Vektorraum (s. Abschnitt 3.1.1). Sie unterscheiden sich allerdings in der Art, wie sie diesen Vektor erzeugen.

⁴ Es sei denn auch die Anforderungen enthält Informationen zum Autor. Allerdings lässt sich aus einer Übereinstimmung des Autors der Anforderung und des Quelltextes nicht notwendigerweise eine Verbindung dieser Beiden folgern.

Dieses Vorgehen erleichtert die anschließende Ähnlichkeitsberechnung, da hierzu Vektordistanzmaße genutzt werden können. Entscheidend bei dieser Art der Repräsentation ist allerdings die Güte, mit der sie die Semantik der Artefakte im Vektorraum darstellen. Idealerweise sollten semantisch ähnliche Artefakte auf ähnliche Vektoren abgebildet werden. Die Abbildung in den Vektorraum sollte also möglichst semantikerhaltend sein.

Ziel 5: Die Abbildung der Artefakte in einen Vektorraum soll semantikerhaltend sein.

Betrachtet man nun bestehende Verfahren hinsichtlich dieses Ziels, so lassen sich verschiedene Schwachstellen ausmachen. Eine Repräsentation über BoW-Vektoren, mit oder ohne TF-IDF-Gewichtung, hat Schwierigkeiten mit Synonymen und Dokumenten, die nicht syntaktisch ähnlich sind (s. Abschnitt 3.1.1). Durch den Einsatz von Themenmodellierung wird versucht, möglichst repräsentative Begriffe für die Artefakte zu bestimmen bzw. die Artefakte als Verteilung über die Gesamtmenge der Wörter aufzufassen. Die Ergebnisse mit LDA und LSI deuten auf eine bessere Repräsentation hin. Allerdings haben auch diese Verfahren Probleme mit Formulierungen, die nicht dieselben Begriffe beinhalten (s. Abschnitt 3.1.1). All diese Verfahren basieren also auf der Annahme, dass übereinstimmende Syntax bzw. Begriffe, ein geeigneter Indikator für semantische Ähnlichkeit ist. Sie repräsentieren also Artefakte ähnlich, die viele oder einige als wichtig identifizierte Begriffe teilen. Enthalten die Artefakte aber nicht dieselben Begrifflichkeiten oder befinden sich die Begriffe auf unterschiedlichen Abstraktionsniveaus, wie es bei Anforderungen und Quelltext häufig der Fall ist (s. Abbildung 4.1), so sind diese Repräsentationen nicht in der Lage, die geteilte Semantik abzubilden.

In der Grundlagenforschung der Computerlinguistik und in anderen verwandten Aufgabenstellungen werden vermehrt Sprachmodelle zur Repräsentation eingesetzt, um diesem Problem zu begegnen (s. Abschnitte 2.9.3 und 3.6). Sprachmodelle erlernen semantische Zusammenhänge zwischen Wörtern, Sätzen und Dokumenten aus großen Textkorpora, indem sie der Hypothese folgen, dass Wörter/Sätze/Dokumente, die häufig in ähnlichen Kontexten auftreten, auch eine ähnliche Semantik haben (s. Abschnitt 2.9). Sie sind also in der Lage, auch semantische Ähnlichkeiten abzubilden, die über die Verwendung derselben Begriffe und Formulierungen hinausgehen.

Im Rahmen der Verfahren zur automatischen Generierung von TLs zwischen Anforderungen und Quelltext wurden bisher *word2vec*- und *doc2vec*-Einbettungen eingesetzt (s. Abschnitt 3.1.3). Die Ergebnisse, die diese Verfahren erzielen konnten, waren allerdings zum Teil sogar schlechter als die vorangehenden Ansätze ohne Sprachmodelle. Eine Begründung hierfür könnte das Training der Modelle sein. Zwei der Verfahren [Che+19; Zha+21] trainieren das verwendete statische Sprachmodell nur auf den Artefakten selbst. Einen Vorteil, den dies bietet, ist, dass hierdurch alle zu repräsentierenden Wörter bereits im Training auftraten und somit auch repräsentiert werden können; es also keine Probleme mit Wörtern außerhalb des Vokabulars gibt (s. Abschnitt 2.9.2). Statische Worteinbettungen erlernen semantische Zusammenhänge zwischen den Wörtern über die Wörter, mit denen diese zusammen auftreten (s. Abschnitt 2.9.2). Wörter, die häufig mit denselben

anderen Worten gemeinsam auftreten, erhalten eine ähnlichere Repräsentation als Worte, die nur wenige Kontexte teilen. Nutzt man jetzt allerdings die Projekte selbst als Trainingsdaten, so ist die Menge der gleichen Kontexte aufgrund der Größe der Projekte sehr gering. Zusätzlich können bei einem solchen Vorgehen nur semantische Zusammenhänge erfasst werden, die in den Projekten auch auftreten. Treten beispielsweise die Wörter *Healthcare* und *Patient* nur mit unterschiedlichen Wörtern in ihrem Kontext auf, kann das Worteinbettungsmodell den semantischen Zusammenhang zwischen diesen nicht erlernen. Aufgrund dieser Problematik werden Sprachmodelle zumeist auf sehr großen möglichst diversen Trainingsmengen vortrainiert. Hierbei ist die Annahme, dass bei genügend Daten im Training, eine Vielzahl der Zusammenhänge erfasst werden können. Die vielversprechenden Ergebnisse der immer größer werdenden Sprachmodell auf diversen Aufgaben der Sprachverarbeitung und des Sprachverständnisses bestätigen diese Annahme [Boj+17; Dev+19; Liu+19; Yan+19]. Aus diesem Grund nehme ich an, dass auch für die TLR ein Sprachmodell, welches auf einem sehr großen Datensatz vortrainiert ist, zu einer besseren Repräsentation der Semantik führt, als auf den Projekten selbst trainierte Modelle:

Annahme 3: Das Nutzen eines auf einem großen, diversen Textkorpus vortrainierten Sprachmodells wirkt sich positiv auf dessen Fähigkeit der semantischen Repräsentation der Artefakte in der TLR aus.

Des Weiteren sorgt die Repräsentation der Artefakte durch ein Worteinbettungsverfahren dafür, dass ein Artefakt durch die Menge der Vektoren der in ihm enthaltenen Wörter repräsentiert wird. Es wird also nicht ein Vektor pro Artefakt erstellt, wie bei BoW oder *doc2vec*, sondern eine Menge von Vektoren als Repräsentation genutzt. Um nun die Semantik des Artefakts abzubilden, müssen also die Repräsentationen der Wörter als Menge interpretiert oder die Vektoren aggregiert werden. Ein Aggregieren der Vektoren kann dazu führen, dass der resultierende Vektor in einem Bereich des Vektorraums liegt, der einer ganz anderen Semantik entspricht. Dies ist vor allem dann gefährlich, wenn ein Artefakt verschiedene Aspekte darstellt, z. B. wenn eine Klasse mehrere unterschiedliche Funktionalitäten bereitstellt. So könnte ein weiterer Faktor für die schlechten Ergebnisse der bestehenden, mit Sprachmodellen arbeitenden Verfahren, ihre Art der Aggregation sein. Zhao et al. [ZCS17] nutzen beispielsweise das Mittel der maximalen Ähnlichkeiten der Wortpaarungen, bestehend aus einem Wort aus dem Quell- und einem aus dem Zielartefakt. Dies kann dazu führen, dass die Information darüber, wie sich die Wörter allgemein zueinander verhalten, verloren geht.

Neben den bereits für die Anforderung-zu-Quelltext Nachverfolgbarkeit genutzten statischen Sprachmodellen, werden in den letzten Jahren vor allem neuronale, kontextsensitive Sprachmodelle wie *BERT* zur Repräsentation von textueller Semantik genutzt. Auch im Bereich der Nachverfolgbarkeit wurden diese bereits für die Abbildung zwischen VCS-Einbuchungen und Belangen genutzt und erzielten überzeugende Ergebnisse (s. Abschnitt 3.4). Ein Problem der Anwendung dieser Sprachmodelle für die TLR zwischen Anforderungen und Quelltext ist die Struktur des Quelltextes. Die Informationen des Quelltextes liegen nicht notwendigerweise als natürlichsprachliche Sätze vor, sondern vielmehr als Menge von Bezeichnern. Selbst die Kommentare des Quelltextes müssen

nicht aus vollständigen Sätzen bestehen, sondern enthalten oftmals ebenso stichpunktartige Beschreibungen. Da diese Form von Sprachmodellen allerdings auf Sequenzen von Wörtern trainiert werden, um den dynamischen Kontext und damit die Bedeutung zu repräsentieren, ist unklar, wie übertragbar die vortrainierten Modelle auf Quelltext sind. Da diese Sprachmodelle ebenso eine noch deutlich größere Menge an Trainingsdaten benötigen als statische Sprachmodelle, wäre das Zurückgreifen auf vortrainierte Modelle oder das Feinanpassen für den Rahmen dieser Arbeit unumgänglich.

Das entgegengesetzte Problem könnte es mit vortrainierten Sprachmodellen geben, die sowohl auf natürlichsprachlichem Text als auch Quelltext trainiert wurden, wie *CodeBERT* [Fen+20] oder *UniXcoder* [Guo+22a]. Hier sind die natürlichsprachlichen Beschreibungen hauptsächlich Quelltextdokumentationskommentare und somit implementierungsnäher als die Beschreibungen in den Anforderungen. Bei einem Einsatz dieser, wäre also die Übertragbarkeit der Repräsentation auf Anforderungen zu prüfen.

4.6. Semantische Ähnlichkeitsvergleiche

Betrachtet man bestehende Verfahren zur TLR zwischen Anforderungen und Quelltext (s. Abschnitt 3.1), so setzen diese weitestgehend auf Vektordistanzmaße zur Abbildung der Ähnlichkeit zweier Artefakte. Hierbei werden die Artefakte durch Vektoren in einem gemeinsamen Vektorraum repräsentiert. Die eigentliche Ähnlichkeit ergibt sich daraufhin aus dem Abstand der Vektoren. Die beiden üblichsten Maße hierfür sind die Kosinusähnlichkeit und der euklidische Abstand (s. Abschnitt 2.10). Erstere ignoriert die Länge des Vektors, was je nach Repräsentation von Vorteil sein kann. Nutzt man Vektordistanzmaße als Form der Ähnlichkeit, so hängt die Güte der Abbildung vor allem von der Güte der Repräsentation der Semantik durch den genutzten Vektor ab. Werden semantisch ähnliche Artefakte nicht auf ähnliche Vektoren abgebildet oder sind sich alle Vektoren, unabhängig von ihrer Semantik, nahezu gleich ähnlich, so kann ein solches Vorgehen zu keinem nutzbaren semantischen Ähnlichkeitsvergleich führen.

Über die Vektordistanzmaße hinaus setzen bestehende Verfahren außerdem auf mengenbasierte Ähnlichkeiten [Loh+13; ZCS17; RC20]. Der Jaccard-Koeffizient⁵ (s. Abschnitt 2.10) beispielsweise bestimmt die Ähnlichkeit zweier Dokumente über den Anteil der Begriffe, die in beiden Dokumenten enthalten sind. Ein Nachteil dieser Art von Ähnlichkeitsvergleichen ist ihre Einschränkung auf das Auftreten von gleichen Begriffen. Sie haben also Probleme mit Synonymen und verwandten Begriffen sowie mit semantischen Zusammenhängen, die sich erst aus dem Zusammenspiel mehrerer Begriffe ergeben. Sie bieten somit vielmehr einen textuellen als einen semantischen Ähnlichkeitsvergleich.

Eine weitere Technik zur Bestimmung von Ähnlichkeiten stellt das Nutzen von maschinellem Lernen dar. Hierbei wird beispielsweise ein ANN oder ein Sprachmodell wie *BERT* darauf trainiert vorherzusagen, welche Dokumente eines Korpus ähnlich zueinander sind.

⁵ Ebenso der Dice-Koeffizient [Dic45], welcher in den Jaccard-Koeffizient überführt werden kann.

Hierzu benötigt es eine große Menge von Dokumenten mit einer Musterlösung für die Ähnlichkeit. Da das Erstellen einer solchen Musterlösung nicht einfach ist, werden oftmals mehrere Korpora für verschiedene verwandte Aufgaben⁶ dazu genutzt, um mittels Transferlernens ein geeignetes Modell zu lernen. Ein Nachteil dieser Form der Ähnlichkeitsberechnung ist die nötige Menge an Trainingsdaten und die daraus resultierende Frage nach der Generalisierbarkeit des gelernten Modells. Wurde das Modell nur auf Daten trainiert, die aus bestimmten Domänen stammen, so ist unklar, ob sie ohne Weiteres auch auf z. B. Quelltext angewandt werden können. Des Weiteren besteht ein Großteil des Quelltextes nicht aus natürlichsprachlichen Sätzen. Die vorhandenen Modelle [Cer+18; Con+17; RG19; Gio+21] werden allerdings weitestgehend auf natürlichsprachlichen Dokumenten und Sätzen trainiert, weshalb eine direkte Anwendbarkeit der Modelle auf Quelltext nicht unbedingt gegeben sein muss. Auch hier könnten die bimodalen, auf dualen Korpora aus natürlicher und Programmiersprache trainierten, Sprachmodelle, wie *CodeBERT* oder *UniXcoder* (s. Abschnitt 3.7), genutzt werden. Diese müssten aber ebenfalls erst auf die Aufgabe feinangepasst werden, um sie für die Ähnlichkeitsberechnung einzusetzen.

Da bisherige Kombinationen aus Repräsentation und Ähnlichkeitsbestimmung meist nur in bestimmten Fällen gute Ergebnisse liefern, werden Ansätze oftmals kombiniert. So werden gerade die weniger semantischen Verfahren, wie Jaccard oder TF-IDF, miteinander und mit aufwändigeren Verfahren zur Repräsentation, wie z. B. LDA oder JS, kombiniert, unter der Annahme, dass diese ihre gegenseitigen Schwächen ausgleichen (vgl. Abschnitt 3.1.1). In diesem Fall ist entscheidend wie die Kombination durchgeführt wird. Deshalb setzen die erfolgreichsten Kombinationsverfahren auf ein Training der Gewichtung der einzelnen Verfahren oder ein Ableiten der Gewichtung aus anderen Merkmalen der Artefakte (s. [MEH18; Mor+20b]). Moran et al. [Mor+20b] erzielten dabei sogar gute Ergebnisse, wenn sie die Gewichtung über Projektgrenzen hinweg übertragen. Somit stellt dieses Vorgehen eine Möglichkeit dar, auch auf neue, unbekannte Projekte angewandt zu werden, welches ein Ziel dieser Arbeit ist.

Über die in bestehenden Arbeiten verwendeten Verfahren zur Ähnlichkeitsberechnung hinaus haben sich in der Computerlinguistik weitere vielversprechende Verfahren zur Bestimmung von Dokumentenähnlichkeit entwickelt. Die Wortüberführungsdistanz (engl.: *Word Movers Distance*, WMD) nutzt hierbei Mengen von Einbettungsvektoren (s. Abschnitt 2.10). Sie eignet sich also für den Fall einer feingranularen Repräsentation der Artefakte. Anstatt die Menge an Vektoren zu mitteln oder die Vektoren einzeln abzubilden, nutzt WMD die minimale kumulative Distanz, die sich ergibt, wenn man die Vektoren, die Dokument A repräsentieren, auf die Vektoren, die Dokument B repräsentieren, abbildet. Sie repräsentiert also die minimale Distanz im Vektorraum, die benötigt wird, um die durch die Vektoren repräsentierte Semantik der Dokumente ineinander zu überführen. Zusätzlich werden die einzelnen Abbildungen der Wörter mit ihrer normierten Auftretenshäufigkeit im jeweiligen Dokument gewichtet, was dazu führt, dass häufige Wörter eine höhere Relevanz erhalten. Die WMD hat sich für Ähnlichkeitsvergleiche auf Onlineartikeln, ins-

⁶ Beispielsweise Korpora für die Inferenz natürlicher Sprache (engl.: *natural language inference*), welche semantische Implikation von Sätzen abbilden, oder Korpora für die Vorhersage, ob zwei Sätze innerhalb eines Absatzes/Dokumentes auftreten

besondere in Kombination mit Worteinbettungen, als überlegenes Verfahren zu TF-IDF, BoW oder LDA erwiesen [Kus+15]. Insbesondere die Eigenschaft, mit Worteinbettungen und beliebigen Mengen von diesen umzugehen, bietet Potenzial für einen Einsatz dieses Ähnlichkeitsverfahrens für feingranulare Abbildungen zwischen Anforderungen und Quelltext.

4.7. Ebene der Abbildung

Ein weiterer Faktor, der eine Rolle bei der Abbildung der Artefakte aufeinander spielt, ist die Ebene, auf der die Abbildung durchgeführt wird. Bestehende Verfahren operieren weitestgehend auf Artefaktebene (s. Abschnitt 3.1). Sie bestimmen also eine Repräsentation pro Artefakt, beispielsweise aus allen in diesem Artefakt enthaltenen Wörtern, und bilden zwischen diesen Artefaktrepräsentationen ab. Ein solches Vorgehen bietet den Vorteil, dass die Ähnlichkeit der Artefakte direkt abgeleitet werden kann. Sofern die Artefaktrepräsentation in der Lage ist, die Semantik aller Aspekte des Artefakts abzubilden und in Relation zu anderen Artefakten zu stellen, ist dies das einfachste und effizienteste Vorgehen. Die Ergebnisse der bestehenden Verfahren (s. Abschnitt 3.1.1) deuten aber darauf hin, dass dies nicht der Fall ist. Sie erzielen noch zu geringe Präzision bei angemessener Ausbeute. Ein Grund hierfür könnte sein, dass bisherige Methoden zur Repräsentation auf Artefaktebene zu grobgranular für eine zuverlässige Abbildung sind. Bietet eine Klasse beispielsweise unterschiedliche Funktionalitäten an und eine Anforderung beschreibt nur eine dieser Funktionalitäten, so können Bezeichner, die auf die anderen Funktionalitäten hindeuten, die Abbildung erschweren. Ebenso ist es beispielsweise bei Anwendungsfallbeschreibungen häufig, dass jeder Schritt der Ablaufbeschreibung Bezug zu einer anderen Funktionalität des Systems nimmt und somit mehrere Klassen adressiert.

Möchte man trotzdem auf bekannte Arten der Repräsentation zurückgreifen, kann es also von Vorteil sein, nicht auf Artefaktebene, sondern feingranularere Repräsentationen zu wählen und zunächst zwischen diesen abzubilden, um Verfälschungen zu minimieren. Im Gegenzug hat ein solches Vorgehen allerdings den Nachteil, dass eine Abbildung mit weniger Kontextinformationen auskommen muss. Um diesen Nachteil auszugleichen und auch Aussagen über die Zusammenhänge auf Artefaktebene geben zu können, lassen sich verschiedene Arten der Aggregation der feingranularen Ähnlichkeiten einsetzen. Die einfachste Möglichkeit ist es, jeweils die maximale, minimale oder durchschnittliche Ähnlichkeit der Artefaktelemente aus dem Quell- und Zielartefakt als Repräsentanten für die Ähnlichkeit zwischen den Artefakten zu nutzen. Ein solches Vorgehen hat verschiedene Nachteile. Die maximale Ähnlichkeit könnte die eigentliche Ähnlichkeit der Artefakte übertreiben, wenn sich z. B. zwei schlecht repräsentierte oder wenig semantiktragende Elemente der Artefakte sehr ähnlich sind. Ebenso kann die minimale Ähnlichkeit dazu führen, dass sich viele Artefakte gleich ähnlich sind, welche eigentlich eine ganz andere Semantik tragen. Die durchschnittliche Ähnlichkeit hat dasselbe Problem, welches die direkte Repräsentation auf Artefaktebene hat: Artefakte, die mehrere Funktionalitäten oder Aspekte beinhalten, können zu verfälschten Abbildungen führen.

In einer idealen Welt sollte die Relation der vertikalen Nachverfolgbarkeit bitotal sein, also jedes Artefakt mit höherem Abstraktionsniveau zu einem Artefakt mit niedrigerem Abstraktionsniveau in Verbindung stehen und umgekehrt. Aufgrund von Inkonsistenzen zwischen den Artefakttypen sind die Mengen aber oftmals unvollständig (z. B. fehlen Anforderungen zu bestimmten Teilen des Quelltextes oder es existieren Anforderungen, die gar nicht im Quelltext umgesetzt werden). Dies führt dazu, dass in der Praxis weder von Links- noch von Rechtstotalität der Relation ausgegangen werden kann. Allerdings kann aufgrund der Umsetzungs- bzw. Verfeinerungsrelation zwischen den Artefakttypen angenommen werden, dass Artefakte mit höherem Abstraktionsniveau zu mehreren Artefakten mit niedrigerem Abstraktionsniveau in Verbindung stehen. Artefakte mit niedrigerem Abstraktionsniveau korrespondieren hingegen tendenziell eher zu lediglich einem oder nur wenigen abstrakteren Artefakten. So wird eine Anforderung oftmals durch das Zusammenspiel mehrerer Quelltextklassen umgesetzt, eine Quelltextklasse trägt aber zumeist lediglich zur Umsetzung einer oder weniger Anforderungen bei. Dies in Kombination mit der Beobachtung, dass Klassen zumeist einen Zweck oder im Fall von Querschnittsanliegen einige wenige Zwecke erfüllen, diese aber potenziell über mehrere Elemente anbieten, kann zur Aggregation genutzt werden. Um den vorherrschenden oder die vorherrschenden Aspekte des Quelltextartefakts zu identifizieren, kann ein Mehrheitsentscheid eingesetzt werden. Hierbei hat jedes repräsentierte Element der Klasse eine Stimme und stimmt mit dieser für seine ähnlichste Anforderung ab. Diejenige oder diejenigen Anforderung(en), welche die meisten Stimmen erhalten haben, sind potenzielle Kandidaten für eine TL.

Allerdings kann eine einzelne Stimme pro Element je nach Verteilung der Ähnlichkeiten bzw. Güte der Repräsentation ebenso zu semantisch falschen Stimmabgaben führen⁷. Möchte man trotzdem ausnutzen, dass gerade die Vielzahl von Verweisen auf dieselbe Anforderung ein Indikator für eine TL ist, kann anstatt einer Stimme pro Element auch eine Mehrzahl von Stimmen zugelassen werden. Beispielsweise könnten alle Ähnlichkeiten eines Quelltextelements zu einer Anforderung, die über einer gewissen Güte, also einem bestimmten Schwellenwert, liegen, zu einer Stimmabgabe führen. Hat ein Quelltextelement beispielsweise zu den Anforderungen *A* und *B* eine maximale Ähnlichkeit von 89 % bzw. 90 %, zu allen weiteren Anforderungen aber nur eine Ähnlichkeit von unter 60 %, wäre ein solches Vorgehen vorteilhaft. Würde in dieser Situation nur die ähnlichste Stimme zählen, würde das Element nur für Anforderung *B* abstimmen. Die Information, dass Anforderung *A* eventuell ebenso relevant sein könnte, geht verloren. Würde man allerdings alle Zusammenhänge, die z. B. über einem Schwellenwert von 80 % liegen, zulassen, würde das Quelltextelement sowohl für Anforderung *A* als auch *B* abstimmen. Gibt es zusätzlich noch weitere Elemente des Quellartefakts, welche ebenso Stimmen für *A* bzw. *B* abgeben, kann das Zulassen mehrerer Stimmen pro Element zu einem klareren Bild der mehrheitlichen Anforderungen und damit einer besseren Entscheidung bezüglich der möglichen Kandidaten für TL führen.

⁷ Ein ähnliches Ergebnis wie es bei einer Aggregation über die maximalen Ähnlichkeiten auftreten würde.

5. Feingranulare Wiederherstellung von Nachverfolgbarkeitsverbindungen

In practice, requirements-to-code traces occur in various granularities. For example, traces may link requirements to methods in the source code or they may link requirements to classes, packages, or components.

(Kuang et al. [Kua+15])

Die Analyse in den vorangegangenen Kapiteln hat verschiedene Lücken in bestehenden Ansätzen zur automatischen Nachverfolgbarkeit von Anforderungen und Quelltext aufgedeckt und Anforderungen sowie Ziele an ein verbessertes Verfahren gestellt. Basierend auf diesen Erkenntnissen und den getroffenen Annahmen möchte ich nun in diesem Kapitel den grundlegenden Ansatz dieser Arbeit präsentieren. Dieser Ansatz zur automatischen TLR, im Folgenden als *Fine-grained Traceability Link Recovery* (**FTLR**) bezeichnet, setzt auf feingranulare, worteinbettungsbasierte Zusammenhänge zwischen den Artefakten, um semantische Ähnlichkeit zur Identifikation von TLs zu nutzen. **FTLR** identifiziert mögliche TLs ohne Wissen über vorhandene bzw. initiale TLs und kann somit in einer breiten Masse an Projekten, auch nachträglich, eingesetzt werden. Somit widmet es sich der unüberwachten Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext (s. Ziel 1). Außerdem ist das Verfahren so entworfen, dass es in der Lage sein soll, mit uneingeschränkter natürlicher Sprache sowohl auf Anforderungs- als auch Quelltextseite umzugehen (s. Anforderung 1).

FTLR identifiziert TLs zwischen Anforderungen und Quelltextdateien mittels der drei in Abbildung 5.1 dargestellten Schritte:

- 1) Repräsentation der Artefaktelemente mittels Worteinbettungen
- 2) Bestimmung der Ähnlichkeit zwischen den Elementen
- 3) Aggregation der feingranularen Zusammenhänge zur Identifikation der TLs

Der Einsatz feingranularer, worteinbettungsbasierter Zusammenhänge basiert dabei auf der Beobachtung, dass Artefakte häufig mehrere Aspekte abdecken¹ und somit eine Repräsentation auf Artefaktebene zu grobgranular sein kann (s. Abschnitt 4.7). Die Hypothese hierbei ist, dass eine feingranulare Abbildung, auf Ebene der bedeutungstragenden Artefaktelemente, wie beispielsweise Sätze in Anforderungen oder Methoden in Klassen, es

¹ So enthalten beispielsweise Ablaufbeschreibungen eines Anwendungsfalls oftmals mehrere Schritte und damit einzelne Funktionalitäten des Systems oder es existieren Klassen, deren Methoden mehrere unterschiedliche Zwecke erfüllen.

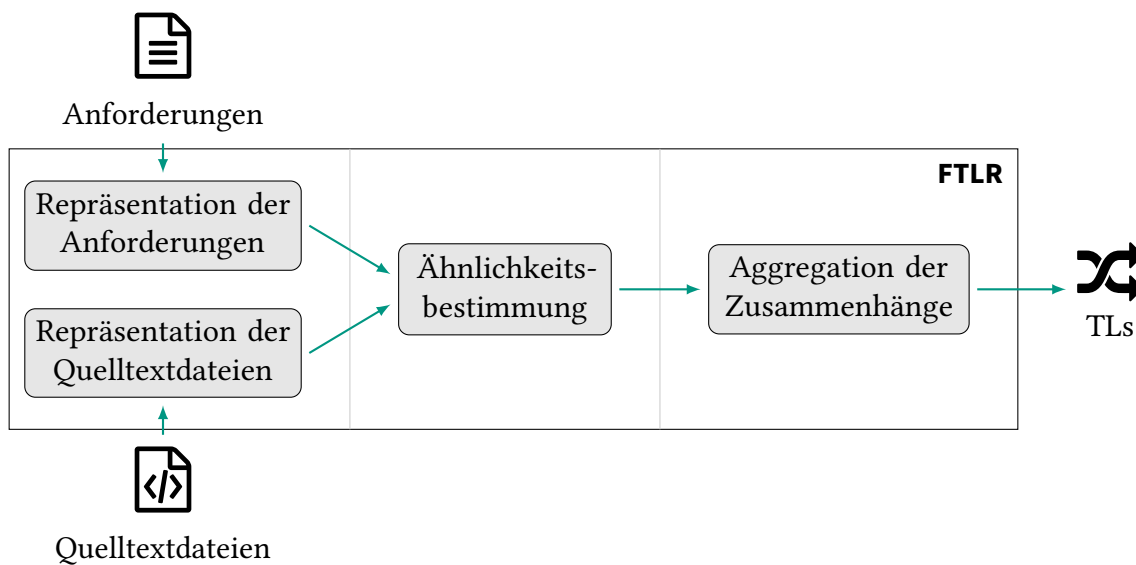


Abbildung 5.1.: Überblick über **FTLRs** Teilschritte zur automatischen Identifikation von Nachverfolgbarkeitsverbindungen (engl.: *trace links*, TLs)

ermöglicht, diese Aspekte besser zu unterscheiden. Der Einsatz von Worteinbettungen als Repräsentationsgrundlage zielt dabei darauf ab, **FTLR** in die Lage zu versetzen, einen semantischen Vergleich der Artefakte durchzuführen und somit sich der Überbrückung der semantischen Lücke anzunähern (s. Ziel 2 und Unterziel 2.1). Hierfür wird die Wortüberferungsdistanz (s. Abschnitt 2.10) erstmalig für den Ähnlichkeitsvergleich in einem Verfahren zur TLR eingesetzt. Abschließend aggregiert **FTLR** die feingranularen Ähnlichkeiten über einen Mehrheitsentscheid pro Artefakt. Dies dient dem Filtern der möglichen Kandidaten für TLs und der Identifikation der letztlich relevanten Zusammenhänge.

Teile der Beiträge dieses Kapitels wurden von mir bereits in vorangegangenen Arbeiten [Hey19; Hey+21] veröffentlicht. So habe ich die Motivation und Ziele eines semantikbasierten Ansatzes und die Idee einer unabhängigen Generierung einer Repräsentation der Semantik der beiden Artefakttypen bereits in einer frühen Vision dieser Arbeit dargelegt [Hey19]. Außerdem habe ich eine erste Version des in diesem Kapitel vorgestellten grundlegenden Ansatzes präsentiert und zu einem Vergleich mit bestehenden Verfahren genutzt [Hey+21].

In den folgenden Abschnitten werde ich zunächst auf die Repräsentation der Artefakte eingehen (s. Abschnitt 5.1), daraufhin die Ähnlichkeitsberechnung zwischen den Artefakten beschreiben (s. Abschnitt 5.2) und anschließend die Aggregation der feingranularen Zusammenhänge auf Artefaktebene erläutern (s. Abschnitt 5.3). In Abschnitt 5.4 werden daraufhin verschiedene Merkmalsvarianten von **FTLR** beschrieben. Abschließend werden in Kapitel 6 die verschiedenen Ausprägungen **FTLRs** evaluiert und der Einfluss der einzelnen Entwurfsentscheidungen bestimmt.

5.1. Repräsentation der Artefakte

Um sich dem Ziel des Schließens der semantischen Lücke zwischen den Artefakten anzunähern, sind in den letzten Jahren Verfahren sowohl in der TLR als auch in der Computerlinguistik allgemein von der Nutzung rein syntaktischer oder textueller Repräsentationen, wie z. B. BoW, für Ähnlichkeitsvergleiche abgekommen. Dies liegt vor allem an dem Erfolg der modernen Sprachmodelle (s. Abschnitt 2.9.3) in den letzten Jahren erzielt haben. Diese stellen aktuell den Stand der Technik für die Repräsentation der Semantik von natürlicher Sprache dar [Boj+17; Dev+19; Yan+19; Liu+19] und werden somit auch immer mehr für die Identifikation von TLs genutzt (vgl. [GCC17; ZCS17; Che+19; Zha+21; Lin+21; Lin+22]). All diesen Sprachmodellen liegt die Annahme zugrunde, dass die Bedeutung eines Wortes, Satzes oder Dokuments durch den Kontext bestimmt wird, in dem dieses steht. Im Fall der Sprachmodelle wird dieser Kontext auf den sprachlichen Kontext beschränkt und der situative oder persönliche/soziale Kontext ignoriert, da dieser nicht einfach aus den textuellen Daten erhoben werden kann. Auf Basis dieser Annahme gehen die Sprachmodelle davon aus, dass aus einer ausreichend großen Menge an Textdokumenten die semantischen Beziehungen der zu repräsentierenden Elemente gelernt werden können. Alle Sprachmodelle repräsentieren außerdem die Eingabe als Vektoren, welche der Annahme folgend, somit die Semantik der Eingabe repräsentieren. Sie zielen also auf eine semantikerhaltende Abbildung in den Vektorraum ab (s. Ziel 5).

Bestehende Verfahren zur Identifikation von TLs zwischen grob- und feingranularen Anforderungen oder Belangen und VCS-Einbuchungen setzen auf Varianten des kontextsensitiven Sprachmodells *BERT* (s. Abschnitte 3.2 und 3.4), wohingegen für Verbindungen zwischen Anforderung und Quelltext bisher auf statische Wort- bzw. Dokumenteinbettungen zurückgegriffen wurde (s. Abschnitt 3.1.3). Ein Grund hierfür könnte die Art, wie die natürlichsprachlichen Elemente im Quelltext vorliegen, sein. Die kontextsensitiven Sprachmodelle werden üblicherweise auf Sequenzen von Wörtern vortrainiert, welche zumeist aus natürlichsprachlichen Sätzen extrahiert werden. Da die Bezeichner im Quelltext allerdings, bis auf Quelltextkommentare, zumeist nicht den Reihenfolgen und Sequenzen eines natürlichsprachlichen Textes folgen, können die Repräsentationen dieser Art von Sprachmodellen Probleme bei der Repräsentation haben. Aus diesem Grund wurden in den letzten Jahren vermehrt bimodale, auf dualen Korpora aus natürlicher und Programmiersprache trainierte, Sprachmodelle entwickelt (s. Abschnitt 3.7). Diese werden gleichzeitig auf Methodendokumentation und -implementierung trainiert und beziehen häufig außerdem strukturelle Informationen, wie den AST oder Datenflussabhängigkeiten mit ein. Durch diese sehr implementierungsnahe Trainingsgrundlage erzielen sie gerade in quelltextbezogenen Aufgaben wie Klondetektion, Quelltextsuche oder Quelltextvervollständigung gute Ergebnisse. Bisher wurden diese bimodalen Sprachmodelle allerdings noch nicht für die TLR zwischen Anforderungen und Quelltext eingesetzt. Ein Grund hierfür könnte die Implementierungsnahe der Trainingsgrundlage sein. Sie könnte dazu führen, dass die sich auf einem höheren Abstraktionsniveau befindlichen natürlichsprachlichen Anforderungen schlechter abgebildet werden. Zusätzlich trainieren auch diese bimodalen Sprachmodelle weiterhin auf Sequenzen von Token und könnten dadurch Probleme mit syntaktisch unzusammenhängenden Mengen von Wörtern haben, sofern sie diese im Training nicht

derart gesehen haben. Statische Einbettungen hingegen repräsentieren einzelne Wörter immer gleich und können somit auch auf syntaktisch unzusammenhängende Mengen von Wörtern angewandt werden, wie es beispielsweise bei den Bezeichnern einer Methode der Fall sein kann.

Aus diesem Grund setzt **FTLR** auf statische Worteinbettungen. Im Gegensatz zu einigen bisherigen Verfahren zur TLR zwischen Anforderungen und Quelltext [ZCS17; Che+19; Zha+21] setzt **FTLR** allerdings nicht ein auf den zu repräsentierenden Artefakten oder rein softwarespezifischen Texten selbst trainiertes Modell ein. **FTLR** macht sich ein vortrainiertes Modell zunutze. Der Gedankengang hinter dieser Entscheidung ist der, dass die Größe und Vielfalt der gesehenen Beispiele sehr viel entscheidender für die Güte der Repräsentationsfähigkeit des Modells ist, als die domänenspezifischen Zusammenhänge (s. Annahme 3). Idealerweise würde das Sprachmodell auf einem sehr großen Korpus von Texten aus der Domäne vortrainiert, um die Zusammenhänge der Wörter in ihrer jeweiligen Bedeutung in der Domäne zu erfassen. Da ein solcher Korpus allerdings nicht existiert setze ich auf vortrainierte *fastText*-Worteinbettungsmodelle [Gra+18], die auf 24 Terabyte Textdaten trainiert wurden. Diese Modelle wurden auf der Wikipedia und dem CommonCrawl-Datensatz, welcher aus verschiedensten Texten des weltweiten Internets (engl.: *world wide web*, WWW) besteht, trainiert. Es handelt sich also nicht um ein domänenspezifisches Modell, welches somit viele Wörter und Zusammenhänge abdeckt und auf neue Domänen angewandt werden kann. Allerdings wird dieses Modell spezielle Zusammenhänge aus spezifischen Kontexten weniger stark gewichten, da statische Worteinbettungen keine Unterscheidung zwischen den verschiedenen Bedeutungen der Wörter durchführen.

Die Wahl für *fastText* als Worteinbettungsmodell bietet außerdem den Vorteil der Repräsentation von Zeichen-N-Grammen (s. Abschnitt 2.9.2) gegenüber den bisher oftmals verwendeten *word2vec*-Einbettungen. *fastText* ermöglicht es damit auch für Wörter, die das Modell nicht während des Trainings gesehen hat, Vektoren zu bilden, indem ihre Subwortrepräsentationen kombiniert werden. Dies ist insbesondere für den Einsatz auf Quelltextartefakten und den dort häufig auftretenden Abkürzungen und Auslassungen [New+19] sowie für domänenspezifische Terminologie relevant. Somit können diese differenzierter repräsentiert und nicht durch den Nullvektor repräsentiert werden, welcher von anderen Einbuchungsverfahren zur Repräsentation von unbekanntem Wörtern genutzt wurde. Einen weiteren Vorteil dieser Wahl des Sprachmodells ist die Verfügbarkeit von Modellen für verschiedene Sprachen, sowie sprachübergreifende Modelle. Dies erlaubt es **FTLR** auf Projekte in allen Sprachen anzuwenden, für die es ein vortrainiertes Modell gibt (aktuell sind dies 157), ohne das Verfahren zu ändern oder den Korpus für ein Selbsttrainieren des Modells auf nur wenige Projekte einschränken zu müssen.

Durch die Wahl eines Worteinbettungsmodells als Grundlage für die semantische Repräsentation der Artefakte arbeitet **FTLR** auf Wort- bzw. Tokenebene (s. Abschnitt 2.8.1), denn diese lassen sich durch das Modell auf einen Vektor abbilden. Einzelne Wortvektoren liefern aber alleinstehend noch keinen Aufschluss über die Semantik des Artefakts. Es muss also die Semantik des Artefakts aus den einzelnen Vektoren der Wörter extrahiert werden. Eine Möglichkeit wäre es, die Wortvektoren eines Artefakts zu mitteln oder einen

Vektor aus der Menge der Vektoren des Artefakts als Repräsentanten zu bestimmen. Ein solches Vorgehen würde den darauf folgenden Ähnlichkeitsvergleich deutlich erleichtern, da nun nur noch einzelne Vektoren miteinander verglichen werden müssen. Hierbei können allerdings Informationen verloren gehen. Ein Mitteln beispielsweise kann dazu führen, dass eine Repräsentation entsteht, die irgendwo zwischen den eigentlichen Aspekten der Eingabe liegt und potenziell etwas ganz anderes bedeutet. Die Wahl eines Repräsentanten hingegen würde die Semantik des gesamten Artefakts auf die Semantik eines einzelnen Wortes reduzieren. Da statische Wortembeddings ein Wort, unabhängig vom Kontext, immer gleich repräsentieren, kann dies aber dazu führen, dass die eigentliche Semantik des Artefakts gar nicht erfasst wird (s. Abschnitt 4.5).

Möchte man die Wortinformationen des zu repräsentierenden Elements weitestgehend erhalten, kann stattdessen auch eine Repräsentation mittels Einbettungs-Multimengen (engl.: *bags-of-embeddings*, BoEs) genutzt werden. Analog zur Wort-Multimenge (engl.: *bag-of-words*, BoW) fassen BoEs das zu repräsentierende Element als die Menge seiner Wortvektoren auf. Besteht also beispielsweise ein Artefakt aus den Wörtern *calculate*, *scalar* und *product* und sind $\vec{v}_{calculate}$, \vec{v}_{scalar} und $\vec{v}_{product}$ ihre entsprechenden Wortembeddingsvektoren so besteht das BoE dieses Artefakts aus der Menge $\{\vec{v}_{calculate}, \vec{v}_{scalar}, \vec{v}_{product}\}$. BoEs erhalten also die Beiträge aller enthaltenen Wörter zur Semantik, verlagern aber die Entscheidung, was die geteilte Semantik des Artefakts ist, an die eigentliche Ähnlichkeitsberechnung.

Aufgrund der informationserhaltenden Eigenschaften von BoEs nutzt **FTLR** diese zur Repräsentation der Artefakte. Allerdings repräsentiert **FTLR** die Artefakte, also eine Quelltextdatei oder eine Anforderung, nicht jeweils als ein einziges BoE, sondern spaltet die Artefakte in ihre Elemente auf. Diese Entscheidung basiert auf der in Abschnitt 4.7 gemachten Beobachtung, dass bisherige Repräsentationen, welche größtenteils auf Artefaktebene arbeiten zu grobgranular für eine präzise Identifikation der Zusammenhänge zwischen den Artefakten sind. Ich bezeichne dieses Vorgehen als feingranular und spreche im Weiteren von feingranularen Zusammenhängen, sofern Zusammenhänge zwischen Artefaktelementen anstatt ganzer Artefakte gemeint sind.

Definition 10: Zusammenhänge

Grobgranulare Zusammenhänge sind Beziehungen zwischen ganzen Artefakten.

Feingranulare Zusammenhänge sind Beziehungen zwischen Artefaktelementen oder Artefaktelementen und ganzen Artefakten.

Artefaktelemente definiere ich hierbei wie folgt:

Definition 11: Artefaktelemente

Ein Artefaktelement ist ein bedeutungstragender Teil eines nachverfolgbaren Artefakts.

Im Falle einer Anforderung ist jeder Satz ein Artefaktelement. Im Falle von Quelltextartefakten werden Methoden als Artefaktelemente aufgefasst.

Der Gedankengang hinter dieser Wahl ist der, dass (gut geschriebene) Anforderungssätze wie auch Methodennamen und -beschreibungen eine zusammenhängende Bedeutung beschreiben sollten. Theoretisch wären noch feingranularere Elemente für die Quelltextseite denkbar, wie z. B. die Inhalte des Methodenrumpfes, doch wie bereits in Abschnitt 4.4 diskutiert, enthalten diese zumeist hauptsächlich implementierungsnahe Informationen. Außerdem zeigten erste Experimente meinerseits mit Nachverfolgbarkeitsdatensätzen, dass diese mehr Rauschen als nützliche Informationen beitragen.

Somit besteht der erste Schritt der Repräsentation der Artefakte in einem Aufspalten in ihre entsprechenden Artefaktelemente. In den Abschnitten 4.3 und 4.4 wurde dargelegt, dass nicht alle Wörter eines Artefaktelements einen gleichen Beitrag zur Semantik des Elements leisten. Gerade, wenn die Semantik durch Worteinbettungen repräsentiert wird, können diese wenig bedeutungstragenden Einheiten die Präzision der Repräsentation vermindern. Aber auch in den bisher eingesetzten IR-Verfahren spielt die Auswahl der zu repräsentierenden Wörter oftmals eine große Rolle für den Erfolg des Verfahrens. In der Computerlinguistik und vor allem in der Verarbeitung natürlicher Sprache wird dieses Problem oftmals durch eine Vorverarbeitung der Eingaben adressiert. Auch **FTLR** nutzt eine für die verschiedenen Artefakttypen angepasste Vorverarbeitung, welche der eigentlichen Repräsentation durch Worteinbettungen vorangestellt wird.

Es ergibt sich also für den Repräsentationsschritt von **FTLR** der in Abbildung 5.2 dargestellte Ablauf. In den nächsten beiden Abschnitten werde ich im Detail auf die konkreten Schritte für die jeweiligen Artefakttypen eingehen und mögliche Erweiterungen und Varianten aufzeigen.

5.1.1. Anforderungsrepräsentation

Auf der Seite der Anforderungen arbeitet **FTLR** auf Satzebene als Anforderungselemente. Somit besteht der Schritt des Aufspaltens der Anforderungen in ihre Elemente aus einer Bestimmung der in einer Anforderung enthaltenen Sätze. Eine Anforderung ist hierbei definiert als eine Textdatei, deren Inhalt aus einer einzelnen natürlichsprachlichen Anforderung besteht, also z. B. eine einzelne Anwendungsfallbeschreibung oder Anwender:innenerzählung. Um diese Sätze zu bestimmen, setzt **FTLR** Standardverfahren zur Portionierung und Satzerkennung in natürlicher Sprache ein (s. Abschnitte 2.8.1 und 2.8.2). Konkret setzt **FTLR** auf den Portionierer und Satzerkenner des *Natural Language*

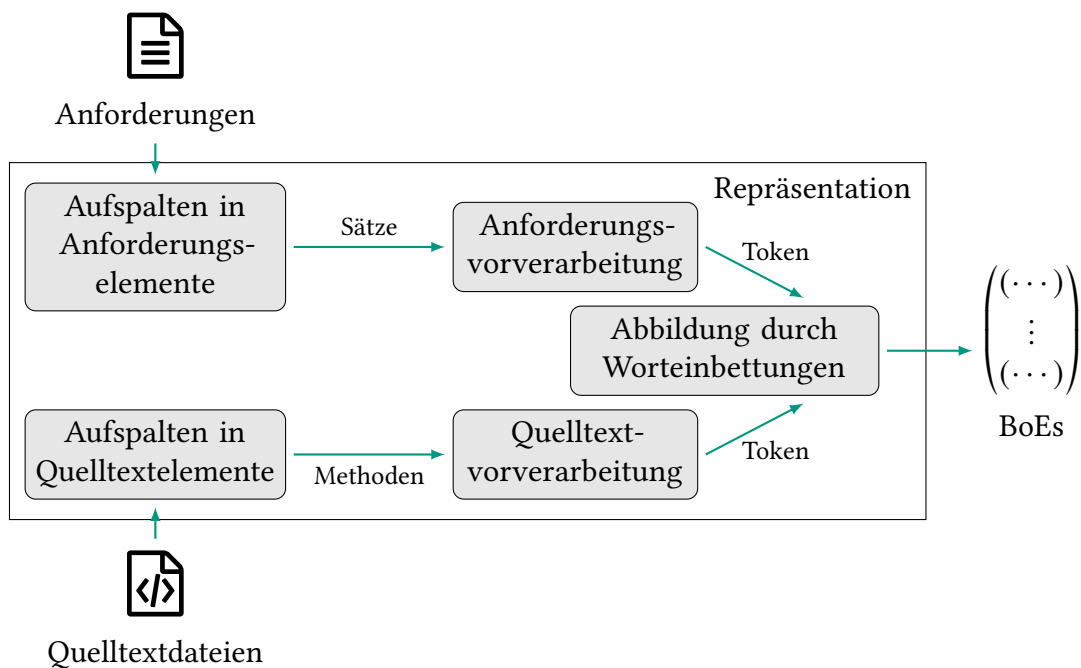


Abbildung 5.2.: Überblick der Teilschritte zur Repräsentation der Artefaktelemente in **FTLR** durch Einbettungs-Multimengen (engl.: *bags-of-embeddings*, BoEs)

Toolkits (NLTKs)². Diese liefern für ein Textdokument die enthaltenen Sätze, oder durch Zeilenumbrüche getrennte Texteinheiten, und die in ihnen enthaltenen Token.

Zusätzlich kann **FTLR** so konfiguriert werden, dass es vordefinierte Anwendungsfallvorlagen erkennt und diese für die Satzerkennung nutzt. Werden die Beschriftungen einer solchen Schablone, wie beispielsweise *Description* oder *Flow of Events*, erkannt, so werden die darauf folgenden Textabschnitte als zu diesem Element der Vorlage zugehörig identifiziert, ohne ihre Beschriftungen als Teil des Anforderungselements zu betrachten. Da die Ablaufbeschreibung zumeist aus einer Aufzählung von Schritten und somit mehreren Sätzen besteht, wird diese zusätzlich weiter mittels des Satzerkenners unterteilt. Somit würde die Anwendungsfallbeschreibung aus Abbildung 4.2 in den in Abbildung 5.3 dargestellten Anforderungselementen resultieren. Für die Ablaufbeschreibung werden vier Elemente erstellt, wohingegen die beiden Sätze der Nachbedingung zu einem Element zusammengefasst werden. Eine normale Anforderung ohne Schablone, wie beispielsweise „*The system shall return the time and date. The date is formatted as YYYY-MM-DD.*“, würde hingegen in der in Abbildung 5.4 dargestellten Anforderungselementmenge resultieren.

Die Sätze bzw. Anforderungselemente werden daraufhin einer Vorverarbeitung unterzogen. Hierbei werden zunächst nicht textuelle Elemente, wie Querverweise, Sonderzeichen und Zahlen durch die Anwendung regulärer Ausdrücke entfernt. Diese können durch das verwendete Word-Embeddingsmodell nicht sinnvoll auf eine semantische Repräsentation

² <https://www.nltk.org/>, zuletzt besucht am 26.04.2023.

element _{Name}	=	{ "AdvancedSearch" }
element _{Beschreibung}	=	{ "The", "tourist", "is", "searching", "for", "a", "site", "using", "the", "potential", "offered", "by", "the", "Advanced", "Search", "." }
element _{Akteur}	=	{ "Initialized", "by", "Tourist" }
element _{Vorbedingungen}	=	{ "The", "Tourist", "has", "successfully", "authenticated", "to", "the", "system", "." }
element _{Ablauf1}	=	{ "1.", "Enable", "the", "advanced", "search", "feature", "from", "your", "personal", "area", "." }
element _{Ablauf2}	=	{ "2.", "View", "the", "advanced", "search", "form", "." }
element _{Ablauf3}	=	{ "3.", "Fill", "in", "the", "form", "of", "advanced", "search", "and", "submit", "." }
element _{Ablauf4}	=	{ "4.", "Gets", "the", "position", "of", "relying", "on", "the", "tourist", "event", "of", "the", "use", "location", "and", "process", "the", "request", "." }
element _{Nachbedingungen}	=	{ "The", "system", "displays", "a", "list", "of", "results", ".", "Interruption", "of", "the", "connection", "to", "the", "server", "ETOUR", "." }
element _{Qualität}	=	{ "The", "system", "requires", "the", "transaction", "in", "more", "than", "15", "seconds", "." }

Abbildung 5.3.: Portionierung einer Anwendungsfallbeschreibung mit Schablone aus dem eTour-Projekt [CoE] in Anforderungselemente

„The system shall return the time and date. The date is formatted as YYYY-MM-DD.“



element _{satz1}	=	{ "The", "system", "shall", "return", "the", "time", "and", "date", "." }
element _{satz2}	=	{ "The", "date", "is", "formatted", "as", "YYYY-MM-DD", "." }

Abbildung 5.4.: Portionierung einer Anforderung in Anforderungselemente

abgebildet werden und stellen somit eine Quelle für Rauschen dar. Daraufhin werden Token in Binnenmajuskelschreibweise in einzelne Token aufgespalten und alle Token in Kleinbuchstaben umgesetzt. Das Aufspalten der Binnenmajuskelschreibweise sorgt dafür, dass derart zusammengesetzte Wörter in ihre einzelnen Token aufgespalten werden und somit ihre Semantik durch das Worteinbettungsmodell repräsentiert werden kann. Die Umwandlung in Kleinschrift führt dazu, dass alle Ausprägungen eines Wortes, egal ob diese am Anfang oder mitten im Text stehen, zu den gleichen Token und somit der gleichen Repräsentation durch das Worteinbettungsmodell führen. Anschließend werden die Grundformen der Token mittels einer Lemmatisierung (s. Abschnitt 2.8.4) bestimmt und eine Stoppwortentfernung (s. Abschnitt 2.8.5) durchgeführt. Für die Lemmatisierung setzt **FTLR** auf *spaCy*³ und nutzt für die Stoppwortentfernung die Stoppwortlisten des

³ <https://www.spacy.io/>, zuletzt besucht am 26.04.2023.

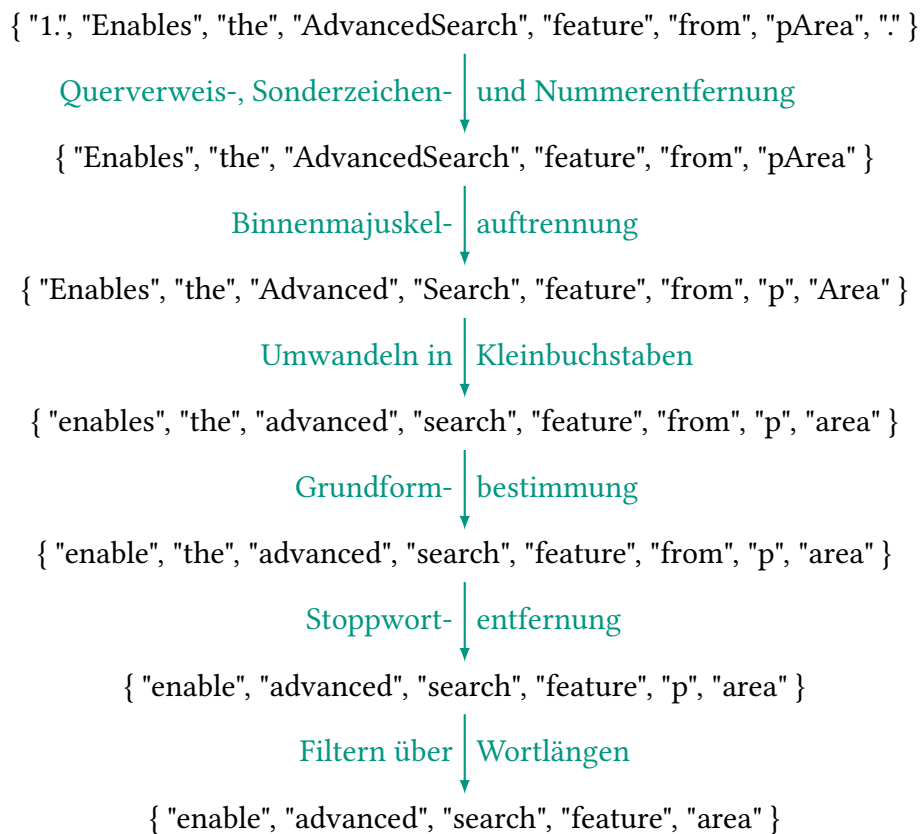


Abbildung 5.5.: Illustration der Vorverarbeitungsschritte für Anforderungselemente anhand eines Beispiels

NLTKs (vgl. Anhang B.1). Die Lemmatisierung führt zu einer weiteren Vereinheitlichung der Token, sodass Beugung und Konjugationen keine Rolle mehr bei der Repräsentation spielen. Die Stoppwortentfernung hingegen ist ein weiterer Schritt, um das Rauschen in der Repräsentation zu verringern. Hier werden Wörter, die wenig zur Semantik der Eingabe beitragen (z. B. Artikel oder Präpositionen), entfernt. Abschließend wird ein Wortlängenfilter eingesetzt, um Token, die aus weniger als drei Zeichen bestehen, zu entfernen. Der Gedankengang hinter diesem Wortlängenfilter ist, dass Token, die nach der Stoppwortentfernung übrig bleiben und aus weniger als drei Zeichen bestehen, oftmals Abkürzungen darstellen, welche schwer zu repräsentieren und interpretieren sind.

Wendet man diese Vorverarbeitungsschritte auf das bereits portionierte Anforderungselement „1. Enables the AdvancedSearch feature from pArea.“ an, so entsteht der in Abbildung 5.5 dargestellte Ablauf der Verarbeitung, welcher letztlich zu der Tokenmenge { "enable", "advanced", "search", "feature", "area" } als Repräsentationsgrundlage führt. Im ersten Schritt wird die Aufzählung 1. und das Satzzeichen . entfernt. Anschließend spaltet die Binnenmajuskelauftrennung das Token *AdvancedSearch* in *Advanced* und *Search*, sowie das Token *pArea* in *p* und *Area* auf. Daraufhin werden alle Token in Kleinbuchstaben umgewandelt, was dazu führt, dass aus *Enables*, *Advanced*, *Search* und *Area* die Token *enables*, *advanced*, *search* und *area* werden. Die darauf folgende Grundformbestimmung reduziert

enables auf seine Grundform *enable* und im nächsten Schritt werden die Stoppwörter *the* und *from* entfernt. Abschließend entfernt der Wortlängenfilter das aus einem Zeichen bestehende Token *p*.

Die nach der Vorverarbeitung pro Anforderungselement übrig gebliebene Tokenmenge wird daraufhin von **FTLR** mittels des Worteinbettungsmodells in eine Einbettungs-Multimenge umgewandelt. Es wird also jedes Token durch seinen Worteinbettungsvektor ersetzt und die Menge dieser Vektoren bildet das BoE, welches das Anforderungselement repräsentiert. Zusätzlich merkt sich **FTLR** die Zuordnung der Anforderungselemente zu der sie enthaltenden Anforderung, um später eine Aggregation auf Artefaktebene zu ermöglichen.

5.1.2. Quelltextrepräsentation

Auf der Quelltextseite arbeitet **FTLR** auf Methodenebene als kleinste zu verarbeitende Einheit. Um die Quelltextdateien in ihre Artefaktelemente aufzuspalten, erstellt **FTLR** zunächst den AST der Eingabe (einschließlich Kommentare). Hierdurch lassen sich die Zugehörigkeiten der verschiedenen Quelltextelemente zu ihren umschließenden Umgebungen sowie die zu ihnen zugehörigen Kommentare einfach extrahieren. Aufgrund der in Abschnitt 4.4 analysierten Zusammenhänge zwischen Attributen und ihren *Getter/Setter*-Methoden verzichtet **FTLR** auf eine Repräsentation der Attribute als Artefaktelemente, sondern betrachtet ausschließlich Methoden. Ebenso basierend auf der in Abschnitt 4.4 geführten Analyse und der daraus gefolgerten Annahme 2, dass der Zweck einer Klasse durch ihre öffentlich sichtbaren Elemente beschrieben werden sollte, betrachtet **FTLR** nur die öffentlich sichtbaren Methoden. Dies kann dazu führen, dass für manche Klassen keine Artefaktelemente existieren. In diesen Fällen nutzt **FTLR** den Klassennamen als Artefaktelement anstatt der Methoden.

FTLR repräsentiert eine Methode durch ihre natürlichsprachlichen Bezeichner. Dies sind zunächst der Methodename selbst, die Namen der Parameter und die Bezeichner der Typen der Parameter. Zusätzlich zu diesen methodenspezifischen Bezeichnern wird außerdem der Klassename der umschließenden Klasse zur Repräsentation jeder Methode hinzugefügt. Hierdurch kann **FTLR** Methoden mit dem gleichen Namen in verschiedenen Klassen unterscheiden. So können z. B. zwei Klassen `ShoppingCart` und `Stock` mit einer Methode `add(item:Item)` durch Hinzufügen des Klassennamens einfacher semantisch unterschieden werden. Außerdem spielt der Klassename als kontextgebende Einheit der Methoden generell eine große Rolle bei der Interpretation ihrer Semantik (s. Abschnitt 4.4). Somit wird eine Quelltextdatei in ihre öffentlich sichtbaren Methoden zerlegt und diese jeweils durch die folgende Menge an Bezeichnern repräsentiert:

```
{ Methodename, Parametername1, Parametertyp1, ...,  
  ParameternameN, ParametertypN, Rückgabotyp, Klassename }
```

Ebenso wie auf der Anforderungsseite werden auch die Methoden bzw. Quelltextelemente einer Vorverarbeitung unterzogen. Diese umfasst dieselben Schritte wie auf der Anforderungsseite mit einer Änderung. Zusätzlich zu den Stoppwortlisten des NLTKs wird eine



Abbildung 5.6.: Illustration der Vorverarbeitungsschritte für Quelltextelemente anhand eines Beispiels

weitere Stoppwortentfernung von programmiersprachenspezifischen Stoppwörtern und häufigen Begriffen in der Programmierung von Software (vgl. Anhang B.2) durchgeführt. Hierbei werden Begriffe wie *get*, *set*, *array* oder *servlet* entfernt. Der Gedankengang hier ist, dass diese im Fall von *get* und *set* Standardpräfixe für Attributzugriffe darstellen und somit in vielen Methoden vorkommen, ohne diese wirklich zu differenzieren. Andere Begriffe wie *array*, *servlet* oder *bean* stellen hingegen Implementierungsdetails dar, die oftmals nicht in den Anforderungen beschrieben werden und somit für die Abbildung Rauschen darstellen. Bei der Auswahl dieser Begriffe wurde konservativ vorgegangen, um nicht zu viele möglicherweise doch hilfreiche Begriffe auszuschließen, weshalb nur neun solcher Begriffe zur Liste der programmiersprachenspezifischen Stoppwörter, den Sprachkonstrukten und Standarddatentypen, hinzugefügt wurden. Abbildung 5.6 zeigt den Ablauf der Extraktion einer Methode aus einem Quelltextartefakt mit anschließender Vorverarbeitung. Zunächst werden der Methodename `retrieveParticipants`, der Parametername `year`, der Paramatertyp `int` und der Rückgabertyp `String[]` extrahiert und um den Klassennamen `Courses` ergänzt. Durch die Sonderzeichenentfernung werden nur die Klammern der Arrayschreibweise der Java-Syntax entfernt. Daraufhin wird der

Methodenname durch die Auftrennung der Binnenmajuskelschreibweise in seine Bestandteile `retrieve` und `Participants` zerlegt. Durch die Kleinschreibung und anschließende Grundformbestimmung werden anschließend aus `Participants` das Token `participant` und aus `Courses` das Token `course`. Die linguistische Stoppwortentfernung entfernt in diesem Beispiel keine Wörter, aber die quelltextspezifische Stoppwortentfernung entfernt die beiden Token `int` und `string`, weil sie primitive bzw. Standarddatentypen in Java darstellen. Der Schritt des Wortlängenfilters ändert in diesem Beispiel nichts, weshalb die abschließende Tokenmenge zur Repräsentation dieses Quelltextelements { "retrieve", "participant", "year", "course" } ist.

Ebenso wie auf Anforderungsseite werden auch die für die Quelltextelemente nach der Vorverarbeitung übrig gebliebenen Token mittels des Worteinbettungsmodells in eine Einbettungs-Multimenge umgewandelt. Genauso wird auch hier die Zuordnung des Quelltextelements zu seinem umschließenden Quelltextartefakt gespeichert, um die spätere Aggregation zu ermöglichen.

5.2. Ähnlichkeitsbestimmung

Nachdem die Repräsentationen der Artefaktelemente bestimmt wurden, kann der eigentliche Ähnlichkeitsvergleich durchgeführt werden. Da **FTLR** dasselbe Worteinbettungsmodell sowohl für die Repräsentation der Anforderungselemente, als auch der Quelltextelemente nutzt, können die Eigenschaften des gemeinsamen Vektorraumes ausgenutzt werden. Wörter die häufig in denselben Kontexten auftreten, haben aufgrund des Trainings des Worteinbettungsmodells auch Worteinbettungsvektoren, die im Vektorraum nahe beieinander liegen⁴ (s. Abschnitt 2.9.2).

In Abschnitt 4.6 habe ich verschiedene Möglichkeiten eines semantischen Ähnlichkeitsvergleichs auf Basis einer Vektorrepräsentation analysiert. Bestehende Arbeiten [HDS06; Loh+13; ZCS17] setzen die Kosinusähnlichkeit für diesen Zweck ein. Um diese auf BoE anwenden zu können, müssten die einzelnen Worteinbettungen aber zunächst zu einem Vektor zusammengefasst werden oder die Ähnlichkeitsbestimmung wortweise durchgeführt werden und daraufhin die maximale, durchschnittliche oder minimale Ähnlichkeit als Repräsentant für die Ähnlichkeit der beiden Artefaktelemente herangezogen werden. Ersteres setzt voraus, dass der gemittelte Vektor aller Worteinbettungen eines Artefaktelements einen Vektor darstellt, der repräsentativ für die Semantik ist. Sollten mehrere Aspekte oder Zwecke in einem Artefaktelement vorhanden sein, kann ein solches Aggregieren allerdings auch in einem Vektor resultieren, der irgendwo zwischen diesen Aspekten liegt und eventuell sogar der Semantik eines ganz anderen Aspektes entspricht. Letzteres Vorgehen ignoriert die geteilte Semantik mehrerer Wörter, was ebenso zu einem Verlust an semantischer Aussagekraft führen kann. Aus diesem Grund nutzt **FTLR** die Wortüberführungsdistanz (engl.: *Word Movers Distance*, WMD) [Kus+15] (s. Abschnitt 2.10) als Ähnlichkeitsmaß für die feingranularen Zusammenhänge.

⁴ bezüglich ihres Kosinus- bzw. euklidischen Abstands

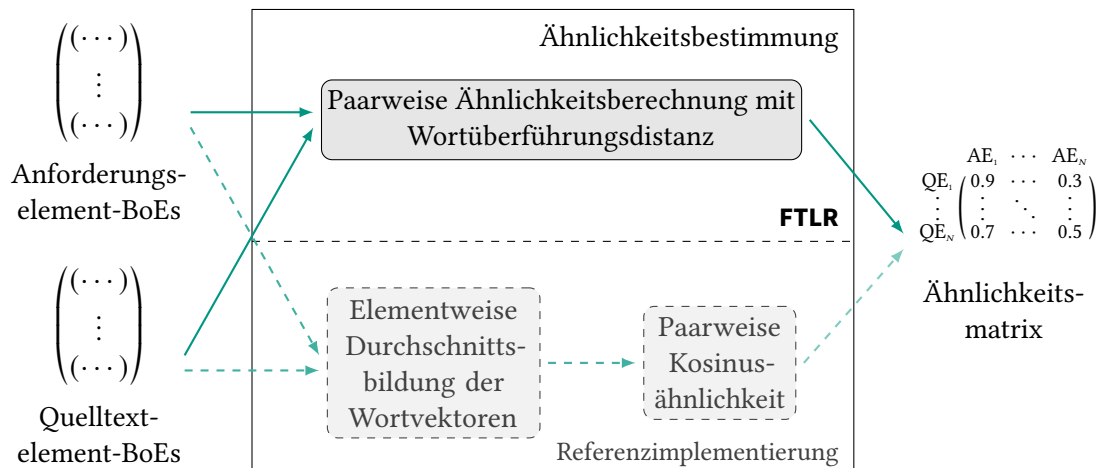


Abbildung 5.7.: Gegenüberstellung der Teilschritte **FTLRs** zur Ähnlichkeitsbestimmung, zu denen der Referenzimplementierung mittels Kosinusähnlichkeit

Die WMD bestimmt die Ähnlichkeit zweier BoEs, indem sie die minimale kumulative Distanz berechnet, die benötigt wird, um jede Einbettung des einen BoE auf seine nächste Einbettung im anderen BoE zu überführen. Die Ähnlichkeit bestimmt sich also durch die Überführungskosten, weshalb zwei BoEs sich ähnlicher sind, sofern ihre WMD geringer ist. Durch dieses Vorgehen werden keine Zusammenhänge zwischen den Wörtern durch eine Aggregation oder die Auswahl eines Repräsentanten verworfen oder verfälscht. Die möglichen Werte für die WMD zweier BoEs mit normalisierten Worteinbettungsvektoren, wie den *fastText*-Worteinbettungen, liegen im Bereich $[0, 2]$. Um einer Ähnlichkeitsfunktion zu entsprechen und allgemeingültigere Schwellenwerte definieren zu können, projiziert **FTLR** diese in den Bereich $[0, 1]$.

Um den Effekt des Einsatzes der WMD gegenüber einer Aggregation mit anschließendem Einsatz der Kosinusähnlichkeit bemessen zu können, verfügt **FTLR** auch über eine Referenzimplementierung eines solchen Vorgehens. Hierbei werden die einzelnen Worteinbettungen eines BoEs durch eine Durchschnittsbildung zu einem Vektor zusammengefasst und anschließend die Ähnlichkeit eines Paares aus einem Anforderungs- und einem Quelltextelement über die Kosinusähnlichkeit ihrer aggregierten Vektoren bestimmt. In diesem Fall zeugt ein höherer Wert von einer höheren Ähnlichkeit.

In beiden Varianten erzeugt **FTLR** eine Ähnlichkeitsmatrix, welche die feingranularen Ähnlichkeiten zwischen den Paaren von Anforderungs- und Quelltextelementen enthält. Dieses Vorgehen ist in Abbildung 5.7 dargestellt und entspricht dem zweiten Schritt in Abbildung 5.1.

Artefaktebene Um auch den Effekt der feingranularen Abbildung bemessen zu können, kann **FTLR** auch so konfiguriert werden, dass es die BoEs eines Artefakts aggregiert. Hierbei wird ein neues BoE aus den Einträgen aller BoEs des Artefakts gebildet und die Ähnlichkeitsberechnung, wie für die Artefaktelemente, auf diesen Artefakt-BoEs

Tabelle 5.1.: Übersicht der konfigurierbaren Ähnlichkeitsbestimmungsverfahren

	Elementebene	Artefaktebene
WMD	<i>ElementWMD</i> (FTLR)	<i>ArtefaktWMD</i> (AWMD)
Kosinusähnlichkeit	<i>ElementCosSim</i> (ECosS)	<i>ArtefaktCosSim</i> (ACosS)

durchgeführt. Dies führt zu den in Tabelle 5.1 dargestellten vier möglichen Konfigurationen für die Ähnlichkeitsbestimmung, wobei im Weiteren, wenn von **FTLR** die Rede ist, die Variante der WMD auf Elementebene gemeint ist.

5.3. Aggregation

Auf Basis der feingranularen Zusammenhänge zwischen den Artefaktelementen bildet **FTLR** nun im letzten Schritt Kandidaten für mögliche TLs. Da nicht alle Artefaktelemente eine Entsprechung im jeweils anderen Artefakttyp haben müssen⁵, ist die Herausforderung dieses Schrittes, diejenigen feingranularen Zusammenhänge zu identifizieren, die am meisten zur Gesamtbeziehung zwischen den Artefakten beitragen. Aus diesem Grund aggregiert **FTLR** die feingranularen Zusammenhänge zwischen den Artefaktelementen zu TLs zwischen Artefakten. Wie bereits in Abschnitt 4.7 diskutiert, können die Eigenschaften einer vertikalen Nachverfolgbarkeitsrelation zwischen Artefakten, die in einer Umsetzungs- bzw. Verfeinerungsrelation stehen, für eine solche Aggregation ausgenutzt werden. Auf Basis der Beobachtungen in Abschnitt 4.7 ergibt es Sinn, die Aggregation der feingranularen Zusammenhänge von der Quelltextseite anzugehen. Zum einen ist es zumeist so, dass die durchschnittliche Anzahl der Relationen, in denen ein Artefakt auf höherem Abstraktionsniveau zu Artefakten des niedrigeren Abstraktionsniveaus steht, deutlich höher als umgekehrt ist⁶. Zum anderen erfüllt eine Quelltextklasse zumeist einen oder manchmal einige wenige Zwecke und bietet diese über ihre verschiedenen Elemente an. Die Aggregation von der Quelltextseite anzugehen, ermöglicht es mittels eines Mehrheitsentscheides, die vorherrschenden erfüllten Zwecke über die vorherrschenden feingranularen Zusammenhänge eines Quelltextartefakts zu bestimmen. Hiermit lassen sich im Gegensatz zu Aggregationsmethoden, wie der Durchschnittsbildung oder einem einfachen Nutzen der maximalen Ähnlichkeit, mögliche falsch-positive Beziehungen filtern. Wichtig zu beachten ist aber, dass ein solches Vorgehen ebenso wie die anderen Aggregationsmethoden Schwierigkeiten mit Aspekten/Zwecken hat, die nur durch ein einzelnes Artefaktelement repräsentiert werden. Insbesondere wenn ein anderer Aspekt durch eine größere Menge von Artefaktelementen repräsentiert wird, kann auch dieses Vorgehen den Zusammenhang nicht identifizieren.

⁵ Die Relation muss weder links- noch rechtstotal sein.

⁶ Zumeist besitzt eine Anforderung TLs zu mehreren Quelltextartefakten, wohingegen ein Quelltextartefakt meist nur zur Umsetzung einer oder weniger Anforderungen beiträgt.

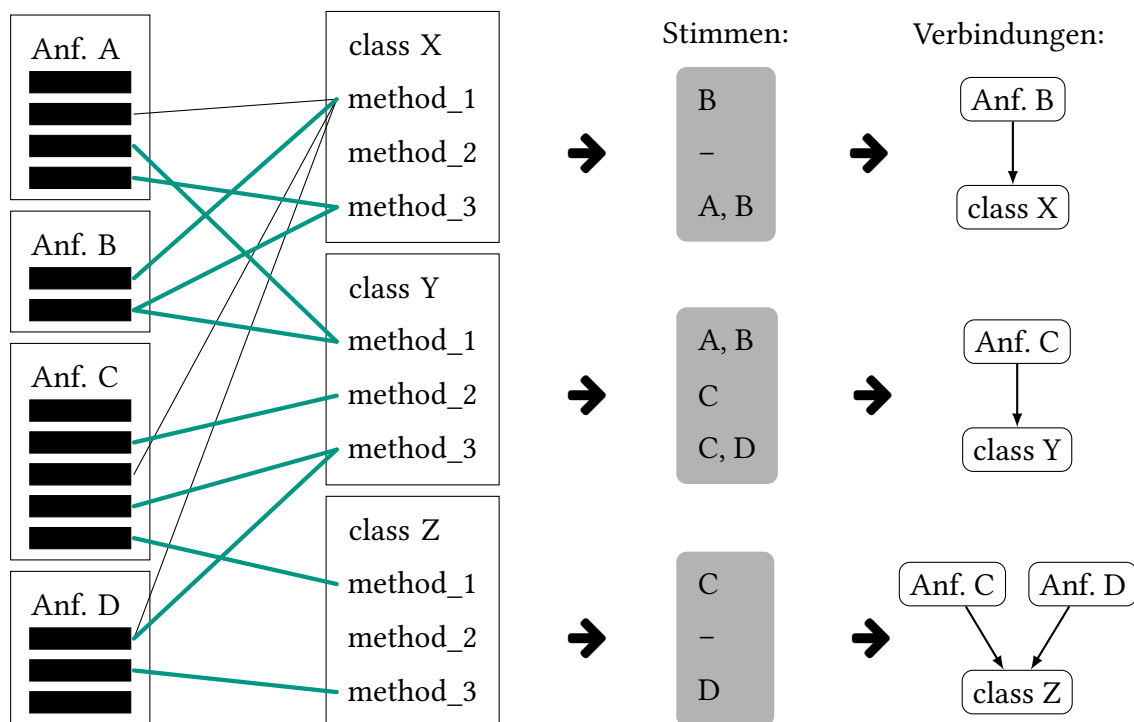


Abbildung 5.8.: Schematisches Beispiel des Aggregationsschrittes von **FTLR**, ausgehend von feingranularen Zusammenhängen zwischen Methoden und Anforderungselementen, über die Stimmabgaben jeder Methode bis hin zu den resultierenden Nachverfolgbarkeitsverbindungskandidaten. Breite Linien verdeutlichen Ähnlichkeiten über dem Mehrheitsschwellenwert. Weitere Ähnlichkeiten wurden aus Lesbarkeitsgründen weggelassen

FTLR setzt zur Aggregation mittels Mehrheitsentscheid das in Abbildung 5.8 schematisch dargestellte Vorgehen ein. Ausgehend von den feingranularen Ähnlichkeiten zwischen den Anforderungs- und Quelltextelementen berechnet **FTLR** zunächst für jedes Quelltextelement (qe) dasjenige Anforderungselement (ae) eines jeden Anforderungsartefakts (A), zu dem es am ähnlichsten ist (dargestellt als Linien auf der linken Seite von Abbildung 5.8)⁷.

$$\text{sim}(qe, A) = \min_{ae \in A} \text{WMD}(qe, ae) \quad (5.1)$$

Diese Relation dient als Repräsentant für die Ähnlichkeit zwischen dem feingranularen Quelltextelement und der grobgranularen, gesamten Anforderung. Somit wird der Zusammenhang zwischen einer Quelltextmethode und einer Anforderung durch ihre ähnlichste Beziehung definiert. Da das Vorhandensein eines solchen Zusammenhangs allerdings nicht notwendigerweise bedeutet, dass das Quelltextelement tatsächlich in einer für die

⁷ Jedes Quelltextelement hat eine solche Relation zu einem Anforderungsartefakt, allerdings zeigt Abbildung 5.8 aus Gründen der Lesbarkeit nur für das erste Quelltextelement (class A.method_1) alle diese Zusammenhänge.

Nachverfolgbarkeit relevanten Beziehung zu dieser Anforderung steht⁸, filtert **FTLR** diese Beziehungen über einen Schwellenwert. Dieser Schwellenwert, im Weiteren Mehrheitschwellenwert genannt, liegt bei einer Wortüberführungsdistanz von 0,59. Dieser wurde durch eine Vorstudie auf einer zufällig ausgewählten Untermenge, der durch das *Center of Excellence for Software & Systems Traceability* (CoEST) [CoE] frei zur Verfügung gestellten Nachverfolgbarkeitsdatensätze eTour, iTrust, SMOS und eAnci, bestimmt. Somit betrachtet **FTLR** nur diejenigen Beziehungen zwischen Quelltextelementen und Anforderungen weiter, die ein Wortüberführungsdistanz unterhalb von 0,59 haben, denn eine niedrigere Wortüberführungsdistanz bedeutet eine höhere Ähnlichkeit. In Abbildung 5.8 sind diese Beziehungen mit breiten Linien dargestellt.

Auf Basis der verbleibenden Beziehungen führt **FTLR** den Mehrheitsentscheid durch. Jedes Quelltextelement stimmt für diejenigen Anforderungen ab, zu denen es noch verbliebene Beziehungen hat. So stimmt beispielsweise `method_1` von `class X` in Abbildung 5.8 nur für Anforderung B ab, da ihre ähnlichsten Beziehungen zu den anderen Anforderungen nicht den Schwellenwert unterschreiten. Abschließend erstellt **FTLR** nur für diejenigen Anforderungen, welche die meisten Stimmen pro Quelltextartefakt bekommen haben, einen Nachverfolgbarkeitsverbindungskandidaten. Somit wird in den meisten Fällen nur eine TL pro Klasse identifiziert. Erhalten allerdings mehrere Anforderungen gleich viele Stimmen, wie es für `class Z` in Abbildung 5.8 der Fall ist, so werden TLs zu all diesen Anforderungen erstellt. Dieses Vorgehen zielt auf Präzision ab und kann dazu führen, dass für Klassen, die einen Aspekt umsetzen, der für viele Anforderungen relevant ist, TLs verpasst werden. Auf Basis der Beobachtungen in Abschnitt 4.7 und dem Problem bestehender Verfahren hinsichtlich Präzision ist das ein vertretbarer Nachteil.

Der Mehrheitsschwellenwert wurde bewusst relativ hoch gesetzt, damit eine größere Menge an Elementen an der Entscheidung teilnehmen kann. Der Gedankengang hier ist, dass mehr Stimmberechtigte bei der Abstimmung die übereinstimmenden Zusammenhänge noch klarer abgrenzen können. Dies bedeutet aber auch, dass in einem Fall, in dem für alle Elemente nur niedrige Ähnlichkeiten existieren, also eventuell keine TLs existieren, trotzdem eine Entscheidung und damit ein Nachverfolgbarkeitsverbindungskandidat erstellt würde. Um dies zu verhindern, setzt **FTLR** nach dem Mehrheitsentscheid noch einen zweiten Schwellenwert als Filter für die identifizierten Nachverfolgbarkeitsverbindungskandidaten ein. Die Idee dieses Filters ist, dass ein vorherrschender Zweck (und damit ein identifizierter Nachverfolgbarkeitsverbindungskandidat) mindestens eine Stimme eines Elements bekommen hat, das eine höhere Ähnlichkeit bzw. niedrigere Wortüberführungsdistanz zu der abgestimmten Anforderung hat als dieser Schwellenwert. Dieser Wert, im Weiteren als Abschlusschwellenwert bezeichnet, liegt bei einer Wortüberführungsdistanz von 0,44. Somit bestimmt **FTLR** für jeden identifizierten Nachverfolgbarkeitsverbindungskandidat dasjenige Element mit der höchsten Ähnlichkeit und entfernt diejenigen Kandidaten, welche eine Wortüberführungsdistanz oberhalb dieses Schwellenwertes haben, also eine niedrigere maximale Ähnlichkeit aufweisen. Es wäre ebenso denkbar, die durchschnittliche

⁸ Ist ein Quelltextelement zu allen Anforderungselementen semantisch unähnlich so ist auch die ähnlichste Beziehung immer noch unähnlich und damit irrelevant für die Nachverfolgbarkeit.

```

public class Courses {
    /**
     * Obtains the names of persons in this course
     * as an array
     * @param the year of the course
     * @return an array of names of persons in this course
     */
    public String[] retrieveParticipants(int year)

```

{ "retrieveParticipants", "year", "int", "String[]", "Courses", "Obtains", "the", "names",
"of", "persons", "in", "this", "course", "as", "an", "array" }

Vorver- | arbeitung

{ "retrieve", "participant", "year", "course", "obtain", "name", "person", "course" }

Abbildung 5.9.: Illustration der Extraktion eines Quelltextelements mit Kommentar anhand eines Beispiels

oder minimale Ähnlichkeit pro Nachverfolgbarkeitsverbindungskandidaten zu nutzen. Allerdings schnitt das Maximum in der Vorstudie am besten ab.

5.4. Varianten

In den vorangehenden Abschnitten wurde das grundlegende Verfahren hinter **FTLR** präsentiert. Das grundlegende Verfahren kann mit Hilfe von drei Erweiterungen variiert werden. Diese verwenden ggf. nicht verfügbare Informationen, weshalb sie optional aktiviert werden können. So verfügt beispielsweise nicht jedes Projekt über qualitativ hochwertige Methodenkommentare oder Anwendungsfallbeschreibung, die einer Schablone folgen. Die Erweiterungen können alleinstehend oder in Kombination neue Varianten von **FTLR** definieren.

5.4.1. Methodenkommentare


Wie bereits in Abschnitt 4.4 analysiert, können Methodenkommentare eine sehr hilfreiche Quelle für die Semantik der Funktionalität einer Methode darstellen. Da diese nicht immer vorhanden sind, von variierender Qualität sein können, sowie häufig Inkonsistenzen zur eigentlichen Semantik des Quelltextes aufweisen, werden Methodenkommentare als optionale Informationsquelle betrachtet. **FTLR** bindet Methodenkommentare in die Repräsentation einer Methode ein, indem die Token des Kommentars der Menge an Bezeichnern der Methode vor der Vorverarbeitung hinzugefügt werden. Liegen die Kommentare außerdem in einem vorgegebenen Format wie beispielsweise Javadoc für Java vor, so können sogar die einzelnen Felder des Kommentars extrahiert werden. In diesen Fällen nutzt **FTLR** nur den beschreibenden Teil des Kommentars. Parameter-, Rückgabe- und

Ausnahmenbeschreibungen werden ignoriert, da diese oftmals der bereits in der Beschreibung vorhandenen Information entsprechen oder Implementierungsdetails beschreiben. Außerdem entfernt **FTLR** spezielle Etiketten, wie die Javadoc-Etiketten (s. Anhang C).

Abbildung 5.9 zeigt den Ablauf der Extraktion der Methode aus Abbildung 5.6 mit vorhandenem Methodenkommentar und die resultierende Tokenmenge nach der Vorverarbeitung. Zusätzlich zu den Bezeichnern aus der Methodensignatur extrahiert **FTLR** nun auch den beschreibenden Teil des Javadoc-Kommentars, nachdem dieser in seine Token portioniert wurde (in grau dargestellt). Die Vorverarbeitung erfolgt daraufhin mit denselben Schritten, die auch in der grundlegenden Variante von **FTLR** für Quelltextelemente angewandt werden. Dies führt dazu, dass die initiale Menge an Token auf folgende Tokenmenge zur Repräsentation dieses Quelltextelements reduziert wird:

```
{ "retrieve", "participant", "year", "course", "obtain",  
  "name", "person", "course" }
```

An diesem Beispiel lässt sich gut erkennen, warum Methodenkommentare hilfreich für die Bestimmung der Semantik einer Methode sein können. Die Information, dass es sich um Namen handelt, die von der Methode zurückgegeben werden, lässt sich nur aus der Methodenbeschreibung entnehmen, weil der Rückgabebetyp der Methode mit einem String-Array diese Information nicht bietet. Idealerweise wäre der Methodename dieser Methode spezifischer, z. B. `retrieveParticipantsNamesForYear`, aber dies ist in der Praxis häufig nicht der Fall, da sich beispielsweise Schreibezeit gespart wird [New+19]. Außerdem lässt sich aus dem Kommentar entnehmen, dass es sich bei den Teilnehmern um Personen und nicht z. B. Tiere handelt. In diesem Beispiel tritt ein Begriff mehrfach, sowohl im Kommentar als auch der Signatur der Methode, auf und ist somit auch mehrfach Teil der Repräsentation. Dies erhöht das Gewicht des Begriffs. Dieses Verhalten ist durch die Wahl einer Multimenge als Repräsentation explizit erwünscht. Dies wurde bewusst so gewählt, da angenommen wird, dass diese Begriffe durch ihre doppelte Nennung auch semantisch wichtiger sind. In diesem Beispiel ist dies der Klassenname `course`, welcher für die Bedeutung der Funktion den Kontext angibt aus dem bzw. für den etwas extrahiert wird.

Varianten von **FTLR**, welche zusätzlich Methodenkommentare in die Repräsentation miteinbeziehen, werden im Weiteren mit  markiert.

5.4.2. Filterung von Vorlagenelementen

In Abschnitt 4.3 wurde angenommen, dass nicht alle Teile einer Anforderung gleichermaßen zur Beschreibung der Semantik beitragen oder zum Teil Zusammenhänge beschreiben, die nicht im Quelltext abgebildet werden. Basierend auf dieser Annahme kann **FTLR** das Rauschen in der Abbildung zwischen den Artefaktelementen verringern, wenn es in der Lage ist, diese Teile zu ignorieren. Liegt die natürlichsprachliche Anforderung als eine Anwendungsfallbeschreibung vor, die einer Schablone folgt, so können gewisse Elemente dieser Schablone gefiltert werden. Die Beschriftungen der Elemente einer solchen Schablone geben Aufschluss darüber, welche Information in ihnen enthalten sein können.



Abbildung 5.10.: Beispiel für das Filtern von Vorlagenelementen in **FTLR**

Auf Basis der Analyse von Anwendungsfallbeschreibungsvorlagen in Abschnitt 4.3 kann **FTLR** so konfiguriert werden, dass (sofern vorhanden) nur der Name des Anwendungsfalls, seine Kurzbeschreibung und die Elemente der Ablaufbeschreibung als Anforderungselemente aufgefasst werden und alle anderen Anforderungselemente ignoriert werden⁹. Der Grund hierfür ist, dass bei diesen Feldern mit Sicherheit gesagt werden kann, dass diese die Semantik der Anforderung beschreiben und somit Entsprechungen im Quelltext haben sollten. Bei den ignorierten Elementen kann dies nicht mit Sicherheit gesagt werden. Für Akteure und nichtfunktionale Anforderungen ist dies zudem aufgrund ihrer Semantik eher unwahrscheinlich. Abbildung 5.10 zeigt die Filterung der Anforderungselemente für das Beispiel aus Abbildung 5.3. Die ursprüngliche Anforderungselementmenge, bestehend aus 10 Elementen, wird auf die sechs Elemente, welche den Namen, die Gesamtbeschreibung und die Ablaufbeschreibung repräsentieren, reduziert.

Da die Aufteilung in Anforderungselemente (s. Abschnitt 5.1.1) bereits alle Elemente einer Anwendungsfallbeschreibung anhand der Markierungen in einzelne Anforderungselemente aufspaltet und die Beschriftungen entfernt, müssen die entsprechenden Elemente nur noch aus der Liste der Anforderungselemente einer Anforderung entfernt werden. Sollte diese Erweiterung von **FTLR** aktiviert sein, die Anforderungen aber nicht in Form einer Vorlage vorliegen, so wird mit diesen wie in der grundlegenden Variante verfahren. Varianten von **FTLR**, welche diese Filterung anhand von Anwendungsfallbeschreibungsvorlagen durchführen, werden im Weiteren mit **▼** gekennzeichnet.

5.4.3. Ausnutzen von Aufrufabhängigkeiten

Wie bereits in Abschnitt 4.4 beschrieben, konnten bestehende Verfahren zur automatischen TLR [Pan+13; Kua+15] zeigen, dass Informationen der aufrufenden und aufgerufenen Methoden hilfreich für die Identifikation von TLs einer Methode sein können. Die aufrufenden bzw. aufgerufenen Methoden können zur Interpretation des Zwecks der Methode

⁹ Ignoriert werden hier vor allem der Akteur, die Vor- und Nachbedingungen und ggf. nichtfunktionale Anforderungen.

beitragen, denn sie bilden den Aufrufkontext dieser Methode ab. Neben dem strukturellen Klassenkontext können sie als weitere Informationsquelle durch **FTLR** genutzt werden. **FTLR** integriert diese Information, indem es die Ähnlichkeit der aufrufenden und aufgerufenen Methoden zu einer Anforderung in die Ähnlichkeit der betrachteten Methode miteinbezieht. Hierzu wird die durchschnittliche Ähnlichkeit aller aufrufenden und aufgerufenen Methoden mit der Ähnlichkeit der Methode selbst verrechnet. Der Durchschnitt wird hier verwendet, um die Verteilung der Ähnlichkeiten zu normalisieren. Um die Methode selbst weiterhin als wichtigste Informationsquelle für ihren Zweck zu betrachten, nutzt **FTLR** eine gewichtete Aggregation. Da bereits die Ergebnisse von Panichella et al. [Pan+13] gezeigt haben, dass Methodenaufrufabhängigkeiten nicht in allen Fällen zu Verbesserungen führen, gewichtet **FTLR** die durchschnittliche Ähnlichkeit der aufrufenden und aufgerufenen Methoden nur mit 10 %.

Auch diese Informationsquelle ist in **FTLR** optional aktivierbar. Ist die Option aktiviert, so aktualisiert **FTLR** die Ähnlichkeitswerte (sim) aller Methoden wie folgt:

$$\text{simAA}(am, A) = \frac{9}{10} \text{sim}(am, A) + \frac{1}{10} \frac{\sum_{m \in AA} \text{sim}(m, A)}{|AA|} \quad (5.2)$$

Die aktualisierte Ähnlichkeit (simAA) der aktuellen Methode (am) zu einer Anforderung (A) ergibt sich also zu 90 % aus der ursprünglichen Ähnlichkeit (sim) und zu 10 % aus der durchschnittlichen Ähnlichkeit der aufrufenden und aufgerufenen Methoden (AA) zu dieser Anforderung.

Varianten von **FTLR**, welche diese Form der Integration von Aufrufabhängigkeiten zwischen Methoden durchführen, werden im Weiteren mit \Rightarrow gekennzeichnet.

6. Evaluation des grundlegenden Verfahrens

Every genuine test of a theory is an attempt to falsify it, or to refute it.

(Popper [Pop02])

Um einen ersten Einblick in die Leistungsfähigkeit von **FTLR** zu geben, wird in diesem Kapitel ein Vergleich der verschiedenen Varianten untereinander und gegenüber den Referenzimplementierungen durchgeführt. Somit widmet sich dieses Kapitel dem Erkenntnisgewinn bezüglich der in Abschnitt 1.3 definierten Unterthesen $\mathbf{T}_{1.1}$, $\mathbf{T}_{1.2}$ und $\mathbf{T}_{1.3}$ und bereitet die Grundlage für ein Beantworten der Hauptthese \mathbf{T}_1 in Kapitel 15. Das Kapitel wird Aufschluss darüber geben, welchen Einfluss die getroffenen Entwurfsentscheidungen bezüglich Repräsentation, Abbildung und Aggregation auf die Leistung eines automatischen Verfahrens zur TLR haben. Hierzu werden mehrere Experimente auf sechs in der Forschung zur automatischen TLR häufig verwendeten Datensätzen durchgeführt. Diese Experimente dienen dazu, die folgenden aus den Thesen abgeleiteten Forschungsfragen zu beantworten:

FF₁: Welchen Einfluss haben die Varianten auf die Leistung **FTLRs**?

Diese Forschungsfrage zielt darauf ab, die besten Konfigurationen **FTLRs** zu bestimmen. Hierbei wird bemessen, ob das Einbeziehen der Methodenkommentare, das Filtern der Anwendungsfallvorlagenelemente oder das Ausnutzen von Aufrufabhängigkeiten (s. Abschnitt 5.4) die Leistung **FTLRs** verbessern.

FF₂: Inwieweit beeinflussen die gewählten Schwellenwerte die Leistung **FTLRs**?

FTLRs Leistung hängt von der Übertragbarkeit der gewählten Schwellenwerte ab. Sind diese nicht übertragbar, so kann das Ziel einer unüberwachten TLR nicht erreicht werden. Deshalb analysiere ich, ob die festgelegten Schwellenwerte allgemeingültig sind, indem ihre Leistung mit den optimalen Schwellenwertkombinationen auf verschiedenen Projekten verglichen werden. Zusätzlich vergleiche ich, wie stark die optimalen Schwellenwerte über die untersuchten Projekte hinweg variieren.

FF₃: Wie unterscheidet sich die Leistung **FTLRs** mit WMD im Vergleich zu **FTLR** mit Kosinusähnlichkeit?

Diese Forschungsfrage befasst sich mit der Untersuchung der Entwurfsentscheidung zur Ähnlichkeitsberechnung. Hierzu werden die Ergebnisse **FTLRs** mit WMD, mit denen mit Kosinusähnlichkeit verglichen.

Tabelle 6.1.: Übersicht der verwendeten Datensätze. Außer LibEST bestehen alle Datensätze aus Anwendungsfallbeschreibungen als Quellartefakte und Quelltext als Zielartefakte. LibEST hingegen hat einfache natürlichsprachliche Anforderungen als Quellartefakte. Von den Anwendungsfallbeschreibungen folgen außerdem alle außer iTrust einer Vorlage. TL steht für die Menge an Nachverfolgbarkeitsverbindungen (engl.: *trace links*, TLs) im Goldstandard

Projekt	Domäne	Sprache		Artefaktanzahl		
		Natürl.	Prog.	Quelle	Ziel	TL
eTour	Tourismus	EN	Java	58	116	308
iTrust	Gesundheit	EN	Java	131	226	286
iTrustJSP	Gesundheit	EN	Java/JSP	131	367	399
SMOS	Bildung	IT	Java	67	100	1044
eAnci	Verwaltung	IT	Java	139	55	567
Albergate	Tourismus	IT	Java	17	55	53
LibEST	Vernetzung	EN	C	52	14	204

Tabelle 6.2.: Weitere Details zu den verwendeten Datensätzen. SLOC gibt die Quelltextzeilen der Quelltextartefakte an und die Q×Z-Basislinie stellt die Leistung eines naiven Verfahrens dar, welches jedes Quell- mit allen Zielartefakten verbindet. Die mit * markierten Einträge des iTrustVoll-Datensatzes entsprechen denen von iTrust, da sich die Werte für JSP-Dateien nicht bestimmen lassen. Aus Gründen der Lesbarkeit nutze ich hier und in allen folgenden Tabellen dieser Arbeit die englische Schreibweise von Zahlen (. anstatt , als Dezimaltrennzeichen)

Projekt	SLOC	Ø Methoden pro Klasse		Abdeckung		Q × Z Basislinie	
		public	sonst	Quelle	Ziel	Präz.	F ₁
eTour	12420	7.0	3.6	0.983	0.767	0.046	0.088
iTrust	14570	6.5	0.3	0.802	0.385	0.010	0.020
iTrustJSP	27512	*6.5	*0.3	0.802	0.367	0.008	0.016
SMOS	8872	3.5	2.5	1.000	0.684	0.159	0.274
eAnci	6081	5.1	0.2	0.281	1.000	0.074	0.137
Albergate	10464	2.0	4.0	0.941	0.945	0.057	0.107
LibEST	8333	30.6	0.0	0.786	0.900	0.280	0.438

FF₄: Erzielt die feingranulare Abbildung mit anschließender Aggregation eine besser Abbildungsgüte als eine direkte Abbildung auf Artefaktebene?

Die letzte Forschungsfrage zielt auf einen Erkenntnisgewinn zur Entwurfsentscheidung bezüglich der Ebene der Abbildung ab. Hierbei wird bemessen, ob die gewählte Abbildung auf feingranularer Elementebene derjenigen auf Artefaktebene überlegen ist.

6.1. Datensätze

Um die Leistungsfähigkeit eines Verfahrens zur automatischen TLR bemessen zu können, bedarf es Projekte mit verfügbaren Anforderungen, Quelltext und einem Goldstandard für Nachverfolgbarkeitsverbindungen zwischen diesen. Hierfür greife ich auf eine Menge von Vergleichsdatensätzen aus der Forschungsgemeinschaft zur TLR zurück, welche ich im Folgenden näher beschreiben werde. Diese erlauben mir einen Vergleich mit bestehenden Arbeiten. Das *Center of Excellence for Software & Systems Traceability* (CoEST) [CoE] hat eine Menge solcher Datensätze, welche häufig in der Forschung zur Nachverfolgbarkeit verwendet werden, gesammelt und auf ihrer Webseite zur Verfügung gestellt. Aus dieser Menge habe ich die fünf Datensätze eTour, iTrust, SMOS, eAnci und Albergate ausgewählt. Diese wurden gewählt, da sie zum einen über einen Goldstandard für TLs zwischen Anforderungen und objektorientiertem Quelltext verfügen und zum anderen verschiedene Domänen abdecken. Um zusätzlich einen Einblick in **FTLRs** Leistung auf nicht objektorientiertem Quelltext geben zu können, wird zusätzlich der von Moran et al. [Mor+20b] zur Verfügung gestellte C-Datensatz LibEST verwendet.

Tabelle 6.1 gibt einen Überblick über die Datensätze, inklusive der verwendeten Sprachen und Artefakte. Alle Datensätze außer LibEST umfassen Anwendungsfallbeschreibungen als Anforderungen; LibEST hingegen enthält einfache natürlichsprachliche Anforderungen. Der Originaldatensatz von iTrust weist sowohl Java als auch Jakarta Server Pages (JSP) als Zielartefakte auf. JSP-Dateien enthalten hauptsächlich nicht objektorientierten, sondern HTML-basierten Quelltext. Für diese Art von Quelltext können viele der Merkmale die **FTLR** nutzt, nicht extrahiert werden. Aus diesem Grund werden zwei Varianten von iTrust betrachtet, eine mit den JSP-Dateien (iTrustJSP) und eine nur mit Java-Dateien und TLs zu diesen (iTrust). Dies wirkt sich vor allem auf die Menge der Zielartefakte aus, welche um 141 sinkt und dadurch auch 113 TLs weniger gefunden werden müssen. Für JSP-Dateien nutzt **FTLR** allein den Dateinamen zur Repräsentation. Bis auf Albergate enthalten alle Datensätze Methodenkommentare für die meisten der öffentlichen Methoden. Albergates Quelltextartefakte enthalten insgesamt bis auf ein paar wenige Zeilenkommentare keinerlei Dokumentation.

Neben den Programmiersprachen unterscheiden sich die Datensätze außerdem in der vorherrschenden natürlichen Sprache. iTrust und LibEST sind in englischer Sprache, eAnci und Albergate in italienischer Sprache verfasst. Im Original enthält eTour hauptsächlich in englischer Sprache verfassten Text mit Ausnahme der Bezeichner im Quelltext und den Namen der Anwendungsfallbeschreibungen. SMOS hingegen besteht aus italienischem Text für die Anwendungsfallbeschreibungen und Quelltextkommentare; allerdings liegen die Bezeichner im Quelltext auf Englisch vor. Um das volle Potenzial der vortrainierten Worteinbettungsmodelle nutzen zu können, benötigt es eine einheitliche Sprache pro Projekt. Aus diesem Grund wurden die Bezeichner in eTour und SMOS mithilfe eines Wörterbuchs händisch in die jeweils vorherrschende Sprache übersetzt. So wurde beispielsweise im eTour-Datensatz der Bezeichner *cognome* zu *surname* übersetzt. Die übersetzten Varianten der Datensätze sind im Replikationspaket zu dieser Dissertation [Hey23b] zu finden. Dementsprechend setzt **FTLR** für eTour, iTrust und LibEST ein englischsprachliches

fastText-Worteinbettungsmodell und für SMOS, eAnci und Albergate ein italienisches *fastText*-Worteinbettungsmodell ein. Es existieren auch *fastText*-Worteinbettungen, welche zwei Sprachen aneinander ausrichten [Jou+18] und somit diese in denselben Vektorraum überführen. Allerdings sind diese nur auf dem kleineren Datensatz der Wikipedia trainiert und zeigten in ersten Tests und auch in verwandten Arbeiten (s. Abschnitt 3.4) schlechtere Ergebnisse für die TLR als der Einsatz eines einsprachigen Modells auf den übersetzten Varianten. Prinzipiell ermöglichen diese es aber, 44 unterschiedliche Sprachen miteinander zu mischen und mittels **FTLR** aufeinander abzubilden, indem nur das Worteinbettungsmodell ausgetauscht und die sprachspezifische Vorverarbeitung angepasst wird. Dadurch stellen diese eine sinnvolle Lösung für Projekte in Mischsprache dar, in denen ein Übersetzen zu teuer wäre.

Die Anwendungsfallbeschreibungen von eTour, SMOS, eAnci folgen alle häufig verwendeten Vorlagen, welche mindestens den Namen des Anwendungsfalles, die teilnehmenden Akteure, die Vor- und Nachbedingungen sowie die Ablaufbeschreibung kennzeichnen. eTours und SMOS' Anwendungsfallbeschreibungen enthalten zusätzlich jeweils noch eine kurze Beschreibung des Anwendungsfalles. Albergates Anwendungsfallbeschreibungen folgen ebenso einer Vorlage, diese unterscheidet sich allerdings von derjenigen der anderen dadurch, dass keine dedizierte Ablaufbeschreibung gekennzeichnet ist, sondern der mit Beschreibung gekennzeichnete Block entweder eine Kurzbeschreibung oder eine Art Ablaufbeschreibung enthält. Außerdem wird in Albergate kein Akteur angegeben. Die Anwendungsfallbeschreibung des iTrust-Datensatzes hingegen besteht nur aus der Ablaufbeschreibung. LibESTs Anforderungen sind zwar keine Anwendungsfallbeschreibungen, folgen aber dennoch einer gewissen Struktur, die beim Portionieren ausgenutzt werden kann. Jede Anforderung startet mit der Beschriftung *Requirement X*;, wobei *X* für die Nummer der Anforderung steht, gefolgt von einem Namen wie z. B. *HTTP-BASED CLIENT AUTHENTICATION*. Nach einem Zeilenumbruch folgt daraufhin die Beschreibung der Anforderung. Somit kann **FTLR** den Namen identifizieren und als eigenes Anforderungselement neben den einzelnen Sätzen der Beschreibung repräsentieren. Konkrete Beispiele für die Anforderungen der verwendeten Datensätze befinden sich in Anhang D.

Betrachtet man die Artefaktanzahlen der ausgewählten Datensätze, so umfassen fünf der sieben Datensätze eine kleinere Anzahl an Anforderungen als Quelltextartefakte. Das Verhältnis ist hierbei im Durchschnitt bei ungefähr Eins zu Zwei. Nur eAnci und LibEST haben ein umgekehrtes Verhältnis. Bei LibEST lässt sich dies damit erklären, dass es sich um nicht objektorientierten Quelltext handelt und die Quelltextartefakte über ihre Methoden viele verschiedene Zwecke erfüllen. Dies lässt sich auch an der in Tabelle 6.2 dargestellten durchschnittlichen Anzahl an Methoden pro Quelltextartefakt ablesen. Diese liegt bei LibEST mit 30,6 deutlich über derjenigen der anderen Datensätze, welche zwischen 7 und 2 öffentliche Methoden pro Klasse liegen. Für eAnci hingegen liegt die Anzahl an Methoden pro Klasse im Durchschnitt bei 5,1. Die Abdeckung der beiden Artefakttypen durch die TLs (s. Tabelle 6.2) zeigt aber deutlich, dass ein Großteil der Anforderungen in eAnci keine Entsprechung im Quelltext hat. Nur 28 % der Anforderungsartefakte sind Teil einer TL des Goldstandards, aber 100 % der Quelltextartefakte sind abgedeckt. Somit ist der Datensatz zu eAnci entweder unvollständig hinsichtlich der TLs oder viele der Anforderungen wurden nicht im Quelltext umgesetzt.

Auch die Abdeckungen der anderen Datensätze deuten auf gewisse Inkonsistenzen bezüglich der Anforderungen und ihren Umsetzungen oder den vorhandenen TLs hin, doch diese sind weniger stark ausgeprägt. Die anderen Datensätze weisen im Durchschnitt eine Abdeckung von 90 % für die Anforderungsartefakte auf. Für die Quelltextartefakte ist die durchschnittliche Abdeckung mit 78 % geringer. Hierbei spielt vor allem die niedrige Abdeckung der Quelltextartefakte der beiden iTrust-Varianten eine Rolle. Diese liegt für iTrust bei 38,5 % und für iTrustJSP bei 36,7 %. Ein Großteil der nicht abgedeckten Java-Klassen sind JavaBeans und BeanLoader zum Laden und Speichern von Daten der grafische Benutzeroberfläche (GBO). Die Anforderungen verweisen aber zumeist nur auf die jeweiligen Datenzugriffsobjekte (engl.: *data access objects*, DAOs), welche wiederum die jeweiligen Loader und somit die Beans befüllen. Eine Erklärung hierfür könnte sein, dass eine Art Transitivität der Verbindungen angenommen wurde oder das Befüllen der Benutzeroberflächenelemente nicht als die in den Anforderungen beschriebene Funktionalität angesehen wird.

Niedrige Abdeckungen können in einem Ansatz, der die TLR als IR-Problem auffasst, zu vielen falsch positiven Ergebnisse führen. Dies liegt daran, dass diese Ansätze Nachverfolgbarkeitsverbindungskandidaten dadurch bestimmen, dass die relevantesten Zielartefakte für ein Quellartefakt gesucht werden. Betrachtet man das erreichte F_1 -Maß eines naiven Verfahrens, welches eine TL zwischen jedem Anforderungsartefakt und jedem Quelltextartefakt annimmt (QxZ-Basislinie in Tabelle 6.2), so wird der niedrigste Wert ebenso für iTrust erzielt. Insgesamt sind die Ergebnisse dieser Basislinie sehr niedrig, was verdeutlicht, wie entscheidend die korrekte Auswahl der Nachverfolgbarkeitsverbindungskandidaten ist. Interessant sind hier aber die erreichten Ergebnisse für SMOS und LibEST. SMOS ist der Datensatz mit der größten Anzahl an TLs im Goldstandard, hat aber vergleichsweise wenige Quell- und Zielartefakte. Dies führt dazu, dass selbst ein solches naives Verfahren bereits ein F_1 -Maß von 27,4 % erzielen kann. Für LibEST ist das Ergebnis sogar noch höher bei einem F_1 -Maß von 43,8 %, welches ein ausgefeilteres Verfahren erst übertreffen muss. Der Grund für den hohen Wert für LibEST ist die sehr geringe Anzahl an Quelltextartefakten von 14 und die vergleichsweise hohe Nachverfolgbarkeitsverbindungsanzahl von 204.

Betrachtet man den Umfang bzw. die Größe der Projekte in Quelltextzeilen (engl.: *source lines of code*, SLOC), so decken die gewählten Datensätze den Bereich eines kleinen Projektes mit ca. 6000 Zeilen bis hin zu einem mittleren bis großen Projekt mit 28000 Zeilen ab. Außerdem decken die Projekte fünf verschiedene Domänen ab. Allerdings stammen bis auf LibEST alle Projekte aus der akademischen Welt und stellen somit nicht zwangsweise die Realität in der Praxis dar. Leider ist es schwierig an Industrieprojekte zu gelangen, welche bereits über vorhandene TLs verfügen. Somit beschränke ich mich, wie bisherige Arbeiten zur automatischen TLR auch, auf diese Datensätze unter der Einschränkung, dass sie nur bedingt eine Aussage zur Anwendbarkeit in der Praxis erlauben. Aufgrund der Zusammensetzung, Varietät und den verschiedenen Institutionen und Lehrveranstaltungen denen die Projekte entstammen, lassen sich die Ergebnisse aber vermutlich zumindest teilweise auf nicht-akademische Projekte übertragen.

6.2. Methodik

In allen Experimenten werden die Metriken F_1 -Maß und MAP verwendet (s. Abschnitt 2.11). Ersteres ist das präferierte Maß für Klassifikationsaufgaben und kann in der TLR dafür genutzt werden, zu bemessen, wie gut ein Verfahren die erwarteten TLs bestimmt. Es ist das harmonische Mittel aus Präzision und Ausbeute und bewertet damit, wie gut ein Verfahren alle geforderten TLs findet, ohne dabei gleichzeitig zu viele falsche Verbindungen zu produzieren. Somit sollte es das Ziel aller Verfahren zur automatischen TLR sein, ein hohes F_1 -Maß zu erzielen [Got+12a].

Da bisherige Verfahren noch nicht an die nötigen F_1 -Maße herankommen, die notwendig wären, um die Wiederherstellung vollends zu automatisieren, kann eine Betrachtung der Eignung eines Verfahrens für eine semi-automatische Anwendung sinnvoll sein. Bei der semi-automatischen TLR werden Expert:innen in der manuellen Erstellung der TLs durch die Ausgabe eines Verfahrens unterstützt. Zumeist geschieht dies durch das Bereitstellen einer geordneten Liste von Nachverfolgbarkeitsverbindungskandidaten pro betrachtetem Quellartefakt. Die durch die Expert:innen noch zu verrichtende Arbeit wird in dieser Situation maßgeblich von der Position der korrekten TLs in der Liste bestimmt. Sind diese alle möglichst weit oben in der Liste zu finden, so müssen die Expert:innen weniger Zeit mit dem Finden der korrekten Verbindungen verbringen. Um die Güte dieser Listen zu bewerten, kann ihre durchschnittliche Präzision (engl.: *average precision*, AP) bestimmt und über alle Quellartefakte gemittelt werden, welches dann den Mittelwert der durchschnittlichen Präzision (engl.: *mean average precision*, MAP) ergibt (s. Abschnitt 2.11). Die AP gibt die Präzision bezüglich des Rangs der korrekten Verbindungen in der geordneten Liste an. Eine sehr hohe MAP bedeutet also, dass für alle Quellartefakte alle korrekten TLs am Anfang der Liste stehen.

FTLR ist zunächst auf die vollautomatische Lösung und damit dem Ziel eines hohen F_1 -Maßes ausgelegt. Hierbei dienen die Schwellenwerte zum Filtern der tatsächlich ausgegebenen Kandidaten. Dies sorgt dafür, dass **FTLR** keine vollständige, geordnete Liste der Nachverfolgbarkeitsverbindungskandidaten ausgibt. Intern existiert allerdings aufgrund der Ähnlichkeitsberechnung eine solche Rangfolge der Kandidaten. Um auch eine Aussage über die Eignung **FTLRs** in einer semi-automatischen TLR geben zu können und die Leistung mit anderen Ansätzen zu vergleichen, wird für die Berechnung des MAP **FTLR** ohne die beiden Schwellenwerte ausgeführt. Die geordnete Liste von Nachverfolgbarkeitsverbindungskandidaten pro Anforderung ergibt sich daraufhin, indem zunächst pro Klasse die ähnlichste Stimme eines Quelltextelements zu dieser Anforderung als der Repräsentant der Klasse gewählt wird. Diese Liste von Ähnlichkeiten einer Anforderung mit allen Klassenrepräsentanten wird daraufhin nach den Ähnlichkeiten geordnet. Da neben den Quelltextdateien in manchen Datensätzen auch ein paar wenige Dateien enthalten sind, die keine Quelltextartefakte darstellen und trotzdem TLs im Goldstandard besitzen, werden diese, obwohl sie eigentlich von **FTLR** ignoriert werden, ans Ende der Liste angehängt. Hierdurch erhält man eine vollständige Liste pro Anforderung. Über diese Listen lässt sich daraufhin der MAP berechnen.

Um die in Abschnitt 1.3 aufgestellten Thesen zu überprüfen, können Hypothesentests eingesetzt werden (s. Abschnitt 2.12). Da sich alle Thesen auf eine Verbesserung der Leistung eines Verfahrens durch Einsatz einer bestimmten Technik gegenüber einem anderen Verfahren ohne diese Technik beziehen, können diese auf einen einseitigen Test reduziert werden. Dieser bestimmt, ob durch Einsatz der Technik eine signifikante Verbesserung hinsichtlich einer Metrik erzielt werden konnte. Somit fungiert als Nullhypothese H_0 die Aussage, dass der Einsatz der Technik zu keiner Verbesserung führt. Der Test prüft daraufhin, mit welcher Wahrscheinlichkeit p die Nullhypothese H_0 verworfen wird, obwohl sie wahr ist. Ist dieser Wert gering genug, kann davon ausgegangen werden, dass die Alternativhypothese H_A , also die eigentliche Forschungshypothese, dass eine Verbesserung vorliegt, gilt. Diese Grenze wird als Signifikanzniveau α bezeichnet und wird in der Softwaretechnik meist auf 0,05 festgelegt. Da es sich bei den hier durchgeführten Experimenten um Vergleiche bezüglich einer Metrik auf verschiedenen Datensätzen handelt, wird eine paarweise Betrachtung pro Datensatz benötigt. Der erzielte Wert auf einem Datensatz muss mit dem Wert nach Anwenden der Technik auf demselben Datensatz verglichen werden. Für diese Art von Hypothesentests kann der gepaarte t-Test [Stu08] oder der gepaarte Wilcoxon-Vorzeichen-Rang-Test (s. Abschnitt 2.12) eingesetzt werden. Ersterer ist ein parametrisierter Test, welcher eine Normalverteilung der Grundgesamtheiten annimmt. Zweiterer ist ein nicht-parametrisierter Test, der diese Annahme nicht voraussetzt. Da nicht davon ausgegangen werden kann, dass die erzielten Ergebnisse vor und/oder nach dem Einsatz einer bestimmten Technik normalverteilt sind, ist in diesem Szenario der gepaarte Wilcoxon-Vorzeichen-Rang-Test vorzuziehen. Aus diesem Grund werden für die einzelnen Experimente ebenso die p -Werte eines gepaarten Wilcoxon-Vorzeichen-Rang-Tests angegeben.

6.3. Einfluss der Varianten

Das erste Experiment zielt auf die Beantwortung von Forschungsfrage **FF₁** ab, welche sich auf den Einfluss der Varianten auf die Leistung von **FTLR** bezieht. Es liefert also Erkenntnisse bezüglich These **T_{1.1}**, welche besagt, dass das Einbeziehen von strukturellen Informationen des Quelltextes und der Anforderungen die Leistung des Verfahrens positiv beeinflussen. Die Aufrufabhängigkeiten in den mit **⇒** markierten Varianten können als strukturelle Informationen des Quelltextes und das Filtern der Vorlagenelemente in den mit **▼** markierten Varianten als Ausnutzen von strukturellen Informationen der Anforderungen angesehen werden. In diesem Experiment werden die verschiedenen Varianten von **FTLR** einzeln und in Kombination auf die Datensätze angewandt und ihre Leistung bezüglich des jeweiligen Goldstandards bemessen.

Die Ergebnisse des Experiments sind in Tabelle 6.3 dargestellt. Die Variante, welche alle drei Erweiterungen kombiniert, erzielt das beste Ergebnis hinsichtlich des F_1 -Maßes über alle Datensätze hinweg. Sie erzielt ein durchschnittliches F_1 -Maß von 33 % und das beste F_1 -Maß für die Projekte iTrust und LibEST. Allerdings erzielen die Varianten **●**, **⇒●** und

Tabelle 6.3.: Vergleich des Einflusses der Varianten auf das **FTLR**-Verfahren. \rightleftharpoons markiert Varianten, die Aufrufabhängigkeiten einbeziehen, \bullet welche, die Methodenkommentare betrachten und \blacktriangledown Varianten, die das Filtern von Vorlagenelementen nutzen. Die besten Ergebnisse sind fett dargestellt. p stellt die Ergebnisse eines gepaarten Wilcoxon-Vorzeichen-Rang-Tests auf den Werten des F_1 -Maßes und MAPs gegenüber **FTLR** ohne aktivierte Varianten dar

Projekt	Maß	FTLR-Varianten							
		\rightleftharpoons	\bullet	\blacktriangledown	\rightleftharpoons \bullet	\rightleftharpoons \blacktriangledown	\bullet \blacktriangledown	\rightleftharpoons \bullet \blacktriangledown	
eTour	Prüz.	.271	.272	.267	.376	.270	.383	.373	.379
	Ausb.	.701	.698	.688	.643	.692	.640	.633	.633
	F_1	.391	.392	.385	.475	.388	.479	.469	.474
	MAP	.370	.375	.349	.528	.350	.532	.483	.483
iTrust	Prüz.	.135	.134	.142	.135	.149	.134	.142	.149
	Ausb.	.322	.290	.374	.322	.346	.290	.374	.346
	F_1	.190	.183	.206	.190	.209	.183	.206	.209
	MAP	.227	.213	.284	.227	.271	.213	.284	.271
SMOS	Prüz.	.404	.400	.411	.425	.419	.424	.432	.440
	Ausb.	.340	.333	.356	.314	.352	.311	.340	.335
	F_1	.369	.364	.382	.361	.383	.359	.381	.381
	MAP	.405	.400	.418	.438	.418	.434	.451	.451
eAnci	Prüz.	.224	.223	.204	.312	.204	.311	.273	.274
	Ausb.	.261	.250	.291	.226	.280	.219	.254	.243
	F_1	.241	.236	.240	.262	.236	.257	.263	.258
	MAP	.148	.147	.157	.151	.157	.150	.154	.153
Albergate	Prüz.	.208	.211	.208	.171	.211	.171	.171	.171
	Ausb.	.302	.302	.302	.132	.302	.132	.132	.132
	F_1	.246	.248	.246	.149	.248	.149	.149	.149
	MAP	.447	.447	.447	.406	.447	.406	.406	.406
LibEST	Prüz.	.403	.403	.431	.403	.431	.403	.431	.431
	Ausb.	.490	.490	.623	.490	.623	.490	.623	.623
	F_1	.442	.442	.509	.442	.509	.442	.509	.509
	MAP	.559	.559	.581	.559	.581	.559	.581	.581
	$\emptyset F_1$.313	.311	.328	.313	.329	.312	.329	.330
	p Basis	.9062	.1563	.6563	.1563	.5937	.2188	.2188	
	\emptyset MAP	.359	.357	.373	.385	.370	.383	.393	.391
	p Basis	.8438	.1563	.2188	.1563	.4063	.0782	.0781	

$\bullet\blacktriangledown$ ähnlich hohe durchschnittliche F_1 -Maße. Da alle dieser Varianten die Methodenkommentare nutzen, verdeutlicht dies, dass Methodenkommentare einen Mehrwert für die TLR bieten. Auf allen Projekten die über Methodenkommentare verfügen, außer eTour, verbessert sich das Ergebnis. Auf eTour verschlechtert das Einbeziehen von Methodenkommentaren sogar die Leistung sowohl im F_1 -Maß als auch MAP. Eine mögliche Ursache

Tabelle 6.4.: Durchschnittliches F_1 -Maß und MAP der Varianten und p -Werte eines gepaarten Wilcoxon-Vorzeichen-Rang-Tests, sofern Albergate nicht betrachtet wird

Maß	FTLR-Varianten							
		⇔	☞	⚡	☞	⚡	☞	⚡
$\emptyset F_1$.327	.323	.344	.346	.345	.344	.366	.366
p Basis		.9375	.1563	.3125	.1563	.3125	.0313	.0313
\emptyset MAP	.342	.339	.358	.381	.355	.378	.391	.388
p Basis		.8438	.1563	.0313	.1563	.1563	.0313	.0313

könnte die Qualität der Kommentare in eTour sein. Diese sind zum Teil inkonsistent mit den Methodensignaturen die sie beschreiben. Außerdem besitzen viele öffentliche Methoden gar keine Kommentare, da diese nur im Interface definiert wurden, aber weder ein `@Override` noch `{@inheritDoc}` vorhanden ist.

Die Varianten, die Vorlagenelemente filtern (⚡), erzielen die höchsten durchschnittlichen MAPs und verbessern auch das F_1 -Maß für diejenigen Projekte, die Vorlagen nutzen. Eine Ausnahme hierbei bildet Albergate. Hier sorgt das Filtern der Vorlagenelemente dafür, dass sich das F_1 -Maß deutlich um zehn Prozentpunkte reduziert. Die deutlich gesunkene Ausbeute deutet darauf hin, dass sich Semantik, die zur Abbildung der erwarteten TLs benötigt wird, in den gefilterten Elementen befindet. Eine andere Begründung könnten die Eigenschaften des Projektes liefern. Albergate ist das Einzige der Projekte mit Anwendungsfallbeschreibungen, welches keine dedizierte Ablaufbeschreibung besitzt und der Quelltext von Albergate umfasst keine Kommentare. Zusätzlich hat Albergate die geringste Anzahl an öffentlichen Methoden pro Klasse (vgl. Tabelle 6.2). Diese Eigenschaften könnten dazu führen, dass die gewählten Schwellenwerte für dieses Projekt nicht optimal sind. Die besseren Ergebnisse ohne Vorlagenelementfilter deuten darauf hin, dass durch die Schwellenwerte in Kombination mit der geringen Informationslage viele falsche Verbindungen identifiziert werden. Betrachtet man hingegen optimierte Schwellenwerte¹ für Albergate so ergibt sich, dass die Filterung doch einen positiven Effekt auch auf Albergate hat (s. Tabelle 6.5).

Der Einfluss der Aufrufabhängigkeiten (⇔) ist insgesamt eher gering. Einzeln angewandt verschlechtert diese Erweiterung sogar das Ergebnis. Für die Projekte eTour, iTrust, SMOS und Albergate allerdings wird das beste F_1 -Maß nur durch eine Kombination der Aufrufabhängigkeiten mit einer der anderen Erweiterungen erzielt. Dass der Einfluss dieser Erweiterung so gering ist, liegt aber auch an der konservativen Gewichtung der Ähnlichkeiten aus den Aufrufabhängigkeiten (s. Abschnitt 5.4.3). Die Eigenschaften der einzelnen Projekte könnten unterschiedliche Gewichtungen benötigen, welche aber so nicht generell festgelegt werden können. Die teilweise negativen Beiträge dieser Erweiterung unterstüt-

¹ Optimiert durch vollständige Suche in 0,001er-Schritten.

Tabelle 6.5.: Vergleich des Einflusses der Merkmalsvarianten auf das **FTLR**-Verfahren mit optimierten Schwellenwerten. \rightleftharpoons markiert Varianten, die Aufrufabhängigkeiten nutzen, \bullet Varianten, die Methodenkommentare verwenden und \blacktriangledown Varianten, die Anwendungsfallvorlagen miteinbeziehen. Die besten Ergebnisse im F_1 -Maß pro Projekt sind fett dargestellt

Projekt	Maß	FTLR-Varianten							
		\rightleftharpoons	\bullet	\blacktriangledown	$\rightleftharpoons\bullet$	$\rightleftharpoons\blacktriangledown$	$\bullet\blacktriangledown$	$\rightleftharpoons\bullet\blacktriangledown$	$\rightleftharpoons\blacktriangledown\bullet$
eTour	Prüz.	.331	.333	.323	.505	.325	.508	.472	.479
	Ausb.	.656	.659	.623	.597	.623	.604	.571	.562
	F_1	.440	.442	.425	.548	.427	.552	.517	.517
iTrust	Prüz.	.193	.227	.252	.193	.190	.227	.252	.190
	Ausb.	.241	.196	.255	.241	.322	.196	.255	.322
	F_1	.215	.210	.253	.215	.239	.210	.253	.239
SMOS	Prüz.	.288	.293	.405	.313	.380	.334	.380	.369
	Ausb.	.629	.603	.406	.590	.433	.546	.466	.484
	F_1	.396	.394	.405	.409	.405	.415	.419	.419
eAnci	Prüz.	.214	.225	.193	.311	.188	.287	.255	.266
	Ausb.	.333	.321	.381	.272	.395	.296	.300	.287
	F_1	.261	.264	.256	.290	.255	.292	.276	.277
Albergate	Prüz.	.254	.254	.254	.284	.254	.284	.284	.284
	Ausb.	.547	.547	.547	.509	.547	.509	.509	.509
	F_1	.347	.347	.347	.365	.347	.365	.365	.365
LibEST	Prüz.	.421	.421	.406	.421	.406	.421	.406	.406
	Ausb.	.799	.799	.863	.799	.863	.799	.863	.863
	F_1	.552	.552	.552	.552	.552	.552	.552	.552
	$\emptyset F_1$.368	.368	.373	.396	.371	.397	.397	.395
	p Basis	.5781	.4063	.0469	.4063	.0625	.0156	.0156	

zen aber die Entwurfsentscheidung, die Ähnlichkeiten aus den Aufrufabhängigkeiten nur mit geringen 10 % zu gewichten.

Betrachtet man die Ergebnisse hinsichtlich des MAPs, so erzielt die Variante $\bullet\blacktriangledown$ mit einem MAP von 39,3 % den höchsten durchschnittlichen Wert. Die Variante, die das höchste durchschnittliche F_1 -Maß aufweist ($\rightleftharpoons\bullet\blacktriangledown$), erzielt mit 39,1 % den zweithöchsten Wert. Den dritthöchsten Wert erzielt die alleinige Anwendung des Vorlagenelementfilters (\blacktriangledown). Dies ist interessant, da diese Variante im F_1 -Maß nicht unter der besten Varianten ist. Dies deutet darauf hin, dass die Schwellenwerte für diese Variante nicht optimal sind, denn der MAP ist durch das Vorgehen bei der Berechnung unabhängig von den Schwellenwerten.

Zur Beantwortung der These $T_{1.1}$ sind zwar klar positive Einflüsse der strukturellen Informationen auf F_1 -Maß und MAP des Verfahrens erkennbar, diese sind aber nicht signifikant. Dies zeigen die p -Werte des gepaarten Wilcoxon-Vorzeichen-Rang-Tests bezüglich der als Basis verwendeten Variante ohne Erweiterungen unter der Hypothese, dass die Erweiterungen einen positiven Einfluss haben. Lediglich die MAPs der beiden Varianten ☛ und ☛☛ kommen dem Signifikanzniveau von 0,05 nahe. Dies liegt vor allem daran, dass durch die Abweichungen im Ergebnis für Albergate die durchschnittliche Differenz abgeschwächt wird. Betrachtet man die Ergebnisse ohne Albergate, wie sie in Tabelle 6.4 dargestellt sind, so ist die Verbesserung der beiden Varianten ☛ und ☛☛ sowohl im F_1 -Maß als auch im MAP signifikant bezüglich des Signifikanzniveaus von 0,05.

Um den Effekt der gewählten Schwellenwerte herauszunehmen und zu bestimmen, was mit den Varianten möglich ist, habe ich ein weiteres Experiment durchgeführt. Hierfür werden die Schwellenwerte für **FTLR** in 0,001er-Schritten variiert und die pro Projekt beste Schwellenwertkombination bestimmt. Die Ergebnisse dieses Experiments sind in Tabelle 6.5 dargestellt. Auch hier weisen die beiden Varianten ☛ und ☛☛ eine signifikante Verbesserung des F_1 -Maßes auf. Es lässt sich also auch hier feststellen, dass These $T_{1.1}$ für dieses Experiment zutrifft. Zusätzlich erzielt die Variante, die nur Vorlagenelemente filtert, mit einem durchschnittlichen F_1 -Maß von 39,6 % nur 0,1 Prozentpunkte weniger als die beste Variante, welches die Hypothese des suboptimalen Schwellenwertes für diese Variante unterstützt. Insgesamt ist das beste erzielte durchschnittliche F_1 -Maß von 33 % auf 39,7 % gestiegen. Eine genauere Betrachtung der möglichen Verbesserungen durch angepasste Schwellenwerte folgt in Abschnitt 6.4.

Zusammenfassend lässt sich also für den Einfluss der Varianten (FF_1) festhalten, dass das Hinzunehmen von Methodenkomentaren und das Filtern von Vorlagenelementen einen positiven Einfluss auf die Leistung von **FTLR** hat. In bestimmten Situationen können auch die Aufrufabhängigkeiten zu weiteren Verbesserungen führen. Diese Erkenntnisse unterstützen These $T_{1.1}$, bezüglich der Leistungssteigerung durch Ausnutzen struktureller Informationen in Quelltext und Anforderungen, wie es die Aufrufabhängigkeiten und Anwendungsfallvorlagenelemente sind.

6.4. Einfluss der Schwellenwerte

Die Ergebnisse in Tabelle 6.5 haben gezeigt, dass die gewählten Schwellenwerte die Leistung von **FTLR** beeinflussen. In diesem Abschnitt wird dieser Einfluss nun genauer analysiert und bestimmt, ob fest gewählte Schwellenwerte über Projektgrenzen hinweg einsetzbar sind. Diese Frage ist relevant, da das Zielszenario von **FTLR** der Einsatz auf unbekannte Projekte ohne initiale TLs ist (unüberwachte TLR). Ohne diese initialen TLs können also auch keine Schwellenwerte optimiert werden. Da Projekte unterschiedliche Formulierungen, semantische Lücken und vorhandene Informationen aufweisen, ist eine Verbesserung der erzielten Ergebnisse durch pro Projekt optimierte Schwellenwerte zu erwarten. Die Forschungsfrage (FF_2), die dieser Abschnitt beantworten soll, ist also nicht, ob Verbesserungen durch optimierte Schwellenwerte erzielbar sind, sondern wie groß


Tabelle 6.6.: Vergleich der festgelegten mit der optimierten Schwellenwertkombination (OPT) pro Projekt für **F₁LR** \Rightarrow **Y**

	Präzision		Ausbeute		F ₁ -Maß		Schwellenwert			
							Mehrheits-	Abschluss-		
	Δ		Δ		Δ		Δ	Δ		
eTour	.379		.633		.474		.590	-.016	.440	
eTour _{OPT}	.479	+0.101	.562	-.071	.517	+0.043	.574		.409	-.031
iTrust	.149		.346		.209		.590	-.040	.440	
iTrust _{OPT}	.190	+0.041	.322	-.024	.239	+0.030	.550		.436	-.004
SMOS	.440		.335		.381		.590	+.026	.440	
SMOS _{OPT}	.369	-.071	.484	+0.148	.419	+0.038	.616		.487	+0.047
eAnci	.274		.243		.258		.590	-.021	.440	
eAnci _{OPT}	.266	-.008	.287	+0.044	.277	+0.019	.569		.463	+0.023
Albergate	.171		.132		.149		.590	-.092	.440	
Albergate _{OPT}	.284	+0.113	.509	+0.377	.365	+0.216	.498		.498	+0.058
LibEST	.431		.623		.509		.590	-.006	.440	
LibEST _{OPT}	.406	-.025	.863	+0.240	.552	+0.043	.584		.486	+0.046
∅		+0.025		+0.119		+0.065		-.025		+0.023
∅ o. Albergate		+0.008		+0.067		+0.035		-.011		+0.016

die Unterschiede in der Güte zu den festgelegten Schwellenwerten sind und wie stark sich die Schwellenwerte über die Datensätze und damit verschiedene Projekte hinweg unterscheiden.

Um diese Frage zu beantworten, wurden die beiden Schwellenwerte pro Projekt und Variante in 0,001er-Schritten variiert und die jeweils beste Schwellenwertkombination hinsichtlich des F₁-Maßes bestimmt. Die Tabellen 6.6 bis 6.8 zeigen die Unterschiede in Präzision, Ausbeute und F₁-Maß zwischen den festgelegten und pro Projekt optimierten Schwellenwerten für die drei besten **F₁LR**-Varianten hinsichtlich F₁-Maß und MAP (\Rightarrow **Y**, **Y** und **Y**).

Betrachtet man die Ergebnisse für die \Rightarrow **Y**-Variante (Tabelle 6.6), so fällt auf, dass für einen Teil der Projekte die optimierte Schwellenwertkombination dazu führt, dass die Präzision steigt, aber die Ausbeute sinkt und für andere Projekte dies genau andersherum ist. Nur Albergate hat sowohl eine Verbesserung in der Präzision als auch Ausbeute aufzuweisen. Im Durchschnitt verbessert sich aber die Präzision um 2,5 und die Ausbeute um 11,9 Prozentpunkte. Dies führt dazu, dass das F₁-Maß im Durchschnitt um 6,5 Prozentpunkte steigt. Der Unterschied der Verbesserung in Präzision oder Ausbeute korreliert hierbei mit dem Anheben oder Verringern des Abschlusschwellenwerts. Wird dieser erhöht, so werden weniger Nachverfolgbarkeitsverbindungskandidaten verworfen und die Ausbeute steigt. Da allerdings nicht alle dieser Kandidaten tatsächlich korrekt sind, verringert sich

Tabelle 6.7.: Vergleich der festgelegten mit der optimierten Schwellenwertkombination (OPT) pro Projekt für FTLR-

	Präzision		Ausbeute		F ₁ -Maß		Schwellenwert			
		Δ		Δ		Δ	Mehrheits-		Abschluss-	Δ
eTour	.373		.633		.469		.590		.440	
eTour _{OPT}	.472	+0.099	.571	-0.062	.517	+0.048	.574	-0.016	.408	-0.032
iTrust	.142		.374		.206		.590		.440	
iTrust _{OPT}	.252	+0.110	.255	-0.119	.253	+0.048	.543	-0.047	.402	-0.038
SMOS	.432		.340		.381		.590		.440	
SMOS _{OPT}	.380	-0.053	.466	+0.126	.419	+0.038	.614	+0.024	.481	+0.041
eAnci	.273		.254		.263		.590		.440	
eAnci _{OPT}	.255	-0.018	.300	+0.046	.276	+0.012	.583	-0.007	.463	+0.023
Albergate	.171		.132		.149		.590		.440	
Albergate _{OPT}	.284	+0.113	.509	+0.377	.365	+0.216	.498	-0.092	.498	+0.058
LibEST	.431		.623		.509		.590		.440	
LibEST _{OPT}	.406	-0.025	.863	+0.240	.552	+0.043	.584	-0.006	.486	+0.046
∅		+0.038		+0.102		+0.067		-0.024		+0.016
∅ o. Albergate		+0.023		+0.046		+0.038		-0.010		+0.008

jeweils die Präzision leicht. Ebenso führt das Verringern des Abschlusschwellenwerts zu einer höheren Präzision, da hierdurch anscheinend mehr falsche Kandidaten verworfen werden. Da aber auch hier einige korrekte Verbindungen zu viel verworfen werden, führt dies zu einer leichten Senkung der Ausbeute. Mit Ausnahme von Albergate und LibEST korreliert das Verringern bzw. Anheben des Abschlusschwellenwerts auch mit der bereits vorliegenden, höheren Präzision als Ausbeute bzw. höheren Ausbeute als Präzision mit den festgelegten Schwellenwerten. Dies deutet darauf hin, dass die optimierten Schwellenwerte das Ergebnis einer ausgeglicheneren Verteilung von Präzision und Ausbeute annähern. Dies war zu erwarten, da hinsichtlich des F₁-Maßes optimiert wurde, welches das harmonische Mittel der beiden Maße darstellt. Der Mehrheitsschwellenwert wird bei allen Projekten außer SMOS verringert. Dies deutet darauf hin, dass der festgelegte Schwellenwert von 0,59 zu viele nicht semantiktragende Verbindungen zum Mehrheitsentscheid zulässt.


Die Ergebnisse für die -Variante (Tabelle 6.7) zeigen ein nahezu identisches Verhalten mit nur leichten Abweichungen hinsichtlich der Schwellenwerte für iTrust. Dies war zu erwarten, da der Einfluss der Aufrufabhängigkeiten insgesamt gering ist. Der Unterschied für iTrust könnte daran liegen, dass die Aufrufabhängigkeiten dort den negativsten Einfluss aufweisen und konsequent das Ergebnis verschlechtern (vgl. Tabelle 6.5).

Tabelle 6.8.: Vergleich der festgelegten mit der optimierten Schwellenwertkombination (OPT) pro Projekt für **F_{TLR}- Υ**

	Präzision		Ausbeute		F ₁ -Maß		Schwellenwert			
							Mehrheits-	Abschluss-		
	Δ		Δ		Δ		Δ		Δ	
eTour	.376		.643		.475		.590	-.022	.440	
eTour _{OPT}	.505	+ .129	.597	- .045	.548	+ .073	.568		.410	-.030
iTrust	.135		.322		.190		.590	-.019	.440	
iTrust _{OPT}	.193	+ .058	.241	- .080	.215	+ .024	.571		.403	-.037
SMOS	.425		.314		.361		.590	+.027	.440	
SMOS _{OPT}	.313	- .113	.590	+ .276	.409	+ .047	.617		.527	+ .087
eAnci	.312		.226		.262		.590	+.007	.440	
eAnci _{OPT}	.311	- .001	.272	+ .046	.290	+ .028	.597		.459	+ .019
Albergate	.171		.132		.149		.590	-.092	.440	
Albergate _{OPT}	.284	+ .113	.509	+ .377	.365	+ .216	.498		.498	+ .058
LibEST	.403		.490		.442		.590	+.005	.440	
LibEST _{OPT}	.421	+ .018	.799	+ .309	.552	+ .109	.595		.490	+ .050
∅		+ .034		+ .147		+ .083		-.016		+ .025
∅ o. Albergate		+ .018		+ .101		+ .056		+ .000		+ .018

Für die Υ -Variante (Tabelle 6.8) sind nun außer SMOS auch bei eAnci und LibEST die optimalen Mehrheitsschwellenwerte höher als der festgelegte Wert. Außerdem weist nun neben Albergate auch LibEST sowohl einen Zuwachs der Präzision als auch der Ausbeute auf. Dieses leicht andere Verhalten lässt sich mit den nun nicht mehr als Informationsquelle verwendeten Methodenkommentaren in dieser Variante erklären. Insgesamt sind aber auch hier die Unterschiede zu den anderen beiden Varianten gering. Wie bereits im vorhergehenden Abschnitt (s. Abschnitt 6.3) angenommen, verbessert sich für diese Variante das F₁-Maß durch die Optimierung am meisten. Die durchschnittliche Verbesserung beträgt hier 8,3 Prozentpunkte im Gegensatz zu 6,7 bei Υ und 6,5 bei Υ . Betrachtet man auch hier Albergate als Ausreißer und berechnet die durchschnittliche Verbesserung ohne Albergate, so ergibt sich nur noch eine Verbesserung von 5,6 Prozentpunkten bei Υ und 3,8 bzw. 3,5 bei Υ und Υ . Dies entspricht einer Verbesserung des durchschnittlichen F₁-Maßes von ursprünglich 36,6 % auf 40,3 % bzw. 40,1 % für Υ bzw. Υ und von ursprünglich 34,6 % auf 40,3 % bei Υ .

Um die Varianz der optimierten Schwellenwerte pro Projekt genauer zu betrachten, stellen die Abbildungen 6.1 und 6.2 die Verteilung des Mehrheits- bzw. Abschlusschwellenwerts für jede Variante über die Datensätze hinweg dar. Der Mehrheitsschwellenwert bewegt sich je nach Variante zwischen 0,644 und 0,488 wobei die niedrigen Werte aus dem Albergate-Datensatz stammen und bei fünf der acht Varianten als einfache Ausreißer

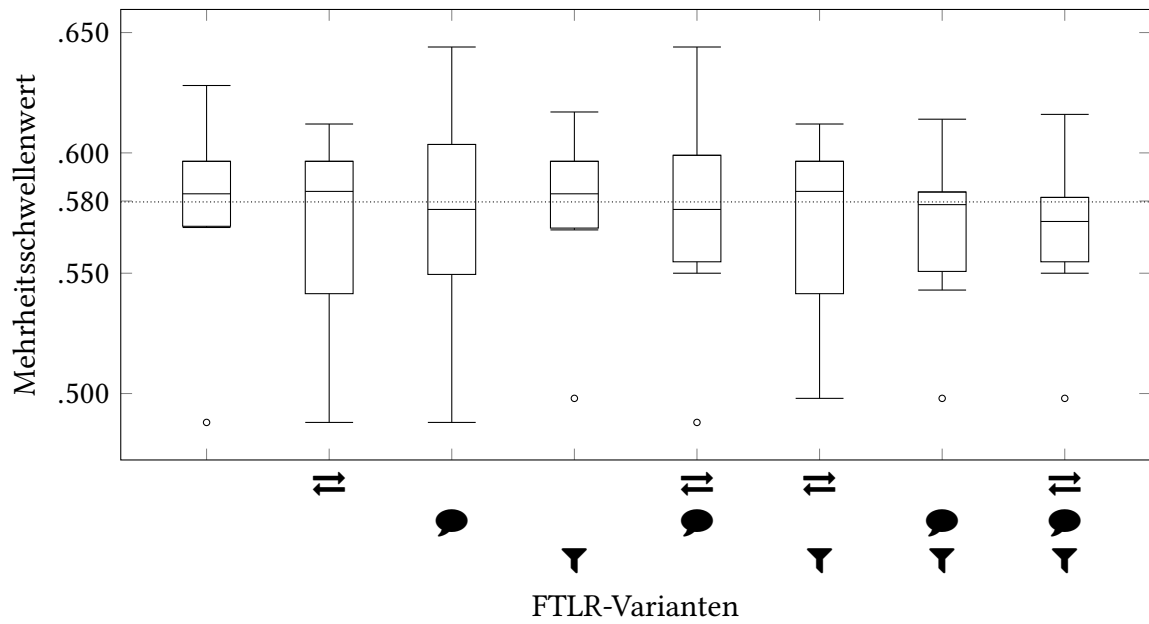


Abbildung 6.1.: Varianz der optimierten Mehrheitsschwellenwerte über die Datensätze hinweg². Der Durchschnitt der Mediane ist durch eine gepunktete Linie gekennzeichnet.

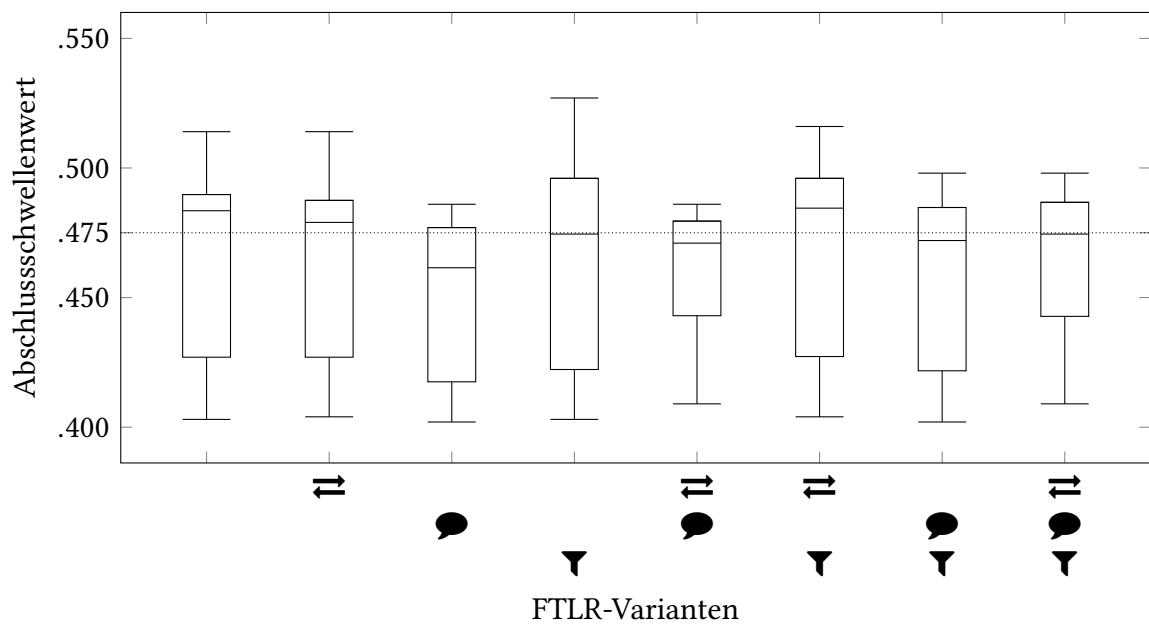


Abbildung 6.2.: Varianz der optimierten Abschlusschwellenwerte über die Datensätze hinweg. Der Durchschnitt der Mediane ist durch eine gepunktete Linie gekennzeichnet

angesehen werden können. Die maximalen Werte nach oben liegen allerdings innerhalb des als Kriterium für Ausreißer verwendeten Abstands des 1,5-Fachen des jeweiligen Interquartilsbereichs. Insgesamt deckt der Interquartilsbereich eine relativ kleine Spanne

² Somit deckt jeder Graph sechs Datenpunkte ab.

von maximal 0,044 ab. Für die Varianten mit der besten Leistung (T, ●T und ⇌●T) ist dieser sogar kleiner als 0,033. Dies bedeutet, dass 50 % der Mehrheitsschwellenwerte pro Variante innerhalb eines Bereichs liegen, der maximal 0,033 groß ist. Außerdem liegen die Mediane über alle Varianten hinweg mit einem Durchschnitt von 0,580 und einer Standardabweichung von 0,0043 sehr nahe beieinander. Dies deutet darauf hin, dass sich ein optimaler Schwellenwert über alle Projekte und Varianten hinweg festlegen lässt, welcher im Bereich um 0,580 liegt. Für die drei besten Varianten ist der Bereich den die Interquartilsbereiche abdecken [0,551; 0,597]. Der festgelegte Mehrheitsschwellenwert von 0,59 liegt also innerhalb dieses Bereichs. Da er allerdings am oberen Ende des Bereichs liegt, ist davon auszugehen, dass ein niedrigerer festgelegter Mehrheitsschwellenwert ein besseres Ergebnis über alle Datensätze hinweg erzielen könnte. Ein Kandidat hierfür wäre der Durchschnitt der Mediane 0,580.

Bei den Abschlusschwellenwerten (s. Abbildung 6.2) sind keine Ausreißer aufgetreten. Diese variieren zwischen 0,402 und 0,527 und decken damit einen um 0,031 kleineren Bereich ab, als die Mehrheitsschwellenwerte. Dafür sind die Interquartilsbereiche insgesamt mit Werten zwischen 0,037 und 0,073 größer als beim Mehrheitsschwellenwert. Aber auch beim Abschlusschwellenwert liegen die Mediane der Varianten wiederum dicht beieinander. Der Durchschnitt der Mediane beträgt 0,475 mit einer Standardabweichung von 0,0069. Der von den Interquartilsbereichen der drei besten Varianten abgedeckte Bereich ist für die Abschlusschwellenwerte im Intervall [0,422; 0,496]. Auch der festgelegte Abschlusschwellenwert (0,440) liegt also in diesem Bereich und stellt somit keine schlechte Wahl dar. Aber auch hier deutet die Streuung darauf hin, dass ein höherer Wert ein besseres Ergebnis über alle Datensätze hinweg ermöglichen könnte.

Um zu bemessen, ob derart angepasste feste Schwellenwerte einen positiven Effekt auf das Ergebnis haben, zeigt Tabelle 6.9 die Ergebnisse mit den beiden Durchschnitten der Mediane als Schwellenwerte im Vergleich zu den anderen Schwellenwertkombinationen. Hier zeigt sich, dass bezüglich des durchschnittlichen F_1 -Maßes eine solche Anpassung einen durchweg positiven Effekt hat. Ausschließlich auf dem eTour- und iTrust-Datensatz verschlechtert sich das F_1 -Maß durch die angepassten Schwellenwerte. Da die Verbesserung auf den anderen Datensätzen allerdings diese Verschlechterung überwiegt, ergibt sich eine Verbesserung im Durchschnitt des F_1 -Maßes. Die durchschnittliche Verbesserung, die über alle Varianten hinweg erzielt wurde, beträgt 0,2 Prozentpunkte. Es lässt sich also festhalten, dass die aus der Vorstudie gefolgerte Schwellenwertkombination zwar gute Ergebnisse erzielt, aber auf Basis eines größeren Wissens eine Schwellenwertkombination, wie die aus den Medianen abgeleitete, potenziell für die weitere Anwendung über Projektgrenzen hinweg vorzuziehen ist. Da ein solches Wissen aber ohne einen Goldstandard an TLs nicht ermittelbar ist, werden im folgenden alle drei Schwellenwertkombinationen in den Experimenten betrachtet. Die aus den Medianen abgeleitete Schwellenwertkombination wird von nun an mit MED und Ergebnisse die mit optimierten Schwellenwertkombinationen erzeugt wurden mit OPT gekennzeichnet. Die ursprünglich festgelegte Schwellenwertkombination wird mit ORG bezeichnet.

Tabelle 6.9.: Vergleich der Ergebnisse der Schwellenwertkombinationen bezüglich des F_1 -Maßes. ORG bezeichnet die Ergebnisse mit der ursprünglich festgelegten Schwellenwertkombination, MED diejenigen mit den angepassten festen Schwellenwerten (Mehrheit: 0,580, Abschluss: 0,475) und OPT die Ergebnisse der pro Projekt optimierten Schwellenwerte

Projekt		FTLR-Varianten							
		\rightleftharpoons	\bullet	\blacktriangledown	\rightleftharpoons	\rightleftharpoons	\bullet	\blacktriangledown	\rightleftharpoons
eTour	ORG	.391	.392	.385	.475	.388	.479	.469	.474
	MED	.355	.358	.358	.472	.365	.478	.461	.468
	OPT	.440	.442	.425	.548	.427	.552	.517	.517
iTrust	ORG	.190	.183	.206	.190	.209	.183	.206	.209
	MED	.166	.171	.172	.166	.174	.171	.172	.174
	OPT	.215	.210	.253	.215	.239	.210	.253	.239
SMOS	ORG	.369	.364	.382	.361	.383	.359	.381	.381
	MED	.364	.363	.387	.370	.386	.367	.390	.390
	OPT	.396	.394	.405	.409	.405	.415	.419	.419
eAnci	ORG	.241	.236	.240	.262	.236	.257	.263	.258
	MED	.247	.248	.242	.265	.242	.265	.268	.268
	OPT	.261	.264	.256	.290	.255	.292	.276	.277
Albergate	ORG	.246	.248	.246	.149	.248	.149	.149	.149
	MED	.335	.335	.335	.252	.335	.252	.252	.252
	OPT	.347	.347	.347	.365	.347	.365	.365	.365
LibEST	ORG	.442	.442	.509	.442	.509	.442	.509	.509
	MED	.524	.524	.538	.524	.538	.524	.538	.538
	OPT	.552	.552	.552	.552	.552	.552	.552	.552
	Ø ORG	.313	.311	.328	.313	.329	.312	.329	.330
	Ø MED	.332	.333	.339	.342	.340	.343	.347	.348
	Ø OPT	.368	.368	.373	.396	.371	.397	.397	.395

Vergleich der iTrust-Datensatzvarianten Wie in Abschnitt 6.1 beschrieben wurde bisher mit dem iTrust-Datensatz, der nur Java-Quelltext enthält, experimentiert, um das volle Potenzial von **FTLR** zu untersuchen. Um einschätzen zu können, ob sich Unterschiede im Verhalten von **FTLR** auf der vollständigen Variante von iTrust (inkl. JSP) ergeben, enthält Tabelle 6.10 den direkten Vergleich der Ergebnisse auf den beiden Datensatzvarianten. Sowohl für die ursprünglich festgelegte (ORG) als auch die MED-Schwellenwertkombination sind die Unterschiede über alle **FTLR**-Merkmalsvarianten hinweg sehr gering. Der größte absolute Unterschied liegt hier bei einem Rückgang von 0,8 Prozentpunkten für die $\rightleftharpoons\bullet$ - und $\rightleftharpoons\blacktriangledown$ -Varianten. Für die optimierte Schwellenwertkombination ist der Unterschied

Tabelle 6.10.: Vergleich der Ergebnisse der Schwellenwertkombinationen auf den beiden iTrust-Varianten bezüglich des F_1 -Maßes und des MAP. ORG bezeichnet die Ergebnisse im F_1 -Maß mit der ursprünglich festgelegten Schwellenwertkombination, MED diejenigen mit den angepassten festen Schwellenwerten (Mehrheit: 0,580, Abschluss: 0,475) und OPT die Ergebnisse der pro Projekt optimierten Schwellenwerte

		FTLR-Varianten							
		\rightleftharpoons	\bullet	\blacktriangledown	\rightleftharpoons	\rightleftharpoons	\bullet	\rightleftharpoons	\bullet
Projekt									
iTrust	ORG	.190	.183	.206	.190	.209	.183	.206	.209
	MED	.166	.171	.172	.166	.174	.171	.172	.174
	OPT	.215	.210	.253	.215	.239	.210	.253	.239
	MAP	.227	.213	.284	.227	.271	.213	.284	.271
iTrustJSP	ORG	.187	.182	.199	.187	.201	.182	.199	.201
	MED	.167	.171	.172	.167	.173	.171	.172	.173
	OPT	.207	.202	.235	.207	.223	.202	.235	.223
	MAP	.187	.179	.239	.187	.232	.179	.239	.232

zumindest für die beste Variante (\bullet bzw. $\bullet\blacktriangledown^3$) etwas größer. Hier verringert sich das F_1 -Maß von 25,3 % auf 23,5 %. Im MAP sind ähnlich große Reduzierungen festzustellen. Eine Erklärung hierfür könnte sein, dass durch die 141 JSP-Dateien zwar die Menge der Zielartefakte um 60 % erhöht wird, diese aber schwieriger korrekt abzubilden sind, da sie nur wenige nutzbare Informationen enthalten. In diesem Fall würde also die Präzision des Verfahrens leiden. Wären die JSP-Dateien besser dokumentiert oder könnte eine einheitliche Struktur zur Identifikation relevanter Informationen genutzt werden, so könnte dieses Problem abgemildert werden.

6.5. Vergleich mit Referenzimplementierungen

Um Aussagen über die Auswirkungen der getroffenen Entwurfsentscheidungen bezüglich der Ähnlichkeitsberechnung (FF_3) und der Ebene der Abbildung (FF_4) treffen zu können, wird in diesem Abschnitt die Leistung von **FTLR** mit den Referenzimplementierungen verglichen. Die in Abschnitt 5.2 beschriebenen Referenzimplementierungen umfassen den Einsatz der WMD auf Artefaktebene sowie den Einsatz der Kosinusdistanz sowohl auf Artefakt- als auch Elementebene. Der Vergleich liefert also Erkenntnisse bezüglich der Unterthesen $T_{1,2}$ und $T_{1,3}$. Diese nehmen an, dass die Wortüberführungsdistanz einer einfachen, aggregierten Vektorabbildung bzw. dass ein feingranulares Abbilden mit anschließender Aggregation einer Abbildung auf der Artefaktebene überlegen sind. Da für die Referenzimplementierungen keine festgelegten Schwellenwerte existieren, werden in

³ Da iTrust keine Anwendungsfallvorlagen verwendet sind die Ergebnisse für die mit \blacktriangledown gekennzeichneten Varianten gleich denen ohne \blacktriangledown .

Tabelle 6.11.: Vergleich der erzielten F_1 -Maße der Referenzimplementierungen mit **FTLR**. ECosS bezeichnet die Referenzimplementierung, welche Kosinusähnlichkeit auf Elementebene und ACosS diejenige, welche diese auf Artefaktebene einsetzt. Mit AWMD wird der Einsatz der Wortüberferungsdistanz auf Artefaktebene gekennzeichnet. Die besten Ergebnisse pro Datensatz und Variante sind fett dargestellt

Verfahren		eTour	iTrust	SMOS	eAnci	Albergate	LibEST	$\emptyset F_1$	
	ORG	FTLR	.385	.206	.382	.240	.246	.509	.328
	MED	FTLR	.358	.172	.387	.242	.335	.538	.339
☞	OPT	FTLR	.425	.253	.405	.256	.347	.552	.373
		ECosS	.407	.273	.363	.223	.202	.578	.341
		ACosS	.385	.252	.377	.205	.195	.562	.329
		AWMD	.435	.266	.424	.246	.279	.536	.364
	ORG	FTLR	.475	.190	.361	.262	.149	.442	.313
	MED	FTLR	.472	.166	.370	.265	.252	.524	.342
⚡	OPT	FTLR	.548	.215	.409	.290	.365	.552	.396
		ECosS	.486	.258	.380	.271	.176	.532	.351
		ACosS	.403	.187	.378	.252	.237	.445	.317
		AWMD	.422	.220	.426	.294	.220	.480	.344
	ORG	FTLR	.469	.206	.381	.263	.149	.509	.329
	MED	FTLR	.461	.172	.390	.268	.252	.538	.347
☞⚡	OPT	FTLR	.517	.253	.419	.276	.365	.552	.397
		ECosS	.443	.273	.382	.237	.176	.578	.348
		ACosS	.404	.252	.390	.224	.237	.571	.346
		AWMD	.433	.266	.447	.291	.220	.536	.366
	ORG	FTLR	.474	.209	.381	.258	.149	.509	.330
	MED	FTLR	.468	.174	.390	.268	.252	.538	.348
☞☞⚡	OPT	FTLR	.517	.239	.419	.277	.365	.552	.395
		ECosS	.445	.272	.387	.241	.176	.578	.350

diesem Experiment optimierte Schwellenwertkombinationen (OPT) für die Referenzimplementierungen verwendet. Daher kann auch ein Vergleich mit **FTLR** nur mit den Ergebnissen für die optimierten Schwellenwertkombinationen sinnvoll durchgeführt werden. Um die Größenordnungen trotzdem besser einschätzen zu können, enthält die Auswertung in Tabelle 6.11 außerdem auch die Ergebnisse von **FTLR** mit der ursprünglichen (ORG) und angepassten (MED) festen Schwellenwertkombination. Als betrachtete Merkmalsvarianten wurden die besten Varianten für die Referenzimplementierungen gewählt, welche mit den besten für **FTLR** übereinstimmen. Da für die Anwendung der Wortüberferungsdistanz auf Artefaktebene (AWMD) das zweitbeste Ergebnis durch die ☞-Variante erzielt wird, ist auch dieses Ergebnis enthalten. Die Aufrufabhängigkeiten lassen sich zudem nur

auf Elementebene einbeziehen, weshalb für die $\Rightarrow\bullet\Upsilon$ -Variante nur die Anwendung der Kosinusähnlichkeit auf Elementebene (ECosS) als Vergleich herangezogen wurde.

Tabelle 6.11 zeigt die erzielten F_1 -Maße der einzelnen Verfahren auf den Datensätzen sowie das durchschnittliche F_1 -Maß über die Datensätze hinweg. Informationen über die erzielte Präzision, Ausbeute und MAP enthalten Tabellen E.2 bis E.4 in Anhang E. Zunächst lässt sich feststellen, dass bezüglich des durchschnittlichen F_1 -Maßes **FTLR** mit optimierten Schwellenwerten die besten Ergebnisse für alle Merkmalsvarianten erzielt. Außerdem erzielt die $\bullet\Upsilon$ -Variante für die beiden auf Artefaktebene abbildenden Verfahren sowie für **FTLR** die besten durchschnittlichen Ergebnisse. Die Anwendung der Kosinusähnlichkeit auf Artefaktebene (ACosS) erreicht ein durchschnittliches F_1 -Maß von 34,6 %, AWMD von 36,6 % und **FTLR** von 39,7 %. Lediglich ECosS erzielt auf der Variante, die nur das Filtern von Vorlagenelementen durchführt (Υ), mit 35,1 % im durchschnittlichen F_1 -Maß sein bestes Ergebnis. Interessanterweise erzielt **FTLR** mit der angepassten Schwellenwertkombination (MED) ähnliche oder sogar bessere Ergebnisse als manche der optimierten Referenzimplementierungen. So ist das erzielte Gesamtergebnis von **FTLR** mit MED besser als die Anwendung von Kosinusähnlichkeit auf Artefaktebene und ähnlich wie die Anwendung von Kosinusähnlichkeit auf Elementebene. Dies bedeutet, dass selbst eine optimierte Schwellenwertkombination pro Projekt, welche eine obere Schranke der mit diesen Verfahren erzielbaren Güte darstellt, keine bessere Leistung als **FTLR** mit einem festgelegten Schwellenwert für alle Projekte erzielt. Vergleicht man zusätzlich die Ergebnisse der beiden Verfahren die WMD nutzen mit ihren Gegenstücken die Kosinusähnlichkeit einsetzen, so zeigt ein gepaarter Wilcoxon-Vorzeichen-Rang-Test auf den Werten des F_1 -Maßes über die Datensätze eine signifikante Verbesserung bezüglich des Signifikanzniveaus von 0,05. Hierzu wurde die beste Variante von ECosS mit **FTLR** auf dieser Variante (Υ) und die beste Variante von ACosS mit der jeweiligen Variante von AWMD ($\bullet\Upsilon$) verglichen. Der gepaarte Wilcoxon-Vorzeichen-Rang-Test ergibt einen p -Wert von 0,0127. Es lässt sich also eine signifikante Verbesserung der Abbildungsgüte durch den Einsatz der Wortüberführungsdistanz gegenüber einer einfachen, aggregierten Vektorabbildung feststellen (These $T_{1.2}$).

Betrachtet man nun die Unterschiede zwischen einer Abbildung auf Elementebene und einer Abbildung auf Artefaktebene unter der Hypothese, dass eine Abbildung auf Elementebene zu einer Verbesserung führt (These $T_{1.3}$), so kann für die beste Merkmalsvariante ($\bullet\Upsilon$) keine signifikante Verbesserung festgestellt werden ($p = 0,2349$). Dies liegt vor allem daran, dass die erzielte Verbesserung zwischen den beiden Kosinusähnlichkeitsverfahren bezüglich des Durchschnitts des F_1 -Maß sehr gering ist (0,002 Prozentpunkte). Für die beiden WMD-basierten Verfahren ist dieser mit 0,021 Prozentpunkten deutlich größer. Ebenso ist die Verbesserung, die auf der \bullet -Variante erzielt wird, nicht signifikant ($p = 0,0789$). Trotzdem ist eine Verbesserung über alle Datensätze hinweg festzustellen. Diese beträgt für die kosinusbasierten Verfahren 0,012 und für die WMD-basierten 0,009 Prozentpunkte im Durchschnitt des F_1 -Maßes. Betrachtet man allerdings die Υ -Variante, so ist eine signifikante Verbesserung mit einem p -Wert von 0,0261 festzustellen. Somit lässt sich abschließend nur für die Υ -Variante zeigen, dass die Abbildung auf Elementebene zu einer signifikanten Verbesserung auf diesen Datensätzen führt. Für die anderen Varianten

ist zwar eine Verbesserung im durchschnittlichen F_1 -Maß feststellbar, diese ist aber nicht signifikant und erlaubt damit keine abschließende Beurteilung.

Insgesamt zeigen diese Ergebnisse, dass die gewählten Entwurfsentscheidungen bezüglich des Verfahrens zur Ähnlichkeitsberechnung und der Ebene der Abbildung zu Verbesserungen der durchschnittlichen Abbildungsgüte geführt haben. Somit unterstützen sie die beiden Unterthesen $T_{1,2}$ und $T_{1,3}$. Betrachtet man allerdings die einzelnen Datensätze so ist **FTLR** nicht auf allen Datensätzen das überlegene Verfahren. Für eTour und Albergate erzielt **FTLR** die besten Werte für das F_1 -Maß. Auf iTrust ist hingegen ECosS das über die Varianten hinweg beste Verfahren. AWMD erzielt die höchsten Werte für das F_1 -Maß auf SMOS und eAnci. Auf LibEST erzielt zwar ECosS das insgesamt beste Ergebnis auf der $\bullet\blacktriangledown$ -Variante, doch auf den anderen Merkmalsvarianten ist dies jeweils ein anderes Verfahren. So erzielt ACosS das beste F_1 -Maß auf \bullet und **FTLR** auf \blacktriangledown . Es zeigt sich also, dass je nach Beschaffenheit des Datensatzes durchaus auch die Kosinusähnlichkeitsverfahren oder die Abbildung auf Artefaktebene vorteilhaft sein können. Der trotzdem signifikante Unterschied im Durchschnitt deutet aber darauf hin, dass **FTLR** besser generalisiert und somit für einen Einsatz über verschiedene Projekte hinweg geeigneter ist.

6.6. Gefährdungen der Gültigkeit

In diesem Abschnitt werden die Gefährdungen für die Gültigkeit der in diesem Kapitel durchgeführten Experimente und den aus ihnen gefolgerten Aussagen diskutiert.

Externe Gültigkeit Die erste und vermutlich auch gravierendste Gefährdung der Gültigkeit der Erkenntnisse dieser Arbeit bezieht sich auf die externe Gültigkeit. Die Experimente wurden auf einer begrenzten Menge von Projekten durchgeführt, welche zum größten Teil aus akademischen bzw. studentischen Projekten resultieren. Deshalb kann es sein, dass die erzielten Ergebnisse nicht repräsentativ für andere Projekte insbesondere aus der Industrie sind und damit die Ergebnisse eventuell nicht auf andere Projekte übertragen werden können. Diese Gefahr teilen die alle bestehenden Verfahren zur automatischen TLR (s. Abschnitt 3.1), da Datensätze mit vorhandenem Goldstandard für Nachverfolgbarkeitsverbindungen rar sind. Allerdings werden die ausgewählten Datensätze in vielen Studien in der Forschungsgemeinschaft zur Nachverfolgbarkeit verwendet und sind dort weitestgehend als Vergleichsmittel akzeptiert. Außerdem sind die Projekte von unterschiedlicher Größe, Sprache und decken verschiedene Domänen ab (s. Tabelle 6.1).

Zusätzlich verfügen alle Datensätze über mehr oder minder gut geschriebene Anforderungen. Es kann also sein, dass die erzielten Ergebnisse nicht ohne Weiteres auf Projekte mit weniger qualitativen Anforderungen übertragen werden können. Insbesondere das Ausnutzen der Anwendungsfallbeschreibungsvorlagen könnte schwieriger anwendbar sein, als es sich in diesen Experimenten herausstellte. **FTLR** benötigt jedoch an sich keine syntaktisch korrekten Sätze, sondern lediglich die für die Abbildung notwendigen semantischen Relationen zwischen den Wörtern der Artefakte.

Interne Gültigkeit Eine Gefahr für die interne Gültigkeit stellt die Übersetzung der Bezeichner in den Projekten dar. Diese Übersetzung wurde mit einem Wissen über das Ziel der automatischen TLR durchgeführt, könnte somit durch eine Form des Experimentator-effekts beeinflusst worden sein. Allerdings wurde die Übersetzung ohne Wissen bezüglich der zugehörigen Anforderungen durchgeführt. Die übersetzten Versionen der Datensätze können im Replikationspaket für **FTLR** [Hey23b] eingesehen und auf diesen Effekt überprüft werden.

7. Zusammenfassung

Dieser Teil der Arbeit hat sich mit der unüberwachten automatischen Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext beschäftigt. Hierzu wurde zunächst das Problem analysiert und aufgedeckt, dass bis zum heutigen Stand eine automatische Abbildung nur über die natürlichsprachlichen Elemente und deren Semantik möglich ist. Gerade das Überbrücken der potenziell vorhandenen semantischen Lücke zwischen den Artefakten erfordert einen semantischen Ähnlichkeitsvergleich.

Als Herangehensweise an dieses Problem wurde der Hauptbeitrag dieser Arbeit, das Verfahren **FTLR**, vorgestellt. Dieses repräsentiert zunächst Anforderungen und Quelltext, indem ihre enthaltenen Elemente durch Einbettungs-Multimengen dargestellt werden. Auf Basis dieser Elementrepräsentationen nutzt **FTLR** die Wortüberführungsdistanz, um Zusammenhänge zwischen den Elementen über ihre semantische Ähnlichkeit zu bestimmen. Diese feingranularen Zusammenhänge werden daraufhin dazu genutzt, um mittels eines Mehrheitsentscheids die vorherrschenden Aspekte pro Quelltextartefakt zu bestimmen und in Nachverfolgbarkeitsverbindungskandidaten zu überführen. Das Bestimmen dieser Zusammenhänge und der letztlichen Nachverfolgbarkeitsverbindungskandidaten wird dabei über zwei Schwellenwerte für die Wortüberführungsdistanz gesteuert. **FTLR** kann mit verschiedenen Merkmalsvarianten konfiguriert werden, die wahlweise Aufrufabhängigkeiten, Methodenkommentare oder ein Filtern von Vorlagenelementen umfassen.

In Experimenten auf sechs Vergleichsdatensätzen aus der Forschungsgemeinschaft zur Nachverfolgbarkeit konnte gezeigt werden, dass das Einbeziehen von strukturellen Informationen der Anforderungen zu einer signifikanten Verbesserung der Abbildungsgüte führen kann (Forschungsfrage **FF**₁ und These **T**_{1.1}). Ebenso konnte gezeigt werden, dass der Einsatz der Wortüberführungsdistanz gegenüber einfachen, aggregierten Vektorabbildungen (Forschungsfrage **FF**₃ und These **T**_{1.2}) und das Abbilden auf feingranularer Elementebene anstatt auf Artefaktebene (Forschungsfrage **FF**₄ und These **T**_{1.3}) zu signifikanten Verbesserungen führt. Eine Betrachtung der Schwellenwerte und ihrer angenäherten Optima pro Projekt zeigte, dass die Wahl der Schwellenwerte einen nicht zu vernachlässigenden Einfluss auf die Leistung des Systems hat, aber die Varianz über die Projekte hinweg eher gering ist (Forschungsfrage **FF**₂). Es kann also eine festgelegte Schwellenwertkombination bestimmt werden, die im Durchschnitt des F_1 -Maßes nur ein um 5 Prozentpunkte niedrigeres Ergebnis als optimierten Werte aufweist (MED: 34,7 % vs. OPT: 39,7 %).

Insgesamt zeigen die Experimente aber auch, dass die erzielte Abbildungsgüte noch weit von der für eine vollständige Automatisierung benötigten entfernt ist. Ein Grund hierfür ist auch bei **FTLR** die noch zu geringe Präzision auf akzeptablen Ausbeuteniveaus. Es

werden also immer noch zu viele TLs zwischen Artefakten erzeugt, die eigentlich nicht zusammenhängen. Um dieses Problem anzugehen, werden in den nächsten Teilen dieser Arbeit weitere Ansätze wie die Klassifikation von Anforderungen vorgestellt, welche als Erweiterung für **FTLR** eingesetzt werden können. Das Ziel ist es, diese Verfahren dazu zu verwenden, eine höhere Abbildungsgüte durch **FTLR** zu erreichen.

Teil III.

Anforderungsklassifikation als Vorfilter

8. Anforderungsklassifikation zur Filterung von Eingaben in FTLR

Natural language can allow requirements to be ambiguous, and requirements of different types and perspectives are in danger of being unintentionally mixed up during documentation.

(Klaus Pohl [Poh16])

Im vorherigen Teil dieser Arbeit wurde **FTLR** als Verfahren zur automatischen TLR vorgestellt. **FTLR** zielt auf eine Automatisierung in einer Situation in der keine bestehenden TLs zur Verfügung stehen ab (Ziel 1). Dies heißt insbesondere, dass das Verfahren ohne vorheriges Wissen über die Ausprägung der Anforderungen und den bedeutungstragenden Teilen des Quelltexts auskommen muss. Die Ergebnisse der Evaluation in Kapitel 6 zeigen, dass unter diesen Rahmenbedingungen noch keine Ergebnisse erzielt werden können, die eine vollständige Automatisierung des Prozesses zulassen würden. **FTLR** erzielt, wie bisherige Verfahren auch, zu niedrige Präzisionswerte auf Ausbeuteniveaus, die es für einen Einsatz in der Praxis benötigen würde.

Ein wichtiger Faktor, der die Präzision bestimmt, ist die Auswahl der für die Bestimmung der TLs relevanten Teile der Artefakte. Dies bestätigen die höheren Präzisionen durch die Filterung der Vorlagenelemente (▼-Varianten). Hier werden die vorhandenen Beschriftungen der Anwendungsfallbeschreibungsvorlagen dazu genutzt, bestimmte Teile der Anforderung herauszufiltern. Aber auch die Vorlagenelemente, welche grundsätzlich als für die Abbildung relevant angenommen wurden, enthalten nicht ausschließlich für die TLR nützliche Informationen. So enthält beispielsweise die Ablaufbeschreibung des eTour-Anwendungsfalls UC57 in Anhang D den Satz „*Fill in the form of advanced search and submit.*“, welcher eine Nutzerinteraktion innerhalb des Anwendungsfalls beschreibt. Diese kann nicht ohne Weiteres auf eine Funktionalität bzw. eine Methode oder Klasse des Systems abgebildet werden, da das Ausfüllen eines Formulars keine explizite Funktionalität des Systems ist. Ebenso können auch Anforderungen, die keiner Vorlage folgen, Teile enthalten, die irrelevant für die Abbildung auf Quelltext sind. So enthalten mehrere Anforderungen des iTrust-Datensatzes die Aussage „*The patient chosen is not the desired patient.*“, welche zwar eine Bedingung für den darauf folgenden Teil der Anforderung darstellt, aber keine Funktionalität des Systems beschreibt. Auch auf derartigen Projekten wäre also eine Auswahl der abzubildenden Teile der Anforderungen relevant. Werden diese Teile nicht herausgefiltert, so können sie zu fälschlicherweise identifizierten Zusammenhängen zwischen Artefakten führen und damit die Präzision der Wiederherstellung von Nachverfolgbarkeitsverbindungen verringern.

Das Bestimmen des Inhalts und der Art bzw. Kategorie von Anforderungen ist ein weitverbreitetes Forschungsthema in der Anforderungstechnik (engl.: *requirements engineering*, RE). Die sogenannte Anforderungsklassifikation (engl.: *requirements classification*) beschäftigt sich mit dem Einsatz von maschinellen Lernverfahren für die Klassifikation von Anforderungen. Dies umfasst Aufgaben wie die Identifikation von Anforderungen in Anforderungsdokumenten [SW13; WV16], die Kategorisierung ganzer Anforderungen in funktional bzw. nichtfunktional [Cle+06; KM17; Aba+17] oder Unterkategorien dieser sowie der Bestimmung der vorliegenden Anliegen (engl.: *concerns*) bzw. Aspekte in Teilen der Anforderungen [Hey+20a]. Die durch solche Verfahren generierten Informationen bieten also das Potenzial, ein genaueres Verständnis der Anforderungen und ihrer Teile zu erhalten. Dieses Verständnis wiederum könnte dazu verwendet werden, Teile der Anforderungen zu filtern und damit die Präzision **FTLRs** zu verbessern.

Um dieses Potenzial genauer zu untersuchen, beschäftigt sich dieser Teil der Arbeit mit dem Einsatz von Anforderungsklassifikation für das Vorfiltern von Anforderungselementen in **FTLR** und liefert damit Erkenntnisse bezüglich These **T₂**:

T₂: Durch geeignete Klassifikation der Anforderungen können irrelevante Teile der Eingabe ausgeschlossen werden und damit die Präzision eines worteinbettungs-basierten Verfahrens zur Nachverfolgbarkeit gesteigert werden.

Hieraus leiten sich die folgenden Ziele dieses Teils der Arbeit ab:

Ziel 6: Erhöhen der Präzision **FTLRs** durch geeignetes Filtern von Anforderungselementen mittels Anforderungsklassifikation

Unterziel 6.1: Bestimmen von Klassen aus der Anforderungsklassifikation, die sich für das Filtern irrelevanter Anforderungselemente eignen

Unterziel 6.2: Identifikation geeigneter Integrationsmöglichkeiten in **FTLR**

Unterziel 6.3: Auswahl eines geeigneten Anforderungsklassifikationsverfahrens zur Bestimmung der Klassen

Um diese Ziele zu erreichen, werde ich zunächst in Abschnitt 8.1 analysieren, welche durch Anforderungsklassifikation ermittelbaren Informationen zur Bestimmung (ir)relevanter Anforderungselemente für die automatische TLR genutzt werden können. Daraufhin wird in Abschnitt 8.2 die Integration der Klassifikation in **FTLR** beschrieben. Um Unterziel 6.3 zu erreichen, wird in Abschnitt 9.1 zunächst ein Überblick über bestehende Ansätze zur Anforderungsklassifikation gegeben und in Abschnitt 9.2 das in dieser Arbeit verwendete Klassifikationsverfahren vorgestellt. In den Abschnitten 9.3 und 9.4 wird die Güte der Klassifikation des gewählten Verfahrens in einer intrinsischen Evaluation bemessen.

Abschließend wird in Kapitel 10 der Einfluss der Klassifikation bei einer Integration in **FTLR** bemessen.

8.1. Informationen aus der Anforderungsklassifikation

In der Anforderungsklassifikation werden typischerweise die folgenden Aufgaben adressiert: Eine trennscharfe Klassifikation von Anforderungen in funktional bzw. nichtfunktional, eine überlappende Klassifikation von vorhandenen funktionalen oder nichtfunktionalen Aspekten in den Anforderungen und die Verfeinerung der Klassifikation von nichtfunktionalen Anforderungen in Unterklassen. Diese Aufgaben beziehen sich auf die Klassifikation von ganzen Anforderungen oder Teilen dieser. Das heißt, als Eingabe für die Klassifikation werden einzelne Anforderungen erwartet. Liegen Anforderungen allerdings nicht einzeln vor, wird oftmals auch der Schritt der Identifikation von Anforderungen in einem Anforderungsdokument als Anforderungsklassifikationsschritt betrachtet. Die Anforderungsklassifikation wird in der RE als vorbereitender Schritt für weitere Analysen gesehen. Sie liefert Informationen über die Ausprägung der Anforderung und den zu erwartenden Inhalt bzw. den adressierten Aspekt des Systems. Diese Information ist für eine Vielzahl von nachgelagerten Analysen hilfreich. So können beispielsweise Entwickler:innen und auch weitere Projektbeteiligte dabei unterstützt werden, dass Anforderungen frühzeitig identifiziert werden [Cle+06; Kna+12] oder es kann verhindert werden, dass Anforderungen übersehen oder missinterpretiert werden. Eine weitere mögliche nachgelagerte Analyse stellt die automatische TLR dar. Bisher wurde Anforderungsklassifikation nicht als Vorverarbeitungsschritt für die Wiederherstellung von Nachverfolgbarkeitsverbindungen genutzt. Daher wird in diesen Abschnitt zunächst analysiert, welche der Informationen, die Anforderungsklassifikation liefern kann, für die automatische TLR genutzt werden können.

Bereits die grundlegendste Aufgabe der Anforderungsklassifikation, die Identifikation von funktionalen und nichtfunktionalen Anforderungen, ist hilfreich für das Erfüllen des Ziels der Bestimmung relevanter bzw. irrelevanter Informationen für die TLR. Auch wenn eine harte Trennung zwischen funktionalen und nichtfunktionalen Anforderungen kontrovers diskutiert wird [Gli07; Li+14; EVF16], so ist die Information, dass eine Anforderung oder ein Teil dieser hauptsächlich nichtfunktionale Eigenschaften beschreibt, ein Grund, diese bei der Bestimmung der Nachverfolgbarkeit herauszufiltern. Denn nichtfunktionale Anforderungen, auch manchmal als weiche Ziele (engl.: *softgoals*) oder Qualitätseinschränkungen bezeichnet [MCN92; Chu+00], beschreiben in erster Linie Eigenschaften des Systems, welche keine funktionale Entsprechung haben und somit höchstens indirekt im Quelltext abgebildet werden. Dies bedeutet insbesondere aber auch, dass es keine Quelltextelemente gibt, die in einer direkten Verfolgbarkeitsbeziehung zu diesen nichtfunktionale Anforderungen stehen. Aus diesem Grund können diese für die automatische TLR herausgefiltert werden. Die Diskussion bezüglich der Trennung zwischen funktionalen und nichtfunktionalen Anforderungen bezieht sich hierbei darauf, dass auch nichtfunktionale Anforderungen zu Entwurfsentscheidungen führen können, die sich in Funktionalitäten

des Systems widerspiegeln. So könnte z. B. die nichtfunktionale Anforderung „Das System sollte verschiedene Sprachen unterstützen.“ indirekt dazu führen, dass eine Sprachwahl-funktionalität implementiert wird. Deshalb schlagen verschiedene Forscher:innen statt einer harten Trennung zwischen funktional und nichtfunktional, eher eine überlappende Klassifikation in Anforderungen, die funktionale bzw. nichtfunktionale Aspekte enthalten, vor [Bro15; EVF16; Dal+19]. Auch in dieser Klassifikationsvariante lassen sich Anforderungen bestimmen, die ausschließlich nichtfunktionale Aspekte umfassen und somit für die TLR irrelevant sein sollten. Geht man außerdem davon aus, dass diejenigen Anforderungen, die neben nichtfunktionalen auch funktionale Aspekte enthalten, die funktionalen Aspekte eher implizit beschreiben, so könnte es ebenfalls sinnvoll sein, diese aus der Betrachtung auszuschließen.

Neben dieser sehr groben Klassifikation von Anforderungen wird oftmals eine Verfeinerung der Klassen vorgenommen. Insbesondere nichtfunktionale Anforderungen werden häufig in Unterklassen kategorisiert. Eines der am weitesten verbreiteten Klassifikationsschemata unterscheidet hierbei zwischen den 11 nichtfunktionalen Unterklassen: Verfügbarkeit (engl.: *Availability*), Ausfallsicherheit (engl.: *Fault Tolerance*), Rechtliches (engl.: *Legal*), Erscheinungsbild (engl.: *Look & Feel*), Wartbarkeit (engl.: *Maintainability*), Betrieb (engl.: *Operational*), Leistungsfähigkeit (engl.: *Performance*), Übertragbarkeit (engl.: *Portability*), Skalierbarkeit (engl.: *Scalability*), Sicherheit (engl.: *Security*) und Benutzerfreundlichkeit (engl.: *Usability*) [Cle+06]. Diese Art der Verfeinerung bietet ein deutlich präziseres Bild bezüglich des nichtfunktionalen Aspekts, den die Anforderung beschreibt. Da es sich aber hierbei ausschließlich um nichtfunktionale Anforderungen handelt, sind diese Art von Informationen weitestgehend irrelevant für die TLR. Möglicherweise ließen sich zwar für diese Unterklassen Wahrscheinlichkeiten ableiten, um auszudrücken, wie wahrscheinlich es ist, dass diese Anforderungen implizite Funktionalität beinhalten¹. Jedoch lässt sich hier keine verlässliche Aussage treffen, weshalb die Bestimmung der Unterklassen nichtfunktionaler Anforderungen für die automatische TLR nicht vorteilhaft ist.

Die Verfeinerung funktionaler Anforderungen spielt in der Anforderungsklassifikation hingegen eine deutlich kleinere Rolle. Ein Grund hierfür könnte sein, dass für bisherige nachgelagerte Analysen vor allem die Aspekte nichtfunktionaler Anforderungen eine Rolle spielen, da z. B. funktionale Anforderungen seltener übersehen oder zu spät erhoben werden. Gerade eine solche Verfeinerung könnte aber bei der Vorverarbeitung für eine automatische TLR hilfreich sein. Glinz [Gli07] unterteilt funktionale Anforderungen anhand der durch sie beschriebenen Anliegen (engl.: *concerns*) in Funktionen (engl.: *Functions*), Daten (engl.: *Data*), Auslöser (engl.: *Stimuli*), Reaktionen (engl.: *Reactions*) und Verhalten (engl.: *Behavior*). Ghazarian [Gha12] identifiziert in einer Studie auf 15 webbasierten Unternehmenssoftwaresystemen folgende Unterkategorien funktionaler Anforderungen: Dateneingabe (engl.: *Data Input*), Datenausgabe (engl.: *Data Output*), Datenvalidierung (engl.: *Data Validation*), Datenspeicherung (engl.: *Data Persistence*), Anwendungslogik (engl.: *Business Logic*), Kommunikation (engl.: *Communication*), Ereignisauslöser (engl.: *Event*

¹ Vermutlich enthalten Erscheinungsbild, Sicherheit und Benutzerfreundlichkeit eher funktionale Aspekte als Rechtliches oder Verfügbarkeit.

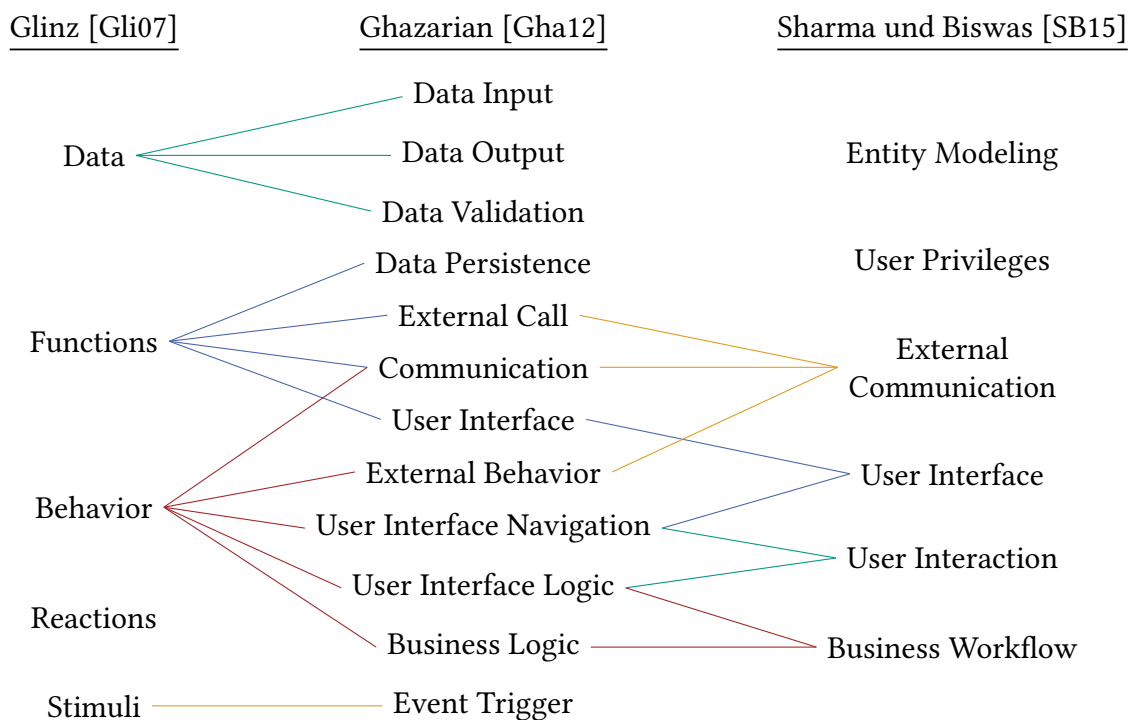


Abbildung 8.1.: Illustration der Unterklassifizierungen von funktionalen Anforderungen aus verschiedenen bestehenden Arbeiten und wie diese zusammenhängen

Trigger), Nutzerschnittstellennavigation (engl.: *User Interface Navigation*), Nutzerschnittstelle (engl.: *User Interface*), Nutzerschnittstellenlogik (engl.: *User Interface Logic*), externer Aufruf (engl.: *External Call*) und externes Verhalten (engl.: *External Behavior*). Sharma und Biswas [SB15] argumentieren, dass Ghazarians Kategorien zu spezifisch wären und schlagen stattdessen eine der Problemdomäne nähere Aufteilung vor. Ihre Taxonomie umfasst die Kernkategorien Entitätsmodellierung (engl.: *Entity Modelling*), Nutzerschnittstelle (engl.: *User Interface*), Nutzerprivilegien (engl.: *User Privileges*), Nutzerinteraktion (engl.: *User Interaction*), Betriebsabläufe (engl.: *Business Workflow*), Betriebseinschränkungen (engl.: *Business Constraints*) und externe Kommunikation (engl.: *External Communication*). Allerdings enthalten einige der Kategorien laut ihrer Beschreibung auch Elemente, die in der Forschungsgemeinschaft weitestgehend als nichtfunktionale Anforderungen angesehen werden. So enthalten Nutzerschnittstellenanforderungen auch Erscheinungsbildbeschreibungen. Die Kategorie Betriebseinschränkung, welche ihrer Beschreibung nach regulatorische Normen, Richtlinien und technische Einschränkungen umfasst, wird normalerweise gänzlich als nichtfunktionale Anforderung des Typs Einschränkung aufgefasst. Dies macht die von Sharma und Biswas vorgeschlagene Aufteilung nicht ohne Weiteres anwendbar für die TLR.

Abbildung 8.1 stellt die Klassen der drei Arbeiten in Bezug zueinander. Diese Grafik wurde anhand der in den Arbeiten verwendeten Definitionen der einzelnen Klassen angefertigt. Hierbei wurden die als nichtfunktional einzuordnenden Klassen aus Sharma und Biswas [SB15] weggelassen, da diese für die TLR nicht geeignet sind. Die Grafik hebt hervor,

dass die Einordnung von Ghazarian [Gha12] letztlich nur eine Verfeinerung der Klassen von Glinz [Gli07] darstellt und dabei nur das Anliegen Reaktion auslöst. Die Kategorien von Sharma und Biswas [SB15] hingegen überdecken nur einen Teil der Klassen von Glinz und Ghazarian, verbinden diese allerdings eher entlang der Problemdomäne als entlang der Art des dahinterliegenden Anliegens. Außerdem konnten die beiden Klassen Entitätsmodellierung und Nutzerprivilegien nicht eindeutig einer der bestehenden Klassen von Ghazarian zugeordnet werden. Sie beziehen sich vielmehr auf Rollen, Entitäten oder Konzepte, welche die Nutzer eines Systems einnehmen können.

Ein weiterer Faktor, der für eine Auswahl geeigneter Klassen in Betracht gezogen werden muss, ist der Detailgrad der Klassifikation. Eine feingranularere Unterteilung ermöglicht es zwar, das jeweilige Anliegen präziser zu bestimmen, kann aber dazu führen, dass Gemeinsamkeiten zwischen den Klassen außer Acht gelassen werden. Zusätzlich brauchen automatische Klassifikationsverfahren ausreichend Trainingsdaten einer jeden Klasse, um diese präzise bestimmen zu können. Eine feinere Unterteilung kann also dazu führen, dass Klassen in den Trainingsdaten unterrepräsentiert sind und damit nicht oder nur sehr unpräzise erkannt werden. Die bestehende Datenlage für Anforderungsklassifikation ist insgesamt als gering einzustufen und die meisten Verfahren basieren auf den zwei Datensätzen PROMISE NFR (PNFR) [Cle+07b] und PURE [FSG18]. Für die Klassifikation von Unterklassen funktionaler Anforderungen existiert nur der von uns erstellte, auf dem PROMISE NFR-Datensatz basierende Datensatz [Hey+20b] mit den drei Klassen Funktion, Verhalten und Daten nach Glinz. Aufgrund des Mangels an Trainingsdaten und der Möglichkeit der Identifikation von Gemeinsamkeiten, habe ich mich in dieser Arbeit für eine gröbere Klassifikation auf Basis der Anliegen nach Glinz [Gli07] entschieden. Diese sind gröber als die Kategorien nach Ghazarian [Gha12], aber umfassender und in der Forschungsgemeinschaft anerkannter, als die Aufteilung von Sharma und Biswas [SB15].

Zusätzlich zu den Kategorien nach Glinz stellen allerdings die nutzerbezogenen Kategorien aus der Arbeit von Sharma und Biswas [SB15] eine vielversprechende Ergänzung dar. Denn Anwendungsfallbeschreibungen und Anwender:innenerzählungen enthalten oftmals Aussagen, die sich lediglich auf das Verhalten von Nutzer:innen beziehen und selten eine direkte Entsprechung im Quelltext, z. B. in Form von Methoden, besitzen. So bezieht sich der Inhalt des Anforderungselements „*The user reads the displayed items.*“ nur auf eine Aktion der Nutzer:in, die nicht im System selbst stattfindet und damit irrelevant für die TLR ist. Solche Teile der Anforderung helfen zwar dEntwickler:innen beim Verständnis des zu implementierenden Systems, sind aber für eine automatische TLR, die auf Methodenebene arbeitet, vermutlich irrelevant. Um auch für auf Nutzer:innen bezogene Kategorien eine angemessene Anzahl an Trainingsdaten pro Klasse erzielen zu können, lassen sich die Klassen unter dem Anliegen Nutzerbezogenes (engl.: *user-related*, NB) zusammenfassen. Geht man außerdem davon aus, dass Reaktionen und Stimuli ebenso eine Form des Verhaltens des Systems darstellen, können die Klassen für die Klassifikation auf Funktion, Verhalten, Daten und Nutzerbezogenes reduziert werden. Folgende Definitionen geben die in dieser Arbeit verwendeten Definitionen der Klassen an:

Definition 12: Klasse Funktion

Beschreibungen der Aufgaben, Funktionalitäten und Operationen, die ein System im Rahmen seines Funktionsumfangs anbieten soll, um einen bestimmten Zweck zu erfüllen

Beispiel:

The system shall allow a real estate agent to query MLS information.

Definition 13: Klasse Verhalten

Beschreibungen des Zusammenwirkens der Funktionalitäten des beschriebenen Systems sowie Reaktionen, die durch einen oder mehrere Stimuli ausgelöst werden

Beispiel:

If there are no expired prescriptions, an empty list of expired prescriptions is displayed.

Definition 14: Klasse Daten

Beschreibungen von Datenelementen oder Datenstrukturen, die Teil des Systemzustands sein sollen

Beispiel:

The audit report shall include the total number of recycled parts used in the estimate.

Definition 15: Klasse Nutzerbezogenes

Beschreibung des Verhaltens der Nutzer:in bzw. dem/der Nutzer:in zuzuordnenden Funktionalitäten des Systems

Beispiel:

The user reads the instructions and completes the form.

Hierbei sei zu erwähnen, dass sich die Klassen keinesfalls gegenseitig ausschließen, sondern einzelne Beschreibungen mehrere dieser Klassen gleichzeitig erfüllen können. Insbesondere enthalten viele Verhaltensbeschreibungen die Funktionalitäten des Systems, welche miteinander interagieren. Ebenso beinhalten viele Funktionsbeschreibungen Hinweise auf das Verhalten des Systems oder die verwendeten Daten. Wird allerdings lediglich die angebotene Funktionalität genannt (wie im Beispiel in Definition 12), so fasse ich dies als reine Funktionsbeschreibung auf.

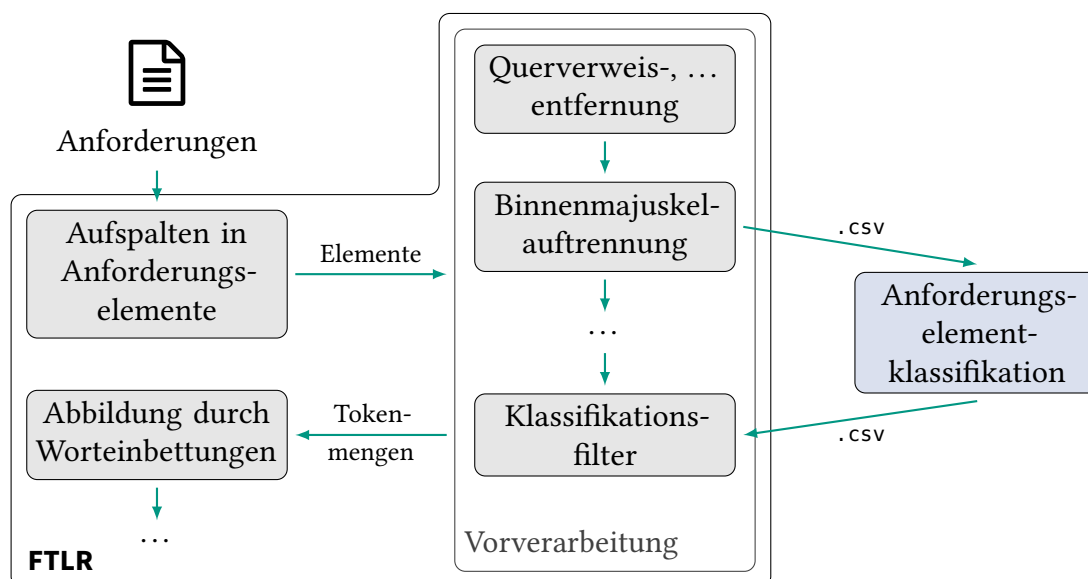


Abbildung 8.2.: Ablauf der Integration der Anforderungselementklassifikation in FTLR

8.2. Integration in FTLR

Die Ergebnisse in Kapitel 6 haben gezeigt, dass die feingranulare Vorgehensweise von **FTLR** vorteilhaft für die TLR ist. Die Abbildung von Anforderungs- auf Quelltextelemente mit anschließender Aggregation erzielt bessere Ergebnisse als eine artefaktweise Betrachtung. Außerdem ist das Ausnutzen von Anwendungsfallbeschreibungsvorlagen zur Filterung der zu betrachtenden Anforderungselemente durchweg vorteilhaft für die Abbildungsgüte. Diese Ergebnisse deuten darauf hin, dass auch ein Filtern einzelner Anforderungselemente durch Informationen aus einer Anforderungsklassifikation hilfreich für die Abbildungsgüte sein könnte. Die meisten Verfahren zur Klassifikation von Anforderungen klassifizieren auf Artefaktebene, d. h. sie klassifizieren ganze Anforderungen. Da diese aber als Eingabe zumeist klassische Anforderungen, bestehend aus ein bis wenigen Sätzen, erwarten und nicht auf ganze Anwendungsfallbeschreibungen angewandt werden, unterscheidet sich diese Art der Eingabe nicht maßgeblich von den Artefaktelementen wie sie **FTLR** betrachtet. Diese bestehen ebenfalls aus einem oder wenigen Sätzen. Eine Klassifikation ganzer Anwendungsfallbeschreibungen wäre zu grob, um ein sinnvolles Vorfiltern der Informationen zu gewährleisten. So kann z. B. eine Anwendungsfallbeschreibung weitestgehend funktionale Aspekte des Systems beschreiben, aber trotzdem einen Satz zu Qualitätsanforderungen oder eine Beschreibung einer Nutzerinteraktion beinhalten. Würde man nun auf Artefaktebene klassifizieren, erhielte man sowohl die Information, dass funktionale Aspekte als auch nichtfunktionale Aspekte beschrieben werden. Diese Information ist für die Weiterverarbeitung in **FTLR** wenig hilfreich, da weder die Anforderung gänzlich ignoriert noch die tatsächlich für die Abbildung irrelevanten Stellen identifiziert werden können. Daher kann und sollte die Klassifikation der Anforderungen für **FTLR**

auf Artefaktelementebene erfolgen. Somit setzt dieser Schritt nach dem Aufspalten der Anforderungselemente an.

Um die für die Verarbeitung durch ein Klassifikationsverfahren nutzbaren Informationen möglichst umfassend zu halten und somit eine Austauschbarkeit der Klassifikationskomponente zu gewährleisten, sollten nur die beiden ersten Schritte der Vorverarbeitung auf die Anforderungselemente angewandt werden. Dies sind die Querverweis-, Sonderzeichen- und Nummerentfernung sowie die Binnenmajuskelauftrennung (s. Abschnitt 5.1.1). So können auch Verfahren, die auf vollständigen natürlichsprachlichen Anforderungen vortrainiert wurden, eingesetzt werden. Die derart vorverarbeiteten Anforderungselemente können daraufhin in Textform an einen oder mehrere Klassifikatoren übergeben werden, welche bestimmen, ob es sich bei dem Anforderungselement um eine Ausprägung der jeweiligen Klassen handelt. Die Ergebnisse der Klassifikation werden daraufhin in Form einer csv-Datei an **FTLR** zurückgespielt und können für weitere Schritte der Analyse genutzt werden. Die Klassifikation der Anforderungselemente kann also im Rahmen von **FTLR** als neuer Vorverarbeitungsschritt angesehen werden. Dieses Vorgehen ist in Abbildung 8.2 dargestellt, wobei die Klassifikation selbst an ein externes Werkzeug ausgelagert werden oder auch manuell erfolgen kann. Durch die Wahl einer Datei als Übergabeformat können bestehende Verfahren zur Klassifikation, welche zumeist auf csv-Dateien arbeiten und ggf. in anderen Programmiersprachen verfasst sind oder aus Leistungsgründen sogar auf anderen Maschinen laufen, ohne eine Anpassung an **FTLR** eingebunden werden. Auf Basis der Klassifikationsergebnisse können daraufhin einzelne Anforderungselemente aus der Repräsentation eines Anforderungsartefakts gefiltert und somit aus der Betrachtung für die Nachverfolgbarkeit ausgeschlossen werden. Sollten sogar alle Anforderungselemente eines Anforderungsartefakts herausgefiltert werden, wird diese gänzlich aus der TLR entfernt.

Von den in Abschnitt 8.1 identifizierten Klassen eignen sich allerdings nur das Vorhandensein einer funktionalen Beschreibung und die nutzerbezogenen Artefaktelemente zur Auswahl relevanter Anforderungselemente. Denn Artefaktelemente, welche eine konkrete Funktion, Verhalten oder Daten beschreiben sind genau die Elemente, welche auf Quelltextartefakte abgebildet werden müssen. Sie würden sich also für eine genauere Abbildung, aber nicht für eine generelle Filterung, eignen. Somit ist die übergeordnete Klassifikation der Elemente als funktionale Aspekte enthaltend sinnvoller. Mit dieser lassen sich dann alle Anforderungselemente ignorieren, welche keine funktionalen Aspekte beinhalten. Man bemerke, dass ein Element, welches keine funktionalen Aspekte beinhaltet, nicht zwangsläufig als nichtfunktionale Anforderung im Sinne der Definition (s. Abschnitt 2.2) anzusehen ist. Es enthält lediglich natürlichsprachlichen Text, der eben keine funktionalen Aspekte beinhaltet. Da solche Formulierungen insbesondere in Anwendungsfallbeschreibungen häufiger auftreten, ist die Klassifikation in funktionale und/oder Qualitätsaspekte (vgl. [Bro15; EVF16; Dal+19]) einer direkten Klassifikation in funktional bzw. nichtfunktional vorzuziehen. Hieraus ergibt sich als möglicher Filter, den **FTLR** auf Basis der Anforderungselementklassifikation einsetzen kann, das Entfernen all jener Anforderungselemente, die keine funktionalen Aspekte beinhalten. Dieser Filter wird im Folgenden als Nichtfunktional-Filter (NF-Filter) bezeichnet. Um zusätzlich auch diejenigen

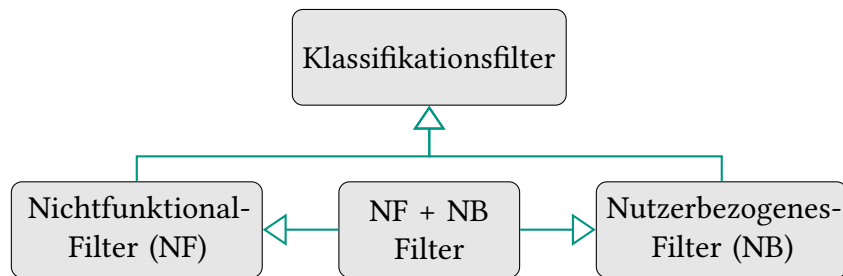


Abbildung 8.3.: Darstellung der in **FTLR** integrierten Anforderungselementfilter

Elemente zu entfernen, die zwar funktionale Aspekte, aber hauptsächlich Nutzerbezogenes beinhalten, kann **FTLR** außerdem den Nutzerbezogenes-Filter (NB-Filter) verwenden.

FTLR erlaubt es außerdem diese Menge an Filtern beliebig zu erweitern und in Kombination miteinander zu verwenden. Abbildung 8.3 zeigt, die beiden soeben beschriebenen Filter und ihre Kombination. Den Einsatz der im vorangehenden Abschnitt identifizierten Klassen Funktion, Verhalten und Daten für ein automatisches TLR-Verfahren wird dabei zukünftigen Arbeiten überlassen. Denn diese ließen sich lediglich für eine passgenauere Abbildung auf Quelltextelemente nutzen. Da **FTLR** allerdings einen auf sprachlichen Ähnlichkeiten basierenden Ansatz umsetzt, würde für derartige Abbildungen ein Verständnis der Kategorien von Quelltextelementen benötigt. Da dieses bisher nicht vorhanden ist, behandelt diese Arbeit lediglich die Güte mit der diese Kategorien auf Anforderungsseite bestimmt werden können.

Um die Leistung und den Einfluss eines derart integrierten Anforderungselementfilters in der automatischen Anwendung bemessen zu können, muss zunächst ein geeignetes automatisches Klassifikationsverfahren ausgewählt werden. Hierzu wird in Kapitel 9 eine Analyse bestehender Verfahren durchgeführt und das gewählte Verfahren vorgestellt. Um die Eignung des Verfahrens zur Klassifikation von Anforderungselementen zu verdeutlichen, enthält Kapitel 9 außerdem eine intrinsische Evaluation auf Anforderungselementen in Nachverfolgbarkeitsdatensätzen. Die eigentliche Evaluation des Einflusses der Anforderungselementfilter auf **FTLR** folgt daraufhin in Kapitel 10.

9. Transferlernen für Anforderungsklassifikation

Resulting requirements specifications are often organized by functionality with non-functional requirements scattered throughout the specification. This can lead to important conflicts going undetected and architectural solutions that fail to meet the stakeholders' real needs.

(Cleland-Huang et al. [Cle+06])

Möchte man die Klassifikation von Anforderungen für nachgelagerte Analysen nutzen, so bedarf es eines Klassifikationsverfahrens, das eine angemessene Güte aufweist. Insbesondere, wenn man die Ergebnisse für das Filtern und damit das Ignorieren von Informationen für nachgelagerte Analysen einsetzen möchte, können Fehler in der Klassifikation zu noch mehr Fehlern in den nachgelagerten Analysen führen. Für die Filterung von Elementen in der TLR kann beispielsweise eine geringe Präzision (s. Abschnitt 2.11) eines Klassifikationsverfahrens dazu führen, dass essenzielle Informationen für die Abbildung auf den Quelltext herausgefiltert und damit nicht genutzt werden können. Dies kann sogar so weit führen, dass die positiven Effekte durch das Entfernen irrelevanter Informationen von den negativen Effekten einzelner Fehler überlagert werden. Aber auch die Ausbeute (s. Abschnitt 2.11) des Klassifikationsverfahrens beeinflusst maßgeblich den Erfolg. Identifiziert das Verfahren nur einen Bruchteil der zu filternden Elemente, so ist der Effekt, der überhaupt durch die Filterung erzielt werden kann, gering. Daher sollte ein Verfahren eine möglichst gute Leistung hinsichtlich beider Werte erzielen, also ein hohes F_1 -Maß aufweisen.

Da für die Anforderungsklassifikation allgemein und insbesondere für die funktionalen Aspekte nur geringe Mengen an Trainingsdaten zur Verfügung stehen, muss ein Verfahren also in der Lage sein, auch mit wenig Trainingsdaten und ungleichmäßig verteilten Klassen umzugehen:

Anforderung 2: Gute Klassifikationsergebnisse auch für Klassen mit geringer Repräsentation in den Trainingsdaten und allgemein geringer Menge an Daten

Durch die Zielsetzung dieser Arbeit, einer potenziellen Anwendung **FTLRs** auf ungesehene Projekte und Domänen (s. Kapitel 4), muss das Klassifikationsverfahren außerdem möglichst übertragbar auf ungesehene Projekte und damit potenziell unbekannte Begrifflichkeiten und Formulierungen sein:

Anforderung 3: Gute Übertragbarkeit des Klassifikationsverfahrens auf neue ungesene Projekte

Die Auswahl bzw. Entwicklung eines solchen Verfahrens und die Bemessung seiner Güte auf Vergleichsdatensätzen wird in diesem Kapitel beschrieben. Teile der in diesem Kapitel besprochenen Beiträge sind bereits in unserer Arbeit zum Transferlernen für Anforderungsklassifikation [Hey+20a] veröffentlicht. Da nach der Veröffentlichung des Konferenzartikels ein Fehler im Evaluationsskript entdeckt wurde, ergeben sich kleinere Abweichungen zu den veröffentlichten Ergebnissen. Eine korrigierte Version der Arbeit wurde veröffentlicht [Hey+22].

In den folgenden Abschnitten werden zunächst bestehende Arbeiten zur Anforderungsklassifikation beschrieben und ihre Eignung diskutiert (s. Abschnitt 9.1). Daraufhin wird das für diese Arbeit ausgewählte Verfahren näher beschrieben (s. Abschnitt 9.2). In Abschnitt 9.3 wird die Güte des Verfahrens auf dem Standard-Vergleichsdatensatz in der Anforderungsklassifikation (PROMISE NFR) bemessen. Abschließend werden in Abschnitt 9.4 die Klassifikationsergebnisse des Verfahrens auf den Vergleichsdatensätzen der TLR dargestellt und analysiert.

9.1. Bestehende Ansätze zur Anforderungsklassifikation

Die automatisierte Extraktion und Klassifikation von Anforderungen aus natürlichsprachlichen Textdokumenten steht seit mehr als einem Jahrzehnt im Fokus der Forschung und hat zu zahlreichen Ansätzen geführt. Die Ansätze von Cleland-Huang et al. [Cle+06; Cle+07c] verwenden IR, um nichtfunktionale Anforderungen in strukturierten und unstrukturierten Dokumenten zu klassifizieren. Sie identifizieren sogenannte Indikatorbegriffe (engl.: *indicator terms*) in den Dokumenten und nutzen diese zur Klassifikation von Anforderungen. Der Ansatz erreicht hohe Ausbeutewerte, ist aber ungenau (Präzision unter 27 %) bei der Klassifikation nichtfunktionaler Anforderungen. Die Autoren veröffentlichten außerdem ihren Datensatz NFR [Cle+07b] als Teil des PROMISE-Repository [SM05], welcher auch heute noch als Standard-Vergleichsdatensatz für die Anforderungsklassifikation gilt.

Hussain et al. [HKO08] verwenden spezielle Wortklassen wie Zahlwörter, Adverbien und Modalverben mit einem Entscheidungsbaum-Klassifikator (C4.5), um die Anforderungsklassifikation in einer anderen Version des PROMISE NFR-Datensatzes zu verbessern, und erzielen vielversprechende Ergebnisse (bis zu 99 % F_1 -Maß). Slankas und Williams [SW13] stellen einen Ansatz vor, der die Verarbeitung natürlicher Sprache auf unbeschränkte natürlichsprachliche Dokumente anwendet, um Sätze zu finden, die nichtfunktionale Anforderungen enthalten, und ihnen Kategorien zuzuordnen. Sie verwenden Abhängigkeitsgraphen und einen K-Nächste-Nachbarn-Klassifikator (engl.: *k-nearest neighbor classifier*), um die spezifischen Kategorien nichtfunktionaler Anforderungen zu identifizieren. Rahimi et al. [RMC14] verwenden eine Reihe von maschinellen Lern- und Datengewinnungsmethoden (engl.: *data mining*) zur Extraktion und Modellierung von Qualitätsbedenken in

Anforderungen. Sie modellieren die Anliegen in einem Zielmodell und verwenden einen LDA-basierten Ansatz, um den Zielen Themen zuzuordnen.

Kurtanović und Maalej [KM17] verwenden eine automatische Merkmalsauswahl für lexikalische, syntaktische und Metadaten-Merkmale, um bestimmte Anforderungsklassen im PROMISE NFR-Datensatz vorherzusagen. Ihr SVM-Klassifikator erreicht Werte für das F_1 -Maß von bis zu 93 % für die funktionale vs. nichtfunktionale Klassifikation. Bei den vier wichtigsten nichtfunktionalen Anforderungsklassen Benutzerfreundlichkeit (engl.: *usability*), Sicherheit (engl.: *security*), Betrieb (engl.: *operational*) und Leistungsfähigkeit (engl.: *performance*) erreichen ihre binären Klassifikatoren Ergebnisse zwischen 72 % und 90 % im F_1 -Maß.

Abad et al. [Aba+17] zeigen, dass die Vorverarbeitung und Vereinheitlichung des PROMISE NFR-Datensatzes das Ergebnis eines Entscheidungsbaum-Klassifikators, wie der von Hussain et al. [HKO08] vorgeschlagene, bei der Klassifikation funktional vs. nichtfunktional von einem F_1 -Maß-Wert von 91 % auf 95 % verbessert. Darüber hinaus bewerten sie verschiedene Klassifikationsalgorithmen für die Klassifikation von Unterklassen nichtfunktionaler Anforderungen. Der binarisierte Naïve-Bayes Klassifikator schneidet sowohl beim unbearbeiteten als auch beim vorverarbeiteten Datensatz am besten ab und erreicht einen F_1 -Maß-Wert von 45 % bzw. 90 %. Ein Nachteil dieses Ansatzes ist die teilweise manuelle Vorverarbeitung, die datensatzspezifisch sein kann.

Dalpiatz et al. [Dal+19] etikettieren den PROMISE NFR-Datensatz neu, um Probleme bei der Etikettierung zu beheben und Anforderungen zu berücksichtigen, die sowohl funktionale als auch nichtfunktionale Aspekte umfassen. Sie reimplementieren den Ansatz von Kurtanović und Maalej [KM17] und evaluieren ihn anhand des neu etikettierten Datensatzes und weiterer Projekte. Zusätzlich schlagen sie eine besser interpretierbare Merkmalsmenge vor, mit der zwar niedrigere, aber dennoch vergleichbare Ergebnisse bei der Aufgabe erzielt werden können.

Andere Ansätze verwenden tiefes maschinelles Lernen zur Klassifikation von Anforderungen. So verwenden Winkler und Vogelsang [WV16] ein CNN, um die Inhaltselemente natürlichsprachlicher Anforderungsspezifikationen entweder als „Anforderung“ oder „Information“ zu klassifizieren und erreichen dabei Werte im F_1 -Maß von 82 %. Navarro-Almanza et al. [NJJ17] setzen ein CNN ein, um die Anforderungen im PROMISE NFR-Datensatz nach allen zwölf Anforderungskategorien mit einem durchschnittlichen F_1 -Maß von 77 % zu klassifizieren. Dekhtyar und Fong [DF17] wenden Worteinbettungen und ein CNN auf die Aufgabe an, nichtfunktionale Anforderungen im PROMISE NFR-Datensatz zu identifizieren. Sie erreichen einen F_1 -Maß-Wert von bis zu 92 %. Amasaki und Leelaprute [AL18] vergleichen verschiedene Wortvektormodelle hinsichtlich ihrer Effektivität für die vier wichtigsten nichtfunktionalen Anforderungsklassen im PROMISE NFR-Datensatz. Die Ergebnisse zeigen, dass Modelle basierend auf *doc2vec* [LM14] und dünn zusammengesetzten Dokumentvektoren (engl.: *sparse composite document vectors*) [Mek+17] die Klassifikation von nichtfunktionalen Anforderungen im PROMISE NFR-Datensatz am besten erfüllen.

Die oben genannten Ansätze sind vielversprechend und zeigen die Fähigkeiten verschiedener Techniken für das Problem der Klassifikation von Anforderungen. Allerdings sind

viele dieser Ansätze für eine Anwendung in der Praxis eher ungeeignet, da sie entweder zu stark an den Datensatz angepasst sind, stark von Wortlaut und Satzstruktur abhängen oder eine manuelle Vorverarbeitung erfordern. Darüber hinaus geben die Ansätze entweder nicht an, ob sie in der Lage sind, von Projektspezifika zu verallgemeinern, oder sie generalisieren nicht gut genug, um auf zuvor nicht gesehene Projekte anwendbar zu sein. Ein Faktor hierfür könnte der Mangel an verfügbaren Trainingsdaten im RE sein, um allgemeingültigere Modelle zu trainieren. Zusätzlich beschäftigen sich diese Arbeiten weitestgehend mit der Klassifikation von nichtfunktionalen Anforderungen und ihrer Abgrenzung zu funktionalen Anforderungen. Eine detailliertere Betrachtung der Aspekte der funktionalen Anforderungen wurden kaum untersucht. Aus diesen Gründen haben wir 2020 einen neuen Ansatz entwickelt, der erstmalig Transferlernen von Sprachmodellen für die Aufgabe der Klassifikation von Anforderungen einsetzt. Dieses Vorgehen verspricht auch mit einer geringen Datenlage, wie sie für funktionale Aspekte besteht, akzeptable Ergebnisse und eine höhere Übertragbarkeit auf ungesehene Projekte zu erzielen. Das Verfahren **NoRBERT** [Hey+20a; Hey+22] setzt dabei auf eine Feinanpassung des Sprachmodells *BERT* (s. Abschnitt 2.9.3) auf dem PROMISE NFR-Datensatz. Hinsichtlich der Klassifikation von Unterklassen nichtfunktionaler Anforderungen und der Erkennung funktionaler und Qualitätsaspekte konnte **NoRBERT** die Leistung im F_1 -Maß gegenüber bisherigen Verfahren teils deutlich steigern. Auch in der Anwendung auf im Training ungesehene Projekte übertrifft **NoRBERT** die bestehenden Verfahren. Zusätzlich haben wir die Anforderungen des PROMISE NFR-Datensatzes hinsichtlich der drei Aspekte funktionaler Anforderungen Funktion, Daten und Verhalten untersucht. Damit haben wir den ersten Datensatz und die ersten Ergebnisse zur automatischen Unterklassifikation funktionaler Anforderungen bereitgestellt. Hier konnte **NoRBERT** in dem für Übertragbarkeit relevanten Versuchsaufbau auf ungesehenen Projekten ein durchschnittliches F_1 -Maß von 77 % erzielen.

Aufbauend auf diesen vielversprechenden Ergebnissen haben seitdem weitere Arbeiten *BERT* und Transferlernen für die Anforderungsklassifikation eingesetzt. So kombinieren Li et al. [Li+22] *BERT* mit einem Graph-Aufmerksamkeitsnetz (engl.: *graph attention network*, GAT) auf den Abhängigkeitsgraphen (s. Abschnitt 2.8.6) der Anforderungen. Auf der Klassifikation von Anforderungen in nichtfunktional bzw. funktional und den vier häufigsten nichtfunktionalen Unterklassen (Benutzbarkeit, Sicherheit, Einsatz und Leistungsfähigkeit) erzielt ihr Modell DBGAT bessere Ergebnisse als **NoRBERT**. In der Klassifikation aller 10 Unterklassen nichtfunktionaler Anforderungen im PROMISE NFR-Datensatz sind die Ergebnisse allerdings ähnlich und gerade bei den Klassen mit geringem Auftreten im Datensatz übertrifft **NoRBERT** DBGAT. Über die Leistung von DBGAT auf einer Klassifikation funktionaler und Qualitätsaspekte oder der Unterasspekte funktionaler Anforderungen geben die Autoren leider keine Auskunft.

Ajagbe und Zhao [AZ22] trainieren das Standardmodell $BERT_{BASE}$ auf Dokumenten aus der Anforderungsdomäne weiter. Sie nutzen dafür den PROMISE NFR-, PURE- sowie zwei Datensätze zu Smartphone-Applikationsbewertungen. Die ersten beiden Datensätze umfassen zusammen ca. 49000 Wörter und die letzten beiden zusammen über 7 Millionen Wörter. Das so entstandene BERT4RE-Modell evaluieren sie daraufhin auf den funktionalen Anforderungen des PURE-Datensatzes [FSG18]. Als Klassifikationsaufgabe nutzen sie hierzu eine Spezialform der Erkennung semantischer Rollen (s. Abschnitt 2.8.7) auf

Basis der neun semantischen Konzepte Agens (engl.: *Agent*), Objekt (engl.: *Object*), Instrument, Container, Dativ, Ort (engl.: *Location*), Zeitlich (engl.: *Temporal*), Ziel (engl.: *Goal*) und Einschränkung (engl.: *Constraint*). Sie konnten zeigen, dass ihr auf Anforderungen angepasstes Modell das ursprüngliche $BERT_{BASE}$ -Modell in dieser Aufgabe im Durchschnitt über die Klassen um 7 % übertrifft. Allerdings geben die Autoren keine Informationen zur Leistung ihres Modells in einem Versuchsaufbau auf ungesehenen Projekten an und bieten somit keine Einblicke zur Übertragbarkeit des Verfahrens.

Dell’Anna et al. [DAD22] haben die Ergebnisse der bisher verfügbaren Klassifikationsverfahren für die Detektion von funktionalen und Qualitätsaspekten (**NoRBERT** [Hey+20a], Kurtanović und Maalej [KM17] und Dalpiaz et al. [Dal+19]) auf einer um 13 weitere Datensätze ergänzten Testmenge überprüft. Hierbei trainieren sie die Modelle auf dem gesamten PROMISE NFR-Datensatz und testen die Leistung hinsichtlich Präzision, Ausbeute und F_1 -Maß auf den 13 Datensätzen. Auf den ungesehenen Daten erzielt **NoRBERT** mit einem F_1 -Maß von 79 % für funktionale Aspekte und 71 % für Qualitätsaspekte die beste Leistung der drei Verfahren. Dies bestätigt die Erkenntnis aus unserer Veröffentlichung, dass **NoRBERT** die höchste Leistung auf dieser Aufgabenstellung erzielt. Allerdings konnten sie zeigen, dass das Verfahren von Dalpiaz et al. [Dal+19] den geringsten Leistungsverlust zwischen Training und Test aufweist und somit voraussichtlich am unempfindlichsten gegenüber Überanpassung ist. Dieses erzielt auf der Testmenge allerdings nur ein F_1 -Maß von 75 % für funktionale Aspekte und 62 % für Qualitätsaspekte.

Somit ist **NoRBERT** momentan noch immer der Stand-der-Technik zur Anforderungsklassifikation von Aspekten funktionaler Anforderungen (Anforderung 2) und eines der besten Verfahren für die anderen Klassifikationsaufgaben im RE. Durch die vielversprechenden Ergebnisse hinsichtlich der Anwendung auf ungesehenen Projekten (Anforderung 3) und der einfachen Anwendbarkeit ohne weitere Vorverarbeitung oder manueller Schritte eignet sich das Verfahren gut für den Einsatz als Vorverarbeitungsschritt für **FTLR**.

9.2. NoRBERT

Das Klassifikationsverfahren **NoRBERT** (*Non-functional and functional Requirements classification using BERT*) [Hey+20a; Hey+22] basiert auf dem Sprachmodell *BERT* [Dev+19] (s. Abschnitt 2.9.3), welches auf der englischen Wikipedia und dem BooksCorpus [Zhu+15] vortrainiert ist. Durch die enorme Menge an Trainingsdaten (3,3 Milliarden Wörter) soll *BERT* generelle Zusammenhänge in der Sprache sowie die semantischen Zusammenhänge zwischen Wörtern kontextsensitiv erlernen. Es konnte gezeigt werden, dass *BERT* in seinen Schichten ähnliche Strukturen wie das klassische NLP-Fließband erlernt [TDP19]. Genau diese Eigenschaften liegen auch der Idee zugrunde, *BERT* für die Klassifikation von Anforderungen zu nutzen. Da die zur Verfügung stehende Trainingsmenge für diese Art von Aufgaben sehr gering ist, können die generellen Zusammenhänge, die *BERT* bereits aus seinem Vortraining erlernt hat, dem Modell beim Training dabei helfen, sich auf die wesentlichen Aspekte für die eigentliche Aufgabe zu konzentrieren. Es braucht nicht erst die generellen syntaktischen und semantischen Zusammenhänge natürlicher Sprache zu

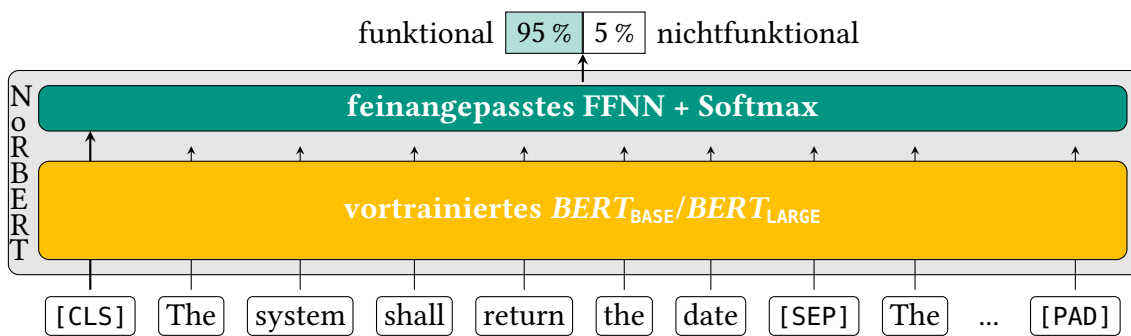


Abbildung 9.1.: Darstellung des Einsatzes des **NoRBERT**-Verfahrens auf einer Beispielingabe

erlernen, sondern kann sich direkt auf das Erkennen und somit Differenzieren der Klassen konzentrieren. Dies kann dabei helfen, bereits mit wenigen Trainingsdaten Ergebnisse zu erzielen, die vormals nur mit bis zu 100-mal mehr Daten möglich waren [HR18]. Der Einsatz *BERTs* für die Anforderungsklassifikation basiert also auf folgender Annahme:

Annahme 4: Die generellen sprachlichen Zusammenhänge, die ein großes Sprachmodell, wie *BERT*, in seinem Vortraining erlernt, ermöglichen bessere Ergebnisse auf Klassifikationsaufgaben mit geringer Trainingsmenge.

NoRBERT nutzt eines von zwei zur Verfügung stehenden vortrainierten *BERT*-Modellen, das $BERT_{BASE}$ - oder das $BERT_{LARGE}$ -Modell, jeweils in der Variante, welche Groß- und Kleinschreibung beachtet. Für diese Modelle stehen außerdem ebenso bereits vortrainierte Portionierer (s. Abschnitt 2.8.1) zur Verfügung. Dies ist auch die einzige Vorverarbeitung, die **NoRBERT** nutzt. Um eines der Modelle nun für die eigentliche Aufgabe der Klassifikation von Anforderungen zu nutzen, wird eine weitere Schicht hinzugefügt, welche als Eingabe die aggregierte Ausgabe der letzten *BERT*-Schicht¹ nimmt (s. Abbildung 9.1). Diese Schicht besteht aus einem einschichtigem FFNN, welches die Ausgabe aus der Summe der gewichteten Eingaben (und einem gewissen Bias) berechnet. **NoRBERT** nutzt dabei die Softmax-Funktion zur Bestimmung der Wahrscheinlichkeitsverteilung der verschiedenen Klassen.

Die Feinanpassung des Modells geschieht durch ein Trainieren des Gesamtmodells auf der jeweiligen Klassifikationsaufgabe. Während des Trainings nutzt **NoRBERT** die Kreuzentropie als Verlustfunktion. Die Verlustfunktion trifft eine Aussage darüber, wie nahe die vorhergesagte Verteilung der tatsächlichen Verteilung ist:

$$H(p, q) = - \sum_x p(x) \log q(x) \quad (9.1)$$

$p(x)$ entspricht dabei der Zielwahrscheinlichkeit und $q(x)$ der tatsächlichen Wahrscheinlichkeit; x repräsentiert die verschiedenen Klassen. $p(x)$ entspricht für eine korrekt vor-

¹ Die Ausgabe des [CLS]-Tokens

hergesagte Klasse x Eins und für alle anderen Klassen y ist $p(y) = 0$. Die Verlustfunktion (9.1) bestraft also falsch oder ungenau vorhergesagte Klassenzugehörigkeiten und belohnt korrekte Vorhersagen.

Eine weitere wichtige Komponente beim Training eines (tiefen) neuronalen Netzes ist der Optimierer, welcher die verschiedenen Gewichte innerhalb des Netzes aktualisiert. Anstatt des klassischen Gradientenabstiegs nutzt **NoRBERT** den *AdamW*-Optimierer, der eine Adaption des populären *Adam*-Optimierers [KB15] ist. AdamW [LH18] setzt eine Korrektur des Gewichtsabfalls ein und kompensiert im Gegensatz zum normalen Adam-Optimierer den Bias nicht. **NoRBERT** nutzt AdamW ohne Aufwärmphase, um die Leistung auf kleinen Datensätzen zu steigern. Außerdem wird ein Gewichtsabfall von 0,01, eine maximale Lernrate von $2e^{-5}$, sowie eine Stapelgröße (engl.: *batch size*) von 16 verwendet, wie auch in der ursprünglichen *BERT*-Veröffentlichung [Dev+19] (vgl. Abschnitt 2.5.2.2). Da Transformer-basierte Sprachmodelle eine festgelegte maximale Sequenzlänge benötigen, wurde diese auf 128 Token festgelegt. Eine niedrigere Sequenzlänge sorgt für eine kürzere Trainingszeit und umgeht potenzielle Speicherprobleme mit dem großen Modell. Allerdings übersteigen die meisten klassischen Anforderungen, welche meist aus 1-3 Sätzen bestehen, diese Größe nicht. Auf dem PROMISE NFR-Datensatz sind bspw. keine zehn Anforderungen länger als 50 Token. Die Epochenanzahl wurde in den Experimenten mit **NoRBERT** systematisch zwischen 10 und 32 für binäre Klassifikation und 10 und 64 für Mehrklassen-Klassifikation variiert. Der Gedankengang ist, dass eine höhere Epochenanzahl es dem Modell erlaubt, sich besser auf die gesehenen Daten anzupassen, aber ein zu hoher Wert das Risiko einer Überanpassung birgt. Auf dem PROMISE NFR-Datensatz ergaben sich dabei eine Epochenanzahl von 10 oder 16 für die binäre Klassifikation und 32 oder 50 für die Mehrklassen-Klassifikation als optimal.

9.3. Ergebnisse auf dem PROMISE NFR-Datensatz

Um die Eignung **NoRBERT**s in Bezug zu existierenden Verfahren zu setzen und **NoRBERT**s allgemeine Relevanz für die RE-Forschungsgemeinschaft zu verdeutlichen, möchte ich in diesem Abschnitt **NoRBERT**s Leistung auf dem PROMISE NFR-Datensatz [Cle+07b] mit verwandten Arbeiten vergleichen. Diese Ergebnisse entstammen zum Teil der aktualisierten Veröffentlichung zu **NoRBERT** und beschränken sich aufgrund des Ziels dieser Arbeit auf die in Abschnitt 8.1 identifizierten relevanten Klassen für die TLR². Diese Evaluation befasst sich also mit der Beantwortung der folgenden Forschungsfragen:

FF₅: Welche Leistung erzielt **NoRBERT** im Vergleich zu bestehenden Ansätzen auf den für die TLR relevanten Anforderungsklassen?

FF₆: Ist **NoRBERT** durch den Einsatz von Transferlernen besser in der Lage, auf ungesehene Projekte angewandt zu werden als bestehende Verfahren?

² Ergebnisse von **NoRBERT** auf der Klassifikation von nichtfunktionalen Unterklassen können der Veröffentlichung [Hey+20a; Hey+22] entnommen werden.

Tabelle 9.1.: Verteilung der Klassen im originalen PROMISE NFR-Datensatz [Cle+07b]

Klasse	Anzahl	Ø Anzahl Wörter
Funktional (F)	255	20
Nichtfunktional (NF)	370	20
↪ Verfügbarkeit (A)	21	19
↪ Fehlertoleranz (FT)	10	19
↪ Rechtliches (L)	13	18
↪ Erscheinungsbild (LF)	38	20
↪ Wartbarkeit (MN)	17	28
↪ Einsatz (O)	62	20
↪ Leistung (PE)	54	22
↪ Portierbarkeit (PO)	1	14
↪ Skalierbarkeit (SC)	21	18
↪ Sicherheit (SE)	66	20
↪ Benutzerfreundlichkeit (US)	67	22
Gesamt	625	20

FF₇: Welche **NoRBERT**-Konfigurationen eignen sich besonders für die Anwendung als Vorfilter für die TLR?

Die ersten beiden Forschungsfragen zielen auf die Leistung **NoRBERTs** ab und geben damit Auskunft darüber, ob die Wahl **NoRBERTs** als Klassifikationsverfahren geeignet für die Anwendung in der TLR ist. Hierbei kann Forschungsfrage **FF₆** als Spezialisierung von Forschungsfrage **FF₅** verstanden werden. **FF₆** zielt explizit auf die für die unüberwachte TLR notwendige Leistung auf ungesesehenen Projekten und damit die Übertragbarkeit des Verfahrens ab. Forschungsfrage **FF₇** hingegen befasst sich mit der Auswahl derjenigen **NoRBERT**-Konfigurationen, welche die beste Leistung auf den für die TLR relevanten Klassen erzielen.

Für die Beantwortung der Fragen werden drei Varianten des PROMISE NFR-Datensatzes verwendet: Die originale Version des Datensatzes [Cle+07b], oftmals verwendet in der Forschungsgemeinschaft, eine umetikettierte Version von Dalpiaz et al. [Dal+19], welche funktionale und Qualitätsaspekte unterscheidet, und die von uns etikettierte Version [Hey+20b], welche die Anforderungen mit funktionalen Aspekten weiter in die Klassen Funktion, Daten und Verhalten aufteilt.

Tabelle 9.1 zeigt die Verteilung der Klassen im originalen PROMISE NFR-Datensatz, sowie ihre durchschnittliche Anzahl an Wörtern. Die insgesamt 625 Anforderungen teilen sich auf 255 funktionale und 370 nichtfunktionale Anforderungen auf. Die nichtfunktionalen werden weiterhin in 11 Unterklassen unterteilt, welche stark in ihrer Auftretenshäufigkeit variieren. Am häufigsten vertreten ist die Unterklasse Benutzerfreundlichkeit (engl.: *usability*) mit 67 Auftreten; die Portierbarkeit (engl.: *portability*) hingegen tritt nur einmal auf. Im Durchschnitt umfassen die Anforderungen 20 Wörter.

Tabelle 9.2.: Überblick über den umetikettierten PROMISE NFR-Datensatz [Dal+19]

Klasse	Anzahl
Funktionale Aspekte (F)	310
Qualitätsaspekte (Q)	382
Beides ($F \cap Q$)	80
Gesamt	612

Tabelle 9.3.: Überblick über die Klassenverteilung der funktionalen Aspekte im umetikettierten PROMISE NFR-Datensatz [Hey+20a]

Klasse	Project														Gesamt
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
Funktion	14	13	33	19	25	29	10	31	11	9	3	7	2	1	207
Daten	6	3	13	6	7	2	5	3	2	5	0	1	2	2	57
Verhalten	2	9	9	13	6	6	5	14	10	33	3	1	1	1	113
Anforderungen	19	22	44	26	37	31	15	38	17	43	3	8	3	3	309

Dalpiaz et al. [Dal+19] halten die strikte Unterteilung zwischen funktionalen und nichtfunktionalen Anforderungen für diskutabel und präferieren stattdessen lediglich festzustellen, ob eine Anforderung funktionale und/oder Qualitätsaspekte enthält. Da sie außerdem falsch etikettierte Anforderungen im Originaldatensatz gefunden haben, stellen sie eine umetikettierte Version des Datensatzes zur Verfügung. Tabelle 9.2 stellt die neue Verteilung der Klassen dar. Dieser Datensatz umfasst nur noch 612 der ehemals 625 Anforderungen wovon 310 funktionale und 382 Qualitätsaspekte beinhalten. 80 Anforderungen umfassen sowohl funktionale als auch Qualitätsaspekte.

Die dritte Variante unterteilt die 310 funktionalen Anforderungen aus dem Datensatz von Dalpiaz et al. [Dal+19] in die drei Unterklassen Funktion, Daten und Verhalten. Diese Unterteilung folgt der anliegenbasierten Klassifikation nach Glinz [Gli07] und umfasst drei der in Abschnitt 8.1 identifizierten Unterklassen funktionaler Anforderungen. Eine Anforderung kann hierbei eine oder mehrere dieser Anliegen beinhalten. Die Bestimmung der Klassen im Datensatz erfolgte unabhängig voneinander durch zwei Personen. Hierbei wurde eine Übereinstimmung in Krippendorffs α [Kri18] von 0,803 für Funktion, 0,814 für Daten und 0,752 für Verhalten erzielt. Diese Werte liegen über der unteren Grenze für akzeptable Übereinstimmung und für Funktion und Daten sogar über dem empfohlenen Wert von 0,8 [Kri04]. Unstimmigkeiten wurden daraufhin durch Diskussion bereinigt, um eine uniforme Musterlösung zu bestimmen. Tabelle 9.3 zeigt die Verteilung der Anliegen auf diejenigen Anforderungen der 14 Projekte des PROMISE NFR-Datensatzes, welche funktionale Anforderungen enthalten. Da der Datensatz von Dalpiaz et al. [Dal+19] eine Anforderung doppelt enthielt, ist die Gesamtanzahl der Anforderungen von 310 auf 309 gesunken. Die Projekte 11–14 enthalten nur sehr wenige Anforderungen, da ein Großteil der Anforderungen dieser Projekte nur Qualitätsaspekte enthält.

Um die Leistung der Verfahren zu bemessen, werden drei unterschiedliche Versuchsaufbauten verwendet. Für eine Vergleichbarkeit mit den meisten bestehenden Verfahren, wird eine stratifizierte zehnfache Kreuzvalidierung (**10-fold**) durchgeführt. Diese teilt die Daten zehnmal in 90 % für das Training und 10 % für das Testen ein, sodass jeder Datenpunkt in genau einem Lauf in der Testmenge liegt. Stratifizierte Stichproben sorgen dafür, dass die Verteilung der Klassen in Trainings- und Testmenge möglichst derjenigen des gesamten Datensatzes entspricht. Um die Leistung auf ungesehenen Daten und insbesondere auf ungesehenen Projekten genauer untersuchen zu können, werden außerdem zwei projektspezifische Aufteilungsstrategien vorgenommen. Bei **p-fold** wird die Aufteilung des Datensatzes, wie ihn Dalpiaz et al. [Dal+19] vorschlagen, durchgeführt. Hierbei wird der Datensatz zehnmal in drei Projekte zum Testen und zwölf Projekte für das Training aufgeteilt, wobei darauf geachtet wurde, eine gleichmäßige Verteilung von funktionalen und Qualitätsaspekten zu erzielen. Die dritte Variante simuliert die Situation eines vortrainierten Modells auf einer vorhandenen Datenbasis und der Anwendung auf ein neues Projekt. Hierzu wird der Datensatz so aufgeteilt, dass jedes Projekt genau einmal die Testmenge darstellt und jeweils auf den $N - 1$ der N Projekte trainiert wird. Es werden also Anzahl der Projekte (N) Durchläufe bemessen. Diese Art der Kreuzvalidierung bezeichnen wir als **IoPo** (engl.: *leave-one-project-out*).

Für stark unausgewogene binäre Klassifikationsaufgaben kann es von Vorteil sein, ein Über- oder Unterabtasten der Trainingsdaten durchzuführen (s. Abschnitt 2.5.1). Um zu überprüfen, ob ein solches Vorgehen auch für **NoRBERT** sinnvoll ist, wurde ebenso mit einem zufälligen Überabtasten (engl.: *oversampling*, OS) und Unterabtasten (engl.: *undersampling*, US) experimentiert. Ein US reduziert die Instanzen der Mehrheit auf die Anzahl der Minderheit, wohingegen ein OS die Menge der Instanzen der Minderheit auf die Menge der Mehrheit vergrößert. In beiden Varianten bleibt die Testmenge unberührt.

Zusätzlich wurde mit frühem Anhalten (engl.: *early stopping*, ES) experimentiert. ES dient der Vermeidung von Überanpassung bei iterativen Lernverfahren. Hierzu wurde ein Schwellenwert von 0,01 für das F_1 -Maß der vorherzusagenden Klasse und ein Abwartswert (engl.: *patience value*) von Drei festgelegt. Das Training wird gestoppt, wenn die Verbesserung zur besten Iteration drei Iterationen (Epochen) unter dem Schwellenwert liegt. Aufgrund dieser vielen Konfigurationsmöglichkeiten werden für die bessere Lesbarkeit im Folgenden nur die jeweils besten Konfigurationen für **NoRBERT** präsentiert. Weitere Ergebnisse können dem ergänzenden Material zur Originalveröffentlichung entnommen werden [Hey+20b].

9.3.1. Identifikation funktionaler und nichtfunktionaler Anforderungen

In diesem Abschnitt wird zunächst die Leistung **NoRBERTs** für die Aufgabe, Anforderungen des originalen PROMISE NFR-Datensatzes als eindeutig funktional oder nichtfunktional zu klassifizieren, betrachtet. Diese Aufgabe wurde gewählt, da sie die am häufigsten verwendete Anforderungsklassifikationsaufgabe in verwandten Arbeiten darstellt und somit eine größere Menge an Verfahren zum Vergleich herangezogen werden können. Um diese Aufgabe zu erfüllen, setzt **NoRBERT** binäre Klassifikation ein; der Klassifikator bestimmt also,

Tabelle 9.4.: Ergebnisse der Klassifikation in funktionale (F) oder nichtfunktionale (NF) Anforderungen auf dem PROMISE NFR-Datensatz. Fett gedruckte Werte verdeutlichen die besten Ergebnisse pro Metrik, Klasse und Versuchsaufbau

Ansatz (Parameter)	F (255)			NF (370)		
	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁
10-fold						
K. & M. (Wortmerkmale o. Auswahl)	.92	.93	.93	.93	.92	.92
A. et al. (Unverarbeitete Daten)	.84	.93	.88	.95	.88	.91
A. et al. (Vorverarbeitete Daten)	.90	.97	.93	.98	.93	.95
D. & F. (<i>word2vec</i> , ep.=100, f.=50)	—	—	—	.93	.92	.92
NoRBERT (BASE, ep.=10)	.91	.90	.90	.93	.94	.93
NoRBERT (LARGE, ep.=10, OS)	.95	.88	.91	.92	.97	.94
DBGAT	.94	.90	.92	.94	.98	.96
p-fold						
NoRBERT (BASE, ep.=10)	.85	.51	.64	.74	.94	.82
NoRBERT (LARGE, ep.=16)	.89	.61	.73	.78	.95	.86
loPo						
NoRBERT (BASE, ep.=10, ES)	.88	.50	.64	.73	.95	.83
NoRBERT (LARGE, ep.=10, US)	.87	.71	.78	.82	.93	.87

ob eine Anforderung entweder funktional (F) oder nichtfunktional (NF) ist. Die Ergebnisse (Präzision (Präz.), Ausbeute (Ausb.) und F₁-Maß) auf der zehnfachen Kreuzvalidierung werden in Bezug zu den Ergebnissen der Verfahren von Kurtanović und Maalej [KM17], Abad et al. [Aba+17], Dekhtyar und Fong [DF17] und DBGAT [Li+22] gesetzt. Tabelle 9.4 zeigt diese Ergebnisse im Vergleich zu den Ergebnissen, welche die genannten Arbeiten angeben. Die Ergebnisse zeigen, dass **NoRBERT** vergleichbare Ergebnisse zu bestehenden Verfahren erzielt und für die Identifikation nichtfunktionaler Anforderungen nur durch das Verfahren von Abad et al. und DBGAT übertroffen wird. Das Verfahren von Abad et al. benötigt allerdings manuell zur Verfügung gestellte Wörterbücher und Regeln, um den Datensatz vorzuverarbeiten und DBGAT verwendet Abhängigkeitsbäume, deren Güte von der syntaktischen Korrektheit der Eingabe abhängt. **NoRBERT** hingegen benötigt keine spezialisierte Vorverarbeitung und kann damit direkt auf natürlichsprachlichen Text angewandt werden. Für die Identifikation funktionaler Anforderungen erzielt **NoRBERT** mit einer Präzision von 95 % das höchste Ergebnis und liegt mit einem F₁-Maß von 91 % nur einen Prozentpunkt hinter DBGAT. Das Verfahren von Kurtanović und Maalej, welches neben dem Verfahren von Abad et al. mit teilmanueller Vorverarbeitung ebenfalls 93 % F₁-Maß erzielt, verwendet ein Modell ohne Merkmalsauswahl, welches die Autoren selbst als vermutlich auf den Datensatz überangepasst klassifizieren.

Das auf *BERT*_{LARGE} basierende Modell von **NoRBERT** erzielt auf der 10-fold-Aufteilung insgesamt bessere Ergebnisse als das *BERT*_{BASE}-basierte und der Einsatz von Überabtaben konnte in diesem Versuchsaufbau die Leistung des Modells steigern. Im p-fold erzielt **NoRBERT**

allerdings ohne eine Form der Abtastung die besten Ergebnisse. Hier erzielt ebenso das größere der beiden Modelle bessere Ergebnisse. Mit 73 % bzw. 86 % gewichtetem F_1 -Maß für funktionale bzw. nichtfunktionale Anforderungen sind die Ergebnisse ungefähr 12 % niedriger als in der zehnfachen Kreuzvalidierung. Ein solcher Rückgang war zu erwarten, da ein Vorgehen, welches Projektgrenzen beachtet, den Umgang des Modells mit ungesesehenen Daten (und dementsprechend ungesesehenen Begriffen und Formulierungsweisen) evaluiert und somit eine schwierigere Aufgabe darstellt. Auf loPo ist der Rückgang mit ungefähr 9 % etwas geringer als im p-fold. Dies lässt sich mit der größeren und damit potenziell diverseren Trainingsmenge in diesem Versuchsaufbau begründen, da für jedes Testprojekt auf den verbleibenden 14 Projekten trainiert wird, wohingegen bei p-fold nur auf jeweils 12 Projekten trainiert wird. Auch auf loPo erzielt das $BERT_{LARGE}$ -basierte Modell bessere Ergebnisse und profitiert in diesem Fall sogar von Unterabtasten.

Leider können die Ergebnisse von **NoRBERT** auf den projektspezifischen Aufteilungsstrategien nicht mit verwandten Arbeiten verglichen werden, da diese eine solche Betrachtung und somit eine Evaluation auf ungesesehenen Daten nicht durchgeführt haben. Auch die Arbeit von Li et al. [Li+22], welche die Ergebnisse einer solchen Evaluation für die Klassifikation nichtfunktionaler Unterklassen berichten, lassen diese Art der Evaluation auf der Identifikation funktionaler und nichtfunktionaler Anforderungen aus. Insgesamt lässt sich aber festhalten, dass **NoRBERT** vergleichbare Ergebnisse zu anderen Verfahren erzielt, die den Stand der Technik darstellen. Auf ungesesehenen Projekten und damit ungesesehenen Begrifflichkeiten und Formulierungsweisen weißt **NoRBERT** nur Einbußen um die 10 % im F_1 -Maß auf. Diese Ergebnisse sind vielversprechend und deuten auf eine angemessene Fähigkeit zur Generalisierung über Projektgrenzen hinweg durch **NoRBERT** hin, welche für einen Einsatz in der unüberwachten TLR notwendig ist.

9.3.2. Identifikation von funktionalen und Qualitätsaspekten

Eine klare Trennung zwischen funktionalen und nichtfunktionalen Anforderungen ist nicht immer eindeutig zu bestimmen. Manche Anforderungen oder Anforderungsteile enthalten sowohl funktionale als auch Qualitätsaspekte. Für den Einsatz in der TLR ist es relevanter, ob ein Anforderungsteil einen funktionalen Aspekt enthält, da dieser potenziell für die Abbildung auf Quelltext relevant ist. Es ist nicht wichtig zu bestimmen, ob es sich um eine Ausprägung einer nichtfunktionalen Anforderung im klassischen Sinne handelt. Daher sind die Ergebnisse von **NoRBERT** auf dem unetikettierten Datensatz von Dalpiaz et al. [Dal+19] aussagekräftiger für eine Eignung in der TLR, als die Ergebnisse auf dem originalen Datensatz. Tabelle 9.5 zeigt die Leistung **NoRBERTs** in der Erkennung funktionaler bzw. Qualitätsaspekte und vergleicht diese zum einzigen verfügbaren anderen Verfahren auf diesem Datensatz, einer Reimplementierung des Ansatzes von Kurtanović und Maalej durch Dalpiaz et al. [Dal+19].

In der zehnfachen Kreuzvalidierung erzielt das beste **NoRBERT**-Modell eine Verbesserung im F_1 -Maß von 11 Prozentpunkten für funktionale Aspekte und 8 Prozentpunkten für Qualitätsaspekte. In diesem Fall ist es sogar das $BERT_{BASE}$ -basierte Modell mit frühem Anhalten, welches die größte Verbesserung erzielt. Auf den beiden projektspezifischen

Tabelle 9.5.: Ergebnisse der binären Klassifikation der Klassen des umetikettierten PROMISE NFR-Datensatzes. Fett gedruckte Werte markieren die höchsten Werte pro Metrik, Klasse und Versuchsaufbau

Ansatz	F (310)			Q (382)		
	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁
10-fold						
K. & M. reimplementiert [Dal+19]	.82	.74	.77	.82	.91	.86
NoRBERT (BASE, ep.=10, ES)	.89	.86	.88	.91	.96	.94
NoRBERT (LARGE, ep.=10)	.86	.86	.86	.93	.95	.94
p-fold						
K. & M. reimplementiert [Dal+19]	.81	.70	.74	.75	.92	.82
NoRBERT (BASE, ep.=10)	.86	.76	.81	.81	.95	.87
NoRBERT (LARGE, ep.=10)	.86	.79	.82	.87	.94	.90
loPo						
NoRBERT (BASE, ep.=10)	.87	.76	.81	.79	.94	.86
NoRBERT (LARGE, ep.=10)	.84	.75	.80	.86	.93	.89

Ausführungsstrategien zeigt aber wiederum das große Modell die bessere Leistung (in beiden Fällen mit 10 Epochen und ohne Über-/Unterabtaben oder frühem Anhalten). In dieser Klassifikationsaufgabe können die Ergebnisse von **NoRBERT** außerdem auch auf dem p-fold mit einer verwandten Arbeit verglichen werden, da Dalpiaz et al. [Dal+19] ebenso diese Evaluationsform durchgeführt haben. Hier zeigt sich, dass **NoRBERT**s bestes Modell das reimplementierte Verfahren von Kurtanović und Maalej im F₁-Maß um 8 Prozentpunkte sowohl für funktionale als auch Qualitätsaspekte übertrifft. Für Qualitätsaspekte weist **NoRBERT** insbesondere eine höhere Präzision als das Vergleichsverfahren auf. Bei der Identifikation funktionaler Aspekte ist sowohl in der Präzision als auch der Ausbeute ein deutlicher Leistungszuwachs festzustellen.

9.3.3. Identifikation von Anliegen in funktionalen Anforderungen

Eine Bestimmung der Anliegen in funktionalen Anforderungen wurde in Abschnitt 8.1 als potenziell hilfreich für die automatische Verarbeitung von Anforderungen identifiziert. Diese Anliegen verdeutlichen, wie die jeweiligen Anforderungen umgesetzt werden können und sind somit relevant für nachgelagerte Aufgaben, wie die automatische TLR oder das automatische Erstellen von Modellen. Um eine automatische Verarbeitung zu ermöglichen, müssen also auch diese Informationen automatisch identifiziert werden. Aus diesem Grund ist es interessant zu bestimmen, wie gut **NoRBERT** in der Lage ist, diese Anliegen zu identifizieren. Hierzu wurde der durch uns erstellte, auf den funktionalen Anforderungen des Datensatzes von [Dal+19] basierende Datensatz verwendet, um binäre Klassifikation mit **NoRBERT** durchzuführen.

Tabelle 9.6.: Binäre Klassifikation der funktionalen Aspekte im umetikettierten PROMISE NFR-Datensatz durch **NoRBERT**. Die größten Werte pro Metrik, Klasse und Versuchsaufbau sind jeweils hervorgehoben

Konfiguration	Funktion (207)			Daten (57)			Verhalten (113)		
	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁
10-fold									
BASE, ep.=10, OS	.86	.94	.90	.71	.51	.59	.85	.69	.76
BASE, ep.=16	.86	.95	.90	.81	.51	.62	.83	.62	.71
BASE, ep.=16, OS	.87	.96	.91	.63	.51	.56	.88	.70	.78
BASE, ep.=16, US	.87	.82	.84	.28	.68	.40	.82	.78	.80
BASE, ep.=32, OS	.85	.94	.89	.68	.53	.59	.89	.66	.76
LARGE, ep.=16, OS	.86	.95	.90	.74	.51	.60	.86	.73	.79
loPo									
BASE, ep.=16	.84	.89	.87	.74	.25	.37	.68	.55	.61
BASE, ep.=16, OS	.85	.86	.86	.66	.37	.47	.73	.66	.69
BASE, ep.=32, OS	.86	.85	.86	.76	.46	.57	.73	.68	.71
LARGE, ep.=16, OS	.86	.85	.85	.61	.30	.40	.77	.59	.67

Tabelle 9.6 zeigt die Ergebnisse, die **NoRBERT** in den verschiedenen Ausführungsstrategien erzielen konnte. Hierbei wurde auf p-fold verzichtet, da die Projekte 11 bis 15 im Datensatz aufgrund weniger funktionaler Anforderungen unterrepräsentiert sind (s. Tabelle 9.3) und somit die Ergebnisse im p-fold verfälschen würden. Die Ergebnisse für die beiden am häufigsten auftretenden Klassen, Funktion und Verhalten, sind vielversprechend, sowohl in der zehnfachen Kreuzvalidierung mit 91 % bzw. 80 % F₁-Maß als auch auf loPo. Für die jeweils besten Modelle ist beim Wechsel auf ungesehene Projekte lediglich ein Rückgang von vier Prozentpunkten auf 87 % für Funktion und 9 Prozentpunkten auf 71 % für Verhalten festzustellen. Allerdings erzielt **NoRBERT** in beiden Vorgehen nur eine geringe Ausbeute für Daten, was dazu führt, dass hier nur ein F₁-Maß von 62 % in 10-fold und 57 % in loPo erzielt werden konnte. Dies kann damit begründet werden, dass die Datenmenge von nur 57 Beispielen für diese Klasse zu gering ist, um von den gesehenen Beispielen zu generalisieren. Ebenso lässt sich der Leistungsunterschied zwischen den zwei Klassen Funktion und Verhalten damit erklären, dass für Funktion fast doppelt so viele Trainingsdaten vorliegen.

Interessanterweise unterscheiden sich die einzelnen Konfigurationen doch recht deutlich in ihrer Leistung auf den verschiedenen Klassen. So erzielt **NoRBERT** mit Unterabtasten auf Verhalten das beste Ergebnis, schneidet aber deutlich schlechter auf den anderen Klassen ab. Überabtasten wirkt sich hingegen außer auf Daten, wo keine Abtastung das beste Ergebnis erzielt, positiv aus. Bei zehnfacher Kreuzvalidierung zeigt über alle Klassen hinweg wieder das $BERT_{LARGE}$ -basierte Modell die besten Ergebnisse, wohingegen bei loPo das $BERT_{BASE}$ -basierte Modell mit 32 Epochen und Überabtasten insbesondere auf Daten und Verhalten deutlich besser abschneidet.

Insgesamt lässt sich festhalten, dass **NoRBERT** für die geringe Menge an vorhandenen Trainingsdaten durchaus vielversprechende Ergebnisse erzielt. Der Rückgang beim Übergang zu ungesehenen Projekten ist in dieser Aufgabe außerdem geringer als in anderen Aufgaben, welches hoffen lässt, dass mit mehr Trainingsdaten noch bessere Ergebnisse möglich sind. Gerade die guten Werte im F_1 -Maß auf Funktion und Verhalten bei loPo sind vielversprechend für eine nachgelagerte automatische Verarbeitung wie die TLR.

9.3.4. Auswahl geeigneter NoRBERT-Konfigurationen

Auf Basis der Ergebnisse auf den verschiedenen Varianten des PROMISE NFR-Datensatzes können nun potenzielle Konfigurationskandidaten für eine Anwendung in der TLR identifiziert und damit Forschungsfrage **FF₇** beantwortet werden. Da **FTLR** und die Aufgabenstellung dieser Arbeit auf den Einsatz auf ungesehenen Projekten ausgelegt ist, sind hierbei vor allem die Ergebnisse auf den projektspezifischen Aufteilungsstrategien relevant. Auf den beiden Aufgaben zur Bestimmung funktionaler Anforderungen oder Anforderungen mit funktionalen Aspekten erzielen jeweils **NoRBERT**-Modelle, die mit 10 Epochen trainiert wurden, die besten Ergebnisse (sowohl für die BASE- als auch die LARGE-Modelle). Außerdem zeigen die Ergebnisse, dass in bestimmten Fällen Über- oder Unterabtasten die Leistung steigern kann, aber zumeist auch Modelle ohne diese Abtaststrategien ähnliche Ergebnisse erzielen. Gerade auf Klassifikationsaufgaben mit wenig Repräsentanten pro Klasse (Daten und Verhalten) erzielt eine höhere Epochenanzahl bessere Ergebnisse³. Auf solchen binären Klassifikationsaufgaben, in denen die Klasse, die vorhergesagt werden soll, nur in einem kleinen Teil der Eingaben auftritt, tendieren die Modelle meist dazu, die dominierende Ausgabe und damit das Nichtauftreten der Klasse zu bevorzugen. Dies liegt daran, dass in diesen Fällen die Trainingsmenge stark von Beispielen ohne ein Auftreten dieser Klasse dominiert wird. Deshalb zeigt sich hier auch der Effekt, dass ein Überabtasten die besten Ergebnisse für eine Anwendung auf ungesehenen Projekten erzielt. Hierdurch wird während des Trainings ein künstliches Gleichgewicht zwischen dem Auftreten und dem Nichtauftreten der Klasse erzeugt. Dies reduziert die Tendenz, das Nichtauftreten zu bevorzugen.

Auf den ersten beiden Klassifikationsaufgaben mit projektspezifischen Aufteilungen erzielt das größere, auf $BERT_{LARGE}$ -basierende Modell bessere Ergebnisse als das $BERT_{BASE}$ -basierte. Auf der loPo-Aufteilung für die Klassifikation funktionaler Anliegen hingegen erzielt das auf $BERT_{BASE}$ basierende Modell mit hoher Epochenanzahl und Überabtasten bessere Ergebnisse. Aufgrund dieser Ergebnisse werden für die weiteren Untersuchungen bezüglich eines Einsatzes für die TLR die $BERT_{BASE}$ und $BERT_{LARGE}$ -basierten Modelle mit 10 und 16 Epochen jeweils mit und ohne OS betrachtet.

³ Dies deckt sich auch mit den Ergebnissen von **NoRBERT** auf der Klassifikation von Unterklassen nicht-funktionaler Anforderungen (s. Hey et al. [Hey+22]).

Tabelle 9.7.: Übersicht der Klassenverteilung auf den Nachverfolgbarkeitsdatensätzen sowie der Gesamtanzahl in Kombination mit den umetikettierten Varianten des PROMISE NFR-Datensatzes. Da keine Informationen zur Klasse Nutzerbezogenes (NB) auf dem PROMISE NFR-Datensatz existiert, ergibt sich für diese Klasse diesselbe Anzahl wie ohne den PROMISE NFR-Datensatz (Sternchen)

Projekt	Elemente	F	Q	Funktion	Verhalten	Daten	NB
eTour	571	457	5	299	342	186	279
iTrust	336	311	1	179	164	169	205
SMOS	522	386	0	243	293	276	251
eAnci	1290	874	21	570	561	328	672
LibEST	565	404	22	209	290	205	5
Gesamt	3284	2432	49	1500	1650	1164	1412
+ PNFR	3908	2741	431	1707	1763	1221	*1412

9.4. Klassifikationsergebnisse auf Nachverfolgbarkeitsdatensätzen

Die Ergebnisse in Abschnitt 9.3 geben Aufschluss über **NoRBERT**s Leistung auf klassischen Anforderungen, wie sie der PROMISE NFR-Datensatz enthält. Sie lassen aber noch keine Aussage darüber zu, wie diese Leistung sich auf Anwendungsfallbeschreibungen allgemein und insbesondere auf die durch **FTLR** produzierten Anforderungselemente übertragen lässt. Außerdem ist noch unklar, welche der Konfigurationen sich für die jeweiligen Klassen am besten eignen. Daher werden in diesem Abschnitt die folgenden weiteren Forschungsfragen untersucht:

FF₈: Welche Leistung erzielt **NoRBERT** auf Anforderungselementen aus ungesehenen Projekten?

FF₉: Welche **NoRBERT**-Konfigurationen erzielen die besten Ergebnisse pro Klasse auf Vergleichsdatensätzen aus der Nachverfolgbarkeit?

Beide Fragen lassen sich mit den in Abschnitt 6.1 vorgestellten und bereits für die Evaluation **FTLR**s verwendeten Vergleichsdatensätzen beantworten. Hierzu wurden die Anforderungselemente der englischsprachlichen Projekte eTour, iTrust und LibEST manuell hinsichtlich der Klassen funktionale Aspekte (F), Qualitätsaspekte (Q), Funktion, Verhalten, Daten und Nutzerbezogenes (NB) etikettiert. Da **NoRBERT** aufgrund der vorhandenen Datenlage (PROMISE NFR und Vortraining des *BERT*-Modells) nur englische Eingaben klassifizieren kann, konnten die italienischen Projekte nicht ohne Weiteres genutzt werden. Um auch auf diesen eine Aussage treffen zu können, wurden die Anforderungen von SMOS und eAnci mittels der Google Übersetzer-API⁴ automatisch übersetzt und die dadurch

⁴ <https://translate.google.com/>, zuletzt besucht am 26.04.2023.

resultierenden Anforderungselemente ebenso manuell etikettiert. Das Etikettieren wurde von mir und einem Masterstudenten der Informatik unabhängig voneinander durchgeführt. Hierbei ergab sich eine sehr gute Übereinstimmung für die Klassen funktionale Aspekte (F), Verhalten und Nutzerbezogenes (NB) mit Werten für Krippendorffs α [Kri18] von 0,897 (F), 0,887 (Verhalten) und 0,905 (NB). Auch die Übereinstimmung für die Klassen Funktion und Daten liegen mit α -Werten von 0,742 und 0,785 deutlich über der unteren Grenze für eine akzeptable Übereinstimmung von 0,66 [Kri04]. Lediglich für Qualitätsaspekte wurde nur eine Übereinstimmung von 0,501 erreicht. Unstimmigkeiten wurden daraufhin durch Diskussion bereinigt, um eine uniforme Musterlösung zu bestimmen. Der resultierende Goldstandard ist auf Zenodo verfügbar [Hey23a].

Die Verteilung der Klassen auf diese Projekte ist in Tabelle 9.7 dargestellt. Zunächst lässt sich festhalten, dass durch die Aufteilung in Anforderungselemente mit insgesamt 3284 Elementen gegenüber den 612 Anforderungen des PROMISE NFR-Datensatzes eine große Menge an neuen Datenpunkten für die Klassifikation entstanden ist. Insbesondere die 139 Anwendungsfallbeschreibungen des eAnci-Projekts resultieren in nun 1290 Elementen und damit über einem Drittel der Elemente. Betrachtet man nun allerdings wie viele von eAncis Elementen tatsächlich funktionale Aspekte enthalten, so sind dies nur 874. Somit sind 416 dieser 1290 Elemente Texte, die zu filtern sind. Die anderen Datensätze resultieren in vergleichbar großen Elementmengen um die 550. Nur iTrust, welches nur die Ablaufbeschreibungen selbst als Anforderungen enthält, resultiert in lediglich 336 Elementen für die ursprünglich 131 Anforderungen. Dafür enthalten aber auch bis auf 25 alle Elemente funktionale Aspekte. Ein Filtern von Anforderungselementen ohne funktionale Aspekte kann also die Eingabemenge für die TLR auf iTrust nur bedingt reduzieren.

Die Menge an Anforderungselementen, die Qualitätsaspekte enthalten, ist sehr gering. Nur 49 der 3284 Elemente beschreiben Qualitätsaspekte. Dies lässt sich damit erklären, dass die Anforderungen dieser als Vergleichsdatsätze für Nachverfolgbarkeit verwendeten Projekte, hauptsächlich funktionale Anforderungen beinhalten und die Felder zu Qualitätsanforderungen der Vorlagen in den meisten Anwendungsfallbeschreibungen leer sind. Betrachtet man die Diskrepanz zwischen den Elementen, welche keine funktionalen Aspekte enthalten und den Elementen die Qualitätsaspekte umfassen, so lässt sich festhalten, dass ein Großteil dieser Elemente zwar keine Funktionalität beschreiben, aber auch keine klassischen nichtfunktionalen Anforderungen darstellen. Dies liegt an den Inhalten von Anwendungsfallbeschreibungen, welche Elemente enthalten, die für sich genommen eigentlich keine Anforderungen darstellen, wie bspw. eine Benennung des Akteurs oder Verweise auf Quellen. Außerdem enthalten die Anwendungsfallbeschreibungen oftmals Elemente, die lediglich Nutzerbezogenes, aber keinerlei funktionale Aspekte beschreiben. Daher ist auch die Anzahl der Elemente, die der Klasse Nutzerbezogenes zuzuordnen sind, auf den Projekten mit Anwendungsfallbeschreibungen deutlich höher als auf LibEST. LibEST enthält lediglich fünf Repräsentanten der Klasse Nutzerbezogenes, was zusätzlich daran liegt, dass das Projekt eine Bibliothek darstellt, die nicht durch eine(n) Nutzer:in verwendet wird. Die geringere Zahl auf iTrust lässt sich durch das Fehlen der Beschreibung des Akteurs und der Vor- und Nachbedingungen erklären, welche zum Teil ebenso Nutzerbezogenes enthalten.

Die beiden Klassen Funktion und Verhalten weisen auf allen Datensätzen relativ viele Beispiele auf, wobei auf eTour, SMOS und LibEST die Menge der Beschreibungen von Verhalten größer ist als die von Funktionen. Dies liegt beispielsweise daran, dass LibEST viele Sätze enthält, die das Zusammenspiel des Netzwerkstapels erklären und nicht explizit die angebotenen Funktionen beschreiben. Außerdem gibt es in den meisten Datensätzen einige Elemente, die sowohl die Funktion selbst als auch das ihr zugeordnete Verhalten beschreiben. Dies bestätigt die Entscheidung auf binäre Klassifikatoren pro Klasse zu setzen, da Mehrklassen-Klassifikation dies nicht abbilden könnte. Die Klasse Daten ist, wie auch schon auf dem PROMISE NFR-Datensatz, seltener als die anderen Anliegen und kommt insgesamt nur auf 1164 Repräsentanten.

FTLR zielt darauf ab, TLs auf ungesehenen Projekten zu bestimmen. Dementsprechend sollte auch die Leistung **NoRBERTs** in einer derartigen Situation bemessen werden. Um dies zu simulieren wird abwechselnd jedes der Nachverfolgbarkeitsprojekte als dieses ungesehene Projekt aufgefasst und **NoRBERT** auf den anderen Projekten in Kombination mit dem PROMISE NFR-Datensatz trainiert. Dies entspricht einer Art loPo-Aufteilungsstrategie, nur das diesmal nicht auch die Leistung auf den PROMISE NFR-Projekten bemessen wird. Somit hat **NoRBERT** jeweils die maximale Trainingsmenge zur Verfügung, die ohne das jeweilige Testprojekt verfügbar ist. Lediglich für das Training der Klasse Nutzerbezogenes existieren keine Informationen aus dem PROMISE NFR-Datensatz, weshalb hier nur auf den jeweils anderen vier Nachverfolgbarkeitsprojekten trainiert wird.

Als Metriken werden sowohl die Genauigkeit als auch das F_1 -Maß für die Vorhersage des Auftretens der Klasse (s. Abschnitt 2.11) betrachtet. Die Genauigkeit verdeutlicht hierbei zwar wie gut **NoRBERT** die Gegebenheiten im Testprojekt allgemein bestimmen kann, wird aber bei den seltenen Klassen von den richtig negativ Klassifikationen dominiert und verdeutlicht somit nicht, wie gut der Klassifikator in der Erkennung dieser seltenen Auftreten ist. Das F_1 -Maß allein hingegen bestimmt zwar diese Güte, bildet aber nicht ab, ob der Klassifikator auch Fehler bei den Anforderungselementen macht, die nicht der Klasse zugeordnet werden sollten. Somit ist die Genauigkeit als Maßzahl für die Gesamtgüte auf dem Projekt zu sehen und das F_1 -Maß kann als weitere Informationsquelle für eine detailliertere Betrachtung der potenziell zu filternden Fälle genutzt werden. Um zusätzlich auch Aussagen über das **NoRBERT**-Modell treffen zu können, werden die Metriken sowohl ungewichtet (\emptyset) als auch mit den jeweiligen Auftreten im Projekt gewichtet (\emptyset_w) gemittelt. Hierbei ist die gewichtete Angabe weniger anfällig für abweichende Ergebnisse auf Projekten, die nur sehr wenige Vorkommen einer Klasse aufweisen (z. B. die fünf NB in LibEST). Der ungewichtete Durchschnitt wird hingegen weniger von abweichenden (potenziell sehr guten) Ergebnissen auf Projekten mit sehr vielen Vorkommen einer Klasse (z. B. eAnci) beeinflusst.

Die notwendigen Skripte zur Replikation der Ergebnisse sowie weitere Evaluationsergebnisse sind auf Zenodo verfügbar [Hey23c].

Tabelle 9.8.: Ergebnisse bezüglich der Genauigkeit verschiedener **NoRBERT**-Konfigurationen auf den beiden Klassen funktionale Aspekte (F) und Nutzerbezogenes (NB). \emptyset gibt den ungewichteten und \emptyset_w den gewichteten Durchschnitt über die Projekte hinweg an. Die größten durchschnittlichen Werte pro Klasse bzw. pro Projekt werden fett dargestellt

Klasse	Modell	Ep.	OS	eTour	iTrust	SMOS	eAnci	LibEST	\emptyset	\emptyset_w
F	BASE	10	✗	.909	.923	.736	.831	.690	.818	.819
			✓	.911	.923	.759	.836	.731	.832	.832
		16	✗	.748	.887	.720	.709	.696	.752	.738
			✓	.905	.908	.743	.848	.687	.818	.823
	LARGE	10	✗	.904	.905	.749	.846	.664	.813	.819
			✓	.918	.926	.782	.833	.665	.825	.825
		16	✗	.912	.943	.738	.838	.696	.825	.826
			✓	.907	.958	.753	.860	.710	.838	.840
NB	BASE	10	✗	.741	.744	.803	.928	.733	.790	.841
			✓	.715	.735	.900	.872	.747	.794	.826
		16	✗	.739	.735	.772	.906	.795	.789	.824
			✓	.725	.798	.784	.904	.717	.785	.831
	LARGE	10	✗	.678	.711	.766	.930	.009	.619	.816
			✓	.688	.783	.761	.908	.768	.781	.820
		16	✗	.722	.759	.784	.922	.591	.756	.833
			✓	.739	.759	.768	.933	.596	.759	.839
\emptyset	BASE	10	✗	.825	.833	.769	.879	.712	.804	.830
			✓	.813	.829	.830	.854	.739	.813	.829
		16	✗	.743	.811	.746	.807	.745	.771	.781
			✓	.815	.853	.763	.876	.702	.802	.827
	LARGE	10	✗	.791	.808	.758	.888	.336	.716	.817
			✓	.803	.854	.771	.870	.717	.803	.822
		16	✗	.817	.851	.761	.880	.643	.790	.830
			✓	.823	.859	.761	.897	.653	.798	.839

9.4.1. Identifikation der Klassen für die Anforderungselementfilter

Zunächst möchte ich einen Einblick in die Leistung **NoRBERTs** auf den beiden als Anforderungselementfilter in **FTLR** implementierten Klassen, Anforderungen mit funktionalen Aspekten (im Folgenden nur noch mit F bezeichnet) und Nutzerbezogenes (NB), geben. Tabelle 9.8 zeigt die erzielten Genauigkeiten der Konfigurationen mit $BERT_{BASE}$ bzw. $BERT_{LARGE}$, 10 oder 16 Epochen und mit oder ohne Überabtaben (OS). Hierbei zeigt sich, dass **NoRBERT** auf der Klasse F sowohl hinsichtlich des ungewichteten als auch gewichteten Durchschnitts jeweils die besten Ergebnisse mit OS erzielt. Dies gilt ebenso für die Ergebnisse auf den

jeweiligen Projekten. Es erzielt also immer das jeweilige Modell mit OS eine höhere Genauigkeit als das gleiche Modell ohne OS. Die einzige Ausnahme stellen die beiden $BERT_{LARGE}$ -basierten Modelle mit 10 Epochen auf eAnci dar. Hier erzielt das Modell ohne OS mit 84,6 % eine leicht höhere Genauigkeit als das Modell mit OS (83,3 %).

Die beste Konfiguration auf F ist das $BERT_{LARGE}$ -basiertes Modell mit 16 Epochen und OS, welches eine gewichtete durchschnittliche Genauigkeit von 84 % erzielt. Es scheint also so, als würde ein Ausgleichen der Verteilung von Positiv- und Negativbeispielen im Training für eine bessere Leistung des resultierenden Klassifikators sorgen. Eine potenzielle Erklärung ist, dass der Klassifikator sich hierdurch weniger auf den Anteil des Auftretens der Klasse im Trainingsdatensatz anpasst und dadurch besser in der Lage ist, auf ungesesehenen Projekten mit einer anderen Verteilung die Auftreten der Klasse zu erkennen. Das Modell ist also weniger überangepasst.

Auf NB sind die Ergebnisse heterogener, da sich je nach Projekt und $BERT$ -Modell ein Überabtauen positiv oder negativ auf das Ergebnis auswirkt. Im Durchschnitt über die Projekte erreichen aber die $BERT_{BASE}$ -basierten Modelle bessere Ergebnisse als die $BERT_{LARGE}$ -basierten. Das $BERT_{BASE}$ -basierte Modell mit 10 Epochen und OS erzielt mit 79,4 % das beste Ergebnis in der durchschnittlichen Genauigkeit. Betrachtet man allerdings die gewichtete durchschnittliche Genauigkeit, so erzielt dieses Modell mit 82,6 % nur das fünfthöchste Ergebnis. Den höchsten Wert erzielt mit 84,1 % das $BERT_{BASE}$ -basierte Modell mit 10 Epochen ohne OS. Der hohe einfache Durchschnittswert des Modells mit OS ergibt sich vor allem aus der deutlich höheren Genauigkeit auf SMOS (90 % gegenüber 80,3 %). Dasselbe Modell schneidet aber auf den anderen Datensätzen schlechter ab. Hier kann der gewichtete Durchschnitt ein exakteres Bild zu den tatsächlich richtig klassifizierten Beispielen liefern. Da außerdem der Wert im einfachen Durchschnitt des $BERT_{BASE}$ -basierten Modells mit 10 Epochen ohne OS nur 0,4 Prozentpunkte hinter dem besten Ergebnis liegt, kann dieses Modell als bestes Modell auf NB festgehalten werden.

Ein weiterer Faktor, der bei der Bewertung für die Klasse NB eine Rolle spielt, ist die Art der Berechnung der Genauigkeit. Die $BERT_{BASE}$ -basierten Modelle erzielen vor allem deshalb bessere Ergebnisse im ungewichteten Durchschnitt als die $BERT_{LARGE}$ -basierten, da diese eine höhere Genauigkeit auf LibEST aufweisen. Dies ist ein anschauliches Beispiel dafür, warum die Genauigkeit trügerisch sein kann und der gewichtete Durchschnitt vorzuziehen ist. Da LibEST nur fünf Beispiele für NB aufweist, erzielt ein Klassifikator, der alle Elemente als nicht-NB klassifiziert, eine Genauigkeit von 99,1 %. Betrachtet man die Ergebnisse im F_1 -Maß, für diese fünf Auftreten der Klasse in LibEST (s. Tabelle 9.9), sieht man, dass alle Konfigurationen nicht in der Lage sind, diese zu identifizieren. Betrachtet man das F_1 -Maß auf den anderen Projekten in Tabelle 9.9, so erzielt das $BERT_{BASE}$ -basierte Modell mit 10 Epochen ohne OS nicht mehr eindeutig das beste Ergebnis, wie es in der Genauigkeit der Fall war. Im F_1 -Maß erzielt die Konfiguration mit $BERT_{LARGE}$, 16 Epochen und OS mit einem gewichteten durchschnittlichen Wert von 81,8 % dasselbe Ergebnis wie das $BERT_{BASE}$ -basierte Modell mit 10 Epochen ohne OS.

Die weiteren Ergebnisse zeigen außerdem, dass für das F_1 -Maß auch auf der Klasse NB ein Überabtauen einen positiven Einfluss hat. Dies bestätigt die Erkenntnis zur Klasse F aus der Genauigkeit, welche sich ebenso für das F_1 -Maß zeigt. Da vor allem die Leistung

Tabelle 9.9.: Ergebnisse, bezüglich des F_1 -Maßes des Auftretens einer Klasse, verschiedener **NoRBERT**-Konfigurationen auf den beiden Klassen funktionale Aspekte (F) und Nutzerbezogenes (NB). \emptyset gibt den ungewichteten und \emptyset_w den gewichteten Durchschnitt über die Projekte hinweg an. Die größten durchschnittlichen Werte pro Klasse bzw. pro Projekt werden fett dargestellt

Klasse	Modell	Ep.	OS	eTour	iTrust	SMOS	eAnci	LibEST	\emptyset	\emptyset_w
F	BASE	10	✗	.946	.958	.845	.885	.807	.888	.886
			✓	.947	.958	.857	.887	.830	.896	.893
		16	✗	.851	.940	.836	.806	.811	.849	.837
			✓	.944	.951	.850	.896	.795	.887	.888
	LARGE	10	✗	.943	.948	.852	.894	.774	.882	.883
			✓	.951	.961	.868	.888	.780	.890	.888
		16	✗	.948	.970	.848	.891	.815	.894	.892
			✓	.945	.978	.855	.904	.811	.898	.898
NB	BASE	10	✗	.648	.747	.780	.929	.062	.633	.818
			✓	.598	.739	.900	.872	.065	.635	.800
		16	✗	.639	.734	.731	.909	.079	.619	.796
			✓	.614	.812	.757	.907	.059	.630	.806
	LARGE	10	✗	.513	.719	.745	.933	.018	.585	.782
			✓	.541	.793	.725	.909	.058	.605	.784
		16	✗	.611	.774	.756	.924	.033	.620	.808
			✓	.653	.772	.743	.934	.034	.627	.818
\emptyset	BASE	10	✗	.797	.852	.813	.907	.434	.761	.852
			✓	.772	.848	.879	.879	.447	.765	.847
		16	✗	.745	.837	.784	.857	.445	.734	.816
			✓	.779	.882	.803	.902	.427	.758	.847
	LARGE	10	✗	.728	.833	.798	.913	.396	.734	.833
			✓	.746	.877	.797	.898	.419	.747	.836
		16	✗	.780	.872	.802	.908	.424	.757	.850
			✓	.799	.875	.799	.919	.422	.763	.858

beim Erkennen eines Auftretens der beiden Klassen eine Rolle für die Anwendung in **FTLR** spielt, deutet dies darauf hin, dass ein Klassifikator mit OS vorzuziehen ist. Lässt man nun die zu erwartende schlechte Leistung auf LibEST außen vor, so wäre das $BERT_{LARGE}$ -basierte Modell mit 16 Epochen und OS das beste Modell auf NB. Des Weiteren zeigen die Ergebnisse, dass je nach Projekt ganz unterschiedliche Konfigurationen am besten abschneiden. Für eine Anwendung in **FTLR** ist allerdings die Leistung in einer Anwendung auf neue ungesehene Projekte relevant, weshalb der Durchschnitt über die Projekte eine höhere Aussagekraft hat. Da für jede Klasse ohnehin ein einzelner binärer Klassifikator trainiert werden muss, kann jeweils die beste Konfiguration pro Klasse gewählt werden.

Aus den Ergebnissen in Tabellen 9.8 und 9.9 ergibt sich für die Klasse F somit eindeutig **NoRBERT** auf Basis eines $BERT_{LARGE}$ mit 16 Epochen und Überabtasten als favorisiertes Modell, da dieses sowohl in Genauigkeit als auch F_1 -Maß am besten abschneidet. Für die Klasse NB kann, der obigen Argumentation folgend, als optimale Konfiguration das $BERT_{LARGE}$ -Modell mit ebenfalls 16 Epochen und Überabtasten gewählt werden.

9.4.2. Identifikation der relevanten Anliegen in funktionalen Anforderungen

In einer zweiten Betrachtung soll zusätzlich **NoRBERTs** Leistung auf den Nachverfolgbarkeitsdatensätzen hinsichtlich der Identifikation von Anliegen in funktionalen Anforderungen untersucht werden, welche zukünftig potenziell relevant für die TLR sein könnten. Hierzu zeigt Tabelle 9.10 die Genauigkeit der verschiedenen **NoRBERT**-Konfigurationen auf den Klassen Funktion, Verhalten und Daten. Auf jeder dieser Klassen erzielt ein $BERT_{LARGE}$ -basiertes Modell die besten Ergebnisse. Hier kann es eine Rolle spielen, dass diese Klassen teils deutlich seltener im Datensatz auftreten und somit ein größeres Modell mit potenziell mehr erlerntem Wissen aus dem Vortraining besser die diskriminierenden Eigenschaften ableiten kann. In der Genauigkeit zeigen sich auf diesen Klassen Modelle ohne OS als vorteilhaft. Im Durchschnitt über die Klassen erzielt das $BERT_{LARGE}$ -basierte Modell mit 16 Epochen und ohne OS mit 71,9 % den besten Wert.

Auf der Klasse Funktion erzielen nahezu alle $BERT_{LARGE}$ -basierten Modelle denselben Wert im gewichteten Durchschnitt (70,6 %). Auch im ungewichteten Durchschnitt liegen die Ergebnisse nahe beieinander. Betrachtet man auch hier die Leistung im F_1 -Maß für positive Beispiele der Klassen (s. Tabelle 9.11), so zeigt sich, dass das Modell mit 16 Epochen ohne OS mit 68 % das beste F_1 -Maß im gewichteten Durchschnitt erzielt. Auch im ungewichteten Durchschnitt liegt dieses Modell mit nur 0,1 Prozentpunkten hinter dem besten Ergebnis durch das $BERT_{LARGE}$ -basierte Modell mit 10 Epochen (67,7 % zu 67,8 %). Auf den Klassen Verhalten und Daten allerdings zeigt sich ein ähnlicher Effekt zwischen Genauigkeit und F_1 -Maß wie bereits auf NB. In der Genauigkeit scheinen Modelle ohne OS besser abzuschneiden als ihre jeweiligen Varianten mit OS. Betrachtet man jedoch das F_1 -Maß, so erzielen wieder die Modelle mit Überabtasten bessere Ergebnisse. So erzielt das $BERT_{LARGE}$ -basierte Modell mit 16 Epochen ohne OS auf der Klasse Verhalten die beste ungewichtete und gewichtete durchschnittliche Genauigkeit von 74,9 % bzw. 75,8 %. Im F_1 -Maß erzielt dieses Modell allerdings nur noch die zweitbesten Durchschnittswerte. Hier erzielt, insbesondere durch einen deutlich besseren Wert auf LibEST, dieselbe Konfiguration mit OS das beste Ergebnis (\emptyset -Genauigkeit: 74,6 %).

Auf der Klasse Daten ist dieser Unterschied noch deutlicher. Erzielen in der Genauigkeit die beiden $BERT_{LARGE}$ -basierten Modelle ohne OS mit 70,9 % bzw. 71,4 % die besten Ergebnisse im ungewichteten bzw. gewichteten Durchschnitt, so erreicht im F_1 -Maß das $BERT_{LARGE}$ -Modell mit 16 Epochen und OS eindeutig die höchste Leistung. Dieses erzielt mit 65,1 % zu 58,9 % bzw. 63,3 % zu 61 % deutlich bessere Ergebnisse als die besten Modelle aufgrund der Genauigkeit. Da die Klasse Daten diejenige mit der größten Diskrepanz zwischen

Tabelle 9.10.: Ergebnisse bezüglich der Genauigkeit verschiedener **NoRBERT**-Konfigurationen auf den Klassen Funktion, Verhalten und Daten. \emptyset gibt den ungewichteten und \emptyset_w den gewichteten Durchschnitt über die Projekte hinweg an. Die größten durchschnittlichen Werte pro Klasse bzw. pro Projekt werden fett dargestellt

Klasse	Modell	Ep.	OS	eTour	iTrust	SMOS	eAnci	LibEST	\emptyset	\emptyset_w
Funktion	BASE	10	✗	.758	.667	.638	.669	.618	.670	.674
			✓	.734	.670	.632	.665	.588	.658	.663
		16	✗	.494	.533	.506	.513	.575	.524	.519
			✓	.701	.679	.642	.719	.600	.668	.681
	LARGE	10	✗	.823	.688	.657	.681	.680	.706	.706
			✓	.783	.634	.695	.718	.637	.693	.706
		16	✗	.816	.696	.672	.705	.600	.698	.706
			✓	.774	.676	.661	.715	.653	.696	.705
Verhalten	BASE	10	✗	.765	.705	.774	.774	.542	.712	.725
			✓	.764	.685	.743	.789	.533	.703	.720
		16	✗	.452	.464	.515	.509	.508	.490	.493
			✓	.772	.690	.745	.795	.513	.703	.722
	LARGE	10	✗	.771	.732	.770	.719	.611	.721	.721
			✓	.792	.679	.789	.673	.637	.714	.712
		16	✗	.778	.720	.912	.779	.558	.749	.758
			✓	.779	.720	.770	.750	.565	.717	.724
Daten	BASE	10	✗	.748	.735	.705	.770	.584	.708	.713
			✓	.723	.753	.649	.771	.573	.694	.697
		16	✗	.569	.619	.492	.653	.474	.562	.565
			✓	.751	.774	.678	.782	.504	.698	.702
	LARGE	10	✗	.685	.747	.692	.809	.595	.705	.714
			✓	.837	.503	.529	.799	.630	.660	.668
		16	✗	.743	.777	.634	.779	.612	.709	.709
			✓	.823	.774	.711	.792	.363	.693	.700
\emptyset	BASE	10	✗	.757	.702	.706	.738	.581	.697	.704
			✓	.740	.702	.675	.742	.565	.685	.693
		16	✗	.505	.539	.504	.558	.519	.525	.526
			✓	.741	.714	.688	.765	.539	.690	.702
	LARGE	10	✗	.759	.722	.706	.736	.628	.711	.714
			✓	.804	.605	.671	.730	.635	.689	.696
		16	✗	.779	.731	.739	.755	.590	.719	.724
			✓	.792	.723	.714	.752	.527	.702	.709

Tabelle 9.11.: Ergebnisse, bezüglich des F_1 -Maßes des Auftretens einer Klasse, verschiedener **NoBERT**-Konfigurationen auf den Klassen Funktion, Verhalten und Daten. \emptyset gibt den ungewichteten und \emptyset_w den gewichteten Durchschnitt über die Projekte hinweg an. Die größten durchschnittlichen Werte pro Klasse bzw. pro Projekt werden fett dargestellt

Klasse	Modell	Ep.	OS	eTour	iTrust	SMOS	eAnci	LibEST	\emptyset	\emptyset_w
Funktion	BASE	10	✗	.796	.673	.695	.591	.585	.668	.657
			✓	.770	.643	.691	.557	.588	.650	.636
		16	✗	.590	.475	.570	.364	.518	.503	.477
			✓	.750	.658	.699	.645	.599	.670	.670
	LARGE	10	✗	.846	.673	.685	.609	.578	.678	.672
			✓	.801	.539	.735	.617	.494	.637	.646
16		✗	.834	.675	.722	.648	.504	.677	.680	
		✓	.789	.633	.718	.642	.564	.669	.672	
Verhalten	BASE	10	✗	.826	.635	.823	.772	.636	.739	.755
			✓	.825	.562	.805	.787	.612	.718	.745
		16	✗	.589	.400	.626	.466	.611	.538	.539
			✓	.832	.636	.809	.789	.616	.737	.756
	LARGE	10	✗	.834	.656	.819	.741	.611	.732	.743
			✓	.846	.585	.836	.694	.655	.723	.733
16		✗	.837	.671	.922	.776	.498	.741	.755	
		✓	.839	.691	.827	.753	.622	.746	.755	
Daten	BASE	10	✗	.676	.712	.683	.551	.402	.605	.599
			✓	.624	.765	.613	.518	.493	.602	.589
		16	✗	.428	.605	.447	.180	.361	.404	.376
			✓	.641	.770	.636	.521	.480	.610	.596
	LARGE	10	✗	.633	.766	.626	.566	.510	.620	.610
			✓	.758	.669	.692	.505	.306	.586	.578
16		✗	.665	.796	.647	.496	.342	.589	.575	
		✓	.746	.785	.661	.531	.532	.651	.633	
\emptyset	BASE	10	✗	.766	.673	.734	.638	.541	.670	.671
			✓	.740	.657	.703	.621	.564	.657	.657
		16	✗	.536	.493	.548	.337	.497	.482	.464
			✓	.741	.688	.715	.652	.565	.672	.674
	LARGE	10	✗	.771	.698	.710	.639	.566	.677	.675
			✓	.802	.598	.754	.605	.485	.649	.653
16		✗	.779	.714	.764	.640	.448	.669	.670	
		✓	.791	.703	.735	.642	.573	.689	.687	

Tabelle 9.12.: Beste **NoRBERT**-Konfigurationen auf den Nachverfolgbarkeitsdatensätzen

Klasse	NoRBERT-Konfiguration		
	Basismodell	Epochenanzahl	Überabtasten
F	LARGE	16	✓
NB	LARGE	16	✓
Funktion	LARGE	16	✗
Verhalten	LARGE	16	✓
Daten	LARGE	16	✓

positiven und negativen Auftreten in den Datensätzen ist, zeugt eine hohe Leistung im F_1 -Maß der positiven Auftreten der Klasse von einer besseren Eignung des Modells. Die Datensätze enthalten teils deutlich weniger positive Vorkommen der Klasse Daten als negative Vorkommen und sind somit stark unausgeglichen. Ist ein Klassifikator trotzdem in der Lage, die Auftreten der Klasse zu erkennen, hat er besser von den Trainingsdaten abstrahiert als ein Klassifikator, der eine höhere Genauigkeit erzielt, aber ein schlechteres F_1 -Maß aufweist. Denn eine hohe Genauigkeit kann in solch unausgeglichenen Datensätzen durch das Bevorzugen von Nichtauftreten der Klasse und damit einer hohen Zahl von richtig negativen Klassifikationen erzielt werden. Dies erklärt auch warum ein Überabtasten sich in diesem Fall positiv auf das F_1 -Maß auswirkt. Denn auch hier kann durch ein Ausgleichen der Verteilung von Positiv- und Negativbeispielen im Training des Modells eine Überanpassung des Modells auf die Mehrheitsverhältnisse verhindert werden.

9.4.3. Zusammenfassung

Zusammenfassend lässt sich aus dieser intrinsischen Evaluation festhalten, dass die Ergebnisse der Klassifikation der Anforderungselemente in den für **FTLR** relevanten Projekten vielversprechend sind. Gerade auf den beiden für die Filterung ausgewählten Klassen F und NB erzielt **NoRBERT** mit einem F_1 -Maß von 89,8 % bzw. 81,8 % sehr gute Ergebnisse. Insbesondere gemessen an der kleinen Datenmenge, die zum Training zur Verfügung steht, sowie den verschiedenen Formen von Anforderungen und damit Anforderungselementen in Trainings- und Testmenge sind diese Ergebnisse vielversprechend. Eine abschließende Eignung für ein Filtern auf Basis dieser Klassifikation in **FTLR** lässt sich aber erst durch den Einsatz in der TLR und damit einer extrinsischen Evaluation bewerten, welche im nächsten Kapitel beschrieben wird.

Auch die Ergebnisse die **NoRBERT** auf den Klassen für Anliegen in funktionalen Anforderungen zeigt, deuten auf eine Eignung einer automatischen Klassifikation für eine weitere Verwendung hin. Ein nächster Schritt, um diese Klassen tatsächlich für die TLR zu verwenden, wäre eine entsprechende Klassifikation auf Quelltextseite und ein geeignetes Verfahren diese Information für die Abbildung zu nutzen. Diese Klassen für die TLR zu verwenden, liegt aber, wie bereits in Abschnitt 8.2 beschrieben, nicht im Fokus dieser Arbeit. Die Ergebnisse bestätigen die Ergebnisse auf dem PROMISE NFR-Datensatz und

zeigen, dass **NoRBERT** für die Klassifikation von Anliegen in funktionalen Anforderungen perspektivisch beispielsweise auch von anderen Forscher:innen genutzt werden kann. Informationen zu Anliegen in funktionalen Anforderungen könnten beispielsweise ebenso hilfreich für Entwickler:innen während des Entwurfs oder für ein automatisches Erstellen von Modellen aus Anforderungen sein.

Tabelle 9.12 fasst die jeweils besten Konfigurationen pro Klasse zusammen und beantwortet damit Forschungsfrage **FF₉**. Die Zusammenfassung verdeutlicht, dass für einen Großteil der Klassen ähnliche Konfigurationen die besten Ergebnisse liefern. Insbesondere erweist sich das $BERT_{LARGE}$ -basierte Modell mit 16 Epochen als geeignetstes Modell. Und auch ein Überabtasten hat in vier von fünf Fällen einen positiven Effekt auf die erzielte Leistung.

9.5. Gefährdungen der Gültigkeit

In diesem Abschnitt möchte ich Gefährdungen für die Gültigkeit der in diesem Kapitel durchgeführten Experimente und den aus ihnen abgeleiteten Aussagen diskutieren.

Externe Gültigkeit Um die Generalisierbarkeit **NoRBERTs** zu zeigen, wurden verschiedene Evaluationsstrategien eingesetzt. Die loPo- und p-fold-Strategien sollen hierbei Generalisierbarkeit bemessen. Da allerdings die Projekte der Datensätze nicht notwendigerweise gute Repräsentanten für die Vielfalt möglicher Projekte darstellen, existiert hier eine mögliche Gefahr für die externe Gültigkeit der Ergebnisse. Außerdem sind sowohl der PROMISE NFR-Datensatz, als auch die Nachverfolgbarkeitsdatensätze verzerrte Abbilder der Realität, da sie entweder mit Fokus auf nichtfunktionale oder funktionale Anforderungen erstellt wurden. Dies führt jeweils zu unausgeglichene Datensätzen und damit zu einer möglichen Verschattung der Ergebnisse durch eine einfache Vorhersage der Mehrheitsklasse. Zusätzlich wurden alle Anforderungen der Projekte des PROMISE NFR-Datensätzen und einige der Anforderungen der Nachverfolgbarkeitsdatensätze von Studierenden verfasst und spiegeln somit nicht unbedingt Industriestandards wider. Deshalb kann die Gültigkeit von Aussagen zur Generalisierbarkeit basierend auf diesen Datensätzen nicht garantiert werden. Da die Datensätze allerdings als Vergleichsdatsätze für diese Aufgaben in den Forschungsgemeinschaften zur Anforderungsklassifikation und zur TLR gelten und dementsprechend weit verbreitet verwendet werden, erlauben sie zumindest einen Vergleich verschiedener Ansätze.

Konstruktvalidität Um potenzielle Gefährdungen für die Konstruktvalidität abzumildern, wurden weitverbreitete Versuchsaufbauten und Metriken verwendet. In den Experimenten wurde ein fester Startwert für die Zufallszahlengeneratoren gewählt, um Reproduzierbarkeit der Ergebnisse zu gewährleisten. Für die Experimente auf dem umetikettierten PROMISE NFR-Datensatz wurde derselbe Startwert (42) wie in der Arbeit von Dalpiaz et al. [Dal+19] gewählt, um die Vergleichbarkeit der Ansätze zu garantieren. In allen anderen Experimenten wurde der zufällig ausgewählte Startwert 904727489 verwendet. Außerdem

wurde, um unsystematisches Ausprobieren zu verhindern, der Hyperparameter Epochenanzahl systematisch erhöht. Dies folgt der Annahme, dass eine optimale Epochenanzahl hinsichtlich Trainingsleistung und Überanpassung existiert.

Interne Gültigkeit Eine Gefahr für die interne Gültigkeit stellt die Erstellung des Goldstandards für die funktionalen Anliegen im umetikettierten PROMISE NFR und des Goldstandards für die Klassifikation der Nachverfolgbarkeitsdatensätze dar. Diese wurden mit dem Ansatz im Hinterkopf erstellt. Dies birgt das Risiko einer Verzerrung. Um eine Überprüfung der Klassifikation und ein Nachvollziehen der Ergebnisse zu ermöglichen, wurden beide Datensätze veröffentlicht (s. [Hey+20b] und [Hey23a]).

10. Evaluation der Anforderungselementfilterung

Nach der Evaluation des grundlegenden Verfahrens in Kapitel 6 beschäftigt sich dieses Kapitel nun mit Experimenten zur automatischen Anforderungselementfilterung und liefert demnach Erkenntnisse bezüglich These T_2 . Diese nimmt an, dass es eine Klassifikation von Anforderungen erlaubt, irrelevante Teile der Eingabe von der Wiederherstellung von Nachverfolgbarkeitsverbindungen auszuschließen und damit die Präzision eines wortbettungs-basierten Verfahrens wie **FTLR** zu steigern. Um diese These zu untersuchen, wird in den folgenden Abschnitten die Leistung **FTLRs** mit einer Filterung von Anforderungselementen auf Basis von **NoRBERT** in Bezug zu den Ergebnissen ohne eine derartige Filterung gestellt. Hierbei werden die in Abschnitt 9.4.3 identifizierten, besten **NoRBERT**-Konfigurationen dazu verwendet, die Klassifikationsergebnisse auf den Datensätzen eTour, iTrust, SMOS, eAnci und LibEST zu erzeugen. Um ein möglichst realistisches Szenario für einen späteren Einsatz von **NoRBERT** in Kombination mit **FTLR** geben zu können, werden die Klassifikationsergebnisse, wie in Abschnitt 9.4 beschrieben, erzeugt. Es wird also für jedes Projekt ein Modell auf dem PROMISE NFR-Datensatz sowie den anderen vier Projekten trainiert und mit diesem die Klassifikation für das jeweilig zu analysierende Projekt durchgeführt.

Da die Ergebnisse, die **NoRBERT** auf den beiden als Filter verwendeten Klassen (NF und NB) erzielt, zwar vielversprechend aber nicht perfekt sind, kann es sein, dass durch fehlerhafte Klassifikation fälschlicherweise relevante Teile der Eingabe entfernt werden. Um zu bemessen, welche Ergebnisse mit einer optimalen Ausgangslage aus der Klassifikation erreichbar wären, wird **FTLR** außerdem auch mit der Musterlösung der Klassifikation ausgeführt. Hiermit kann dann die obere Schranke für die Leistung des Anforderungselementfilters auf diesen Projekten bestimmt werden.

Zunächst wird in Abschnitt 10.1 die Leistung **FTLRs** mit Anforderungselementfilter mit derjenigen **FTLR** ohne Anforderungselementfilter und ohne **▼**-Option verglichen. Da **▼** selbst einen Filter darstellt, der auf Basis von Anwendungsfallvorlagenelementen filtert, kann in diesem Versuchsaufbau zunächst verdeutlicht werden, welchen Einfluss die Anforderungselementfilter auf ungefilterte Eingaben haben. Ein Hintergedanke, hinter dem Einführen der Anforderungselementfilter, war ein Einsatz auf Anforderungen, die keiner Vorlage folgen. Außerdem kann auch bei Anwendungsfallbeschreibungen, die einer Vorlage folgen, ein Anforderungselementfilter ohne explizites Wissen über die Elemente und ihre Benennung eingesetzt werden. Um diesen Effekt zu quantifizieren und zu überprüfen, ob die Leistung der beiden Filterarten vergleichbar ist, wird in Abschnitt 10.2

die Leistung der Anforderungselementfilter mit der Leistung der Filterung auf Basis der Anwendungsfallvorlagenelemente (▼) verglichen. Abschließend wird in Abschnitt 10.3 überprüft, ob eine Kombination der beiden Filtervarianten eine Leistungssteigerung gegenüber ihrer einzelnen Anwendung erzielt. Es werden in diesem Kapitel also die folgenden Forschungsfragen adressiert:

- FF₁₀**: Welche Leistung erzielt **FTLR** mit vollautomatischem Anforderungselementfilter im Vergleich zu **FTLR** ohne jeglichen Elementfilter?
- FF₁₁**: Was ist die obere Schranke der Leistung, die **FTLR** mit Anforderungselementfilter auf den Vergleichsdatensätzen erreichen kann?
- FF₁₂**: Ist die Leistung **FTLRs** mit vollautomatischem Anforderungselementfilter vergleichbar mit derjenigen auf Basis der Filterung von Anwendungsfallvorlagenelementen (▼)?
- FF₁₃**: Kann eine Kombination von Anforderungselementfilter und Filterung auf Basis von Anwendungsfallvorlagenelementen die Leistung **FTLRs** gegenüber den einzelnen Varianten steigern?

10.1. Vergleich mit ungefilterten Varianten

Zunächst soll in diesem Abschnitt beantwortet werden, ob der Einsatz von Anforderungselementfiltern auf Basis der Klassen Nutzerbezogenes (NB) und funktionale Aspekte (F) eine höhere Präzision und auch ein höheres F_1 -Maß als **FTLR**-Varianten ohne Filterung erzielt (Forschungsfrage **FF₁₀**). Hierzu werden die Ergebnisse der drei **FTLR**-Varianten verglichen, die ohne Anwendungsfallvorlagenelementfilter (▼) die besten Ergebnisse erzielten. Dies sind zum einen **FTLR** ohne Erweiterung, **FTLR** mit Methodenkommentaren (☛) und **FTLR** mit Methodenkommentaren und Aufrufabhängigkeiten (☛☛). Als Anforderungselementfilter werden hierbei sowohl die beiden Filter einzeln als auch in Kombination verglichen.

Tabelle 10.1 zeigt die Ergebnisse in Präzision, Ausbeute und F_1 -Maß unter Einsatz der Klassifikationsergebnisse der jeweils besten **NoRBERT**-Konfigurationen (vgl. Tabelle 9.12). Zunächst lässt sich festhalten, dass nahezu alle Arten von Filtern die Präzision des Verfahrens erhöhen. Einzige Ausnahme stellt der NB-Filter auf iTrust dar, welcher sowohl in Präzision als auch Ausbeute niedrigere Werte erzielt. Die erhöhten Präzisionen sorgen allerdings auch zumeist für eine leichte Reduktion der Ausbeute. Dies lässt sich mit fälschlicherweise gefilterten Elementen erklären, die aus den Fehlern des Klassifikators entstehen. Die eigentlich relevanten Elemente stehen damit **FTLR** nicht mehr für die Abbildung auf Quelltextelemente zur Verfügung und sorgen dafür, dass manche der vormals identifizierten TLs nicht mehr erkannt werden. Besonders auffällig ist dies bei den NB-Filtern auf dem LibEST-Projekt. Hier sorgt die Reduktion der Ausbeute um bis zu 40 Prozentpunkte sogar für eine Reduktion des F_1 -Maßes um mehr als ein Drittel. Auch auf eAnci reduzieren die NB-Filter die Ausbeute um bis zu 9,5 Prozentpunkte. Hier überwiegt aber die Steigerung

Tabelle 10.1.: Vergleich der Anforderungsfiltervarianten mit den ungefilterten **FTLR**-Ergebnissen auf der festgelegten Schwellenwertkombination (ORG). \rightleftarrows markiert Varianten, die Aufrufabhängigkeiten und \bullet , die Methodenkommentare verwenden. NF bezeichnet Varianten, die Anforderungselemente ohne funktionale Aspekte und NB Varianten, die Nutzerbezogenes filtern. Die besten Ergebnisse im F_1 -Maß pro Projekt und die beste Präzision, Ausbeute und F_1 -Maß im Durchschnitt über alle Projekte sind fett dargestellt

FTLR-Variante:					\bullet			\rightleftarrows		
Projekt	Filter	Präz.	Ausb.	F_1	Präz.	Ausb.	F_1	Präz.	Ausb.	F_1
eTour	Ohne	.271	.701	.391	.267	.688	.385	.270	.692	.388
	NF	.296	.666	.410	.294	.659	.406	.296	.659	.408
	NB	.314	.656	.424	.309	.646	.418	.312	.649	.421
	NF + NB	.314	.646	.423	.311	.640	.419	.314	.643	.422
iTrust	Ohne	.135	.322	.190	.142	.374	.206	.149	.346	.209
	NF	.156	.318	.210	.156	.367	.219	.165	.339	.222
	NB	.119	.192	.147	.130	.231	.167	.142	.220	.173
	NF + NB	.146	.189	.165	.149	.224	.179	.162	.213	.184
SMOS	Ohne	.404	.340	.369	.411	.356	.382	.419	.352	.383
	NF	.414	.331	.368	.420	.350	.382	.428	.345	.382
	NB	.437	.304	.358	.447	.320	.373	.449	.312	.368
	NF + NB	.437	.304	.358	.447	.320	.373	.449	.312	.368
eAnci	Ohne	.224	.261	.241	.204	.291	.240	.204	.280	.236
	NF	.283	.226	.251	.240	.257	.249	.240	.245	.242
	NB	.463	.175	.254	.392	.201	.266	.392	.196	.261
	NF + NB	.456	.166	.243	.393	.198	.263	.394	.192	.258
LibEST	Ohne	.403	.490	.442	.431	.623	.509	.431	.623	.509
	NF	.417	.456	.436	.446	.593	.509	.446	.593	.509
	NB	.533	.235	.327	.538	.275	.364	.538	.275	.364
	NF + NB	.592	.206	.305	.524	.216	.306	.524	.216	.306
\emptyset	Ohne	.287	.423	.327	.291	.466	.344	.294	.459	.345
	NF	.313	.399	.335	.311	.445	.353	.315	.436	.353
	NB	.373	.312	.302	.363	.334	.317	.367	.330	.318
	NF + NB	.389	.302	.299	.365	.319	.308	.368	.315	.308

der Präzision diesen Rückgang, weshalb die Filtervarianten alle ein besseres F_1 -Maß erzielen als **FTLR** ohne einen der Filter. Selbiges gilt für den eTour-Datensatz. eAnci und eTour sind auch die beiden Projekte, auf denen der NB-Filter ein besseres Ergebnis erzielt als der NF-Filter. Diese beiden Projekte weisen besonders häufig Anforderungselemente auf, die lediglich die Tätigkeiten des Nutzers beschreiben, aber nicht gleichzeitig funktionale Aspekte enthalten. Diese herauszufiltern zeigt sich also als vorteilhaft für die TLR.

Betrachtet man die besten Ergebnisse im F_1 -Maß pro Projekt, so erzielt auf allen Projekten außer SMOS eine Anforderungselementfiltervariante den höchsten Wert. Auf SMOS ist der

Tabelle 10.2.: Vergleich der Ergebnisse im F_1 -Maß der unterschiedlichen Schwellenwertkombinationen auf den Anforderungsfiltervarianten mit den ungefilterten **FTLR**-Ergebnissen. \rightleftarrows markiert Varianten, die Aufrufabhängigkeiten und \bullet , die Methodenkommentare verwenden. NF bezeichnet Varianten, die Anforderungselemente ohne funktionale Aspekte und NB Varianten, die Nutzerbezogenes filtern. Die besten Ergebnisse im Durchschnitt über alle Projekte sind fett dargestellt

Projekt	Variante: Filter				\bullet			\rightleftarrows		
		ORG	MED	OPT	ORG	MED	OPT	ORG	MED	OPT
eTour	Ohne	.391	.355	.440	.385	.358	.425	.388	.365	.427
	NF	.410	.369	.464	.406	.370	.442	.408	.377	.442
	NB	.424	.372	.469	.418	.378	.448	.421	.384	.449
	NF + NB	.423	.371	.469	.419	.377	.451	.422	.383	.451
iTrust	Ohne	.190	.166	.215	.206	.172	.253	.209	.174	.239
	NF	.210	.180	.238	.219	.182	.261	.222	.185	.250
	NB	.147	.156	.170	.167	.169	.190	.173	.165	.182
	NF + NB	.165	.175	.185	.179	.183	.199	.184	.178	.194
SMOS	Ohne	.369	.364	.396	.382	.387	.405	.383	.386	.405
	NF	.368	.367	.402	.382	.390	.410	.382	.389	.411
	NB	.358	.362	.405	.373	.382	.411	.368	.380	.414
	NF + NB	.358	.362	.406	.373	.382	.411	.368	.380	.414
eAnci	Ohne	.241	.247	.261	.240	.242	.256	.236	.242	.255
	NF	.251	.258	.281	.249	.259	.271	.242	.259	.270
	NB	.254	.266	.295	.266	.279	.298	.261	.278	.295
	NF + NB	.243	.263	.293	.263	.276	.295	.258	.276	.292
LibEST	Ohne	.442	.524	.552	.509	.538	.552	.509	.538	.552
	NF	.436	.535	.553	.509	.538	.555	.509	.538	.555
	NB	.327	.400	.517	.364	.478	.548	.364	.478	.548
	NF + NB	.305	.367	.494	.306	.439	.522	.306	.439	.522
\emptyset	Ohne	.327	.331	.372	.344	.339	.378	.345	.341	.375
	NF	.335	.342	.388	.353	.348	.388	.353	.350	.386
	NB	.302	.311	.371	.317	.337	.379	.318	.337	.378
	NF + NB	.299	.308	.369	.308	.331	.375	.308	.331	.375

Unterschied mit 0,1 Prozentpunkten zwischen **FTLR** \rightleftarrows \bullet ohne Filter (38,3 %) und mit NF-Filter (38,2 %) allerdings sehr gering. Auf den anderen Projekten sind die Verbesserungen teils sehr viel deutlicher. So erzielt die NB-Filter-Variante für **FTLR** ohne Erweiterungen auf eTour (42,4 %) eine Verbesserung um 3,3 Prozentpunkte (8,4 %) gegenüber **FTLR** ohne Filterung (39,1 %). Auch auf iTrust ist für **FTLR** \rightleftarrows \bullet eine Verbesserung um 1,3 Prozentpunkte (6 %) zu beobachten, hierbei allerdings durch die NF-Filter-Variante (22,2 % zu vormals 20,9 %).

Im Durchschnitt über alle Projekte erzielt ein alleiniges Filtern von Anforderungselementen ohne funktionale Aspekte (NF) auf allen drei Varianten die besten Ergebnisse im F_1 -Maß. Ein alleiniges Filtern von Nutzerbezogenem (NB) oder einer Kombination der beiden Filter erzielt schlechtere Ergebnisse als die Variante ohne Filter. Dieses Ergebnis ist stark davon beeinflusst, dass auf iTrust und LibEST ein Filtern von Nutzerbezogenem aufgrund ungünstiger Falschklassifikationen deutlich schlechtere Ergebnisse erzielt¹. Im Durchschnitt lässt sich außerdem nochmals bestätigen, dass die Präzision durch alle Filter steigt, die Ausbeute aber für **FTLR** ohne Filter am besten ist. Die Verbesserungen im F_1 -Maß durch den NF-Filter sind in einem Wilcoxon-Vorzeichen-Rang-Test in keiner der drei Varianten signifikant bezüglich des Signifikanzniveaus von 0,05.

Betrachtet man allerdings die erzielten F_1 -Maße für die angepassten festen Schwellenwerte (MED) sowie die optimierten Schwellenwertkombinationen (OPT) in Tabelle 10.2, so sind die erzielten Verbesserungen mit dem NF-Filter auf allen drei **FTLR**-Varianten mit einem p-Wert von 0,03125 signifikant gegenüber **FTLR** ohne Filter. Die Ergebnisse in Tabelle 10.2 verdeutlichen außerdem, dass im Durchschnitt über die Projekte auf allen Schwellenwertbetrachtungen das Anwenden des NF-Filters die besten Ergebnisse im F_1 -Maß erzielt. Dabei wird der höchste durchschnittliche Wert mit einem F_1 -Maß von 38,8 % durch **FTLR** mit NF-Filter und Methodenkommentaren bei optimierten Schwellenwerten erzielt.

Der Einbezug von Methodenkommentaren steigert auch auf den Varianten mit NB-Filter das Ergebnis und lediglich auf den angepassten Schwellenwerten (MED) steigert der Einbezug von Aufrufabhängigkeiten (\Rightarrow) das Ergebnis mit NF-Filter nochmals. Die Ergebnisse mit den auf Basis des Medians angepassten Schwellenwerten (MED) erzielen auf den Varianten \bullet und $\Rightarrow\bullet$ mit NF-Filter leicht schlechtere Ergebnisse als die ursprüngliche Kombination (ORG). Lediglich auf **FTLR** ohne Erweiterungen ist eine Verbesserung um 0,7 Prozentpunkte festzustellen (34,2 % gegenüber 33,5 %). Insgesamt bleiben die Abstände zwischen den Ergebnissen mit den unterschiedlichen Schwellenwertkombinationen ähnlich zu denjenigen ohne Anwendung eines Filters, was darauf hindeutet, dass der Einfluss der Filter sich gleichmäßig auf die bereits als günstig erachteten Schwellenwertkombinationen auswirkt.

Aus dieser Betrachtung lässt sich also festhalten, dass **FTLRs** Leistung durch den Einsatz des NF-Filters zum Teil sogar signifikant verbessert werden kann. Den Hauptanteil daran trägt die deutlich erhöhte Präzision, welche das Ziel des Filterns war. Da hierbei zumeist auch die Gesamtleistung, gemessen im F_1 -Maß, gesteigert wird, kann hiermit eine erste zustimmende Aussage bezüglich These T_2 getroffen werden. Da allerdings einige der Ergebnisse durch Fehlklassifikationen nachteilig beeinflusst wurden, soll im Folgenden das volle Potenzial des Filters anhand des Klassifikationsgoldstandards bemessen werden.

Tabelle 10.3.: Vergleich der Anforderungsfiltervarianten auf Basis des Goldstandards mit den ungefilterten **FTLR**-Ergebnissen auf der festgelegten Schwellenwertkombination (ORG). \rightleftarrows markiert Varianten, die Aufrufabhängigkeiten und \bullet , die Methodenkommentare verwenden. NF bezeichnet Varianten, die Anforderungselemente ohne funktionale Aspekte und NB Varianten, die Nutzerbezogenes filtern. Die besten Ergebnisse im F_1 -Maß pro Projekt und die beste Präzision, Ausbeute und F_1 -Maß im Durchschnitt über alle Projekte sind fett dargestellt

FTLR-Variante:					\bullet			\rightleftarrows		
Projekt	Filter	Präz.	Ausb.	F_1	Präz.	Ausb.	F_1	Präz.	Ausb.	F_1
eTour	Ohne	.271	.701	.391	.267	.688	.385	.270	.692	.388
	NF	.376	.649	.476	.369	.640	.468	.372	.640	.470
	NB	.421	.633	.506	.415	.620	.497	.414	.620	.497
	NF + NB	.421	.633	.506	.415	.620	.497	.414	.620	.497
iTrust	Ohne	.135	.322	.190	.142	.374	.206	.149	.346	.209
	NF	.155	.318	.208	.155	.367	.218	.164	.339	.221
	NB	.119	.136	.127	.131	.157	.143	.139	.147	.143
	NF + NB	.153	.133	.142	.153	.150	.152	.166	.143	.154
SMOS	Ohne	.404	.340	.369	.411	.356	.382	.419	.352	.383
	NF	.428	.320	.366	.435	.343	.384	.447	.342	.387
	NB	.450	.308	.366	.457	.330	.383	.465	.321	.380
	NF + NB	.450	.308	.366	.457	.330	.383	.465	.321	.380
eAnci	Ohne	.224	.261	.241	.204	.291	.240	.204	.280	.236
	NF	.311	.231	.265	.271	.259	.265	.272	.249	.260
	NB	.463	.187	.266	.405	.217	.282	.399	.206	.272
	NF + NB	.463	.187	.266	.405	.217	.282	.399	.206	.272
LibEST	Ohne	.403	.490	.442	.431	.623	.509	.431	.623	.509
	NF	.409	.451	.429	.440	.608	.510	.440	.608	.510
	NB	.403	.490	.442	.431	.623	.509	.431	.623	.509
	NF + NB	.409	.451	.429	.440	.608	.510	.440	.608	.510
\emptyset	Ohne	.287	.423	.327	.291	.466	.344	.294	.459	.345
	NF	.336	.394	.349	.334	.443	.369	.339	.435	.370
	NB	.371	.351	.342	.368	.389	.363	.370	.383	.360
	NF + NB	.379	.342	.342	.374	.385	.365	.377	.380	.363

Ergebnisse unter Verwendung des Klassifikationsgoldstandards

Um die obere Schranke für die Leistung **FTLRs** mit dieser Form von Anforderungselementfilter zu bemessen und damit Forschungsfrage **FF₁₁** zu beantworten, wird nun anstatt der durch die Klassifikatoren automatisch erzeugten Klassifikationsergebnisse, die Zuwei-

¹ Zur Erinnerung, der LibEST-Datensatz enthält nur fünf Auftreten der Klasse NB und der ausgewählte Klassifikator erzielt lediglich eine Genauigkeit von 59,6 %.

sung der Elemente aus dem Goldstandard (s. Tabelle 9.7) verwendet. Tabelle 10.3 zeigt die Ergebnisse auf der originalen Schwellenwertkombination (ORG). Zunächst fällt auf, dass im Gegensatz zu den Ergebnissen aus der automatischen Klassifikation nun auch auf SMOS die besten Ergebnisse durch einen Anforderungselementfilter erzielt werden. Somit lassen sich die schlechteren Ergebnisse aus Abschnitt 10.1 auf Fehlklassifikationen zurückführen. Die Steigerung mit dem NF-Filter auf der $\Rightarrow\bullet$ -Variante fällt zwar mit 0,4 Prozentpunkten sehr gering aus, zeigt aber, dass auch auf diesem Projekt ein Filtern nicht leistungsmindernd sein muss.

Auch auf den anderen Projekten kann eine Steigerung im erzielten F_1 -Maß festgestellt werden, welches bei einem Einsatz einer Musterlösung zu erwarten war. Lediglich auf iTrust sind die Ergebnisse mit dem Goldstandard leicht schlechter als diejenigen aus der automatischen Klassifikation. Dies liegt unter anderem daran, dass die automatische Klassifikation auf iTrust mit einem F_1 -Maß von 97,8 % für die Klasse F bereits ein nahezu perfektes Ergebnis erzielt. Hierbei werden durch den Klassifikator 6 Elemente mehr herausgefiltert als die Musterlösung vorsieht. Diese zusätzlich herausgefilterten Anforderungselemente sorgen dafür, dass eine falsche Nachverfolgbarkeitsverbindung weniger beim Einsatz der automatischen Klassifikation identifiziert wird, als dies mit der Musterlösung der Fall ist. Dies sorgt für die minimale Diskrepanz in der Präzision. Auf den NB-Filter-Ergebnissen ist allerdings eine deutlichere Leistungsdifferenz sichtbar. Hier erzielt der automatische Klassifikator tatsächlich eine bessere Entscheidung hinsichtlich der auszusortierenden nutzerbezogenen Anforderungselemente als die menschlichen Annotatoren. Insbesondere wird eine niedrigere Ausbeute mit der Musterlösung erzielt. Eine Erklärung hierfür stellt die geringe Menge an Anforderungselementen in iTrust dar, welche ausschließlich Nutzerbezogenes enthalten (11 der 205 nutzerbezogenen Anforderungselemente). Aufgrund der Entwurfsentscheidung durch den NB-Filter alle Auftreten der Klasse herauszufiltern und diese Entscheidung nicht von dem Vorhandensein funktionaler Aspekte abhängig zu machen, können Informationen zu funktionalen Aspekten verloren gehen. Die automatische Klassifikation identifiziert anstatt der 205 NB-Auftreten in der Musterlösung nur 150 Elemente als Nutzerbezogenes. Dies führt dazu, dass weniger funktionale Aspekte verloren gehen und damit potenziell eine bessere Identifikation der TLs durchgeführt werden kann. Wie auch schon bei der automatischen Klassifikation erzielen auf eTour und eAnci die Filter, die NB filtern, die besten Werte im F_1 -Maß, wohingegen auf iTrust, SMOS und LibEST die NF-Filter besser abschneiden. Mit einem Zuwachs um 19 % durch den NB-Filter auf eTour (50,6 % zu vormals 42,4 %) und um 6 % durch den NB-Filter auf eAnci (28,2 % gegenüber 26,6 %) steigt die Leistung hier sogar nochmals deutlich mehr als zuvor. Auf LibEST ist nun auch im F_1 -Maß ein leichter Zuwachs der Leistung durch den NF-Filter erkennbar.

Vergleicht man die erzielten durchschnittlichen Werte des NF-Filters, mit denen desselben Filters auf Basis der automatischen Klassifikation (s. Tabelle 10.1), so lässt sich auf der Variante ohne Erweiterungen ein Plus von 1,4 Prozentpunkten, auf derjenigen mit \bullet ein Plus von 1,6 Prozentpunkten und auf **FTLR** mit $\Rightarrow\bullet$ von 1,7 Prozentpunkten im F_1 -Maß feststellen. Es zeigt sich auch hier im Durchschnitt über die Projekte, dass der NF-Filter diejenige Filtervariante darstellt, die konsistent die besten Ergebnisse liefert. Unter Einsatz des Goldstandards ist nun auch mit der originalen Schwellenwertkombination

Tabelle 10.4.: Vergleich der Ergebnisse der unterschiedlichen Schwellenwertkombinationen auf den Anforderungsfiltvarianten auf Basis des Goldstandards mit den ungefilterten **FTLR**-Ergebnissen. \rightleftarrows markiert Varianten, die Aufrufabhängigkeiten und \bullet , die Methodenkommentare verwenden. NF bezeichnet Varianten, die Anforderungselemente ohne funktionale Aspekte und NB Varianten, die Nutzerbezogenes filtern. Die besten Ergebnisse im Durchschnitt über alle Projekte sind fett dargestellt

FTLR-Variante:		\bullet			\rightleftarrows					
Projekt	Filter	ORG	MED	OPT	ORG	MED	OPT	ORG	MED	OPT
eTour	Ohne	.391	.355	.440	.385	.358	.425	.388	.365	.427
	NF	.476	.472	.546	.468	.453	.517	.470	.458	.515
	NB	.506	.490	.552	.497	.483	.529	.497	.486	.527
	NF + NB	.506	.490	.552	.497	.483	.529	.497	.486	.527
iTrust	Ohne	.190	.166	.215	.206	.172	.253	.209	.174	.239
	NF	.208	.178	.234	.218	.182	.259	.221	.184	.249
	NB	.127	.150	.165	.143	.164	.182	.143	.155	.169
	NF + NB	.142	.163	.172	.152	.176	.184	.154	.164	.179
SMOS	Ohne	.369	.364	.396	.382	.387	.405	.383	.386	.405
	NF	.366	.366	.399	.384	.388	.413	.387	.390	.414
	NB	.366	.373	.409	.383	.393	.418	.380	.392	.418
	NF + NB	.366	.373	.409	.383	.392	.417	.380	.391	.418
eAnci	Ohne	.241	.247	.261	.240	.242	.256	.236	.242	.255
	NF	.265	.269	.290	.265	.268	.279	.260	.268	.277
	NB	.266	.271	.297	.282	.285	.296	.272	.283	.295
	NF + NB	.266	.271	.297	.282	.285	.297	.272	.284	.296
LibEST	Ohne	.442	.524	.552	.509	.538	.552	.509	.538	.552
	NF	.429	.522	.554	.510	.543	.553	.510	.543	.553
	NB	.442	.522	.548	.509	.540	.552	.509	.540	.552
	NF + NB	.429	.522	.551	.510	.543	.552	.510	.543	.552
\emptyset	Ohne	.327	.331	.372	.344	.339	.378	.345	.341	.375
	NF	.349	.361	.405	.369	.367	.404	.370	.369	.402
	NB	.342	.361	.394	.363	.373	.395	.360	.371	.392
	NF + NB	.342	.364	.396	.365	.376	.396	.363	.374	.394

eine signifikante Verbesserung auf den Varianten \bullet und \rightleftarrows im Wilcoxon-Vorzeichen-Rang-Test bezüglich des Signifikanzniveaus von 0,05 zu beobachten.

Betrachtet man auch hier die Ergebnisse bezüglich des F_1 -Maßes für die Schwellenwertbetrachtungen MED und OPT (Tabelle 10.4), so sind die Verbesserungen mit dem NF-Filter auf OPT bei allen drei Varianten signifikant gegenüber **FTLR** ohne Filter. Auch mit den angepassten festen Schwellenwerten (MED) sind die erzielten Verbesserungen auf den beiden Varianten mit \bullet signifikant im Wilcoxon-Vorzeichen-Rang-Test bezüglich des Signifikanzniveaus 0,05. Es lässt sich also feststellen, dass mit beiden festgelegten Schwellen-

lenwertkombinationen eine signifikante Verbesserung gegenüber Varianten ohne Filter erzielt werden kann. Damit lässt sich These T_2 auf einer idealen Klassifikationseingabe noch stärker belegen.

Auf MED und OPT zeigen sich außerdem noch weitere interessante Ergebnisse. So erzielt auf eAnci die **FTLR**-Variante mit \Rightarrow und der Kombination der beiden Filter auf MED und OPT das beste Ergebnis. Auf den meisten anderen Projekten und Schwellenwertbetrachtungen hingegen erzielt die Kombination der Filter maximal dasselbe Ergebnis wie der bessere der beiden Filter. Auf SMOS erzielt auf der ursprünglichen Schwellenwertkombination der NF-Filter bei allen drei Varianten die besten Ergebnisse. Die besten Ergebnisse auf MED und OPT werden auf SMOS aber durch die Anwendung des NB-Filters erreicht. Hierbei sind also die angepassten Schwellenwerte in der Lage, das Potenzial der Filterung von Nutzerbezogenem besser auszunutzen als die ursprünglichen Schwellenwerte. Im Durchschnitt über alle Projekte ist auf ORG und OPT die alleinige Anwendung des NF-Filters optimal. Auf MED hingegen erzielt der Kombinationsfilter das beste durchschnittliche F_1 -Maß. Auf dieser Schwellenwertbetrachtung profitiert **FTLR** am meisten von den Musterlösungsinformationen zur Klasse Nutzerbezogenes (s. Ergebnisse auf eTour und eAnci).

Betrachtet man die erzielten Verbesserungen des NF-Filters durch den Einsatz der Musterlösung gegenüber der automatischen Klassifikation, so sind diese im Schnitt 1,7 Prozentpunkte im F_1 -Maß. Das beste Gesamtergebnis von 40,5 % wird hierbei von der **FTLR**-Variante ohne Erweiterungen und optimierten Schwellenwerten erzielt. Dieses Ergebnis liegt dabei sogar höher, als das beste Gesamtergebnis **FTLRs**, welches in der \Rightarrow -Variante ein durchschnittliches F_1 -Maß von 40,4 % auf diesen Projekten erzielt.

10.2. Vergleich mit Filterung von Vorlagenelementen

Der Einsatz des Filterns von Anwendungsfallvorlagenelementen hat sich bereits in der Evaluation in Abschnitt 6.3 als eine große Leistungssteigerung auf Anwendungsfallbeschreibungen herausgestellt. In vielen Fällen liegen Anforderungen allerdings nicht in Form von Anwendungsfallbeschreibungen vor. Und auch wenn Anwendungsfallbeschreibungen vorliegen, so muss das Wissen über die auszuschließenden Vorlagenelemente zunächst explizit gegeben sein², um diese automatisch auszuschließen. Somit wäre eine automatische Klassifikation, die ähnliche oder sogar bessere Ergebnisse für ein Filtern der Eingabeelemente liefert, eine sinnvolle Ergänzung des **FTLR**-Verfahrens. Hierbei könnten auch auf anderen Anforderungsarten und ohne manuelle Bestimmung der Namen der Elemente irrelevante Teile der Eingabe herausgefiltert werden.

Die Ergebnisse in Abschnitt 10.1 haben gezeigt, dass vor allem der NF-Filter bessere Ergebnisse, als Varianten, die keine Filterung der Anforderungselemente durchführen, erzielt. In diesem Abschnitt wird nun überprüft, ob diese vielversprechenden Ergebnisse vergleichbar

² Im Fall von häufig verwendeten Vorlagen, wie in den Projekten eTour, SMOS, eAnci und Albergate können diese aber wiederverwendet werden.

Tabelle 10.5.: Vergleich der F_1 -Maß-Ergebnisse der Anforderungsfiltervarianten mit **FTLR** mit Anwendungsfallvorlagenfilter auf den unterschiedlichen Schwellenwertkombinationen. \rightleftharpoons markiert Varianten, die Aufrufabhängigkeiten, \bullet , die Methodenkommentare verwenden und \blacktriangledown , die das Filtern von Vorlagenelemente nutzen. NF bezeichnet das Filtern von Anforderungselementen ohne funktionale Aspekte auf Basis der automatischen Klassifikation und NF_{Gold} auf Basis des Goldstandards. Die besten Ergebnisse im Durchschnitt über alle Projekte sind fett dargestellt

FTLR-Variante:		\rightleftharpoons			\bullet			$\bullet\rightleftharpoons$		
Projekt	Filter	ORG	MED	OPT	ORG	MED	OPT	ORG	MED	OPT
eTour	\blacktriangledown	.479	.478	.552	.469	.461	.517	.474	.468	.517
	NF	.411	.371	.468	.406	.370	.442	.408	.377	.442
	NF_{Gold}	.479	.476	.548	.468	.453	.517	.470	.458	.515
iTrust	\blacktriangledown	.183	.171	.210	.206	.172	.253	.209	.174	.239
	NF	.203	.184	.227	.219	.182	.261	.222	.185	.250
	NF_{Gold}	.202	.182	.224	.218	.182	.259	.221	.184	.249
SMOS	\blacktriangledown	.359	.367	.415	.381	.390	.419	.381	.390	.419
	NF	.364	.366	.402	.382	.390	.410	.382	.389	.411
	NF_{Gold}	.366	.366	.405	.384	.388	.413	.387	.390	.414
eAnci	\blacktriangledown	.257	.265	.292	.263	.268	.276	.258	.268	.277
	NF	.246	.258	.282	.249	.259	.271	.242	.259	.270
	NF_{Gold}	.260	.269	.294	.265	.268	.279	.260	.268	.277
LibEST	\blacktriangledown	.442	.524	.552	.509	.538	.552	.509	.538	.552
	NF	.436	.535	.553	.509	.538	.555	.509	.538	.555
	NF_{Gold}	.429	.522	.554	.510	.543	.553	.510	.543	.553
\emptyset	\blacktriangledown	.344	.361	.404	.366	.366	.403	.366	.368	.401
	NF	.332	.343	.386	.353	.348	.388	.353	.350	.386
	NF_{Gold}	.347	.363	.405	.369	.367	.404	.370	.369	.402

sind oder **FTLR** mit dem NF-Filter sogar besser abschneidet als **FTLR** mit einem Filtern von Anwendungsfallvorlagenelementen (\blacktriangledown). Dieser Abschnitt adressiert somit Forschungsfrage FF_{12} . Hierzu zeigt Tabelle 10.5 die erzielten Werte im F_1 -Maß auf den drei besten Varianten mit \blacktriangledown (\rightleftharpoons , \bullet und $\bullet\rightleftharpoons$) und den verschiedenen Schwellenwertkombinationen. Verglichen werden hier die Filtervarianten \blacktriangledown , NF-Filter mit automatischer Klassifikation (NF) und NF-Filter mit Eingabe aus dem Goldstandard (NF_{Gold}).

Zunächst lässt sich festhalten, dass im Durchschnitt die NF-Filter-Variante mit automatischer Klassifikation auf allen Schwellenwertkombinationen und Varianten schlechter abschneidet als die \blacktriangledown -Variante. Dies liegt vor allem daran, dass durch Fehlklassifikationen die Präzision geringer ausfällt, als es möglich wäre (s. Tabellen E.7 und E.8 in Anhang E.2). Das Filtern von Anwendungsfallvorlagenelementen profitiert hier von dem manuell gegebenen Wissen über irrelevante Elemente. Der Einfluss relevanter Elemente innerhalb

der zum Filtern genutzten Vorlagenelementtypen³, sowie irrelevanter Elemente in den als relevant angesehenen Vorlagentypen, ist also geringer als jener der relevanten Elemente, die aufgrund von Fehlklassifikationen ausgeschlossen werden. Auf den Projekten ohne Anwendungsfallvorlagen (iTrust und LibEST) erzielt **FTLR** allerdings mit dem NF-Filter auf Basis der automatischen Klassifikation bessere Ergebnisse. Insbesondere auf iTrust sind die Ergebnisse auf allen Schwellenwertkombinationen deutlich besser als **▼**. Da die **▼**-Variante hier keine Elemente filtern kann, weil keine Anwendungsfallvorlagenelemente für iTrust existieren, zeigt sich hier das volle Potenzial der Klassifikation auf Anforderungen, die keinen Vorlagen folgen. Mit einer durchschnittlichen Verbesserung von 1,2 Prozentpunkten bzw. um 6 % ist dieses Potenzial vielversprechend.

Betrachtet man nun die Ergebnisse, welche mit einer idealen Klassifikationsleistung möglich wären (NF_{Gold}), so zeigt sich, dass **FTLR** mit NF-Filter auf Basis des Goldstandards sogar bessere Ergebnisse auf allen Schwellenwertkombinationen erzielt. Im Durchschnitt sind diese Unterschiede allerdings gering. Auf SMOS wird hier neben iTrust der größte Leistungszuwachs erzielt. Mit den originalen festen Schwellenwerten (ORG) auf **⇒** beträgt dieser 0,7 Prozentpunkte (36,6 % gegenüber 35,9 %).

Zusammenfassend lässt sich also feststellen, dass ein Anwenden von Anforderungselementfiltern vergleichbare und potenziell sogar bessere Ergebnisse erzielen kann, als die an Anwendungsfallbeschreibungen auf Basis von Vorlagen gebundene Filterung. Da ein solcher Filter auf verschiedene Typen von Anforderungen angewandt werden kann und keine manuelle Identifikation von irrelevanten Vorlagenelementtypen für eine gegebene Vorlage benötigt, ist ein solches Vorgehen breiter anwendbar. Wenn in der Zukunft durch beispielsweise mehr Trainingsdaten oder bessere Klassifikationsverfahren die Leistung eines solchen Filters noch gesteigert werden könnte, wäre dieser einem starren Anwendungsfallvorlagenelementfilter vorzuziehen. Aufgrund der positiven Ergebnisse auf iTrust und LibEST sollte außerdem auf Anforderungen, die nicht einer Vorlage folgen, schon jetzt ein auf Klassifikation basierender Anforderungselementfilter eingesetzt werden.

10.3. Kombination der Filtervarianten

Abschnitt 10.2 hat gezeigt, dass ein automatischer Anforderungselementfilter (NF) den Anwendungsfallvorlagenelementfilter (**▼**) ersetzen kann und aus Gründen des flexibleren Einsatzes auch sollte. Da aber in manchen Fällen Anforderungen trotzdem in Form von Anwendungsfallbeschreibungen auf Basis von Vorlagen vorliegen werden, kann diese Information auch weiterhin genutzt werden. Die beiden Filtervarianten lassen sich dazu kombinieren. Diese Kombination sorgt dafür, dass auch in Anwendungsfallbeschreibungen in Vorlagenform zunächst die Vorlagenelementtypen entfernt werden, die der Anwendungsfallvorlagenelementfilter herausfiltert. Von den verbleibenden Anforderungselementen werden dann durch den NF-Filter diejenigen entfernt, die als keine funktionalen Aspekte enthaltend klassifiziert wurden. Ob eine solche Kombination einen Mehrwert

³ Akteur, Vor- und Nachbedingungen sowie Qualitätsanforderungen

Tabelle 10.6.: Vergleich der Ergebnisse im F_1 -Maß **FTRLRs** mit dem NF-Filter in Kombination mit dem Anwendungsfallvorlagenfilter auf den unterschiedlichen Schwellenwertkombinationen. \rightleftarrows markiert Varianten, die Aufrufabhängigkeiten, \bullet , die Methodenkommentare verwenden und \blacktriangledown , die das Filtern von Vorlagenelemente nutzen. NF bezeichnet das Filtern von Anforderungselementen ohne funktionale Aspekte auf Basis der automatischen Klassifikation und NF_{Gold} auf Basis des Goldstandards. Die besten Ergebnisse im Durchschnitt über alle Projekte sind fett dargestellt

FTLR-Variante:		\rightleftarrows			\bullet			$\bullet\rightleftarrows$		
Projekt	Filter	ORG	MED	OPT	ORG	MED	OPT	ORG	MED	OPT
eTour	\blacktriangledown	.479	.478	.552	.469	.461	.517	.474	.468	.517
	\blacktriangledown + NF	.479	.478	.552	.469	.461	.517	.474	.468	.517
	\blacktriangledown + NF_{Gold}	.479	.478	.552	.469	.461	.517	.474	.468	.517
iTrust	\blacktriangledown	.183	.171	.210	.206	.172	.253	.209	.174	.239
	\blacktriangledown + NF	.203	.184	.227	.219	.182	.261	.222	.185	.250
	\blacktriangledown + NF_{Gold}	.202	.182	.224	.218	.182	.259	.221	.184	.249
SMOS	\blacktriangledown	.359	.367	.415	.381	.390	.419	.381	.390	.419
	\blacktriangledown + NF	.359	.367	.415	.380	.390	.418	.380	.390	.418
	\blacktriangledown + NF_{Gold}	.358	.366	.414	.378	.389	.417	.378	.389	.417
eAnci	\blacktriangledown	.257	.265	.292	.263	.268	.276	.258	.268	.277
	\blacktriangledown + NF	.246	.263	.289	.258	.266	.275	.250	.266	.275
	\blacktriangledown + NF_{Gold}	.251	.261	.286	.258	.267	.273	.253	.267	.274
LibEST	\blacktriangledown	.442	.524	.552	.509	.538	.552	.509	.538	.552
	\blacktriangledown + NF	.436	.535	.553	.509	.538	.555	.509	.538	.555
	\blacktriangledown + NF_{Gold}	.429	.522	.554	.510	.543	.553	.510	.543	.553
\emptyset	\blacktriangledown	.344	.361	.404	.366	.366	.403	.366	.368	.401
	\blacktriangledown + NF	.345	.365	.407	.367	.367	.405	.367	.369	.403
	NF	.332	.343	.386	.353	.348	.388	.353	.350	.386
	\blacktriangledown + NF_{Gold}	.344	.362	.406	.367	.368	.404	.367	.370	.402
	NF_{Gold}	.347	.363	.405	.369	.367	.404	.370	.369	.402

für die Leistung des Systems bietet, soll nun in diesem Abschnitt analysiert werden (Forschungsfrage FF_{13}). Auch bei dieser Kombination kann betrachtet werden, welchen Einfluss die Klassifikationsgüte auf das Ergebnis hat. Daher wird auch hier in NF und NF_{Gold} unterschieden.

Tabelle 10.6 zeigt die Ergebnisse im F_1 -Maß auf den besten \blacktriangledown -Varianten. Hierbei zeigt sich, dass auf eTour keine Veränderung durch die Kombination der beiden Filter gegenüber einer alleinigen Anwendung des Filters für Vorlagenelemente erzielt wird. Dies deutet darauf hin, dass auf diesem Projekt der Anwendungsfallvorlagenelementfilter (\blacktriangledown) bereits nahezu alle irrelevanten Elemente herausfiltert. Dies bestätigt ebenso der Goldstandard für die Klassifikation, welcher nur drei Anforderungselemente in eTour als nicht funktionale Aspekte enthaltend vorsieht, die entweder der Name, die Beschreibung oder ein Teil der Ablaufbeschreibung sind. Auf iTrust und LibEST hingegen zeigen sich dieselben

Ergebnisse wie zuvor. Da iTrust und LibEST keine Vorlagenelemente von Anwendungsfallbeschreibungen aufweisen, filtert der Anwendungsfallvorlagenelementfilter hier keine Anforderungselemente.

Auf SMOS und eAnci erzielt die Kombination von \blacktriangledown mit dem NF-Filter teils leicht schlechtere Ergebnisse als die alleinige Anwendung des Vorlagenelementfilters. Es kann aber insbesondere im Vergleich mit den Ergebnissen der alleinigen Anwendung des NF-Filters auf Basis der automatischen Klassifikation ein deutlicher Leistungszuwachs beobachtet werden. Auf SMOS erzielt beispielsweise die Kombination $\blacktriangledown + \text{NF}$ auf \rightleftharpoons mit OPT-Schwellenwerten ein 1,4 Prozentpunkte höheres F_1 -Maß als **FTLR**- \rightleftharpoons mit NF-Filter. Auch auf eAnci profitiert der NF-Filter auf Basis der automatischen Klassifikation auf allen Varianten und Schwellenwertkombinationen von der Kombination.

Die Betrachtung der durchschnittlichen Werte im F_1 -Maß verdeutlicht ebenso, dass der NF-Filter mit automatischer Klassifikation am meisten von der Kombination mit dem Vorlagenelementfilter profitiert. Mit einer durchschnittlichen Verbesserung von 1,8 Prozentpunkten über alle Schwellenwertbetrachtungen und Varianten hinweg und besseren Ergebnissen in allen Varianten als eine alleinige Anwendung des Vorlagenelementfilters ist dieses Ergebnis sehr positiv. Es zeigt, dass in dieser Form auch die automatische Klassifikation gewinnbringend eingesetzt werden kann. Durch die besseren Ergebnisse auf iTrust und LibEST kann die Kombination von $\blacktriangledown + \text{NF}$ im Durchschnitt über die Projekte sogar auf allen Schwellenwertbetrachtungen ein höheres F_1 -Maß erzielen, als die alleinige Anwendung des Vorlagenelementfilters. Auch im Vergleich mit den Ergebnissen durch den Goldstandard, erzielt \blacktriangledown in Kombinationen mit dem NF-Filter auf Basis der automatischen Klassifikation bessere Ergebnisse auf den optimierten Schwellenwertkombinationen (OPT) und bei **FTLR**- \rightleftharpoons auch auf der MED-Schwellenwertkombination. Es muss aber festgehalten werden, dass die Unterschiede hier allgemein gering sind. Dies lässt sich damit erklären, dass die \blacktriangledown -Variante auf drei der fünf Projekte bereits einen Großteil der irrelevanten Anforderungselemente herausfiltert. Zusätzlich kann der irreführende Beitrag eines Elements, welches bereits wenig Ähnlichkeiten zu den Quelltextelementen hat, auch ohne einen solchen Filter bereits durch die Schwellenwerte ignoriert werden. Die maximal erreichbare Verbesserung durch ein solches filterbasiertes Vorgehen ist also begrenzt und nicht jedes korrekt herausgefilterte Element sorgt auch automatisch für eine Verbesserung des TLR-Ergebnisses. Das insgesamt beste Ergebnis **FTLRs** auf diesen Projekten wird dennoch durch die Kombination von $\rightleftharpoons\blacktriangledown$ mit dem NF-Filter auf Basis der automatischen Klassifikation und optimierten Schwellenwerten mit einem F_1 -Maß von 40,7 % erzielt.

11. Zusammenfassung

Auf Basis der positiven Ergebnisse des Filterns von Vorlagenelementen in Anwendungsfallbeschreibungen und der noch verbesserungswürdigen Präzision des **FTLR**-Verfahrens wurde in diesem Teil der Arbeit das Potenzial einer Anforderungsklassifikation als Vorverarbeitungsschritt für die automatische Wiederherstellung von Nachverfolgbarkeitsverbindungen untersucht. Hierzu wurde zunächst analysiert, welche Kategorien üblicher Anforderungsklassifikationen für ein solches Vorfiltern infrage kommen. Diese Analyse ergab, dass zum einen die allgemeine Erkennung, ob ein Anforderungselement einen funktionalen Aspekt enthält, großes Potenzial bietet, das Rauschen in der Eingabe zu vermindern. Zum anderen wurde ein Bestimmen der vorliegenden Anliegen (engl.: *concerns*) in funktionalen Anforderungen, wie z. B. Funktion, Verhalten oder Daten, als potenziell hilfreiche Information identifiziert. Zusätzlich wurde auf Basis der vorliegenden Anwendungsfallbeschreibungen die Kategorie Nutzerbezogenes eingeführt. Diese umfasst Anforderungselemente, welche die Interaktion des Nutzers bzw. der Nutzerin oder rein dem/der Nutzer:in zuzuordnende Funktionalitäten des Systems beschreiben.

Anschließend wurde eine Integration dieser Informationen in **FTLR** diskutiert. Als Ergebnis dieser Diskussion wurde eine Umsetzung eines Vorverarbeitungsschrittes, der auf Basis einer Anforderungsklassifikation, Anforderungselemente aus der Verarbeitung entfernt, beschrieben. Dieser Klassifikationsfilter nutzt die Information, ob ein Anforderungselement funktionale Aspekte und/oder Nutzerbezogenes enthält.

Daraufhin wurde zur automatischen Klassifikation der Anforderungselemente das Verfahren **NoRBERT** vorgestellt. Dieses verwendet ein feinangepasstes *BERT*-Sprachmodell zur Klassifikation. Auf Basis mehrerer Standardklassifikationsaufgaben aus der Anforderungsklassifikation wurde die Leistung des Verfahrens mit der Leistung bestehender Verfahren zur Anforderungsklassifikation verglichen. Dieser Vergleich zeigt die Eignung **NoRBERTs** insbesondere auf im Training ungesesehenen Projekten und auf der Klassifikation funktionaler Anliegen. Diese Eigenschaften sind besonders relevant für den Einsatz in der TLR.

Um auch die Leistung **NoRBERTs** auf Anforderungselementen aus Projekten der TLR zu bemessen, wurde ein Goldstandard für fünf Vergleichsdatensätze der TLR erstellt. Auf diesem Goldstandard konnte **NoRBERT** für das Erkennen von Anforderungselementen mit funktionalen Aspekten ein F_1 -Maß von 89,8 % und für die Identifikation von Nutzerbezogenem ein F_1 -Maß von 81,8 % erreichen. Da jeweils keinerlei Elemente des gerade evaluierten Projekts zum Training genutzt wurden, sind diese Ergebnisse sehr vielversprechend und zeigen nochmals **NoRBERTs** Eignung, auf ungesehene Projekte angewandt zu werden.

Zusätzlich wurde evaluiert, wie **NoRBERT** funktionale Anliegen auf den TLR-Datensätzen erkennt und auch hier konnte das Verfahren mit Ergebnissen im F_1 -Maß von 68 % für Funktion und 75,6 % für Verhalten vielversprechende Ergebnisse aufweisen. Auf der seltener auftretenden Kategorie Daten waren die Ergebnisse mit einem F_1 -Maß von 63,3 % ebenso positiv. Da diese Kategorien aber zunächst nicht von **FTLR** genutzt werden, bieten diese Ergebnisse vor allem Erkenntnisse für Anwendungen in anderen Bereichen oder zukünftigen Arbeiten.

Abschließend wurde der Einfluss der Klassifikationsfilter auf die TLR durch **FTLR** bemessen. Auf den Datensätzen eTour, iTrust, SMOS, eAnci und LibEST konnte gezeigt werden, dass der Einsatz des Filterns von Anforderungselementen, die keine funktionalen Aspekte enthalten (NF), die Präzision des Verfahrens erhöht. Auch die Gesamtleistung, gemessen im F_1 -Maß, konnte durch diesen Filter erhöht werden und war sogar zum Teil signifikant höher als die Leistung von **FTLR**-Varianten ohne Filterung. Somit konnte These T_2 in diesem Teil der Arbeit bestätigt werden. Das Filtern von Nutzerbezogenem ergab leider keine signifikante Verbesserung, weder in alleiniger Anwendung noch in Kombination mit dem NF-Filter.

Hinsichtlich eines Vergleichs mit dem bereits vorhandenen Vorlagenelementfilter konnte der Klassifikationsfilter vergleichbare und teils sogar bessere Ergebnisse erzielen. Dies erlaubt die Folgerung, dass der auf Anwendungsfallbeschreibungsvorlagen beschränkte Filter durch den flexibleren Klassifikationsfilter ersetzt werden kann. Eine Kombination der beiden Filtervarianten erzielte sogar das insgesamt beste durchschnittliche Ergebnis für die TLR von 40,7 % (F_1 -Maß) für diese Datensätze. Insgesamt zeigen die Experimente aber auch, dass selbst mit dieser Erweiterung **FTLR** nicht die Abbildungsgüte erreicht, die für eine vollständige Automatisierung benötigt würde.

Teil IV.

Einsatz bimodaler Sprachmodelle in FTLR

12. Bimodale, kontextsensitive Sprachmodelle als Repräsentationsbasis

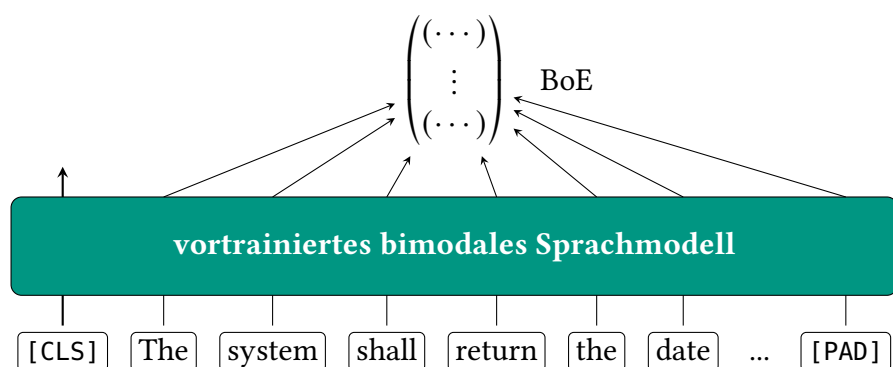
You shall know a word by the company it keeps.

(Firth [Fir57])

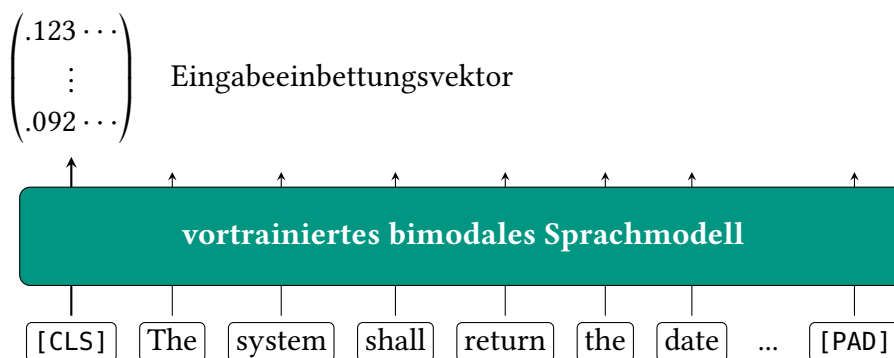
FTLR nutzt als Sprachmodell zur Repräsentation der Artefaktelemente vortrainierte *fastText*-Worteinbettungsmodelle. Dabei handelt es sich um statische Sprachmodelle. Dies bedeutet, ein Wort wird immer genau auf denselben Vektor abgebildet, unabhängig von der Bedeutung des Wortes im aktuellen Kontext. Zusätzlich wurden die verwendeten Modelle auf großen natürlichsprachlichen Korpora vortrainiert (Wikipedia und aus dem Internet gesammelte Dokumente, s. Abschnitt 2.9.2), welche nicht notwendigerweise die Zusammenhänge in der Softwaretechnik und -entwicklung bzw. der jeweiligen Anwendungsdomäne der analysierten Software ausreichend umfassen. Dies kann dazu führen, dass das Modell Zusammenhänge weniger stark gewichtet als dies aufgrund der Semantik des jeweiligen Kontextes notwendig wäre. So könnte beispielsweise eine Ähnlichkeit zwischen `array` und `datatype` geringer ausfallen als dies im Programmierkontext der Fall sein sollte.

In bestehenden Arbeiten zur Nachverfolgbarkeit von Belangen zu VCS-Einbuchungen, der Fehlerlokalisierung und vor allem der Quelltextsuche werden daher zuletzt vermehrt kontextsensitive, bimodale, auf dualen Korpora mit natürlichsprachlichen und programmiersprachlichen Dokumentpaaren vortrainierte Sprachmodelle verwendet (s. Abschnitte 3.4, 3.6 und 3.7). Kontextsensitive Sprachmodelle bieten dabei den Vorteil, dass sie ein Wort in seinem Kontext repräsentieren. Erste kontextsensitive Sprachmodelle, wie *ELMo* oder *BERT*, wurden hierfür allerdings auf konsekutiven Sequenzen von Wörtern in Form von natürlichsprachlichem Text trainiert. Da Folgen von Begriffen im Quelltext sich deutlich von natürlichsprachlichem Text unterscheiden können (vgl. Abschnitt 4.4), zeigten diese Modelle nicht immer überzeugende Ergebnisse auf quelltextbezogenen Aufgaben, wie Quelltextsuche oder Fehlerlokalisierung [Fen+20; Guo+22a]. Aus diesem Grund wurden bimodale Sprachmodelle für natürliche und Programmiersprachen entwickelt. Diese Modelle wie *CodeBERT* [Fen+20], *SynCoBERT* [Wan+21a] oder *UniXcoder* [Guo+22a] erzielten auf diesen Aufgaben deutliche Verbesserungen in der Leistung.

In diesem Kapitel möchte ich daher untersuchen, ob sich diese Ergebnisse ebenso auf die Aufgabenstellung von **FTLR**, der unüberwachten Wiederherstellung von Nachverfolgbarkeitsverbindungen von Anforderungen und Quelltext, übertragen lassen oder ob **FTLR**



(a) Variante 1: Einbettungs-Multimenge (BoE) aus den Ausgaben für die einzelnen Eingabetoken



(b) Variante 2: Einzelner Eingabeeinbittungsvektor aus der Ausgabe des Sondertokens

Abbildung 12.1.: Darstellung der verschiedenen Repräsentationsmöglichkeiten auf Basis eines bimodalen Transformer-basierten Sprachmodells

mit *fastText* bessere Ergebnisse erzielt. Somit liefert dieses Kapitel Erkenntnisse bezüglich These **T₃**:

T₃: Der Einsatz eines bimodalen, kontextsensitiven Sprachmodells für die Repräsentation der Artefaktelemente verbessert die Leistung eines feingranularen Verfahrens zur Nachverfolgbarkeit.

Hierzu werden in Abschnitt 12.1 zunächst die Integrationsmöglichkeiten eines solchen Sprachmodells in **FTLR** beschrieben und das vielversprechendste Modell ausgewählt. Daraufhin präsentiert und analysiert Kapitel 13 die Ergebnisse eines Vergleichs der Leistung **FTLRs** mit dem gewählten Sprachmodell und **FTLRs** mit *fastText*. Dieses Kapitel baut auf den initialen Erkenntnissen der Bachelorarbeit „Vergleich verschiedener Sprachmodelle für den Einsatz in automatisierter Rückverfolgbarkeitsanalyse“ von Lachenicht [Lac22] auf und erweitert diese, um weitere Betrachtungen und Experimente.

12.1. Integration in FTLR

Die in der Literatur verwendeten bimodalen Sprachmodelle für natürliche und Programmiersprache basieren alle auf der Transformer-Architektur, wie sie auch das in Abschnitt 9.2 verwendete *BERT*-Modell benutzt. Für die überwachte Anwendung werden diese Modelle zumeist mit einem weiteren FFNN als Ausgabeschicht kombiniert, welches darauf trainiert wird zu klassifizieren, ob zwischen den Eingaben eine TL besteht [Lin+21]. Da für eine solche überwachte Anwendung bestehende TLs vorhanden sein müssen, kann diese Art der Anwendung nicht für den Einsatz ohne vorhandene TLs genutzt werden. Somit müssen diese Sprachmodelle für einen Einsatz in **FTLR** anders integriert werden.

Für die unüberwachte Anwendung solcher Modelle wird oftmals die Ausgabe der letzten Schicht des Modells als Repräsentationen der Eingabe aufgefasst [RG19]. Hierbei können zwei unterschiedliche Vorgehen gewählt werden. Zum einen können die einzelnen Ausgaben pro Token als eine Einbettungs-Multimenge (BoE) aufgefasst werden (s. Abbildung 12.1a). Zum anderen kann die aggregierte Ausgabe des Sondertokens (welches für jede Eingabe als erstes Token hinzugefügt wird) als Einbettung der gesamten Eingabe aufgefasst werden (s. Abbildung 12.1b). Letzteres Vorgehen nutzt bspw. auch **NoBERT** als Eingabe für seine Ausgabeschicht. Bei ersterem Vorgehen erhält man also eine Menge von N Vektoren mit N der Anzahl der Eingabetoken. Bei letzterem Vorgehen hingegen wird die gesamte Eingabe durch einen einzigen Vektor repräsentiert. Diese zwei Formen der Repräsentation lassen sich daraufhin ohne Weiteres in **FTLR** integrieren. Für das BoE kann wiederum die Wortüberferungsdistanz verwendet werden, wie es **FTLR** auch für die *fastText*-Einbettungs-Multimengen verwendet. Bei der Repräsentation durch einen einzelnen Eingabeeinbettungsvektor kann die Kosinusähnlichkeit wie bei der Referenzimplementierung *ElementCosSim* (ECosS) (s. Abschnitt 5.2) verwendet werden. In diesem Fall muss aufgrund des bereits vorhandenen einzelnen Vektors keine elementweise Durchschnittsbildung der Wortvektoren durchgeführt werden. Letztlich ändert sich in **FTLR** also nur der Schritt der Erzeugung der Repräsentation.

Der größte Unterschied besteht hier also darin, dass anstatt einer einzelnen Anfrage pro Wort der Eingabe an das *fastText*-Worteinbettungsmodell nun eine gemeinsame Anfrage der gesamten Eingabe an ein bimodales Sprachmodell gestellt wird. Aus der Ausgabe dessen letzter Schicht wird daraufhin die jeweilige Repräsentation extrahiert. Dieses Vorgehen führt dazu, dass die Vorverarbeitung im Repräsentationsschritt leicht angepasst werden sollte, da nun der für das jeweilige Modell vorgesehene Portionierer verwendet werden muss. Durch den Einsatz der *Hugging Face Transformers*-Rahmenarchitektur [Wol+20] lässt sich dies allerdings für Transformer-basierte Sprachmodelle uniform gestalten.

Eine entscheidendere Fragestellung ist allerdings, ob die bisherige Vorverarbeitung, welche unter anderem Stoppwörter entfernt und eine Lemmatisierung durchführt, für diese Art von Sprachmodellen geeignet ist. Da die Transformer-basierten Sprachmodelle auf Sequenzen von Token trainiert wurden, in denen sie durch ihren Aufmerksamkeitsmechanismus selbständig die relevanten Wörter identifizieren, könnte sich ein Entfernen von Stoppwörtern negativ auf die Repräsentation auswirken. Insbesondere, da die bimodalen Sprachmodelle auf vollständigen Sequenzen von natürlichsprachlicher Dokumentation

und Programmen vortrainiert wurden, könnte dieser Effekt verstärkt werden. Zusätzlich wurden die Modelle nicht auf lemmatisierten, an Binnenmajuskeln aufgetrennten Eingaben, sondern lediglich tokenisierten Eingaben vortrainiert. Daher ist ebenso fraglich, ob eine Lemmatisierung und Binnenmajuskelauftrennung als Vorverarbeitung notwendig oder sogar hinderlich ist. Um dies zu untersuchen, kann **FTLR** mit einem Transformer-basierten, bimodalen Sprachmodell sowohl mit der vollen Vorverarbeitung wie in Abschnitt 5.1 beschrieben als auch mit einer auf Querverweis-, Sonderzeichen- und Nummerentfernung sowie Umwandlung in Kleinbuchstaben reduzierten Vorverarbeitung ausgeführt werden.

12.2. Wahl geeigneter Sprachmodelle

In den verwandten Arbeiten zur TLR zwischen Belangen und VCS-Einbuchungen, den Arbeiten zur Fehlerlokalisierung und den Ansätzen zur Quelltextsuche wurde in den letzten Jahren eine Vielzahl von bimodalen, kontextsensitiven Sprachmodellen eingesetzt (s. Abschnitte 3.4, 3.6 und 3.7). Insbesondere für die Quelltextsuche umfasst diese Liste mindestens die Modelle *CodeBERT* [Fen+20], *GraphCodeBERT* [Guo+22b], *SYNCoBERT* [Wan+21a], *PLBART* [Ahm+21], *CodeT5* [Wan+21b] und *UniXcoder* [Guo+22a]. In den anderen beiden Forschungsfeldern wurde bisher vor allem *CodeBERT* eingesetzt. Ein Grund hierfür könnte die abstraktere Aufgabenstellung sein. Bei der Quelltextsuche wird zumeist die Aufgabe der Abbildung von Methodenkommentaren auf ihre Methoden bemessen, wohingegen bei der Fehlerlokalisierung und insbesondere bei der TLR natürlichsprachliche Eingaben aus einer anderen Quelle als dem Quelltext selbst stammen. In dieser Situation kann es hilfreich sein, dass *CodeBERT*, im Gegensatz zu den anderen Modellen, keine zusätzlichen strukturellen Informationen aus dem Quelltext, wie Datenflüsse oder den abstrakter Syntaxbaum, miteinbezieht. Da außerdem *CodeBERT* in der überwachten TLR zwischen Belangen und VCS-Einbuchungen und in der Fehlerlokalisierung vielversprechende Ergebnisse erzielt, stellt das Modell einen potenziellen Kandidaten, auch für die Anwendung in der TLR von Anforderungen zu Quelltext, dar.

Von den weiteren bimodalen Sprachmodellen ist *UniXcoder* das vielversprechendste Modell. Zum einen übertrifft *UniXcoder* die Leistung der anderen Modelle in der Quelltextsuche teils deutlich (Verbesserung von 7 % im MRR gegenüber *CodeBERT*). Zum anderen könnte die Idee, den AST nicht für die eigentliche Anwendung zu verwenden, sondern lediglich zum Lernen der Repräsentation einzusetzen, auch für den Einsatz in der TLR von Anforderungen zu Quelltext von Vorteil sein. Zusätzlich erzielt *UniXcoder* von den bimodalen, kontextsensitiven Sprachmodellen die besten Ergebnisse auf der Aufgabe der Suche semantisch ähnlicher Quelltextabschnitte (s. Abschnitt 3.7), wenn alle Modelle ohne Fein Anpassung auskommen müssen. Da auch die Anwendung für die unüberwachte TLR, also eine Anwendung ohne vorhandene TLs und ohne weiteres Training, eine nicht feinangepasste Version des ausgewählten Sprachmodells nutzt, ist *UniXcoder* hier vielversprechend.

In den im Folgenden dargestellten Experimenten beschränke ich mich auf die Ergebnisse und Analyse von **FTLR** mit *UniXcoder*, da dieses Sprachmodell auch auf den TLR-Vergleichsdatensätzen deutlich bessere Ergebnisse in einer nicht feinangepassten Version erzielte als *CodeBERT*.

13. Evaluation mit bimodalen Sprachmodellen

Two primary factors impede the advancement of deep learning traceability solutions. The first is the sparsity of training data, given that DL techniques require large volumes of training data. [...] The second impedance is the practicality of applying multi-layer neural networks in a large industrial project as training and utilizing deep neural networks is significantly slower than more traditional information retrieval or machine learning techniques.

(Lin et al. [Lin+21])

Zusätzlich zu den beiden vorangehenden Evaluationen in Kapitel 6 und 10 soll im Rahmen dieser Evaluation nun **FTLRs** Leistung unter Einsatz von bimodalen, kontextsensitiven Sprachmodellen bemessen und mit der Leistung unter Einsatz von *fastText* verglichen werden. Hierbei werden die bereits in Kapitel 6 und 10 verwendeten Vergleichsdatensätze eTour, iTrust, SMOS, eAnci und LibEST verwendet. Da allerdings für *UniXcoder* nur ein englischsprachliches vortrainiertes Modell zur Verfügung steht, müssen hierfür die zwei italienischen Datensätze SMOS und eAnci ins Englische übersetzt werden. Hierzu wurde auch der Quelltext der beiden Projekte mittels der Google Übersetzer-API¹ automatisch übersetzt und die bereits für die Klassifikation durch **NoRBERT** übersetzten Anforderungen genutzt (s. Abschnitt 9.4). Da es sich hierbei um eine automatische Übersetzung handelt, kann hierbei nicht sichergestellt werden, dass dieselben Begriffe/Konzepte immer gleich übersetzt werden. Außerdem erzielen auch automatische Übersetzungsverfahren keine perfekten Übersetzungen. Somit ist davon auszugehen, dass eine solche Übersetzung gewisse Fehler einführt, welche die Güte der Abbildung für die TLR beeinflussen kann. Allerdings konnten Lin et al. [LLC21] in ihrer Arbeit „*Information Retrieval versus Deep Learning Approaches for Generating Traceability Links in Bilingual Projects*“ zeigen, dass auf bilingualen Projekten eine automatisch übersetzte Version in Kombination mit englischen *fastText*-Worteinbettungen besser in der TLR abschneidet, als bilinguale Modelle oder eine Anwendung ohne Übersetzung. Diese Ergebnisse lassen darauf schließen, dass eine Übersetzung durchaus förderlich sein kann (vgl. Abschnitt 3.4). Um die Auswirkungen der Übersetzung zu adressieren, beschäftigt sich dieses Kapitel zunächst mit der Forschungsfrage:

FF₁₄: Wie wirkt sich die automatische Übersetzung der beiden italienischen Datensätze, SMOS und eAnci, ins Englische auf die Leistung **FTLRs** aus?

¹ <https://translate.google.com/>, zuletzt besucht am 26.04.2023.

Tabelle 13.1.: Vergleich der Auswirkung einer automatischen Übersetzung auf **FTLRs** Leistung mit *fastText*-Worteinbettungen und optimierten Schwellenwerten

FTLR-Variante:			🗨️		📄		🗨️📄	
	F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP
eAnci	.261	.148	.256	.157	.290	.151	.276	.154
eAnci _{übersetzt}	.258	.156	.254	.160	.285	.155	.273	.155
SMOS	.396	.405	.405	.418	.409	.438	.419	.451
SMOS _{übersetzt}	.367	.346	.368	.358	.385	.403	.394	.412

Hierzu werden in Abschnitt 13.1 die Ergebnisse **FTLRs** mit *fastText*-Einbettungen auf den übersetzten Datensätzen, mit denen auf den originalen Datensätzen verglichen.

Anschließend wird zunächst folgende Forschungsfrage adressiert:

FF₁₅: Mit welchen Einbindungsvarianten erzielt **FTLR** mit *UniXcoder* die besten Ergebnisse bezüglich F₁-Maß und MAP?

Zur Beantwortung dieser Frage werden in Abschnitt 13.2 die Leistung der Einbindungsvarianten über die Vergleichsdatsätze hinweg analysiert, um die besten Varianten zu ermitteln.

Anschließend werden in Abschnitt 13.3 die beiden folgenden Forschungsfragen adressiert, um Erkenntnisse bezüglich These **T₃** zu erhalten:

FF₁₆: Wie verhält sich die Leistung **FTLRs** mit *UniXcoder* im Vergleich zu **FTLR** mit *fastText*-Einbettungen?

FF₁₇: Ist **FTLR** mit *UniXcoder* ähnlich gut geeignet für die unüberwachte Anwendung auf ungesehene Projekte wie **FTLR** mit *fastText*?

FF₁₆ liefert Erkenntnisse zu den Leistungsunterschieden der beiden Varianten bezüglich der Abbildungsgüte. Da neben der Leistung im F₁-Maß bzw. MAP auch die Übertragbarkeit auf ungesehene Projekte für die Anwendung in der unüberwachten TLR eine entscheidende Rolle spielt, wird die zweite Frage betrachtet. Hierfür wird zum einen die Varianz der Schwellenwerte **FTLRs** mit *UniXcoder* über die Projekte hinweg betrachtet und zu anderen analysiert, ob sich eine feste Schwellenwertkombination identifizieren lässt, die eine ähnliche oder bessere Leistung als **FTLR** mit *fastText* und fester Schwellenwertkombination erzielt.

13.1. Auswirkung der Übersetzung

Zunächst wird die Auswirkung der Übersetzung auf **FTLR** mit *fastText*-Einbettungen gemessen und damit Forschungsfrage **FF₁₄** adressiert. Hierzu zeigt Tabelle 13.1 die Ergebnisse **FTLRs** mit *fastText*-Einbettungen auf den übersetzten Datensätzen und englischem Modell

im Vergleich zu den originalen, italienischen Datensätzen und **FTLR** mit einem italienischen *fastText*-Modell. Hierbei werden die Ergebnisse mit optimierten Schwellenwerten, also die obere Schranke der erzielbaren Leistung, verglichen.

Es zeigt sich, dass auf eAnci über alle Varianten hinweg auf dem übersetzten Datensatz ein leicht schlechteres F_1 -Maß, aber bessere MAPs erzielt werden. Insgesamt sind die Unterschiede hier aber mit einer durchschnittlichen Differenz von -0,3 bzw. +0,4 Prozentpunkten sehr klein. Deshalb kann für eAnci festgehalten werden, dass eine Übersetzung keine größere Auswirkung auf **FTLR** mit *fastText*-Worteinbettungen hat.

Auf SMOS hingegen zeigen die Ergebnisse einen stärkeren Rückgang der Leistung sowohl im F_1 -Maß als auch im MAP. Im Durchschnitt über die Varianten reduziert sich das F_1 -Maß um 2,9 und der MAP sogar um 4,8 Prozentpunkte. Allerdings schwanken die Differenzen über die **FTLR**-Varianten hinweg. So ist die Differenz im F_1 -Maß auf \bullet mit 3,7 Prozentpunkten am höchsten, wohingegen auf \blacktriangledown und $\bullet\blacktriangledown$ nur ein Rückgang um 2,5 Prozentpunkte zu verzeichnen ist. Selbiges gilt für den MAP, welcher maximal um 6 Prozentpunkte zurückgeht, aber auf den beiden Varianten mit \blacktriangledown nur um maximal 3,9 Prozentpunkte reduziert wird. Dies deutet darauf hin, dass das Filtern von Anwendungsfallbeschreibungselementen dazu führt, dass Teile der Anforderung, die potenziell fehlerhafte Übersetzungen enthalten können, herausgefiltert werden. Dies bestätigt die höhere Präzision, die diese Varianten auf SMOS erzielen (s. Tabelle E.9 in Anhang E.3). Die Differenz wird hier also durch eine geringere Ausbeute erzielt. Dies bedeutet, dass durch die Übersetzung Zusammenhänge zwischen Anforderungs- und Quelltextelementen nicht mehr identifizierbar sind, die im originalen Datensatz, im Italienischen, noch identifizierbar waren.

Bezüglich Forschungsfrage **FF**₁₄ kann also festgehalten werden, dass eine Übersetzung bei eAnci nahezu keinen Unterschied für die Leistung **FTLRs** macht; bei SMOS hingegen durchaus eine Leistungseinbuße, sowohl im F_1 -Maß als auch MAP festzustellen ist. Da *UniXcoder* nur auf die übersetzte Variante von SMOS angewandt werden kann, ist also aufgrund dieser Ergebnisse auch bei **FTLR** mit *UniXcoder* ein schlechteres Ergebnis zu erwarten.

13.2. Vergleich der Einbindungsvarianten

In diesem Abschnitt sollen die besten Einbindungsvarianten von *UniXcoder* in **FTLR** ermittelt werden (**FF**₁₅). Hierzu werden die beiden Metriken F_1 -Maß und MAP für die verschiedenen Varianten auf den fünf Vergleichsdatsätzen bestimmt. Um die tatsächlich beste Konfiguration zu bestimmen, werden, wie bereits in derselben Untersuchung für **FTLR** mit *fastText* (s. Abschnitt 6.3), die beiden Schwellenwerte in 0,001er-Schritten variiert. Hierdurch wird für jeden Datensatz individuell diejenige Schwellenwertkombination ermittelt, die das beste Ergebnis erzielt. Dies stellt also die obere Schranke der erzielbaren Leistung mit dem Sprachmodell und **FTLR** dar. Aus Platzgründen wurden die Ergebnisse der **FTLR**-Variante mit Aufrufabhängigkeiten weggelassen, da diese keine Verbesserung der

Tabelle 13.2.: Vergleich von **FTLRs** Leistung mit den UniXcoder-Einbindungsvarianten. Varianten, die Kosinusähnlichkeit nutzen, sind mit KOS und Varianten, die die Wortüberferungsdistanz nutzen, mit WMD gekennzeichnet. UXC bezeichnet **FTLR** mit *UniXcoder* und vollständiger Vorverarbeitung und UXC* mit reduzierter Vorverarbeitung. Die besten Werte je Spalte werden fett dargestellt und das beste durchschnittliche Ergebnis je **FTLR**-Variante wird unterstrichen

Variante		eTour		iTrust		SMOS		eAnci		LibEST		∅		
		F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP	
KOS	UXC	.414	.376	.216	.237	.333	.317	.289	.172	.556	.589	.362	.338	
	UXC*	.403	.396	.248	.250	.301	.331	.292	.178	.583	.652	<u>.365</u>	.361	
WMD	UXC	.416	.386	.229	.236	.319	.320	.230	.155	.585	.619	.356	.343	
	UXC*	.414	.449	.237	.244	.314	.329	.238	.155	.574	.633	.356	<u>.362</u>	
☐	KOS	UXC	.419	.344	.237	.243	.338	.321	.261	.165	.567	.561	<u>.364</u>	.327
	UXC*	.343	.347	.306	.291	.295	.322	.263	.159	.585	.664	.358	.357	
☐	WMD	UXC	.393	.337	.248	.268	.348	.343	.213	.146	.595	.588	.359	.336
	UXC*	.392	.432	.276	.299	.326	.339	.228	.145	.595	.647	.363	<u>.372</u>	
☐	KOS	UXC	.523	.498	.216	.237	.362	.376	.347	.174	.556	.589	.401	.375
	UXC*	.408	.416	.248	.250	.304	.342	.346	.187	.583	.652	.378	.369	
☐	WMD	UXC	.447	.481	.229	.236	.352	.398	.231	.152	.574	.504	.367	.354
	UXC*	.389	.465	.237	.244	.318	.353	.211	.146	.663	.561	.364	.354	
☐	KOS	UXC	.485	.455	.237	.243	.377	.388	.314	.174	.567	.561	<u>.396</u>	<u>.364</u>
	UXC*	.359	.359	.306	.291	.301	.336	.295	.161	.585	.664	.369	.362	
☐	WMD	UXC	.438	.435	.248	.268	.377	.429	.234	.149	.501	.455	.360	.347
	UXC*	.374	.465	.276	.299	.327	.367	.206	.131	.549	.545	.347	.361	

Ergebnisse mit *UniXcoder* erzielt. Die entsprechenden Ergebnisse können in Anhang E.4 eingesehen werden.

Tabelle 13.2 zeigt die Ergebnisse dieser Auswertung. Diese lassen sich nun hinsichtlich der zwei Dimensionen Repräsentation und Vorverarbeitung analysieren. Betrachtet man nur die durchschnittlichen Ergebnisse über die Projekte hinweg so erzielt die Einbindungsvariante mit Eingabeeinbettungsvektor und Kosinusähnlichkeit durchweg die besten Ergebnisse im F₁-Maß und für die beiden **FTLR**-Varianten mit Anwendungsfallbeschreibungselementfilter (☐) auch die besten MAPs. Lediglich auf den beiden Varianten ohne ☐ wird der beste MAP durch die Einbindungsvariante mit BoE und WMD erzielt. Diese bleibt aber im Durchschnitt über die Projekte in ihrer Leistung hinter der insgesamt besten Variante, **FTLR** mit Kosinusähnlichkeit und vollständiger Vorverarbeitung, zurück, welche einen MAP von 37,5 % erzielt. Diese Variante erzielt auch das insgesamt beste durchschnittliche F₁-Maß von 40,1 %. Betrachtet man allerdings die einzelnen Projekte für sich, so sieht das Ergebnisabbild etwas heterogener aus. Im F₁-Maß erzielt auf den

Projekten eTour, iTrust, SMOS und eAnci die Einbindungsvariante mit Kosinusähnlichkeit die besten Ergebnisse. Im MAP hingegen werden die besten Ergebnisse auf iTrust, SMOS und LibEST von der Einbindungsvariante mit BoE und WMD erzielt.

Differenziert man nun auch noch hinsichtlich der Vorverarbeitung so werden auf den Projekten eTour, SMOS und eAnci die besten Ergebnisse im F_1 -Maß und bis auf eAnci auch im MAP, jeweils mit vollständiger Vorverarbeitung (UXC) erzielt. Bei iTrust und LibEST hingegen werden die besten Ergebnisse mit der reduzierten Vorverarbeitung (UXC*) erreicht. Diese Verbesserungen sind zwar deutlich auf diesen beiden Projekten, führen aber auch zu deutlichen Verschlechterungen der Ergebnisse auf den anderen Projekten, was dazu führt, dass zumindest im F_1 -Maß die vollständige Vorverarbeitung der reduzierten vorzuziehen ist. Im MAP ist der Unterschied allerdings mit 0,2 Prozentpunkten sehr gering. Die Hypothese, dass die Vorverarbeitungsschritte Stoppwortentfernung, Lemmatisierung und Binnenmajuskelauftrennung für ein auf nicht derart vorverarbeiteten Daten vortrainiertes, bimodales Sprachmodell zu schlechteren Ergebnissen in der Repräsentation führen, kann also im Allgemeinen nicht bestätigt werden. Auf einzelnen Projekten kann eine reduzierte Vorverarbeitung allerdings durchaus zu Verbesserungen führen. Nutzt man diese reduzierte Vorverarbeitung auch mit **FTLR** und *fastText* so ergeben sich allerdings auf LibEST ähnliche Ergebnisse wie mit *UniXcoder*. Es bleibt also lediglich das iTrust-Projekt auf dem die reduzierte Vorverarbeitung für **FTLR** mit *UniXcoder* bessere Ergebnisse liefert.

Da die insgesamt besten Ergebnisse sowohl im F_1 -Maß als auch beim MAP von der Einbindungsvariante mit Eingabeeinbettungsvektor und Kosinusähnlichkeit sowie vollständiger Vorverarbeitung erzielt wurde, wird diese zusammen mit ihrer Variante mit reduzierter Vorverarbeitung für die weitere Betrachtung herangezogen. Letztere deckt damit zusätzlich die besten Ergebnisse im F_1 -Maß auf iTrust und im MAP auf LibEST ab. Durch diese Reduktion fallen somit nur die besten Ergebnisse im F_1 -Maß auf LibEST und im MAP auf iTrust und SMOS weg, welche durch zwei unterschiedliche WMD-Varianten erzielt werden.

13.3. Vergleich von FTLR mit fastText und UniXcoder

Mit den beiden besten Eingabevarianten lässt sich nun abschließend evaluieren, ob der Einsatz von *UniXcoder* gegenüber *fastText* einen Vorteil für die automatische, unüberwachte Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext mittels **FTLR** bietet. Zunächst werden zur Beantwortung von Forschungsfrage **FF**₁₆ die Ergebnisse bezüglich F_1 -Maß und MAP der besten *UniXcoder*-Einbindungsvarianten mit denen der besten *fastText*-Varianten verglichen. Auch hierbei werden wieder pro Projekt optimierte Schwellenwerte verwendet und damit die maximal mögliche Leistung verglichen. Es werden die **FTLR**-Varianten **T**, **⦿T**, **⇌T** und **⇌⦿T** betrachtet, da **FTLR** mit *fastText* auf diesen am besten abschneidet hat (s. Abschnitt 6.3) und auch **FTLR** mit *UniXcoder* auf diesen Varianten die besten Ergebnisse erzielt.

Tabelle 13.3.: Vergleich von **FTLRs** Leistung mit unterschiedlichen Sprachmodellen auf den besten Eingabevarianten. Als Sprachmodelle werden *fastText* (fT) und *UniXcoder* mit vollständiger Vorverarbeitung (UXC) und reduzierter Vorverarbeitung (UXC*) betrachtet. Die besten Werte je Spalte werden fett dargestellt

Variante		eTour		iTrust		SMOS		eAnci		LibEST		∅	
		F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP
T	UXC	.523	.498	.216	.237	.362	.376	.347	.174	.556	.589	.401	.375
	UXC*	.408	.416	.248	.250	.304	.342	.346	.187	.583	.652	.378	.369
	fT	.548	.528	.215	.227	.409	.438	.290	.151	.552	.559	.403	.381
T	UXC	.485	.455	.237	.243	.377	.388	.314	.174	.567	.561	.396	.364
	UXC*	.359	.359	.306	.291	.301	.336	.295	.161	.585	.664	.369	.362
	fT	.517	.483	.253	.284	.419	.451	.276	.154	.552	.581	.403	.391
T	UXC	.520	.500	.206	.228	.359	.370	.347	.173	.556	.589	.398	.372
	UXC*	.406	.416	.230	.227	.298	.340	.346	.187	.583	.652	.373	.364
	fT	.552	.532	.210	.213	.415	.434	.292	.150	.552	.559	.404	.378
T	UXC	.483	.460	.228	.227	.374	.383	.317	.173	.567	.561	.394	.361
	UXC*	.365	.356	.294	.271	.299	.331	.295	.162	.585	.664	.368	.357
	fT	.517	.483	.239	.271	.419	.451	.277	.153	.552	.581	.401	.388

Tabelle 13.3 zeigt die Ergebnisse dieses Experiments. Im Durchschnitt über die fünf Projekte, die als Vergleichsdatensätze dienen, erzielt **FTLR** in seiner originalen Variante mit *fastText*-Worteinbettungen sowohl im F₁-Maß als auch im MAP durchweg bessere Ergebnisse als mit einer der *UniXcoder*-Varianten. Allerdings sind die Unterschiede mit 0,3 Prozentpunkten im F₁-Maß (40,4 % gegenüber 40,1 %) und 1,6 Prozentpunkten im MAP (39,1 % zu 37,5 %) sehr gering. Auch ein Wilcoxon-Vorzeichen-Rang-Test ergibt keine signifikanten Unterschiede, sowohl im F₁-Maß als auch beim MAP, zwischen den Varianten. Selbst wenn man für SMOS und eAnci die Ergebnisse mit *fastText* auf den übersetzten Varianten miteinbezieht (s. Tabelle 13.1) verändert sich an dieser Erkenntnis nichts. Lediglich das insgesamt beste durchschnittliche F₁-Maß erzielt dann *UniXcoder* mit T. Der beste Wert mit *fastText* reduziert sich in diesem Fall auf 39,8 %². Den besten durchschnittlichen MAP erzielt aber weiterhin *fastText* mit 38,3 %³.

Betrachtet man allerdings die einzelnen Projekte, so zeigt sich, dass *UniXcoder* durchaus bessere Ergebnisse erzielen kann als *fastText*. Insbesondere im F₁-Maß auf iTrust und eAnci sowie im MAP auf LibEST ergeben sich teils große Verbesserungen. So erzielt **FTLR** mit *UniXcoder* und reduzierter Vorverarbeitung in der T-Variante auf iTrust mit einem F₁-Maß von 30,6 % eine Verbesserung von 5,3 Prozentpunkten gegenüber den 25,3 % von **FTLR** mit *fastText*. Mit derselben Variante wird ebenso eine Verbesserung von 3,3 Prozentpunkten im F₁-Maß auf LibEST erreicht (58,5 % zu vormals 55,2 %). Da allerdings diese Variante ebenso die schlechtesten Ergebnisse dieses Vergleichs im F₁-Maß auf eTour

² Auf der T-Variante

³ Ebenso auf der T-Variante

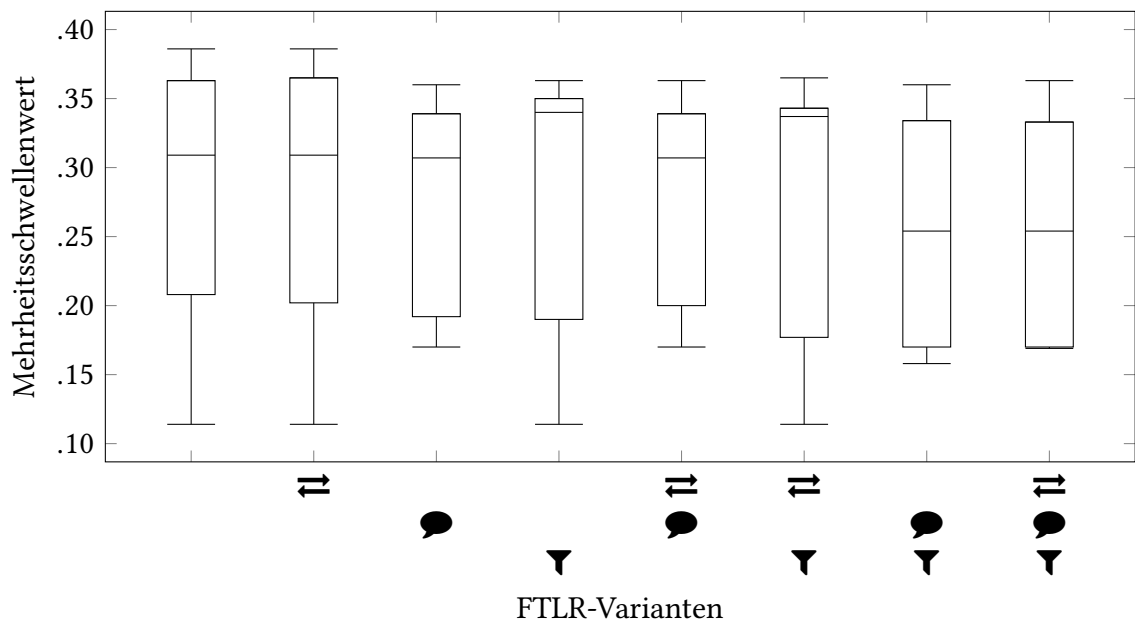


Abbildung 13.1.: Varianz der optimierten Mehrheitsschwellenwerte mit UniXcoder, vollständiger Vorverarbeitung und Kosinusähnlichkeit über die Datensätze hinweg. Jeder Graph deckt somit 5 Datenpunkte ab

und SMOS erzielt, ist diese Variante nicht einmal die beste *UniXcoder*-Variante und wird im Durchschnitt von **FTLR** mit *fastText* um 3,4 Prozentpunkte übertroffen (40,3 % gegenüber 36,9 %). Die Verbesserungen auf eAnci, welche durch beide *UniXcoder*-Varianten erzielt werden, sind hier interessanter. Diese treten in der *UniXcoder*-Variante mit vollständiger Vorverarbeitung auf, welche auch auf eTour und SMOS akzeptable Ergebnisse erzielt. Auf eAnci erzielt **FTLR** mit *UniXcoder* und ▼ ein F_1 -Maß von 34,7 % gegenüber den 29 % von **FTLR** mit *fastText*. Allerdings sind die Ergebnisse der *UniXcoder*-Variante mit vollständiger Vorverarbeitung wiederum vergleichbar mit oder sogar schlechter auf iTrust und LibEST als *fastText*. Es zeigt sich also, dass vor allem die veränderte Vorverarbeitung auf den zwei Projekten die Verbesserung erzeugt und diese leider auf den anderen Projekten Nachteile in der Leistung bringt.

Um auch Forschungsfrage **FF**₁₇ zu adressieren und Erkenntnisse bezüglich der Übertragbarkeit auf ungesehene Projekte zu liefern, kann die Spanne der optimierten Schwellenwerte über die Projekte hinweg betrachtet werden. Denn ist es nicht möglich eine allgemeingültige Schwellenwertkombination zu finden, muss mit einer deutlich schlechteren Leistung bei der Anwendung auf neue Projekte gerechnet werden. Aus der Darstellung der Spannen in Abbildungen 13.1 und 13.2 kann gefolgert werden, dass diese insbesondere für den Mehrheitsschwellenwert sehr stark variieren. Mit einem durchschnittlichen Interquartilsabstand von 0,16 und einer maximalen Spanne von 0,27 lässt sich hier nur schwer ein geeigneter fester Schwellenwert identifizieren. Für den Abschlusschwellenwert ist die maximale Spanne zwar nur bei 0,19. Der Interquartilsabstand liegt allerdings auch hier im Durchschnitt bei 0,15. Die entsprechenden durchschnittlichen Interquartilsbereiche von **FTLR** mit *fastText* sind mit 0,05 für den Mehrheitsschwellenwert und 0,07 für den Abschlusschwellenwert deutlich kleiner.

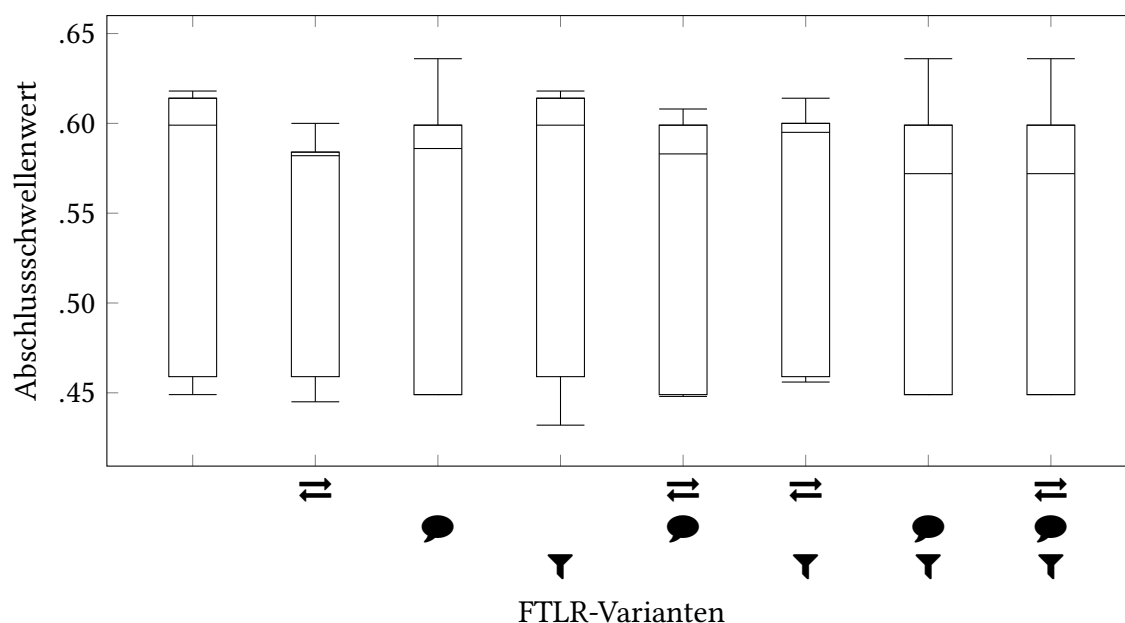


Abbildung 13.2.: Varianz der optimierten Abschlusschwellenwerte mit UniXcoder, vollständiger Vorverarbeitung und Kosinusähnlichkeit über die Datensätze hinweg. Jeder Graph deckt somit 5 Datenpunkte ab

Versucht man auch für *UniXcoder* den Durchschnitt der Mediane über die Konfigurationen, als feste Schwellenwertkombination zu verwenden, ergeben sich die Schwellenwerte 0,302 für den Mehrheitsschwellenwert und 0,586 für den Abschlusschwellenwert. Mit dieser Schwellenwertkonfiguration erzielt **FTLR** mit *UniXcoder*, vollständiger Vorverarbeitung und Kosinusähnlichkeit auf 💬⚡ mit einem F_1 -Maß von 25,7 % das beste Ergebnis. Dieses liegt 9 Prozentpunkte unter dem Ergebnis (F_1 -Maß von 34,7 %), welches **FTLR** mit *fastText* und der Median-basierten festen Schwellenwertkombination erzielt. Die Varianz der Schwellenwerte sowie diese Ergebnisse verdeutlichen, dass **FTLR** mit *UniXcoder* weniger geeignet für die unüberwachte Anwendung auf ungesehene Projekte ist als **FTLR** mit *fastText*. Außerdem ist auch die benötigte Rechenkapazität für *UniXcoder*, eine GPU mit mindestens 16 GB RAM, deutlich höher als die Anwendung von *fastText*-Einbettungen, welche allein auf der CPU mit einem RAM von 8 GB laufen kann. Daher ist im Allgemeinen *fastText* weiterhin der Anwendung von *UniXcoder* vorzuziehen.

13.4. Gefährdungen der Gültigkeit

Auch für die in diesem Kapitel durchgeführten Experimente und die gefolgerten Aussagen bestehen Gefahren bezüglich ihrer Gültigkeit. Wie schon bei den Experimenten zuvor stellt auch hier die externe Gültigkeit die gravierendste Gefahr für die Gültigkeit der erlangten Erkenntnisse dar. Die begrenzte Menge der evaluierten Projekte und die Möglichkeit, dass diese nicht repräsentativ für die eigentliche Anwendung sind, stellen weiterhin eine Gefahr für die Übertragbarkeit der hier gewonnenen Erkenntnisse dar.

Zusätzlich wurde *UniXcoder* auf zwei bestimmte Arten in das bestehende Verfahren **FTLR** integriert. Die Erkenntnisse beziehen sich also nur auf diese Arten der Integration und insbesondere auch nur auf eine Anwendung in **FTLR**. Dies bedeutet also nicht, dass es nicht noch andere Anwendungsmöglichkeiten des *UniXcoder*-Sprachmodells für die automatische TLR zwischen Anforderungen und Quelltext gibt, in denen eine bessere Leistung als **FTLR** mit *fastText* erzielt werden kann. Da die verwendeten Techniken zur Integration allerdings den Standard für Transformer-basierte Sprachmodelle in einer unüberwachten Anwendung darstellen, sollten zumindest für diese die Erkenntnisse eine gewisse Gültigkeit haben.

14. Zusammenfassung

Ziel dieses Teils dieser Arbeit war es, die Leistung **FTLRs** mit einem bimodalen, auf natürlicher und Programmiersprache trainierten, kontextsensitiven Sprachmodell zu bemessen und zu analysieren, ob diese einen Mehrwert für die Anwendung in der unüberwachten Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext bieten. Hierzu wurde das bimodale Sprachmodell *UniXcoder*, welches den Stand der Technik für bimodale Sprachmodelle auf der verwandten Aufgabe der Quelltextsuche darstellt, betrachtet. Zur Integration dieser Form von Sprachmodellen wurde sowohl eine Einbettungs-Multimenge-basierte Repräsentation mit anschließendem WMD-Ähnlichkeitsvergleich als auch die Verwendung der Repräsentation der Eingabe durch einen einzelnen Eingabeeinbettungsvektor und anschließender Berechnung der Kosinusähnlichkeit betrachtet.

Die Ergebnisse auf den Vergleichdatensätzen zeigen, dass die Eingabeeinbettungsvektor-Repräsentation im Durchschnitt bessere Ergebnisse im F_1 -Maß erzielt. Zusätzlich wurde untersucht, ob eine von **FTLRs** Vorverarbeitung abweichende, reduzierte Vorverarbeitung besser für die Anwendung dieser Art von Sprachmodellen in der gegebenen Aufgabe passt. Hierbei konnte festgestellt werden, dass im Durchschnitt über die fünf Vergleichdatensätze die reduzierte Vorverarbeitung schlechtere Ergebnisse im F_1 -Maß erzielt. Lediglich auf den Projekten iTrust und LibEST konnten deutliche Verbesserungen mit der veränderten Vorverarbeitung erzielt werden. Auf den anderen Projekten verschlechterte diese Form der Vorverarbeitung allerdings die Ergebnisse teils so deutlich, dass kein verbessertes Gesamtergebnis festgestellt werden konnte.

Ein Vergleich der Ergebnisse der besten **FTLR**-Varianten mit *fastText* und *UniXcoder* ergab, dass **FTLR** mit *UniXcoder* im Durchschnitt über die Vergleichdatensätze keine Verbesserung gegenüber **FTLR** mit *fastText* erzielen konnte. Da weder eine einfache noch eine signifikante Verbesserung der Leistung durch die Verwendung von *UniXcoder* festgestellt werden konnte, kann These **T₃** nicht bestätigt werden. Zusätzlich variieren die optimalen Schwellenwerte für **FTLR** mit *UniXcoder* über die Projekte hinweg so stark, dass eine für eine spätere Anwendung in der Praxis geeignete, feste Schwellenwertkombination kaum zu ermitteln ist. Ein Test mit dem Median der Schwellenwerte ergab deutlich schlechtere Ergebnisse als es mit **FTLR** und *fastText* der Fall war (9 Prozentpunkte schlechteres F_1 -Maß). Da außerdem der Aufwand für das bimodale Sprachmodell in der Ausführung deutlich höher ist, kann festgehalten werden, dass sich dessen Einsatz für die automatische TLR zwischen Anforderungen und Quelltext mittels **FTLR** nicht eignet.

Teil V.

Epilog

15. Vergleich mit bestehenden Ansätzen

Abschließend wird in diesem Kapitel nun der Vergleich der in dieser Arbeit entwickelten Verfahren mit bestehenden Ansätzen durchgeführt. Dies ordnet die erzielte Leistung **FTLRs** und seiner Erweiterungen in das Forschungsfeld der TLR ein und liefert Erkenntnisse bezüglich These T_1 :

T_1 : Eine unüberwachte Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext mittels feingranularer, worteinbettungsbasierter Beziehungen steigert die Leistung gegenüber bestehenden unüberwachten Verfahren.

Da das Ziel dieser Arbeit die unüberwachte TLR ist, spielt hierbei vor allem die Leistung **FTLRs** mit festgelegten Schwellenwerten eine Rolle. Diese Varianten können ohne Weiteres auf neue Projekte angewandt werden und sind somit besonders geeignet für den Einsatz in der Praxis. Aus diesem Grund wird zunächst folgende Forschungsfrage beantwortet:

FF₁₈: Welche Leistung erzielt **FTLR** mit festgelegten Schwellenwerten im Vergleich zu bestehenden unüberwachten Ansätzen?

Zusätzlich wird in diesem Kapitel außerdem ein Vergleich mit überwachten Verfahren durchgeführt und damit folgende Forschungsfrage adressiert:

FF₁₉: Wie groß ist der Leistungsunterschied zwischen Ansätzen zur unüberwachten und überwachten TLR?

Dies dient der Einordnung, ob sich der Aufwand für die Bereitstellung initialer TLs lohnt bzw. wie groß die Differenz zwischen diesen beiden Aufgaben tatsächlich ist.

Zunächst werden zur Beantwortung der Fragen in Abschnitt 15.1 die verwandten Arbeiten zur TLR zwischen Anforderungen und Quelltext auf ihre Eignung für einen Vergleich untersucht und bestimmt, auf welchen Datensätzen verglichen werden kann. Daraufhin enthält Abschnitt 15.2 den Vergleich mit unüberwachten Verfahren. In Abschnitt 15.3 wird anschließend die Leistung im Vergleich zu überwachten Ansätzen analysiert.

15.1. Auswahl der Vergleichdatensätze und Verfahren

Die in dieser Arbeit zur Evaluation verwendeten Datensätze wurden so gewählt, dass sie die nötigen Informationen für die unüberwachte TLR zwischen Anforderungen und Quelltext enthalten (s. Abschnitt 6.1). Sie umfassen natürlichsprachliche Anforderungen und Quelltext in Form von Quelltextdateien, sowie Goldstandard-TLs. Zusätzlich wurden

Tabelle 15.1.: Übersicht der verwendeten Vergleichsdatensätze bestehender Arbeiten zur automatischen Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext. ● markiert Ansätze, welche dieselbe Version des Datensatzes benutzen wie diese Arbeit und ○ Ansätze, die eine andere Version verwenden

Ansatz	eTour	iTrust	SMOS	eAnci	LibEST
Unüberwacht					
Gethers et al. [Get+11]		○	●	●	
DTC [Loh+13]		○			
Panichella et al. [Pan+13]	○		●		
Kuang et al. [Kua+15]		○			
Mahmoud und Niu [MN15]	○	○			
S2Trace [Che+19]	●	●			
COMET [Mor+20b]	●	●	●		●
Rodriguez und Carver [RC20]	●				
Zhang et al. [Zha+21]	●	○			
Tracerefiner [Ham+21]		○			
Überwacht					
WQI [ZCS17]	●	●			
TRAIL [MEH18]	○	○	●	●	
ALCATRAL [Mil+19]	○	○	●	●	

sie unter dem Gesichtspunkt ausgewählt, dass sie schon in Experimenten bestehender Ansätze verwendet wurden und dadurch ein Vergleich mit den in dieser Arbeit erzielten Ergebnissen möglich ist.

Betrachtet man nun die in Abschnitt 3.1 analysierten bestehenden Ansätze zur TLR zwischen Anforderungen und Quelltext, so bietet sich ein heterogenes Bild der Nutzung von Datensätzen. Viele der Ansätze beschäftigen sich neben der TLR von Anforderungen und Quelltext auch mit anderen Formen der TLR. Daher sind viele der genannten Datensätze, wie z. B. CCHIT oder CM1, für diesen Vergleich gar nicht geeignet (s. Tabelle 3.2). Andere Datensätze, die zwar zur TLR zwischen Anforderungen und Quelltext genutzt werden, enthalten anstatt Quelltextdateien lediglich Klassenbeschreibungen (s. Tabelle 3.1). Dies betrifft beispielsweise den häufig verwendeten EasyClinic-Datensatz. Da **FTLR** aus solchen Klassenbeschreibungen keine strukturellen Informationen des Quelltextes ziehen kann und diese normalerweise auch nicht in einem Projekt vorhanden sind, eignen sich diese ebenso wenig für einen Vergleich. Der EBT-Datensatz enthält zwar grundsätzlich Java-Quelltext, dieser ist aber unvollständig und syntaktisch nicht korrekt, weshalb auch hier keine strukturellen Informationen durch statische Quelltextanalyse gewonnen werden können. Des Weiteren nutzen manche Arbeiten nicht öffentlich verfügbare oder nicht mehr auffindbare Datensätze, wie z. B. den LEDA- oder SE450-Datensatz. Betrachtet man also nur die tatsächlich geeigneten Datensätze so werden von diesen eTour, iTrust, SMOS und eAnci am häufigsten verwendet. Da diese und zusätzlich LibEST, außerdem die Datensätze

sind, auf denen auch die Klassifikation als Vorfilter und die *UniXcoder*-Adaption evaluiert wurden, beschränke ich mich in dieser Auswertung auf diese Datensätze.

Tabelle 15.1 gibt einen Überblick der Nutzung dieser Datensätze durch bestehende Ansätze. Auf den ersten Blick sieht es so aus als könnten fast alle Ansätze auf *iTrust* und *eTour* mit **FTLR** verglichen werden. Allerdings verwenden einige Arbeiten andere Versionen der Datensätze, welche durch \circ gekennzeichnet sind (s. auch Tabelle 3.1). Bei *eTour* handelt es sich dabei um eine Version mit 366 anstatt 308 Musterlösungsverbindungen. Leider konnte ich diese Version des Datensatzes nicht mehr auffinden. Bei *iTrust* ist die Situation sogar noch etwas schwieriger, da es sich nicht nur um eine Variante des Datensatzes, sondern um mehrere, teils sehr unterschiedliche Versionen handelt. Hier verändert sich nicht nur die Zahl der Musterlösungsverbindungen, sondern auch die Anzahl der Artefakte variiert stark. Außerdem ist leider oftmals die Version nicht explizit angegeben oder die angegebene Version ist nicht mehr vollständig online verfügbar (s. Tabelle 3.1). Daher kann ich mich hier nur auf die Arbeiten mit derselben Version (*iTrust*₃₉₉) stützen. Diese Arbeiten beinhalten die beiden Verfahren *S2Trace* [Che+19] und *WQI* [ZCS17], die ebenso Einbettungen verwenden, sowie das *COMET*-Verfahren [Mor+20b], welches insbesondere durch die Kombination verschiedener Verfahren durch ein *HBN* sehr gute Ergebnisse in der unüberwachten TLR erzielte.

Ein weiteres Problem beim Vergleich stellen die in den Veröffentlichungen enthaltenen Informationen zur Güte dar. Viele, insbesondere der älteren Arbeiten geben lediglich Präzision-Ausbeute-Kurven als Datengrundlage an, ohne konkrete Werte zu nennen. In diesen Fällen kann das maximal mögliche F_1 -Maß z. B. nur grob abgeschätzt werden. Selbiges gilt für Balkendiagramme ohne Nennung der konkreten Zahlen, sodass auch hier der eigentliche Wert nur geschätzt werden kann. Da diese Arbeiten außerdem zumeist entweder gar keine oder nicht mehr verfügbare Replikationspakete oder Zusatzpakete mit konkreten Ergebnissen besitzen, können diese nur erschwert für einen Vergleich verwendet werden. Dies betrifft u.a. die Arbeiten von Gethers et al. [Get+11] und Zhang et al. [Zha+21]. Erstere zeigen in der Veröffentlichung gar keine Ergebnisse für *SMOS* oder *eAnci* und verweisen hierzu nur auf den nicht mehr vorhandenen Online-Anhang. Bei letzteren kann zumindest eine grobe Abschätzung des F_1 -Maßes für *eTour* aus der Präzision-Ausbeute-Kurve entnommen werden, da die Skalen hochauflösend genug sind.

Positiv möchte ich an dieser Stelle Moran et al. [Mor+20b] hervorheben, welche ein umfassendes Zusatzpaket zur Verfügung gestellt haben. Dieses umfasst die Ergebnislisten der IR-basierten TLR inklusive Ähnlichkeitsbewertung, aus denen sich zusätzlich zu den in der Arbeit beschriebenen AP-Werten auch MAP und das maximal mögliche F_1 -Maß bestimmen lassen. Zusätzlich haben die Autoren auf Nachfrage weitere Ergebnisse zur Verfügung gestellt, sodass *COMET* dasjenige Verfahren darstellt zu dem **FTLR** am umfassendsten verglichen werden kann. Der Quelltext zur Replikation der Ergebnisse dieser Dissertation ist auf Zenodo verfügbar [Hey23b; Hey23c].

Eine Möglichkeit eine breitere Masse an Vergleichswerten zu erhalten, wäre es eine Nachimplementierung der Verfahren durchzuführen. Idealerweise könnte dabei auf Replikationspakete zurückgegriffen werden, welche allerdings für keines der Verfahren zur Verfügung stehen. Da nicht alle Veröffentlichungen die benötigten Informationen













zur Nachimplementierung enthalten und um den Aufwand in Maßen zu halten, wurde sich auf das vielversprechendste Verfahren konzentriert. Das Verfahren von Rodriguez und Carver [RC20] (s. Abschnitt 3.1.2), welches multikriterielle Optimierung einsetzt, erzielte laut den Autoren eine nahezu doppelt so gute Leistung auf dem eTour-Datensatz als alle bestehenden Verfahren. Aus diesem Grund haben wir im Rahmen des Prozesses zur Veröffentlichung der ersten Version von **FTLR** [Hey+21] versucht, die Ergebnisse auf eTour zu replizieren. Unsere Reimplementierung konnte allerdings nur ein maximales F_1 -Maß von 27 % anstatt den in der Veröffentlichung angegebenen 71,9 % erzielen. Auf Nachfrage bezüglich einiger Unklarheiten zu Implementierungsdetails bekamen wir leider bis heute keine Antwort. Auch Keim et al. [Kei+21] haben das Verfahren unabhängig von uns nachimplementiert und konnten ebenso keine überzeugenden Ergebnisse in der TLR zwischen Architekturdokumentation und Architekturmodell mit diesem erzielen (s. Abschnitt 3.5). Das Verfahren nutzt als einzige Form des semantischen Ähnlichkeitsvergleichs TF-IDF in Kombination mit Jaccard-Koeffizienten, welche auch in vielen anderen Verfahren benutzt werden [Ant+02; ZSC10; Loh+13; Pan+13; ZCS17]. Da trotz des genetischen Algorithmus die Entscheidung über die letztlichen Verbindungskandidaten auf Basis dieser Ähnlichkeit getroffen wird, passt das Ergebnis der Reimplementierung besser zu den bestehenden Ergebnissen als die in der Veröffentlichung genannten. Da die Experimente mit der Nachimplementierung außerdem ergaben, dass die Güte des Verfahrens stark von den zufällig gezogenen Startindividuen abhängt, wird dieses nicht weiter zum Vergleich herangezogen.



Somit werden für den Vergleich der unüberwachten Verfahren die Ansätze von Panichella et al. [Pan+13], S2Trace von Chen et al. [Che+19], COMET von Moran et al. [Mor+20b] und Zhang et al. [Zha+21] auf den Datensätzen eTour, iTrust, SMOS und LibEST verwendet. Im Fall der überwachten TLR können WQI von Zhao et al. [ZCS17], TRAIL von Mills et al. [MEH18] und ALCATRAL von Mills et al. [Mil+19] auf den Datensätzen eTour, iTrust, SMOS und eAnci verwendet werden.

15.2. Unüberwachte Wiederherstellung von Nachverfolgbarkeitsverbindungen

Das Hauptziel dieser Arbeit ist es, ein Verfahren zu entwickeln, welches die Leistung in der automatischen unüberwachten TLR zwischen Anforderungen und Quelltext verbessert. Um tatsächlich die unüberwachte Leistung **FTLRs** zu bemessen, also den Einsatz **FTLRs** ohne Nachverfolgbarkeitsverbindungsinformationen über das Projekt, müssen die festgelegten Schwellenwertkombinationen betrachtet werden. Diese würden auch in einer Anwendung in der Praxis auf ungesehene Projekte die erste Wahl darstellen, da keine Informationen zu den Ähnlichkeiten zwischen Artefaktelementen in einem neuen Projekt vorhanden sind. Die Ergebnisse **FTLRs** mit den beiden festgelegten Schwellenwertkombinationen, ORG und MED, auf den unterschiedlichen Varianten (s. Abschnitte 6.4 und 10.1) zeigen, dass ohne Anforderungselementfilter **FTLR** mit **Y** und **FTLR** mit **Y** die besten Ergebnisse auf den

Tabelle 15.2.: Vergleich von **FTLRs** Leistung mit bestehenden Ansätzen zur unüberwachten Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext. Die besten Werte pro Metrik und Projekt sind fett dargestellt

Ansatz	eTour		iTrust		iTrustJSP		SMOS		LibEST	
	F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP
Panichella et al.								.361		
S2Trace	.158				.267					
COMET _{MAP}	.437	.467	.282	.252	.267	.260	.276	.294	.625	.677
COMET _{NUTS}	.435	.455	.215	.210	.231	.251	.274	.293	.624	.673
COMET _{VI}	.436	.452	.216	.212	.231	.258	.291	.314	.625	.675
FTLR- ORG	.475	.528	.190	.227	.187	.187	.361	.438	.442	.559
FTLR- MED	.472	.528	.166	.227	.167	.187	.370	.438	.524	.559
FTLR-  ORG	.469	.483	.206	.284	.199	.239	.381	.451	.509	.581
FTLR-  MED	.461	.483	.172	.284	.172	.239	.390	.451	.538	.581
FTLR- +NF ORG	.406	.363	.219	.279	.209	.233	.382	.424	.509	.591
FTLR- +NF MED	.370	.363	.182	.279	.181	.233	.390	.424	.538	.591
FTLR-  +NF ORG	.474	.484	.222	.267	.211	.226	.380	.450	.509	.591
FTLR-  +NF MED	.468	.484	.185	.267	.182	.226	.390	.450	.538	.591

Datensätzen zeigen. Ohne  werden die besten Ergebnisse durch **FTLR** mit  und den NF-Filter erzielt.

Daher wird zur Beantwortung von Forschungsfrage **FF**₁₈ die Leistung dieser Varianten mit der Leistung bestehender Arbeiten verglichen. Als Vergleichsgrundlage werden hierbei die Verfahren von Panichella et al. [Pan+13], S2Trace von Chen et al. [Che+19], COMET von Moran et al. [Mor+20b] und das Verfahren von Zhang et al. [Zha+21] herangezogen. Das Verfahren von Panichella et al. [Pan+13] nutzt strukturelle Informationen des Quelltexts, wie z. B. Methodenaufrufabhängigkeiten und Vererbungsbeziehungen, um ein auf dem probabilistischen Jensen-Shannon-Modell basierendes Abbildungsverfahren zu verbessern. S2Trace nutzt *doc2vec* zur Repräsentation der Artefakte und kombiniert diese mit Sequenzinformationen, um die Reihenfolge der Wörter miteinzubeziehen. Bei COMET handelt es sich um ein kombinierendes Verfahren, welches auf Basis eines Hierarchischen Bayes'schen Netzes die Ähnlichkeiten verschiedener IR-Verfahren kombiniert. Zur Abschätzung der Wahrscheinlichkeit einer TL kann COMET die Approximationsverfahren Maximum-a-posteriori-Schätzung (MAP), Markov-Ketten-Monte-Carlo-Verfahren mit No-U-Turn Abtastung (NUTS) und Variationsinferenz (VI) einsetzen. Das Verfahren von Zhang et al. [Zha+21] setzt ebenso auf Worteinbettungen und gewichtet *word2vec*-Worteinbettungen mittels *self-attention* und überführt diese durch eine lineare Transformation in einen „semantischen“ Vektor.

In Tabelle 15.2 sind die Ergebnisse des Vergleichs der Ansätze dargestellt. Hierbei lässt sich feststellen, dass auf eTour und SMOS **FTLR** mit allen Varianten sowohl im F₁-Maß

als auch im MAP die besten Ergebnisse erzielt. Lediglich die Anwendung des NF-Filters in Kombination mit \bullet erzielt schlechtere Ergebnisse auf eTour als COMET. Auf SMOS liegt der beste MAP, erzielt durch **FTLR- \bullet \blacktriangledown** , mit 45,1 % um 9 Prozentpunkte über dem besten Wert eines bestehenden Verfahrens (36,1 %), erzielt durch das Verfahren von Panichella et al. [Pan+13]. Auf eTour kann dieselbe Variante das auf *doc2vec* basierende Verfahren S2Trace um 31,1 Prozentpunkte übertreffen, indem es ein F_1 -Maß von 46,9 % erzielt. Die **FTLR**-Variante, die nur \blacktriangledown benutzt, erzielt sogar ein F_1 -Maß von 47,5 %. Im Vergleich zu COMET ist die Verbesserung allerdings nicht mehr ganz so gravierend. Die beste Konfiguration $\text{COMET}_{\text{MAP}}$, welche Maximum-a-posteriori-Schätzung (MAP) verwendet, erzielt mit einem F_1 -Maß von 43,7 % nur 3,8 Prozentpunkte weniger als die beste **FTLR**-Variante (\blacktriangledown ORG). Hier ist anzumerken, dass für die Bestimmung des F_1 -Maßes von COMET der maximal mögliche Wert aus den zur Verfügung stehenden Kandidatenlisten berechnet wurde. Es wurde also eine Optimierung des Schwellenwertes durchgeführt¹. Aus diesem Grund sind die MAP-Ergebnisse aussagekräftiger für die unüberwachte Anwendung. Betrachtet man MAP ist der Unterschied mit 52,8 % durch **FTLR- \blacktriangledown** im Vergleich zu 46,7 % durch $\text{COMET}_{\text{MAP}}$ deutlicher.





Auf eTour kann auch ein Vergleich zum Ansatz von Zhang et al. [Zha+21] durchgeführt werden, der allerdings nur auf einer Schätzung aus den Präzision-Ausbeute-Kurve der Veröffentlichung basiert². Das maximal erreichbare F_1 -Maß kann hier mit ungefähr 48 % geschätzt werden und ergibt sich aus einer Präzision von ungefähr 40 % und einer Ausbeute von 60 %. Dieser Wert liegt mit 0,5 Prozentpunkten knapp über dem besten von **FTLR** mit festgelegten Schwellenwerten erzielten Ergebnis. Da es sich hierbei allerdings ebenso um die optimale Schwellenwertkonfiguration für das Verfahren von Zhang et al. handelt, sollte auch hier ein Vergleich zu **FTLR** mit optimierten Schwellenwerten herangezogen werden. **FTLR** erzielt mit optimierten Schwellenwerten auf allen Varianten, die \blacktriangledown verwenden, ein F_1 -Maß von mindestens 51,7 % und im Maximum sogar 55,2 % (s. Abschnitt 6.3). Somit ist **FTLR** in der Lage, ein besseres Ergebnis auf eTour zu erzielen als die Vergleichsarbeit.



Die Ergebnisse **FTLRs** im Vergleich zu bestehenden Ansätzen sind somit vielversprechend auf eTour und SMOS. Betrachtet man allerdings iTrust und LibEST, so ergibt sich ein anderes Bild. Auf dem vollständigen iTrust-Datensatz inklusive JSP-Dateien (iTrustJSP) erzielt **FTLR** mit allen Varianten sowohl im F_1 -Maß als auch MAP schlechtere Ergebnisse als COMET. Hier ist auch das Ergebnis, welches S2Trace erreicht, besser als **FTLR**. S2Trace und $\text{COMET}_{\text{MAP}}$ erzielen beide ein F_1 -Maß von 26,7 %. **FTLR** hingegen erzielt maximal 21,1 % unter Kombination aller Varianten ($\rightleftharpoons\bullet\blacktriangledown$) mit dem NF-Filter. Im MAP ist die Differenz zwischen dem besten Ergebnis von 26 % durch $\text{COMET}_{\text{MAP}}$ zu den 23,9 % durch **FTLR- \bullet \blacktriangledown** geringer. Als ein Grund für dieses schlechte Abschneiden kann der schlechtere Umgang **FTLRs** mit JSP-Dateien angeführt werden. **FTLR** verwendet hier nur den Dateinamen als Informationsquelle und kann somit keine Informationen zur Struktur des Quelltextes und auch keine feingranulare Abbildung nutzen. Hier kann also eine der Kernfunktionalitäten **FTLRs**, der Mehrheitsentscheid, nicht genutzt werden. Betrachtet man hingegen die Er-

¹ Vergleicht man das erzielte F_1 -Maß mit dem von **FTLR- \blacktriangledown** ebenfalls mit auf eTour optimierten Schwellenwerten, so fällt der Unterschied größer aus. **FTLR- \blacktriangledown OPT** erzielt ein F_1 -Maß von 54,8 %.

² Aus diesem Grund wurde der Wert nicht in die Tabelle aufgenommen.

Tabelle 15.3.: Vergleich des durchschnittlichen F_1 -Maßes bzw. MAP der Ansätze auf verschiedenen Datensatzmengen

Ansatz	eTour + SMOS							
	+ LibEST							
	+ iTrustJSP		+ iTrust		+ iTrustJSP		+ iTrust	
	$\emptyset F_1$	\emptyset MAP	$\emptyset F_1$	\emptyset MAP	$\emptyset F_1$	\emptyset MAP	$\emptyset F_1$	\emptyset MAP
COMET _{MAP}	.401	.425	.405	.423	.327	.340	.332	.338
COMET _{NUTS}	.391	.418	.387	.408	.313	.333	.308	.319
COMET _{VI}	.396	.425	.392	.413	.319	.341	.314	.326
FTLR- ORG	.366	.438	.367	.438	.341	.384	.342	.398
FTLR- ORG	.390	.439	.391	.450	.350	.391	.352	.406
FTLR- +NF ORG	.377	.403	.379	.414	.332	.340	.336	.355
FTLR- +NF ORG	.394	.438	.396	.448	.355	.387	.359	.400

gebnisse auf iTrust ohne JSP-Dateien, so bleibt zwar der Unterschied im F_1 -Maß ähnlich, aber **FTLR** erzielt nun bessere Werte im MAP als COMET. Mit 28,4 % erzielt **FTLR-** ein um 3,2 Prozentpunkte höheres MAP als die 25,2 % durch COMET. Dies bedeutet **FTLR** ist besser in der Lage, die relevanten Nachverfolgbarkeitsverbindungskandidaten in den vorderen Positionen der Kandidatenliste zu platzieren als COMET. Es eignet sich also auf iTrust besser für eine semi-automatische Anwendung, bei der eine Vorschlagsliste an Expert:innen übergeben wird. Der schlechtere Wert im F_1 -Maß deutet aber darauf hin, dass die Schwellenwertkombination und somit der Mehrheitsentscheid nicht das volle Potenzial der feingranularen Ähnlichkeiten ausnutzt. Mit optimierten Schwellenwerten für iTrust kann **FTLR** mit **** und dem NF-Filter ein F_1 -Maß von 26,1 % erzielen (s. Abschnitt 10.3).

Auf LibEST unterliegt **FTLR** sowohl im F_1 -Maß als auch MAP in allen Varianten der Leistung COMET's. Das beste erzielte F_1 -Maß von 53,8 % liegt 8,7 Prozentpunkte unter dem F_1 -Maß, welches die COMET-Konfigurationen erzielen (62,5 %). Im MAP zeigt sich eine ähnliche Differenz zwischen dem besten Ergebnis durch COMET_{MAP} mit 67,7 % und den **FTLR**-Varianten, die den NF-Filter nutzen, mit 59,1 %. Eine Erklärung liegt auch hier in der Art der Quelltextartefakte. Bei LibEST handelt es sich um ein Projekt in der Programmiersprache C und somit nicht um objektorientierten Quelltext. Da der Fokus **FTLRs** auf einer Verbesserung der Leistung der TLR zu objektorientiertem Quelltext liegt, ist dieser niedrigere Wert zu erwarten gewesen. Die Annahmen bezüglich der wenigen, idealerweise sogar dem einen Zwecke, die ein Quelltextartefakt anbieten, halten für eine Bibliothek in der Programmiersprache C nicht mehr. LibEST besteht aus 14 Quelltextdateien mit durchschnittlich 31 Methoden pro Datei (s. Tabelle 6.2), die teils ganz unterschiedliche Zwecke erfüllen. Hier führt der Einsatz des Mehrheitsentscheids zur Aggregation dazu, dass nicht alle Verbindungen gefunden werden können, da nur diejenigen Verbindungen erstellt werden, die pro Klasse die meisten Stimmen bekommen haben. In der Evaluation der *UniXcoder*-Einbindung in **FTLR** hatte sich allerdings gezeigt, dass eine veränderte Vorverarbeitung auf LibEST zu einer deutlichen Leistungssteigerung im MAP führen kann.

Führt man **FTLR-☛** mit der auf Querverweis-, Sonderzeichen- und Nummerentfernung sowie Umwandlung in Kleinbuchstaben reduzierten Vorverarbeitung aus, kann ein MAP von 66,9 % erzielt werden. Dieses liegt nur noch 0,8 Prozentpunkte unter dem erzielten Wert durch COMET.

Betrachtet man die durchschnittliche Leistung der Verfahren auf den Vergleichsdatensätzen in Tabelle 15.3, so erzielt **FTLR** mit **☛** den besten MAP, unabhängig davon, ob iTrust oder iTrustJSP betrachtet wird. Mit iTrust liegt der durch **FTLR** erzielte durchschnittliche MAP bei 45 % im Gegensatz zu 42,3 % durch $\text{COMET}_{\text{MAP}}$. Im durchschnittlichen F_1 -Maß allerdings erzielt COMET leicht höhere Werte, unabhängig davon, ob iTrust oder iTrustJSP betrachtet wird. Dieser Leistungsvorsprung wird allerdings zum größten Teil durch den Vorsprung auf dem nicht objektorientierten LibEST erzeugt. Nimmt man LibEST aus der Durchschnittsbildung heraus, so erzielen alle dargestellten **FTLR**-Varianten auch ein höheres durchschnittliches F_1 -Maß als COMET. Mit 35,9 % zu 33,2 %, unter Betrachtung iTrusts ohne JSP-Dateien, erzielt **FTLR-☛** +NF ein um 2,7 Prozentpunkte bzw. 8 % höheres F_1 -Maß. Der Unterschied im Durchschnitt unter Betrachtung des vollständigen iTrust-Datensatzes (+ iTrustJSP) ist mit 2,8 Prozentpunkten bzw. 8,6 % sogar noch leicht größer. Bei der Betrachtung ohne LibEST wird außerdem der Unterschied im MAP noch deutlicher. Dieser liegt mit 40,6 % durch **FTLR-☛** zu 33,8 % durch $\text{COMET}_{\text{MAP}}$ bei 6,8 Prozentpunkten bzw. über 20 %.

Leider ist die Stichprobenzahl mit drei bzw. vier Datensätzen zu klein, um eine Aussage über die Signifikanz der Steigerung der Leistung durch **FTLR** zu treffen. Es kann aber festgehalten werden, dass **FTLR** in der Lage ist, im Durchschnitt über die Datensätze, welche die Anforderung **FTLRs** nach objektorientiertem Quelltext erfüllen, eine Verbesserung der Leistung sowohl im F_1 -Maß als auch im MAP gegenüber COMET zu erzielen.

Abbildung 15.1 zeigt die Präzision-Ausbeute-Kurven von **FTLR-☛** im Vergleich mit $\text{COMET}_{\text{MAP}}$. Um diese Kurven für **FTLR** zu bestimmen, wurden der Mehrheitschwellenwert und der Abschlusschwellenwert in 0,001er-Schritten variiert und für jede Kombination die Ausbeute und Präzision bestimmt. Sollten mehrere Kombinationen dieselbe Ausbeute erzielen wurde diejenige in den Graph übernommen, welche die höhere Präzision aufweist. Somit dienen diese Kurven zur Verdeutlichung der möglichen Leistung der Verfahren. Idealerweise sollte sich die Kurve der oberen rechten Ecke annähern, denn dort liegt ein F_1 -Maß von 100 %.

Die Kurven unterstützen die Erkenntnisse aus Tabelle 15.2. Auf dem SMOS-Datensatz erzielt **FTLR** auf allen Ausbeuteniveaus eine höhere Präzision als COMET. Für eTour übertrifft **FTLR** COMET bei allen Ausbeuteniveaus oberhalb 22 %. Auf LibEST ist die schlechtere Präzision **FTLRs** auch in den Kurven zu erkennen. Man sieht aber, dass sich der Verlauf der Kurven ähnelt, **FTLR** allerdings nicht dasselbe Präzisionsniveau erreicht. Die Kurven auf dem iTrust-Datensatz verdeutlichen gut wie der Unterschied im F_1 -Maß zustande kommt. Das maximale F_1 -Maß durch COMET wird bei einer Ausbeute von 23 % erzielt. Bis zu diesem Ausbeuteniveau erreicht COMET eine höhere Präzision. Bei höheren Ausbeuten gleichen sich die beiden Kurven nahezu und ab einem Ausbeuteniveau von 46 % übersteigt die durch **FTLR** erzielte Präzision diejenige von COMET sogar. Da ein Ausbeuteniveau von 23 % allerdings wenig hilfreich für eine Anwendung in der Praxis

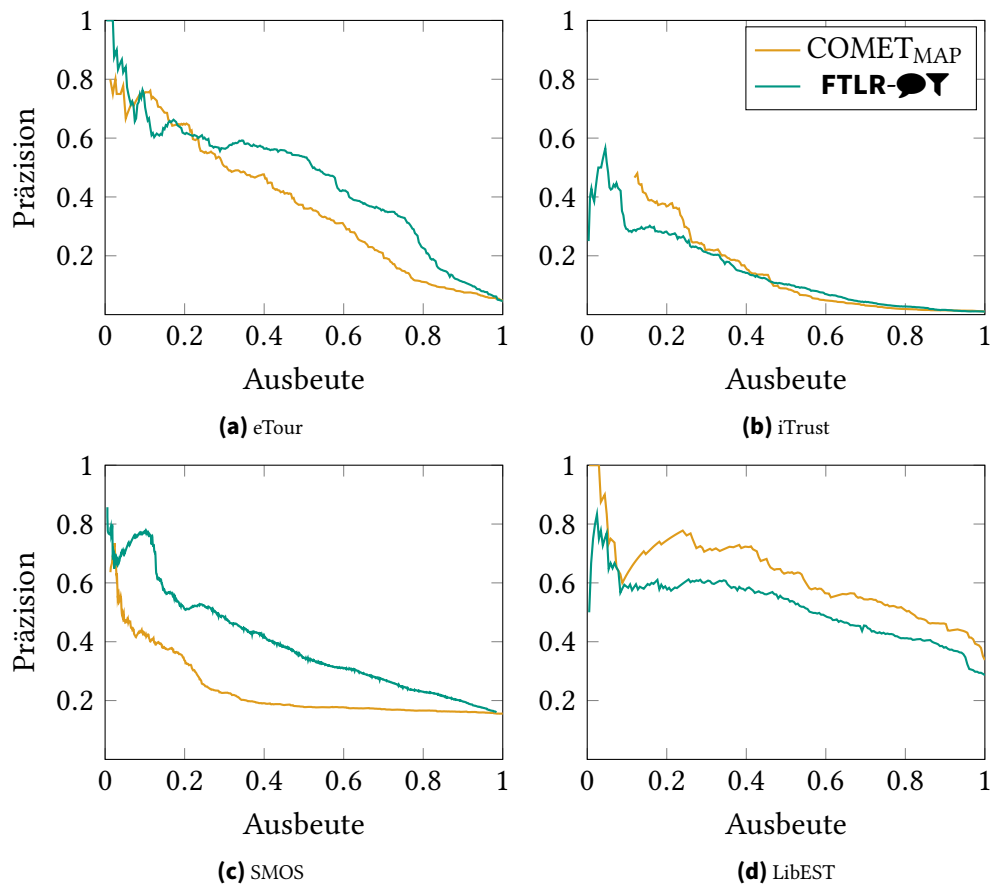






Abbildung 15.1.: Präzision-Ausbeute-Kurven von **FTLR** im Vergleich mit $\text{COMET}_{\text{MAP}}$, sofern die Schwellenwerte in 0,001er-Schritten zwischen Null und Eins variiert werden

ist, sind die Ergebnisse auf höheren Niveaus als wichtiger einzuschätzen. Diese sind allerdings für beide Verfahren so gering, dass hier noch deutliches Verbesserungspotenzial zu erkennen ist.

15.3. Überwachte Wiederherstellung von Nachverfolgbarkeitsverbindungen



Dieser Abschnitt versucht, die Leistung von Verfahren zur unüberwachten TLR in Bezug zu Verfahren zur überwachten TLR zu setzen. Für die überwachte TLR stehen neben den bereits für die unüberwachte TLR zur Verfügung stehenden Artefakten ebenso bereits korrekte Nachverfolgbarkeitsverbindungen aus dem jeweiligen Projekt zur Verfügung. Diese können dazu verwendet werden, das Verfahren auf die Zusammenhänge im jeweiligen Projekt anzupassen. Dies kann z. B. durch das Bestimmen optimaler Merkmale für das jeweilige Projekt geschehen oder bei Verfahren des maschinellen Lernens, durch ein Training der Vorhersagekomponente erfolgen. Somit arbeiten die Verfahren mit mehr konkreten, projektbezogenen Informationen und können damit bessere Entscheidungen

Tabelle 15.4.: Vergleich von **FTLRs** Leistung mit bestehenden Ansätzen zur überwachten Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext

Ansatz	eTour			iTrustJSP			SMOS			eAnci		
	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁
WQI	.088	.415	.145	.198	.322	.245						
ALCATRAL _{10%}							.513	.444	.476	.539	.507	.522
TRAIL							.871	.735	.797	.755	.805	.779
FTLR- ORG	.376	.643	.475	.136	.302	.187	.425	.314	.361	.312	.226	.262
FTLR- OPT	.505	.597	.548	.193	.224	.207	.313	.590	.409	.311	.272	.290
FTLR- ORG	.373	.633	.469	.141	.339	.199	.432	.340	.381	.273	.254	.263
FTLR- OPT	.472	.571	.517	.236	.234	.235	.380	.466	.419	.255	.300	.276

für die TLR treffen, als es unüberwachten Verfahren möglich ist. Um den Einfluss dieser Information zu verdeutlichen, soll in diesem Abschnitt der Leistungsunterschied zwischen Ansätzen zur unüberwachten und überwachten TLR bemessen werden.

Aus den bestehenden Ansätzen zur überwachten TLR wurden die drei Verfahren WQI von Zhao et al. [ZCS17], TRAIL von Mills et al. [MEH18] und ALCATRAL von Mills et al. [Mil+19] ausgewählt (s. Abschnitt 15.1). WQI nutzt ebenso wie **FTLR** Worteinbettungen zur Repräsentation und nutzt ein rangbasiertes SVM, um eine Merkmalsauswahl zu erlernen. TRAIL nutzt einen Zufallswald-Klassifikator zur binären Klassifikation, ob zwei Artefakte in einer Nachverfolgbarkeitsbeziehung stehen. Die Ergebnisse TRAILs wurden durch 50 zufällige Aufteilungsläufe mit je 90 % der Daten des Goldstandards als Training erzielt. ALCATRAL verbessert TRAIL, indem es aktives Lernen einsetzt, um damit die benötigte Menge der initialen TLs zu verringern. Für diesen Vergleich nutze ich ALCATRAL mit nur 10 % des Goldstandards als Trainingsmenge, da diese Ergebnisse denen einer unüberwachten Anwendung am nächsten kommen.

Tabelle 15.4 stellt die Ergebnisse der Anwendung von **FTLR-** und **FTLR-** denjenigen der drei Vergleichsarbeiten gegenüber. Da die bestehenden Ansätze in ihren Veröffentlichungen nur Präzision, Ausbeute und F₁-Maß berichten, bzw. im Fall von TRAIL und ALCATRAL gar keine Kandidatenlisten erstellen, auf denen MAP berechnet werden könnte, werden nur diese Metriken für den Vergleich herangezogen. Für **FTLR** wird sowohl die festgelegte Schwellenwertkombination ORG als auch die pro Projekt optimierte Schwellenwertkombination OPT dargestellt. Letztere wird ebenfalls auf Basis der Goldstandard-Verbindungen bestimmt und kommt somit einer überwachten Anwendung **FTLRs** am nächsten. Wichtig ist aber klarzustellen, dass **FTLR** keine Informationen aus den bestehenden TLs zieht und somit nicht wirklich überwacht arbeitet. Außerdem zielt **FTLR** nicht auf eine gute Leistung in einer überwachten Anwendung ab. Es ist also davon auszugehen, dass die Leistung **FTLRs** niedriger ist als die der spezialisierten Verfahren.

Auf dem eTour und dem vollständigen iTrust-Datensatz kann **FTLRs** Leistung mit der des worteinbettungsbasierten WQI-Verfahrens verglichen werden. Auf eTour erzielt **FTLR**

deutlich höhere Präzisions- und Ausbeutewerte und damit ein bis zu 40 Prozentpunkte höheres F_1 -Maß. Dies deckt sich mit der Verbesserung, die **FTLR** gegenüber dem ebenfalls einbettungsbasierten, unüberwachten S2Trace-Verfahren erzielt. Diesen Effekt schreibe ich u.a. der veränderten Nutzung der Einbettungen zu. **FTLR** nutzt ein vortrainiertes *fastText*-Modell, anstatt eines auf den Projekten selbst trainierten Modells. Zusätzlich wird die WMD anstatt der Kosinusähnlichkeit verwendet und auf Elementebene anstatt Artefakalebene abgebildet, welches sich bereits in Abschnitt 6.5 als vorteilhaft herausgestellt hat. Zusammen mit dem Ergebnis des Ansatzes von Zhang et al. [Zha+21] (s. Abschnitt 15.2) lässt sich also festhalten, dass **FTLR** auf eTour das beste Ergebnis aller worteinbettungsbasierten Verfahren erzielt, unabhängig davon, ob diese unüberwacht oder überwacht sind.




Auf iTrust erzielt WQI allerdings ein besseres Ergebnis als **FTLR**. Dies passt zu dem ebenfalls besseren Ergebnis von S2Trace auf diesem Datensatz. Auch hier ist ein möglicher Grund der schlechtere Umgang **FTLRs** mit nicht objektorientiertem Quelltext. Zusätzlich könnte es sein, dass auf iTrust das Training der Worteinbettungen auf dem Projekt selbst einen Vorteil gegenüber dem Einsatz des vortrainierten *fastText*-Modells bietet. Die Anforderungen und auch der Quelltext von iTrust enthalten viele Fachbegriffe aus dem Gesundheitswesen, welche potenziell nicht, oder nur selten, im Trainingskorpus des *fastText*-Modells auftraten und damit schlechter semantisch repräsentiert und abgebildet werden können. Die Verbesserung WQIs gegenüber **FTLR** auf iTrust ist aber deutlich geringer, als diejenige, die **FTLR** auf eTour erzielt. Im Mittel über die beiden Projekte erreicht **FTLR-🗨️** mit ORG-Schwellenwerten ein F_1 -Maß von 33,4 %, wohingegen WQI nur 19,5 % erzielt.

Auf SMOS und eAnci können die Ergebnisse auch mit den beiden klassifikationsbasierten Verfahren, TRAIL und ALCATRAL, verglichen werden. Hier lässt sich festhalten, dass diese eine deutlich bessere Leistung erzielen als **FTLR**. Insbesondere auf eAnci ist der Unterschied mit 29 % zu 52 % bzw. 77,9 % sehr deutlich. Hier scheint sich das Wissen über die bestehenden TLs besonders auszuzahlen. Eine Vermutung ist, dass diese Verfahren in der Lage sind, aus den Trainingsdaten projektspezifische Zusammenhänge abzuleiten, die das Worteinbettungsmodell nicht abbilden kann. Auf SMOS liegt das erzielte F_1 -Maß, durch den mit 10 % der Goldstandardverbindungen trainierten ALACTRAL Ansatz, nur 6 Prozentpunkte über dem von **FTLR-🗨️** mit optimierten Schwellenwerten. Dies sind vielversprechende Ergebnisse **FTLRs**, auch wenn, wie zu erwarten, der Leistungsunterschied zwischen den überwachten und unüberwachten Verfahren groß ist.

15.4. Gefährdungen der Gültigkeit

Zusätzlich zu den bereits in Abschnitt 6.6 angesprochenen Gefährdungen für die Gültigkeit ist die größte Gefahr für die in diesem Kapitel präsentierten Erkenntnisse der Vergleich der Ansätze auf Basis der in den Veröffentlichungen berichteten Ergebnisse anstatt einer Reproduktion der Ergebnisse. Für COMET konnten aus den online verfügbaren Ergebnislisten weitere konkretere Informationen extrahiert und teilweise auch Adaptionen

der berichteten Werte auf die verwendeten Artefakte durchgeführt werden. So konnte **FTLR** mit COMET auch auf dem auf Java-Dateien reduzierten iTrust-Datensatz verglichen werden. Für eTour und SMOS allerdings wurden Übersetzungen einiger der Bezeichner des Quelltextes durchgeführt, um eine konsistente Sprache zu erhalten. Die für **FTLR** berichteten Ergebnisse basieren auf diesen teilweise übersetzten Datensätzen. In diesem Vergleich wurden diese nun aber mit den berichteten Ergebnissen der bestehenden Ansätze auf den originalen (nicht übersetzten) Datensätzen verglichen. Dies stellt eine potenzielle Gefährdung der Gültigkeit der Ergebnisse des Vergleichs dar. COMET, ALCATRAL und TRAIL nutzen allerdings ausschließlich IR-basierte Techniken, welche auf die in den Projekten selbst vorhandenen Formulierungen setzen. Daher benötigen sie keine konsistente Sprache, wie es beim Einsatz von Sprachressourcen, wie vortrainierten Worteinbettungen der Fall ist. Aus diesem Grund sollten die Ergebnisse trotzdem vergleichbar sein. Ähnliches gilt für die einbettungsbasierten Verfahren, die das Einbettungsmodell auf den Projekten selbst trainieren (WQI, S2Trace und Zhang et al. [Zha+21]). Da bei einem solchen Vorgehen alle semantischen Zusammenhänge aus dem gemeinsamen Auftreten der Wörter im Projekt stammen, spielt die unterschiedliche Sprache eine höchstens kleine Rolle.

Auf dem originalen, unübersetzten SMOS-Datensatz erzielt **FTLR-** mit einem zweisprachigen *fastText*-Modell [Jou+18] ein F_1 -Maß von bis zu 32 % und ein MAP von 37,3 %, welches ebenso eine höhere Leistung darstellt, als die Vergleichsarbeiten. Allerdings nutzt dieses Modell keine Subwortinformationen, das Hauptmerkmal von *fastText*, und entspricht somit vielmehr zweier *word2vec*-Modelle, die in einen gemeinsamen Vektorraum überführt wurden. Es zeigt also nicht das volle Potenzial eines Ansatzes basierend auf *fastText*-Einbettungen. Wendet man hingegen das italienische Modell auf den originalen unübersetzten Datensatz an, so erzielt **FTLR-** ebenso ein höheres F_1 -Maß von 32,7 % und MAP von 36,9 % als bestehende unüberwachte Ansätze. Die niedrigeren Werte deuten aber darauf hin, dass gemischte Sprache durchaus einen Effekt auf die Güte der Abbildung durch vortrainierte Worteinbettungsmodelle hat. Da heutige Verfahren zur Identifikation der vorliegenden Sprache Genauigkeiten von über 93 % erzielen [Gra+18] und maschinelle Übersetzung vielversprechende Ergebnisse erreicht, bin ich zuversichtlich, dass die durchgeführte Übersetzung einiger weniger Bezeichner auch automatisiert erfolgen kann. Denn selbst auf dem vollständig maschinell aus dem Italienischen ins Englische übersetzten SMOS-Datensatz erzielte **FTLR-** noch ein F_1 -Maß von bis zu 39,4 % und MAP von 41,2 %.

15.5. Zusammenfassung des Vergleichs

In diesem Kapitel wurde der in dieser Arbeit vorgestellte Ansatz zur unüberwachten TLR (**FTLR**) mit bestehenden Ansätzen verglichen. Dies dient der Bewertung von These T_1 , welche eine Leistungssteigerung des Verfahrens gegenüber unüberwachten Ansätzen annimmt. Hierzu wurden zunächst Arbeiten bestimmt, die sich für einen direkten Vergleich eignen sowie dieselben Vergleichdatensätze in ihrer Evaluation verwenden wie **FTLR**. In einer Analyse der Evaluationen der bestehenden Ansätze aus Abschnitt 3.1 wurde

festgestellt, dass viele der Veröffentlichungen wichtige Informationen für einen direkten Vergleich vermissen lassen. Die wenigsten stellen ihre Ergebnisse oder ihr Verfahren als Zusatzmaterial zur Verfügung, weshalb nur mit den Ergebnissen aus den Veröffentlichungen gearbeitet werden konnte. Diese enthalten aber oftmals keine konkreten Zahlen, welche für einen Vergleich notwendig gewesen wären. Ein weiterer Punkt, welcher den Vergleich einschränkt, sind die unterschiedlichen Versionen der Vergleichsdatensätze. Viele Arbeiten nutzen den eTour- oder den iTrust-Datensatz zur Evaluation, aber einige in anderen Versionen, die nicht mehr auffindbar waren. Durch diese Gegebenheiten wurde die Menge der für den Vergleich zur Verfügung stehenden bestehenden Ansätze leider stark beschränkt.

Im Vergleich mit den unüberwachten Ansätzen von Panichella et al. [Pan+13], S2Trace von Chen et al. [Che+19], COMET von Moran et al. [Mor+20b] und Zhang et al. [Zha+21] konnten teils deutliche Verbesserungen durch **FTLR** festgestellt werden. Auf SMOS erzielt **FTLR** ein um 9 Prozentpunkte höheren MAP als der Ansatz von Panichella et al. (45,1 % gegenüber 36,1 %). Im Durchschnitt über eTour und iTrust erzielt **FTLR** mit einem F_1 -Maß von 33,4 % ein um 12 Prozentpunkte höheren Wert als das ebenfalls einbettungs-basierte S2Trace (F_1 -Maß von 21,3 %). Gegenüber dem ebenfalls worteinbettungs-basierte Verfahren von Zhang et al. erzielt **FTLR** im maximal möglichen F_1 -Maß (mit optimierten Schwellenwerten OPT) auf dem eTour-Datensatz eine Verbesserung von ca. 3,5 Prozentpunkten.

Durch das Bereitstellen der vollständigen Ergebnislisten konnte der Vergleich mit COMET am ausführlichsten durchgeführt werden. Im Bezug auf MAP erzielt **FTLR** im Durchschnitt über die vier Projekte eTour, SMOS, iTrust und LibEST eine Verbesserung von bis zu 2,7 Prozentpunkten gegenüber COMET. Lediglich auf iTrust und LibEST, welche nicht objektorientierten Quelltext beinhalten, schneidet **FTLR** im F_1 -Maß schlechter ab als COMET. Da **FTLR** allerdings auf objektorientierten Quelltext abzielt, war dieser Effekt zu erwarten. Lässt man den LibEST-Datensatz, welcher vollständig aus Quelltext in der Sprache C besteht, aus der Durchschnittsbildung heraus, so erzielt **FTLR** auch im durchschnittlichen F_1 -Maß über die Projekte eine Verbesserung von bis zu 2,8 Prozentpunkten gegenüber COMET. Eine Betrachtung der Präzision-Ausbeute-Kurven **FTLRs** und **COMETs** zeigt außerdem, dass **FTLR** auf höheren Ausbeuteniveaus bessere Präzisionen erzielt als COMET.

Diese Erkenntnisse bestätigen These T_1 . Eine allgemeine Gültigkeit kann aber aufgrund der kleinen Stichprobe und eingeschränkten Auswahl an Vergleichsansätzen nur schwer gefolgert werden. Da COMET allerdings eine Kombination der gängigsten IR-basierten Verfahren in der automatischen TLR darstellt, kann aufgrund der Ergebnisse zumindest angenommen werden, dass **FTLR** den Verfahren VSM, LSI, JS, LDA und Nichtnegativer Matrixfaktorisierung (NMF) überlegen ist.




Zusätzlich zum Vergleich mit unüberwachten Verfahren wurden die Ergebnisse **FTLRs** auch mit überwachten Verfahren verglichen, um den Leistungsunterschied zwischen diesen beiden Aufgaben der TLR zu verdeutlichen. Hierbei zeigte sich, dass das worteinbettungs-basierte WQI-Verfahren [ZCS17], welches ein trainiertes SVM zur Merkmalsauswahl verwendet, auf eTour sogar schlechtere Ergebnisse erzielt als **FTLR**. Die beiden Verfahren

TRAIL und ALCATRAL von Mills et al. hingegen zeigten die zu erwartenden Leistungssteigerungen durch Ausnutzen von vorhandenen Nachverfolgbarkeitsverbindungen im Training. Sie erzielten Verbesserungen von bis zu 50 Prozentpunkten auf eAnci. TRAIL nutzt hierzu aber auch 90 % der TLs zum Training.

16. Zusammenfassung und Ausblick

In dieser Arbeit wurde das Verfahren **FTLR** zur unüberwachten Wiederherstellung von Nachverfolgbarkeitsverbindungen (TLR) zwischen Anforderungen und Quelltext entworfen und um verschiedene (optionale) Erweiterungen ergänzt. Die unüberwachte TLR zielt darauf ab, Nachverfolgbarkeitsinformationen möglichst vielen Projekten in der Praxis verfügbar zu machen. Denn diese Informationen können Entwickler:innen und weiteren relevanten Akteur:innen den Überblick über ein Projekt erleichtern und ihnen die Zusammenhänge zwischen den Artefakten verdeutlichen. Außerdem können die Nachverfolgbarkeitsverbindungen dazu verwendet werden, nachgelagerte Analysen wie die Änderungsauswirkungsanalyse oder die Software-Wiederverwendbarkeitsanalyse zu verbessern oder eine Automatisierung dieser Aufgaben erst zu ermöglichen. Um in möglichst vielen Projektkontexten einsetzbar zu sein, werden in der unüberwachten TLR nur die Artefakte selbst, also die Anforderungen und der Quelltext, als Eingaben verwendet. Auf Basis der Semantik der Artefakte muss ein automatisiertes Verfahren also in der Lage sein, die für die Nachverfolgbarkeit relevanten Zusammenhänge zu identifizieren.







FTLR setzt hierzu auf eine Kombination von feingranularen, worteinbettungsbasierten Ähnlichkeiten, welche über einen Mehrheitsentscheid zu Nachverfolgbarkeitsverbindungen auf Artefaktebene aggregiert werden. Zur Repräsentation wird ein vortrainiertes *fastText*-Worteinbettungsmodell verwendet, welches die Anforderungs- und Quelltextelemente durch die Menge ihrer Wörter repräsentiert. Als Elemente werden auf Anforderungsseite Sätze bzw. falls vorhanden ganze Vorlagenelemente und auf Quelltextseite die öffentlichen Methoden aufgefasst. Diese Elemente werden daraufhin mittels der Wortüberführungsdistanz (engl.: *Word Movers Distance*, WMD) auf ihre Ähnlichkeit untersucht. Durch einen Mehrheitsentscheid pro Quelltextartefakt werden abschließend die vorherrschenden Zusammenhänge jeder Klasse bestimmt und als Kandidaten für eine Nachverfolgbarkeitsverbindung aufgefasst.

Dieses grundlegende Verfahren wurde im Verlauf der Arbeit zudem um weitere Schritte ergänzt. Auf Quelltextseite können zum einen neben der Methodensignatur auch vorhandene Methodenkommentare in die Repräsentation einbezogen werden (**FTLR**-Variante ). Zum anderen können die Zusammenhänge der aufgerufenen und aufrufenden Methoden dazu genutzt werden, die Abbildung der gerade betrachteten Methode zu verbessern (Variante ). Auf der Anforderungsseite ist es entscheidend zu ermitteln, welche Teile der Anforderungen tatsächlich für eine Abbildung auf Quelltext relevant sind. Hierzu wurden in Anwendungsfallbeschreibungen, die einer Vorlage folgen, Vorlagenelemente identifiziert, die keine oder wenig Relevanz für die TLR haben. Diese Vorlagenelemente können vor der Verarbeitung herausgefiltert werden (**FTLR**-Variante ). Da Anforderungen nicht notwendigerweise solchen Vorlagen folgen, ein Filtern aber auch in anderen Projekten hilfreich sein

kann, wurde außerdem ein Verfahren zur Klassifikation von Anforderungen entwickelt (Beitrag 2: **NoRBERT**). Dieses identifiziert irrelevante Anforderungselemente auf Basis einer Klassifikation mit einem *BERT*-basierten Transformer-Sprachmodell. Enthält das Anforderungselement keine funktionalen Aspekte oder ausschließlich Nutzerbezogenes, so werden diese Elemente aus der weiteren Betrachtung für die TLR ausgeschlossen.

Zusätzlich wurde untersucht, in welcher Form ein bimodales, kontextsensitives Sprachmodell anstatt der statischen *fastText*-Worteinbettungen für **FTLR** genutzt werden kann (Beitrag 3). Hierbei wurden zwei unterschiedliche Integrationsmöglichkeiten identifiziert. Zum einen das Ermitteln von Wortvektoren und eine Repräsentation durch Einbettungsmultimenge, wie sie auch mit *fastText* verwendet werden. Zum anderen eine Repräsentation eines Artefaktelements durch einen einzelnen Eingabeeinbettungsvektor und einer Bestimmung der Kosinusähnlichkeit anstatt der WMD, als Ähnlichkeitsmaß. Als geeignetstes bimodales Sprachmodell wurde hierbei *UniXcoder* ausgewählt.

Die getroffenen Entwurfsentscheidungen und ihre Auswirkungen auf den Einsatz in der unüberwachten TLR wurden in drei Evaluationen untersucht. Für diese Evaluationen wurden die Vergleichsdatensätze eTour, iTrust, SMOS, eAnci, Albergate und LibEST verwendet. Diese umfassen natürlichsprachliche Anforderungen zum Teil in Form von Anwendungsfallbeschreibungen sowie bis auf LibEST objektorientierten Quelltext¹.

In der ersten Evaluation wurde das grundlegende Verfahren und die drei Varianten ,  und  sowie ihre Kombinationen untersucht. Das Ziel war es, zum einen zu bestimmen, ob sich das Nutzen struktureller Informationen des Quelltextes und der Anforderungen positiv auf die Leistung auswirkt. Zum anderen sollten die getroffenen Entwurfsentscheidungen für den verwendeten Ähnlichkeitsvergleich und für die feingranulare Abbildung auf ihren Einfluss auf die TLR untersucht werden. Es konnte gezeigt werden, dass das Hinzunehmen von Methodenkommentaren () und das Filtern von Vorlagenelementen () einen positiven Einfluss auf die Leistung von **FTLR** hat. Auch ein Hinzunehmen von Aufrufabhängigkeiten () kann in Kombination mit den anderen beiden Erweiterungen einen positiven Effekt haben. Die beiden untersuchten Entwurfsentscheidungen, das Verwenden der WMD und das Abbilden auf Elementebene übertrafen ebenso die Leistung von Referenzimplementierungen mit anderer Entscheidung.

Für eine unüberwachte Anwendung von **FTLR** auf ungesehene Projekte ist es außerdem wichtig, dass sich eine festgelegte Schwellenwertkombination bestimmen lässt, die eine gute Leistung erzielt. Hierzu wurden die Ergebnisse **FTLRs** mit den zwei festgelegten Schwellenwertkombinationen ORG und MED mit der Leistung auf pro Projekt optimierten Schwellenwerten (OPT) verglichen. Es konnte gezeigt werden, dass die festgelegten Schwellenwertkombinationen im Durchschnitt über die Projekte nur eine um 5 Prozentpunkte schlechtere Leistung (F_1 -Maß von 34,7 %) erbringen, als die pro Projekt optimalen Schwellenwerte (F_1 -Maß von 39,7 %).

Für die zweite Evaluation wurden die Anforderungselemente der Vergleichsdatensätze hinsichtlich des Auftretens von funktionalen Aspekten, Nutzerbezogenem sowie der funk-

¹ LibEST ist in der Programmiersprache C verfasst.

tionalen Aspekte Funktion, Verhalten und Daten untersucht. Der so entstandene Datensatz wurde mit dem PROMISE NFR-Datensatz kombiniert und als Trainingsgrundlage für das Klassifikationsverfahren **NoRBERT** genutzt. Es konnte gezeigt werden, dass **NoRBERT** bei der Erkennung von funktionalen Aspekten ein F_1 -Maß von 89,8 % und bei der Identifikation von Nutzerbezogenem ein F_1 -Maß von 81,8 % auf den Vergleichsdatsätzen erzielen konnte. Bei der Klassifikation der funktionalen Anliegen konnten Genauigkeiten von 70,6 % für die Klasse Funktion, 75,8 % für die Klasse Verhalten und 71,4 % für die Klasse Daten erzielt werden. Ziel dieser Evaluation war es, zu bestimmen, ob sich ein automatisches Klassifikationsverfahren für Anforderungselemente dafür eignet, irrelevante Elemente aus der Verarbeitung **FTLRs** auszuschließen. Daher wurden die Ergebnisse der automatischen Klassifikation dazu genutzt, Anforderungselemente in **FTLR** zu filtern. Die Ergebnisse zeigen, dass die automatische Klassifikation ähnlich gute und teils sogar bessere Ergebnisse erzielen kann wie das Filtern von Vorlagenelementen (**▼**), diese aber unabhängig von dem Vorliegen von Anwendungsfallbeschreibungsvorlagen ist.

In der dritten Evaluation wurde untersucht, ob sich der Einsatz eines bimodalen, kontextsensitiven Sprachmodells anstatt *fastText* für die Leistung **FTLRs** auszahlt. Hierfür wurde zunächst die Integrationsmöglichkeit des Modells mit der besten durchschnittlichen Leistung auf den Datensätzen eTour, iTrust, SMOS, eAnci und LibEST bestimmt. Hier konnte festgestellt werden, dass die *UniXcoder*-Variante, die einen einzelnen Eingabeeinbettungsvektor mit Kosinusähnlichkeit verwendet, bessere Ergebnisse im F_1 -Maß und MAP erzielt als die WMD-basierte Variante. Im Vergleich zu **FTLR** mit *fastText* konnte im Durchschnitt über die Projekte allerdings keine der *UniXcoder*-basierten Varianten eine Verbesserung erzielen. Lediglich auf dem iTrust-Datensatz konnte ein Einsatz *UniXcoders* mit einer reduzierten Vorverarbeitung eine deutliche Verbesserung von 5 Prozentpunkten erreichen. Diese Konfiguration erzielt aber auf den anderen Datensätze teils ebenso deutlich schlechtere Ergebnisse als **FTLR** mit *fastText*. Da außerdem durch stark schwankende optimale Schwellenwerte keine feste Schwellenwertkombination ermittelt werden konnte, die bessere Ergebnisse als **FTLR** mit *fastText* erzielt, ist der Einsatz von *fastText*-Worteinbettungen dem von *UniXcoder* in **FTLR** vorzuziehen.

Abschließend wurden die Ergebnisse **FTLRs** mit bestehenden Ansätzen zur unüberwachten TLR verglichen. Hier zeigte sich, dass **FTLR** im Durchschnitt über die Datensätzen eTour, iTrust und SMOS ein besseres F_1 -Maß und MAP erzielt als bestehende Ansätze. Lediglich auf nicht objektorientiertem Quelltext, wie im LibEST-Projekt und dem JSP-Teil des iTrust-Datensatzes, schneidet **FTLR** schlechter ab als Vergleichsansätze. Es zeigt sich aber auch, dass keiner der Ansätze, auch nicht **FTLR**, die nötige Güte erreicht, die für einen vollautomatischen Einsatz in der Praxis notwendig wäre, da alle Ansätze noch zu unpräzise arbeiten. Daher kann diese Arbeit nur als Zwischenschritt auf dem Weg zu diesem so wünschenswerten Ziel angesehen werden.

16.1. Diskussion der Thesen

In Abschnitt 1.3 dieser Arbeit wurden verschiedene Thesen aufgestellt, anhand derer die Leistung des in dieser Arbeit entwickelten Verfahrens und seiner Erweiterungen in Bezug zueinander und zu bestehenden Arbeiten gesetzt werden können. Sie erlauben also eine detailliertere Beurteilung **FTLRs** hinsichtlich der Erfüllung der Zielstellung, eine automatische unüberwachte Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext zu ermöglichen. Im Folgenden werden die einzelnen Thesen diskutiert und anhand der in den Evaluationen gewonnenen Erkenntnisse bewertet.

T₁: Eine Wiederherstellung von Nachverfolgbarkeitsverbindungen zwischen Anforderungen und Quelltext mittels feingranularer, worteinbettungsbasierter Beziehungen steigert die Leistung gegenüber bestehenden unüberwachten Verfahren.

Die Ergebnisse in Kapitel 15 haben gezeigt, dass eine TLR mittels feingranularer, worteinbettungsbasierter Beziehungen, wie sie **FTLR** durchführt, ein besseres F_1 -Maß und MAP als bestehende Ansätze zur unüberwachten TLR erzielen kann. Insbesondere gegenüber den bestehenden Ansätzen, die ebenfalls Einbettungen verwenden, konnten teils deutliche Verbesserungen durch **FTLR** festgestellt werden. Es zeigte sich aber auch, dass diese Erkenntnis nur für Projekte mit objektorientiertem Quelltext gilt. Da die Entwicklung **FTLRs** allerdings auf objektorientierten Quelltext abzielte und das Verfahren Eigenschaften der Objektorientierung ausnutzt, war dies ein zu erwartender Effekt. Auch wenn aufgrund der schwierigen Datenlage nur wenige bestehende Arbeiten und Datensätze zum Vergleich herangezogen werden konnten, ist von einer gewissen Aussagekraft dieser Erkenntnisse auszugehen. Die zum Vergleich verwendeten Ansätze gehören zu den aktuellsten Verfahren und die Vergleichsdatsätze sind die am häufigsten verwendeten Datensätze in der Forschungsgemeinschaft zur TLR. Auf bestimmten Datensätzen zeigten allerdings alle bestehenden Verfahren inklusive **FTLR** schlechte Ergebnisse in der Abbildungsgüte. Ein maximales F_1 -Maß von 26,7 % auf dem iTrust-Datensatz liegt sogar unter einem für einen semiautomatischen Einsatz in der Praxis als gut erachteten Wert von mindestens 42 % [HDS06]. Somit kann zwar eine Verbesserung durch ein feingranulares, worteinbettungsbasiertes Vorgehen erzielt werden, eine vollständige Automatisierung der Wiederherstellung von Nachverfolgbarkeitsverbindungen in der Praxis muss jedoch kommenden Arbeiten überlassen werden.

T_{1.1}: Der Einbezug struktureller Informationen des Quelltexts und der Anforderungen kann die Leistung eines worteinbettungsbasierten Verfahrens steigern.

In der Evaluation zum Einfluss der Varianten **FTLRs** (s. Abschnitt 6.3) konnte gezeigt werden, dass das Hinzunehmen von Methodenkommentaren und das Filtern von Vorlagenelementen einen positiven Einfluss auf die Leistung **FTLRs** hat. Kombiniert man diese Erweiterungen zusätzlich mit dem Einbeziehen von Aufrufabhängigkeiten, kann eine weitere Verbesserung festgestellt werden. So

konnte eine signifikante Verbesserung, sowohl im F_1 -Maß als auch im MAP, durch **FTLR** mit Einsatz der Kombinationen $\text{FTLR} \rightarrow \text{FTLR}$ und $\text{FTLR} \rightarrow \text{FTLR}$ gegenüber einer einfachen Anwendung **FTLRs** ohne Erweiterungen festgestellt werden. Da es sich bei den Aufrufabhängigkeiten um strukturelle Informationen des Quelltextes und bei den Vorlagenelementen um strukturelle Informationen der Anforderungen handelt, kann diese These auf den untersuchten Datensätzen bestätigt werden.

T_{1.2}: Der Einsatz der Wortüberführungsdistanz zur wortweisen Abbildung der Artefaktelemente verbessert die Abbildungsgüte gegenüber einfachen, aggregierten Vektorabbildungen.

In Abschnitt 6.5 wurde die Leistung **FTLRs** mit einer Referenzimplementierungen verglichen, die anstatt der WMD auf Basis von Einbettungs-Multimengen, die Kosinusähnlichkeit auf einfachen, aggregierten Vektorrepräsentationen einsetzt. Es konnte dabei eine signifikante Verbesserung der Abbildungsgüte durch **FTLR** gegenüber dieser Referenzimplementierung festgestellt werden. Somit kann auch diese These auf den untersuchten Datensätzen bestätigt werden.

T_{1.3}: Ein Aggregieren feingranularer Ähnlichkeiten zur Bestimmung von Nachverfolgbarkeitsverbindungen auf Artefaktebene ermöglicht eine höhere Leistung bei der Abbildung im Vergleich zur direkten Abbildung der gesamten Artefakte.

Ebenso in Abschnitt 6.5 wurde die Leistung **FTLRs** mit einer Referenzimplementierungen verglichen, die anstatt einer Abbildung auf Elementebene, wie sie **FTLR** durchführt, auf Artefaktebene abbildet. Auch hier konnte eine signifikante Verbesserung im F_1 -Maß durch **FTLR** im Gegensatz zur Referenzimplementierung beobachtet werden. Dieses Erkenntnis war sogar unabhängig davon, ob die WMD oder die Kosinusähnlichkeit verwendet wurde. Somit ist hier anzunehmen, dass der Leistungszuwachs an der feingranularen Abbildung mit Mehrheitsentscheid als Aggregationsmethode liegt.

T₂: Durch geeignete Klassifikation der Anforderungen können irrelevante Teile der Eingabe ausgeschlossen werden und damit die Präzision eines wort-einbettungsbasierten Verfahrens zur Nachverfolgbarkeit gesteigert werden.

Die Evaluation des automatischen Klassifikationsverfahrens als Vorfilter für **FTLR** (s. Kapitel 10) zeigt, dass ein automatisches Filtern von Elementen, die keine funktionalen Aspekte enthalten, zu einer signifikanten Verbesserung gegenüber Varianten, die keinen Filter nutzen, führen kann. Insbesondere konnte eine verbesserte Präzision festgestellt werden, die dafür sorgt, dass ein höheres F_1 -Maß erzielt wird. Ein Filtern von Nutzerbezogenem ergab hingegen keine signifikante Verbesserung, weder in alleiniger Anwendung noch in Kombination mit dem anderen Filter. In einer Betrachtung der maximal möglichen Leistung, durch den Einsatz optimierter Schwellenwerte, konnte außerdem das beste F_1 -Maß aller **FTLR**-Varianten auf den Datensätzen eTour, iTrust, SMOS, eAnci und LibEST mit 40,7 % erzielt werden. Diese

Ergebnisse zeigen, dass eine Klassifikation von Anforderungen zu einer erhöhten Präzision und damit einer verbesserten Abbildungsleistung **FTLRs** führen kann.

T₃: Der Einsatz eines bimodalen, kontextsensitiven Sprachmodells für die Repräsentation der Artefaktelemente verbessert die Leistung eines feingranularen Verfahrens zur Nachverfolgbarkeit.

Die Untersuchung der Ergebnisse **FTLRs** mit dem bimodalen, kontextsensitiven Sprachmodell *UniXcoder* (s. Kapitel 13) ergaben, dass sich dessen Einsatz nicht positiv auf die Leistung auswirkt. Weder eine signifikante noch eine einfache Verbesserung konnte im Durchschnitt über die Projekte beobachtet werden. Da *UniXcoder* von den bestehenden bimodalen, kontextsensitiven Sprachmodellen aufgrund seiner Leistung als vielversprechendstes Modell identifiziert wurde, ist diese These als falsch anzunehmen. Zusätzlich sind die Hardwareanforderungen des *UniXcoder*-Modells deutlich höher als die des *fastText*-Modells und eine feste Schwellenwertkonfiguration für die unüberwachte Anwendung ist aufgrund der Varianz der Schwellenwerte über die Projekte schwer zu ermitteln. Diese Erkenntnisse sorgen dafür, dass die Entwurfsentscheidung in **FTLR** *fastText*-Worteinbettungen zu nutzen, derjenigen ein bimodales, kontextsensitives Sprachmodell zu verwenden, vorzuziehen ist.

16.2. Weiterführende Themen und Ausblick

Das vorangegangene Fazit und die Diskussion der Thesen haben gezeigt, dass **FTLR** und seine Erweiterungen in der Lage sind, bestehende Ansätze zur unüberwachten TLR in der Abbildungsgüte zu übertreffen. Doch die erzielte Abbildungsgüte zeigte auch, dass zum Erreichen des Ziels, Nachverfolgbarkeitsinformationen in möglichst vielen Projekten verfügbar zu machen, weitere Schritte nötig sind. Zu diesen weiteren Schritten zählen zum einen Möglichkeiten der Optimierung und Erweiterung **FTLRs** (s. Abschnitt 16.2.1), zum anderen aber auch Möglichkeiten zur Weiterentwicklung von Ansätzen zur unüberwachten TLR (s. Abschnitt 16.2.2), die über die Ansätze dieser Arbeit hinausgehen.

16.2.1. Optimierungen und Erweiterungen

Um die Abbildungsgüte **FTLRs** weiter zu verbessern, könnten Optimierungen in allen drei durchgeführten Schritten angesetzt werden. Die Güte der semantischen Repräsentation der Artefakte hängt stark von dem verwendeten Worteinbettungsmodell ab. **FTLR** würde also von einem verbesserten Worteinbettungsmodell profitieren. Anstatt des auf allgemeinen Textdaten vortrainierten *fastText*-Modells wäre ein auf die Domäne des Projektes angepasstes Modell ein Kandidat für eine solche Optimierung. Zum einen könnte das bestehende Modell auf dem Projekt selbst weitertrainiert werden, zum anderen könnten beispielsweise über eine Websuche der in den Artefakten enthaltenen Fachbegriffe, weitere

relevante Dokumente gefunden und diese ebenfalls in das Training des Modells miteinbezogen werden. Ein solches Vorgehen nutzten beispielsweise Wang et al. [WWF20] zur Verbesserung von wissensbasierter Disambiguierung von Wortbedeutungen.

Neben der besseren Anpassung des verwendeten Sprachmodells auf die Domäne könnte die Repräsentation der Artefaktelemente ebenfalls von einer verbesserten Vorverarbeitung profitieren. Hierbei ist das Ziel, die Menge der Worte, die in der Einbettungs-Multimengen eines Elements enthalten sind, möglichst auf die wesentlichen, semantiktragenden Begriffe zu reduzieren. Eine Möglichkeit wäre der Einsatz von domänenspezifischen Stoppwortlisten. So kann es im Programmierkontext vorteilhaft sein, den Begriff *test* herauszufiltern, wohingegen er im Gesundheitswesen eine größere Bedeutung trägt. Des Weiteren könnte ein kontextabhängiges Auflösen von Abkürzungen dafür sorgen, dass auch aus abgekürzten Bezeichnern ihre Semantik extrahiert und damit eine bessere Abbildung auf die anderen Begrifflichkeiten erreicht werden kann. Da Abkürzungen in Quelltext durchaus häufig auftreten [New+19] bietet dies großes Potenzial, sofern automatisierte Verfahren zur Abkürzungsauflösung eine ausreichende Leistung erzielen. Wie die verbesserten Werte auf SMOS (vgl. Abschnitt 15.4) und die Arbeit von Lin et al. [LLC21] zeigen, profitiert ein worteinbettungsbasiertes Verfahren, wie **FTLR**, von monolingualen Eingaben. Somit stellt eine automatische Übersetzung von multilingualen Artefakten in die vorherrschende Sprache eine weitere mögliche Optimierung dar.

Um eine bessere Grundlage für das Überbrücken der semantischen Lücke (s. Abschnitt 4.2) zwischen den Artefakten zu schaffen, könnte auch eine Wissensanreicherung der Artefakte vorteilhaft sein. Hierzu könnten mittels wissensbasierter Disambiguierung von Wortbedeutungen die Bedeutungen der Wörter in den Anforderungen [HKT21] und im Quelltext in Form ihrer Entsprechungen in der Wikipedia bestimmt werden (s. Abschnitt 2.8.8). Mittels dieser Einstiegspunkte können durch wissensbasierte Themenetikettierung, wie z. B. das Verfahren von Weigelt et al. [Wei+19], gemeinsame Überkonzepte für die Artefaktelemente bestimmt und der jeweiligen BoE-Repräsentation hinzugefügt werden. Hintergedanke eines solchen Vorgehens ist es, dass die gemeinsamen Überkonzepte die Semantik der Elemente auf einer höheren Abstraktionsebene beschreiben und damit potenziell die semantische Lücke verkleinern.

Auch die Ähnlichkeitsbestimmung bietet noch weiteres Optimierungspotenzial. Die Ergebnisse des Vergleichs mit den Referenzimplementierungen (s. Abschnitt 6.5) haben gezeigt, dass auf manchen Projekten durchaus ein Bestimmen der Ähnlichkeit auf Basis der Kosinusähnlichkeit von Vorteil sein kann. Es könnte sich demnach auszahlen zu untersuchen, wann genau dies der Fall ist und wie die Projekte hierfür geartet sein müssen. Wenn sich diese Information aus den Projektmetadaten, wie z. B. Größe und Anzahl der Artefakte, extrahieren ließen, könnte **FTLR** für diese Projekte die Kosinusähnlichkeit anstatt der WMD einsetzen. Eine weitere Möglichkeit wäre es, beide Ähnlichkeitsbestimmungen zu kombinieren und damit potenziell die Abbildungsgüte zu erhöhen. Hierbei wäre zu untersuchen, wie stark eine Gewichtung der jeweiligen Verfahren erfolgen sollte und ob auch diese von den Projektgegebenheiten abhängt. Zusätzlich könnten die in Abschnitt 8.1 identifizierten funktionalen Anliegen, Funktion, Verhalten und Daten, dazu genutzt werden, die Abbildung zwischen den Artefakten zu verbessern. Beispielsweise könnten als Daten

klassifizierte Anforderungselemente mit einer höheren Wahrscheinlichkeit auf Quelltextelemente abgebildet werden, die Datenobjekte bereitstellen. Würde man zusätzlich auch die Quelltextelemente hinsichtlich ihrer Anliegen klassifizieren, könnte die Abbildung auf geteilte Anliegen konzentriert werden und damit irrelevante Abbildungskandidaten ausgeschlossen werden. Dies könnte die Präzision des Verfahrens erhöhen.

Die durch **FTLR** durchgeführte Aggregation der feingranularen Ähnlichkeiten auf Artefaktebene hängt von den verwendeten Schwellenwerten und deren Eignung für das jeweilige Projekt ab. Auch wenn die Ergebnisse mit den festgelegten Schwellenwerten vielversprechend sind, können je nach Projekt Verbesserungen von bis zu 10 Prozentpunkten durch optimierte Schwellenwerte erzielt werden². Für eine weitere Anwendung könnte diejenige Schwellenwertkombination bestimmt werden, die über alle in dieser Arbeit betrachteten Datensätze hinweg die beste durchschnittliche Leistung erzielt. Diese Schwellenwertkombination sollte damit auch auf neuen Projekten vielversprechende Ergebnisse liefern. Außerdem könnte untersucht werden, ob sich die optimalen Schwellenwerte eines Projektes aus dessen Metadaten ableiten lassen. Hierbei wäre es interessant zu überprüfen, ob die Größe und Anzahl der Artefakte oder die Kopplung und Kohäsion des Quelltextes einen Einfluss auf die optimalen Schwellenwerte hat. Sollte dies der Fall sein, könnte **FTLR** die festgelegten Schwellenwertkombinationen auf Basis der Metadaten automatisch auf neue Projekte anpassen.

FTLR betrachtet nur die aufgrund des Mehrheitsentscheids meistgewählten Anforderungen je Quelltextartefakt als Nachverfolgbarkeitsverbindungskandidaten. Diese Entscheidung wurde bewusst mit Fokus auf Präzision getroffen, verhindert aber gerade in Projekten mit vielen Anforderungen und nur wenig Klassen eine bessere Leistung des Verfahrens. Eine mögliche Optimierung wäre es, diese Entscheidung hinsichtlich einer Top-N-Strategie aufzuweichen. Hierbei würden anstatt der meistgewählten die N meistgewählten Anforderungen als Kandidaten vorgeschlagen. Die Schwierigkeit bei einem solchen Vorgehen liegt allerdings in der Bestimmung von N und ob dieses aus den Projektmetadaten abgeleitet werden kann. Ein zu großes N könnte sich negativ auf die Präzision auswirken, ein zu kleines hingegen würde die Ausbeute beschränken.

16.2.2. Möglichkeiten zur Weiterentwicklung

Neben der Optimierung und Erweiterung **FTLRs** haben die Erkenntnisse dieser Arbeit weitere mögliche Herangehensweisen an das Problem der unüberwachten TLR zwischen Anforderungen und Quelltext aufgeworfen. Die Analyse der verwandten Arbeiten in Abschnitt 3.1.1.2 ergab, dass eine Kombination verschiedener IR-basierter Abbildungsverfahren die Abbildungsgüte verbessern kann. Bisher wurden hierbei aber keine worteinbettungsbasierten Verfahren miteinbezogen. Daher würde sich die Untersuchung eines kombinierenden Verfahrens anbieten, welches z. B. **FTLR** mit LDA- und VSM-basierten

² Auf dem Albergate-Datensatz, welcher als Ausreißer ausgeschlossen wurde, sind sogar Verbesserungen von 21 Prozentpunkten beobachtbar.

Verfahren kombiniert. Ein Vorteil eines solchen kombinierenden Verfahrens ist das Ausgleichen der Schwächen bestimmter Ansätze auf manchen Projekten. So könnte z. B. **FTLRs** schwächere Leistung auf nicht objektorientiertem Quelltext durch eines der anderen Verfahren ausgeglichen werden. Die Schwierigkeit stellt allerdings die Art der Kombination dar. Hierbei muss bestimmt werden, in welchen Fällen welches Verfahren höher gewichtet werden sollte. Der Ansatz von Moran et al. [Mor+20b], ein HBN hierfür zu verwenden, wäre aufgrund der guten Ergebnisse ein erster Startpunkt.

Neben der Kombination der Verfahren bieten eventuell auch die kontextsensitiven Sprachmodelle noch Verbesserungspotenzial. Zum einen gilt natürlich auch hierbei, dass ein besser passendes Modell eventuell bessere Ergebnisse erzielen kann als **FTLR** mit *fast-Text*. Es könnten beispielsweise autoregressive Sprachmodelle, wie z. B. *GPT-3* [Bro+20], *GPT-4* [Ope23] oder *Codex* [Che+21], auf ihre Eignung untersucht werden. Zum anderen können diese kontextsensitiven Sprachmodelle anders als in Abschnitt 12.1 beschrieben für die unüberwachte TLR zwischen Anforderungen und Quelltext eingesetzt werden. Eine Möglichkeit wäre es, die Fähigkeit dieser Sprachmodelle zum Transferlernen auszunutzen. Die Modelle wurden bereits auf der verwandten Aufgabe der Quelltextsuche eingesetzt, indem eine Klassifikationsschicht bestimmt, ob eine natürlichsprachliche Beschreibung zu einer Methode passt. Die Idee wäre es, diese bereits trainierte Klassifikationsschicht zu nehmen und mit TLR-Datensätzen feinanzupassen. Die TLR zwischen Anforderungen und Quelltext würde also als Klassifikationsaufgabe aufgefasst, wie es auch bereits für Belang zu VCS-Einbuchungen durchgeführt wurde (s. Abschnitt 3.4). Um hier allerdings keine überwachte TLR durchzuführen, könnten die Modelle auf einem Teil der Projekte weitertrainiert und ihre Leistung auf den anderen, ungesehenen Projekten bemessen werden. Sollte ein solches Vorgehen Verbesserungen in der Abbildungsgüte erzielen, könnte ein solches Modell auf allen verfügbaren Vergleichsdatsätzen der TLR feinangepasst werden, um in der Praxis eingesetzt zu werden.

16.3. Abschließende Bemerkungen

Nachverfolgbarkeitsinformation in allen Projekten verfügbar zu machen, ist ein ambitioniertes Ziel, dessen Erfüllung dafür sorgen würde, dass die Verwaltung und Wartung von Softwareprojekten erleichtert wird. Software-Nachverfolgbarkeit sorgt dafür, dass das oftmals verloren gegangene oder nicht explizit festgehaltene Wissen über die Zusammenhänge zwischen den verschiedenen Artefakten des Softwareentwicklungsprozesses transparent und verfügbar gemacht wird. Sie trägt also zur Verbesserung der Kommunikation der Projektbeteiligten, zur Einhaltung von Vorschriften und Richtlinien, welche diese Information explizit fordern, sowie zur vollständigen Erfüllung und Umsetzung der Anforderungen und Entwurfsentscheidungen bei. Um dieses Ziel in der Praxis zu erreichen, bedarf es eines automatischen Verfahrens, welches eine hohe Abbildungsgüte aufweist. Die Erkenntnisse dieser Arbeit haben gezeigt, dass der verfolgte Ansatz zwar Verbesserungen gegenüber bestehenden Ansätzen erzielen konnte, mit Ergebnissen um die 40 % im F_1 -Maß diese hohe Abbildungsgüte allerdings noch nicht aufweist. Somit kann

diese Arbeit lediglich als ein Schritt in die richtige Richtung angesehen werden. Um das Ziel zu erfüllen, wird aber weitere Forschung notwendig sein.

Literaturverzeichnis

Die Titel der meisten Einträge stellen Verweise dar, welche auf die DOIs oder eine andere Online-Quelle verweisen.

- [AAT10] Asuncion, H. U., Asuncion, A. U. und Taylor, R. N. „Software Traceability with Topic Modeling“. In: *2010 ACM/IEEE 32nd International Conference on Software Engineering*. 2010 ACM/IEEE 32nd International Conference on Software Engineering. Bd. 1. Mai 2010, S. 95–104.
- [Aba+17] Abad, Z. S. H., Karras, O., Ghazi, P., Glinz, M., Ruhe, G. und Schneider, K. „What Works Better? A Study of Classifying Requirements“. In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 2017 IEEE 25th International Requirements Engineering Conference (RE). Sep. 2017, S. 496–501.
- [Ahm+20] Ahmad, W., Chakraborty, S., Ray, B. und Chang, K.-W. „A Transformer-based Approach for Source Code Summarization“. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. ACL 2020. Online: Association for Computational Linguistics, Juli 2020, S. 4998–5007.
- [Ahm+21] Ahmad, W., Chakraborty, S., Ray, B. und Chang, K.-W. „Unified Pre-training for Program Understanding and Generation“. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. NAACL-HLT 2021. Online: Association for Computational Linguistics, Juni 2021, S. 2655–2668.
- [Aho+07] Aho, A. V., Lam, M. S., Sethi, R. und Ulman, J. D. „Compilers: Principles, Techniques, & Tools“. 2nd ed. Boston: Pearson/Addison Wesley, 2007. 1009 S. ISBN: 978-0-321-48681-3.
- [AK19] Akbar, S. und Kak, A. „SCOR: Source Code Retrieval with Semantics and Order“. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). Mai 2019, S. 1–12.
- [AK20] Akbar, S. A. und Kak, A. C. „A Large-Scale Comparative Evaluation of IR-Based Tools for Bug Localization“. In: *Proceedings of the 17th International Conference on Mining Software Repositories*. MSR '20. New York, NY, USA: Association for Computing Machinery, 18. Sep. 2020, S. 21–31.

- [AL18] Amasaki, S. und Leelaprute, P. „The Effects of Vectorization Methods on Non-Functional Requirements Classification“. In: *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Aug. 2018, S. 175–182.
- [ALS14] Agirre, E., López de Lacalle, O. und Soroa, A. „Random Walks for Knowledge-Based Word Sense Disambiguation“. In: *Computational Linguistics* 40.1 (März 2014), S. 57–84.
- [Ant+02] Antoniol, G., Canfora, G., Casazza, G., Lucia, A. D. und Merlo, E. „Recovering Traceability Links between Code and Documentation“. In: *IEEE Transactions on Software Engineering* 28.10 (Okt. 2002), S. 970–983.
- [Ant+17] Antoniol, G., Cleland-Huang, J., Hayes, J. H. und Vierhauser, M. „Grand Challenges of Traceability: The Next Ten Years“. 9. Okt. 2017. arXiv: 1710.03129 [cs].
- [ASP10] Assawamekin, N., Sunetnanta, T. und Pluempitiwiriyaewej, C. „Ontology-Based Multiperspective Requirements Traceability Framework“. In: *Knowledge and Information Systems* 25.3 (1. Dez. 2010), S. 493–522.
- [AZ22] Ajagbe, M. und Zhao, L. „Retraining a BERT Model for Transfer Learning in Requirements Engineering: A Preliminary Study“. In: *2022 IEEE 30th International Requirements Engineering Conference (RE)*. 2022 IEEE 30th International Requirements Engineering Conference (RE). Aug. 2022, S. 309–315.
- [Bal10] Balzert, H. „Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering“. Heidelberg: Spektrum Akademischer Verlag, 2010. ISBN: 978-3-8274-2247-7.
- [Bis06] Bishop, C. M. „Pattern Recognition and Machine Learning (Information Science and Statistics)“. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0-387-31073-8.
- [Ble12] Blei, D. M. „Probabilistic Topic Models“. In: *Communications of the ACM* 55.4 (1. Apr. 2012), S. 77–84.
- [BNJ03] Blei, D. M., Ng, A. Y. und Jordan, M. I. „Latent Dirichlet Allocation“. In: *Journal of Machine Learning Research* 3.4/5 (März 2003), S. 993–1022.
- [Boj+17] Bojanowski, P., Grave, E., Joulin, A. und Mikolov, T. „Enriching Word Vectors with Subword Information“. In: *Transactions of the Association for Computational Linguistics* 5 (Dez. 2017), S. 135–146.
- [BRA14] Borg, M., Runeson, P. und Ardö, A. „Recovering from a Decade: A Systematic Mapping of Information Retrieval Approaches to Software Traceability“. In: *Empirical Software Engineering* 19.6 (1. Dez. 2014), S. 1565–1616.

- [Bro+20] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. und Amodei, D. „Language Models Are Few-Shot Learners“. In: *Advances in Neural Information Processing Systems*. Bd. 33. Curran Associates, Inc., 2020, S. 1877–1901.
- [Bro15] Broy, M. „Rethinking Nonfunctional Software Requirements“. In: *Computer* 48.5 (Mai 2015), S. 96–99.
- [Bus02] Bussmann, H. „Lexikon der Sprachwissenschaft“. Kröner, 2002. 782 S. ISBN: 978-3-520-45203-0.
- [CCC03] Cleland-Huang, J., Chang, C. und Christensen, M. „Event-Based Traceability for Managing Evolutionary Change“. In: *IEEE Transactions on Software Engineering* 29.9 (Sep. 2003), S. 796–810.
- [Cer+18] Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y.-H., Strophe, B. und Kurzweil, R. „Universal Sentence Encoder“. 12. Apr. 2018. arXiv: 1803.11175 [cs]. Preprint.
- [CGZ+12] Cleland-Huang, J., Gotel, O., Zisman, A. et al. „Software and Systems Traceability“. Bd. 2. 3. Springer, 2012.
- [Che+19] Chen, L., Wang, D., Wang, J. und Wang, Q. „Enhancing Unsupervised Requirements Traceability with Sequential Semantics“. In: *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*. 2019 26th Asia-Pacific Software Engineering Conference (APSEC). Dez. 2019, S. 23–30.
- [Che+21] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I. und Zaremba, W. „Evaluating Large Language Models Trained on Code“. 14. Juli 2021. arXiv: 2107.03374 [cs]. Preprint.
- [Chu+00] Chung, L., Nixon, B. A., Yu, E. und Mylopoulos, J. „Non-Functional Requirements in Software Engineering“. Boston, MA: Springer US, 2000. ISBN: 978-1-4615-5269-7.

- [CKV19] Csuvik, V., Kicsi, A. und Vidács, L. „Source Code Level Word Embeddings in Aiding Semantic Test-to-Code Traceability“. In: *2019 IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST)*. 2019 IEEE/ACM 10th International Symposium on Software and Systems Traceability (SST). Mai 2019, S. 29–36.
- [Cle+05a] Cleland-Huang, J., Settimi, R., Duan, C. und Zou, X. „Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability“. In: *13th IEEE International Conference on Requirements Engineering (RE'05)*. 13th IEEE International Conference on Requirements Engineering (RE'05). Aug. 2005, S. 135–144.
- [Cle+05b] Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E. und Christina, S. „Goal-Centric Traceability for Managing Non-Functional Requirements“. In: *Proceedings of the 27th International Conference on Software Engineering - ICSE '05*. The 27th International Conference. St. Louis, MO, USA: ACM Press, 2005, S. 362.
- [Cle+06] Cleland-Huang, J., Settimi, R., Zou, X. und Solc, P. „The Detection and Classification of Non-Functional Requirements with Application to Early Aspects“. In: *14th IEEE International Requirements Engineering Conference (RE'06)*. 14th IEEE International Requirements Engineering Conference (RE'06). Sep. 2006, S. 39–48.
- [Cle+07a] Cleland-Huang, J., Berenbach, B., Clark, S., Settimi, R. und Romanova, E. „Best Practices for Automated Traceability“. In: *Computer* 40.6 (Juni 2007), S. 27–35.
- [Cle+07b] Cleland-Huang, J., Mazrouee, S., Liguó, H. und Port, D. *Nfr*. Zenodo, 17. März 2007. DOI: 10.5281/zenodo.268542.
- [Cle+07c] Cleland-Huang, J., Settimi, R., Zou, X. und Solc, P. „Automated Classification of Non-Functional Requirements“. In: *Requirements Engineering* 12.2 (2. Mai 2007), S. 103–120.
- [CM04] Carreras, X. und Màrquez, L. „Introduction to the CoNLL-2004 Shared Task: Semantic Role Labeling“. In: *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004) at HLT-NAACL 2004*. Boston, Massachusetts, USA: Association for Computational Linguistics, 6. Mai 2004, S. 89–97.
- [CM05] Carreras, X. und Màrquez, L. „Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling“. In: *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*. Ann Arbor, Michigan: Association for Computational Linguistics, Juni 2005, S. 152–164.
- [Coc00] Cockburn, A. „Writing Effective Use Cases“. Addison-Wesley Professional, 6. Okt. 2000. 301 S. ISBN: 978-0-321-60580-1.
- [CoE] CoEST. *Center of Excellence for Software & Systems Traceability*. URL: <http://coest.org/> (besucht am 24. 05. 2022).

- [Con+17] Conneau, A., Kiela, D., Schwenk, H., Barrault, L. und Bordes, A. „Supervised Learning of Universal Sentence Representations from Natural Language Inference Data“. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2017. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, S. 670–680.
- [CVB18] Cleland-Huang, J., Vierhauser, M. und Bayley, S. „Dronology: An Incubator for Cyber-Physical Systems Research“. In: *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*. 2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER). Mai 2018, S. 109–112.
- [DAD22] Dell’Anna, D., Aydemir, F. B. und Dalpiaz, F. „Evaluating Classifiers in SE Research: The ECSE Pipeline and Two Replication Studies“. In: *Empirical Software Engineering* 28.1 (8. Nov. 2022).
- [Dal+19] Dalpiaz, F., Dell’Anna, D., Aydemir, F. B. und Çevikol, S. „Requirements Classification with Interpretable Machine Learning and Dependency Parsing“. In: *2019 IEEE 27th International Requirements Engineering Conference (RE)*. 2019 IEEE 27th International Requirements Engineering Conference (RE). Sep. 2019, S. 142–152.
- [De +12] De Lucia, A., Marcus, A., Oliveto, R. und Poshyvanyk, D. „Information Retrieval Methods for Automated Traceability Recovery“. In: *Software and Systems Traceability*. Hrsg. von Cleland-Huang, J., Gotel, O. und Zisman, A. London: Springer, 2012, S. 71–98.
- [Deb+02] Deb, K., Pratap, A., Agarwal, S. und Meyarivan, T. „A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II“. In: *IEEE Transactions on Evolutionary Computation* 6.2 (Apr. 2002), S. 182–197.
- [Dee+90] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. und Harshman, R. „Indexing by Latent Semantic Analysis“. In: *Journal of the American Society for Information Science* 41.6 (1990), S. 391–407.
- [Dev+19] Devlin, J., Chang, M.-W., Lee, K. und Toutanova, K. „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding“. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. NAACL-HLT 2019. Minneapolis, Minnesota: Association for Computational Linguistics, Juni 2019, S. 4171–4186.
- [DF17] Dekhtyar, A. und Fong, V. „RE Data Challenge: Requirements Identification with Word2Vec and TensorFlow“. In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 2017 IEEE 25th International Requirements Engineering Conference (RE). Sep. 2017, S. 484–489.
- [Dic45] Dice, L. R. „Measures of the Amount of Ecologic Association Between Species“. In: *Ecology* 26.3 (1945), S. 297–302.

- [dMar+21] De Marneffe, M.-C., Manning, C. D., Nivre, J. und Zeman, D. „Universal Dependencies“. In: *Computational Linguistics* 47.2 (13. Juli 2021), S. 255–308.
- [Dow91] Dowty, D. „Thematic Proto-Roles and Argument Selection“. In: *Language* 67.3 (1991), S. 547–619. JSTOR: 415037.
- [Dum94] Dumais, S. T. „Latent Semantic Indexing (LSI) and TREC-2“. In: *The Second Text REtrieval Conference (TREC-2)*. 1994, S. 105–115.
- [ECS09] ECSS. *ECSS-E-40C: Principles and Requirements Applicable to Space Software Engineering*. 2009.
- [EVF16] Eckhardt, J., Vogelsang, A. und Fernández, D. M. „Are "Non-Functional" Requirements Really Non-Functional? An Investigation of Non-Functional Requirements in Practice“. In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE '16. Austin, Texas: Association for Computing Machinery, 14. Mai 2016, S. 832–842.
- [Fah+16] Fahrmeir, L., Heumann, C., Künstler, R., Pigeot, I. und Tutz, G. „Testen von Hypothesen“. In: *Statistik: Der Weg zur Datenanalyse*. Hrsg. von Fahrmeir, L., Heumann, C., Künstler, R., Pigeot, I. und Tutz, G. Springer-Lehrbuch. Berlin, Heidelberg: Springer, 2016, S. 369–398.
- [Fej+22] Fejzer, M., Narębski, J., Przymus, P. und Stencel, K. „Tracking Buggy Files: New Efficient Adaptive Bug Localization Algorithm“. In: *IEEE Transactions on Software Engineering* 48.7 (Juli 2022), S. 2557–2569.
- [Fen+20] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D. und Zhou, M. „CodeBERT: A Pre-Trained Model for Programming and Natural Languages“. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. EMNLP-Findings 2020. Online: Association for Computational Linguistics, Nov. 2020, S. 1536–1547.
- [Fir57] Firth, J. R. „A Synopsis of Linguistic Theory, 1930-1955“. In: *Studies in Linguistic Analysis* (1957).
- [Flo+22] Florez, J. M., Perry, J., Wei, S. und Marcus, A. „Retrieving Data Constraint Implementations Using Fine-Grained Code Patterns“. In: *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE). Mai 2022, S. 1893–1905.
- [FM98] Fellbaum, C. und Miller, G. A. „WordNet: An Electronic Lexical Database“. Hrsg. von Fellbaum, C. *Language, Speech, and Communication*. MIT Press, 1998. 448 S. ISBN: 978-0-262-06197-1.
- [FSG18] Ferrari, A., Spagnolo, G. O. und Gnesi, S. *PURE: A Dataset of Public Requirements Documents*. Zenodo, 12. Sep. 2018. DOI: 10.5281/zenodo.1414117.
- [Gao+22] Gao, H., Kuang, H., Ma, X., Hu, H., Lü, J., Mäder, P. und Egyed, A. „Propagating Frugal User Feedback through Closeness of Code Dependencies to Improve IR-based Traceability Recovery“. In: *Empirical Software Engineering* 27.2 (18. Jan. 2022).

- [Gao+23] Gao, H., Kuang, H., Sun, K., Ma, X., Egyed, A., Mäder, P., Rong, G., Shao, D. und Zhang, H. „Using Consensual Biterns from Text Structures of Requirements and Code to Improve IR-Based Traceability Recovery“. In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. ASE '22. New York, NY, USA: Association for Computing Machinery, 5. Jan. 2023.
- [GBC16] Goodfellow, I., Bengio, Y. und Courville, A. „Deep Learning“. MIT Press, 10. Nov. 2016. 801 S. ISBN: 978-0-262-33737-3.
- [GCC17] Guo, J., Cheng, J. und Cleland-Huang, J. „Semantically Enhanced Software Traceability Using Deep Learning Techniques“. In: *Proceedings of the 39th International Conference on Software Engineering*. ICSE '17. Piscataway, NJ, USA: IEEE Press, 2017, S. 3–14.
- [Get+11] Gethers, M., Oliveto, R., Poshyvanyk, D. und Lucia, A. D. „On Integrating Orthogonal Information Retrieval Methods to Improve Traceability Recovery“. In: *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. 2011 27th IEEE International Conference on Software Maintenance (ICSM). Sep. 2011, S. 133–142.
- [GF94] Gotel, O. und Finkelstein, C. „An Analysis of the Requirements Traceability Problem“. In: *Proceedings of IEEE International Conference on Requirements Engineering*. Proceedings of IEEE International Conference on Requirements Engineering. Apr. 1994, S. 94–101.
- [Gha12] Ghazarian, A. „Characterization of Functional Software Requirements Space: The Law of Requirements Taxonomic Growth“. In: *2012 20th IEEE International Requirements Engineering Conference (RE)*. 2012 20th IEEE International Requirements Engineering Conference (RE). Sep. 2012, S. 241–250.
- [Gio+21] Giorgi, J., Nitski, O., Wang, B. und Bader, G. „DeCLUTR: Deep Contrastive Learning for Unsupervised Textual Representations“. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. ACL-IJCNLP 2021. Online: Association for Computational Linguistics, Aug. 2021, S. 879–895.
- [Gli07] Glinz, M. „On Non-Functional Requirements“. In: *15th IEEE International Requirements Engineering Conference (RE 2007)*. 15th IEEE International Requirements Engineering Conference (RE 2007). Okt. 2007, S. 21–26.
- [Gli22] Glinz, M. *A Glossary of Requirements Engineering Terminology*. Juli 2022.
- [Got+12a] Gotel, O., Cleland-Huang, J., Hayes, J. H., Zisman, A., Egyed, A., Grünbacher, P., Dekhtyar, A., Antoniol, G. und Maletic, J. „The Grand Challenge of Traceability (v1.0)“. In: *Software and Systems Traceability*. Hrsg. von Cleland-Huang, J., Gotel, O. und Zisman, A. London: Springer, 2012, S. 343–409.

- [Got+12b] Gotel, O., Cleland-Huang, J., Hayes, J. H., Zisman, A., Egyed, A., Grünbacher, P., Dekhtyar, A., Antoniol, G., Maletic, J. und Mäder, P. „Traceability Fundamentals“. In: *Software and Systems Traceability*. Hrsg. von Cleland-Huang, J., Gotel, O. und Zisman, A. London: Springer, 2012, S. 3–22.
- [GR71] Golub, G. H. und Reinsch, C. „Singular Value Decomposition and Least Squares Solutions“. In: *Linear Algebra*. Hrsg. von Wilkinson, J. H., Reinsch, C. und Bauer, F. L. Handbook for Automatic Computation. Berlin, Heidelberg: Springer, 1971, S. 134–151.
- [Gra+18] Grave, E., Bojanowski, P., Gupta, P., Joulin, A. und Mikolov, T. „Learning Word Vectors for 157 Languages“. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. LREC 2018. Miyazaki, Japan: European Language Resources Association (ELRA), Mai 2018.
- [Gru93] Gruber, T. R. „A Translation Approach to Portable Ontology Specifications“. In: *Knowledge Acquisition* 5.2 (1. Juni 1993), S. 199–220.
- [Guo+22a] Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M. und Yin, J. „UniXcoder: Unified Cross-Modal Pre-training for Code Representation“. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2022. Dublin, Ireland: Association for Computational Linguistics, Mai 2022, S. 7212–7225.
- [Guo+22b] Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S. K., Clement, C., Drain, D., Sundaresan, N., Yin, J., Jiang, D. und Zhou, M. „GraphCodeBERT: Pre-training Code Representations with Data Flow“. In: International Conference on Learning Representations. 8. März 2022.
- [Ham+21] Hammoudi, M., Mayr-Dorn, C., Mashkoor, A. und Egyed, A. „TraceRefiner: An Automated Technique for Refining Coarse-Grained Requirement-to-Class Traces“. In: *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. 2021 28th Asia-Pacific Software Engineering Conference (APSEC). Dez. 2021, S. 12–21.
- [Haq+20] Haque, S., LeClair, A., Wu, L. und McMillan, C. „Improved Automatic Summarization of Subroutines via Attention to File Context“. In: *Proceedings of the 17th International Conference on Mining Software Repositories*. MSR '20. New York, NY, USA: Association for Computing Machinery, 18. Sep. 2020, S. 300–310.
- [HDO03] Hayes, J. H., Dekhtyar, A. und Osborne, J. „Improving Requirements Tracing via Information Retrieval“. In: *Proceedings. 11th IEEE International Requirements Engineering Conference, 2003*. Proceedings. 11th IEEE International Requirements Engineering Conference, 2003. Sep. 2003, S. 138–147.
- [HDS06] Hayes, J. H., Dekhtyar, A. und Sundaram, S. K. „Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods“. In: *IEEE Trans. Softw. Eng.* 32.1 (Jan. 2006), S. 4–19.

- [Hey+20a] Hey, T., Keim, J., Koziolok, A. und Tichy, W. F. „NoRBERT: Transfer Learning for Requirements Classification“. In: *2020 IEEE 28th International Requirements Engineering Conference (RE)*. 2020 IEEE 28th International Requirements Engineering Conference (RE). Aug. 2020, S. 169–179.
- [Hey+20b] Hey, T., Keim, J., Koziolok, A. und Tichy, W. F. *Supplementary Material of "NoRBERT: Transfer Learning for Requirements Classification"*. 21. Mai 2020.
- [Hey+21] Hey, T., Chen, F., Weigelt, S. und Tichy, W. F. „Improving Traceability Link Recovery Using Fine-grained Requirements-to-Code Relations“. In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). Sep. 2021, S. 12–22.
- [Hey+22] Hey, T., Keim, J., Koziolok, A. und Tichy, W. F. *NoRBERT: Transfer Learning for Requirements Classification*. Karlsruher Institut für Technologie (KIT), 2022.
- [Hey19] Hey, T. „INDIRECT: Intent-Driven Requirements-to-Code Traceability“. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). Mai 2019, S. 190–191.
- [Hey23a] Hey, T. *Dataset for Requirements Classification in Traceability Link Recovery Datasets*. Zenodo, 26. Apr. 2023. DOI: 10.5281/zenodo.7867845.
- [Hey23b] Hey, T. *Fine-Grained Traceability Link Recovery (FTLR)*. Zenodo, 21. Sep. 2023. DOI: 10.5281/zenodo.8367343.
- [Hey23c] Hey, T. *NoRBERT for Requirements Classification in Traceability Link Recovery Datasets*. Zenodo, 15. Sep. 2023. DOI: 10.5281/zenodo.8348363.
- [HHD09] Holbrook, E. A., Hayes, J. H. und Dekhtyar, A. „Toward Automating Requirements Satisfaction Assessment“. In: *2009 17th IEEE International Requirements Engineering Conference*. 2009 17th IEEE International Requirements Engineering Conference. Aug. 2009, S. 149–158.
- [HKO08] Hussain, I., Kosseim, L. und Ormandjieva, O. „Using Linguistic Knowledge to Classify Non-functional Requirements in SRS Documents“. In: *Proceedings of the 13th International Conference on Natural Language and Information Systems: Applications of Natural Language to Information Systems*. NLDB '08. London, UK: Springer-Verlag, 24. Juni 2008, S. 287–298.
- [HKT21] Hey, T., Keim, J. und Tichy, W. F. „Knowledge-Based Sense Disambiguation of Multiword Expressions in Requirements Documents“. In: *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*. 2021 IEEE 29th International Requirements Engineering Conference Workshops (REW). Sep. 2021, S. 70–76.

- [HL17] Huo, X. und Li, M. „Enhancing the Unified Features to Locate Buggy Files by Exploiting the Sequential Nature of Source Code“. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI'17. Melbourne, Australia: AAAI Press, 19. Aug. 2017, S. 1909–1915.
- [HLZ16] Huo, X., Li, M. und Zhou, Z.-H. „Learning Unified Features from Natural and Programming Languages for Locating Buggy Source Code“. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. IJCAI'16. New York, New York, USA: AAAI Press, 9. Juli 2016, S. 1606–1612.
- [Hof99] Hofmann, T. „Probabilistic Latent Semantic Analysis“. In: *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. UAI'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 30. Juli 1999, S. 289–296.
- [HR18] Howard, J. und Ruder, S. „Universal Language Model Fine-tuning for Text Classification“. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2018. Melbourne, Australia: Association for Computational Linguistics, Juli 2018, S. 328–339.
- [HS97] Hochreiter, S. und Schmidhuber, J. „Long Short-Term Memory“. In: *Neural Computation* 9.8 (Nov. 1997), S. 1735–1780.
- [Huo+21] Huo, X., Thung, F., Li, M., Lo, D. und Shi, S.-T. „Deep Transfer Bug Localization“. In: *IEEE Transactions on Software Engineering* 47.7 (Juli 2021), S. 1368–1380.
- [Hus+20] Husain, H., Wu, H.-H., Gazit, T., Allamanis, M. und Brockschmidt, M. „Code-SearchNet Challenge: Evaluating the State of Semantic Code Search“. 8. Juni 2020. arXiv: 1909.09436 [cs, stat]. Preprint.
- [HYS10] Hayashi, S., Yoshikawa, T. und Saeki, M. „Sentence-to-Code Traceability Recovery with Domain Ontologies“. In: *2010 Asia Pacific Software Engineering Conference*. 2010 Asia Pacific Software Engineering Conference. Nov. 2010, S. 385–394.
- [IS04] Ide, N. und Suderman, K. „The American National Corpus First Release“. In: *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*. LREC 2004. Lisbon, Portugal: European Language Resources Association (ELRA), Mai 2004.
- [ISO17] ISO/IEC/IEEE. *ISO/IEC/IEEE International Standard - Systems and Software Engineering–Vocabulary*. Aug. 2017.
- [Jac12] Jaccard, P. „The Distribution of the Flora in the Alpine Zone.1“. In: *New Phytologist* 11.2 (1912), S. 37–50.
- [JHH13] Janzen, A., Hoffmann, A. und Hoffmann, H. „Anforderungsmuster Im Requirements Engineering“. In: *Information Systems, Kassel University* (2013).

- [JHS02] Jones, J., Harrold, M. und Stasko, J. „Visualization of Test Information to Assist Fault Localization“. In: *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*. Proceedings of the 24th International Conference on Software Engineering. ICSE 2002. Mai 2002, S. 467–477.
- [JM09] Jurafsky, D. und Martin, J. H. „Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition“. 2. ed., [Pearson International Edition]. Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, NJ: Prentice Hall, Pearson Education International, 2009. ISBN: 978-0-13-504196-3.
- [JM22] Jurafsky, D. und Martin, J. H. „Speech and Language Processing“. 3. ed. draft. 2022.
- [Jou+18] Joulin, A., Bojanowski, P., Mikolov, T., Jégou, H. und Grave, E. „Loss in Translation: Learning Bilingual Word Mapping with a Retrieval Criterion“. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Okt. 2018, S. 2979–2984.
- [KB15] Kingma, D. P. und Ba, L. J. „Adam: A Method for Stochastic Optimization“. In: *International Conference on Learning Representations (ICLR)*. San Diego, California, 2015.
- [KCV21] Kicsi, A., Csuvik, V. und Vidács, L. „Large Scale Evaluation of Natural Language Processing Based Test-to-Code Traceability Approaches“. In: *IEEE Access* 9 (2021), S. 79089–79104.
- [Kei+21] Keim, J., Schulz, S., Fuchß, D., Kocher, C., Speit, J. und Koziolok, A. „Trace Link Recovery for Software Architecture Documentation“. In: *Software Architecture*. Hrsg. von Biffel, S., Navarro, E., Löwe, W., Sirjani, M., Mirandola, R. und Weyns, D. *Lecture Notes in Computer Science*. Cham: Springer International Publishing, 2021, S. 101–116.
- [KM17] Kurtanović, Z. und Maalej, W. „Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning“. In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 2017 IEEE 25th International Requirements Engineering Conference (RE). Sep. 2017, S. 490–495.
- [Kna+12] Knauss, E., Damian, D., Poo-Caamaño, G. und Cleland-Huang, J. „Detecting and Classifying Patterns of Requirements Clarifications“. In: *2012 20th IEEE International Requirements Engineering Conference (RE)*. 2012 20th IEEE International Requirements Engineering Conference (RE). Sep. 2012, S. 251–260.
- [KNL14] Kassab, M., Neill, C. und Laplante, P. „State of Practice in Requirements Engineering: Contemporary Data“. In: *Innovations in Systems and Software Engineering* 10.4 (1. Dez. 2014), S. 235–241.

- [Kon+11] Kong, W.-K., Huffman Hayes, J., Dekhtyar, A. und Holden, J. „How Do We Trace Requirements: An Initial Study of Analyst Behavior in Trace Validation Tasks“. In: *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*. CHASE '11. New York, NY, USA: Association for Computing Machinery, 21. Mai 2011, S. 32–39.
- [Kon+12] Kong, W.-K., Hayes, J. H., Dekhtyar, A. und Dekhtyar, O. „Process Improvement for Traceability: A Study of Human Fallibility“. In: *2012 20th IEEE International Requirements Engineering Conference (RE)*. 2012 20th IEEE International Requirements Engineering Conference (RE). Sep. 2012, S. 31–40.
- [Kri04] Krippendorff, K. „Reliability in Content Analysis“. In: *Human Communication Research* 30.3 (2004), S. 411–433.
- [Kri18] Krippendorff, K. „Content Analysis: An Introduction to Its Methodology“. SAGE Publications, 9. Mai 2018. 473 S. ISBN: 978-1-5063-9567-8.
- [Kua+15] Kuang, H., Mäder, P., Hu, H., Ghabi, A., Huang, L., Lü, J. und Egyed, A. „Can Method Data Dependencies Support the Assessment of Traceability Between Requirements and Source Code?“ In: *J. Softw. Evol. Process* 27.11 (Nov. 2015), S. 838–866.
- [Kus+15] Kusner, M., Sun, Y., Kolkin, N. und Weinberger, K. „From Word Embeddings To Document Distances“. In: *International Conference on Machine Learning*. PMLR, Juni 2015, S. 957–966.
- [Lac22] Lachenicht, T. N. „Vergleich verschiedener Sprachmodelle für den Einsatz in automatisierter Rückverfolgbarkeitsanalyse“. Abschlussarbeit - Bachelor. Karlsruher Institut für Technologie (KIT), Nov. 2022.
- [Lam+17] Lam, A. N., Nguyen, A. T., Nguyen, H. A. und Nguyen, T. N. „Bug Localization with Combination of Deep Learning and Information Retrieval“. In: *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC). Mai 2017, S. 218–229.
- [LBM21] LeClair, A., Bansal, A. und McMillan, C. „Ensemble Models for Neural Source Code Summarization of Subroutines“. In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). Sep. 2021, S. 286–297.
- [Leh+15] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S. und Bizer, C. „DBpedia – A Large-Scale, Multilingual Knowledge Base Extracted from Wikipedia“. In: *Semantic Web* 6.2 (2015), S. 167–195.
- [Lev66] Levenshtein, V. I. „Binary Codes Capable of Correcting Deletions, Insertions and Reversals“. In: *Soviet Physics Doklady* 10 (1. Feb. 1966).

- [LH18] Loshchilov, I. und Hutter, F. „Fixing Weight Decay Regularization in Adam“. In: (15. Feb. 2018). Preprint.
- [LHL22] Liang, H., Hang, D. und Li, X. „Modeling Function-Level Interactions for File-Level Bug Localization“. In: *Empirical Software Engineering* 27.7 (1. Okt. 2022).
- [Li+14] Li, F.-L., Horkoff, J., Mylopoulos, J., Guizzardi, R. S. S., Guizzardi, G., Borgida, A. und Liu, L. „Non-Functional Requirements as Qualities, with a Spice of Ontology“. In: *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. 2014 IEEE 22nd International Requirements Engineering Conference (RE). Aug. 2014, S. 293–302.
- [Li+22] Li, G., Zheng, C., Li, M. und Wang, H. „Automatic Requirements Classification Based on Graph Attention Network“. In: *IEEE Access* 10 (2022), S. 30080–30090.
- [Lia+19] Liang, H., Sun, L., Wang, M. und Yang, Y. „Deep Learning With Customized Abstract Syntax Tree for Bug Localization“. In: *IEEE Access* 7 (2019), S. 116309–116320.
- [Lin+21] Lin, J., Liu, Y., Zeng, Q., Jiang, M. und Cleland-Huang, J. „Traceability Transformed: Generating More Accurate Links with Pre-Trained BERT Models“. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). Mai 2021, S. 324–335.
- [Lin+22] Lin, J., Poudel, A., Yu, W., Zeng, Q., Jiang, M. und Cleland-Huang, J. „Enhancing Automated Software Traceability by Transfer Learning from Open-World Data“. 3. Juli 2022. arXiv: 2207.01084 [cs]. Preprint.
- [Lin98] Lin, D. „Automatic Retrieval and Clustering of Similar Words“. In: *COLING 1998 Volume 2: The 17th International Conference on Computational Linguistics*. COLING 1998. 1998.
- [Liu+19] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. und Stoyanov, V. „RoBERTa: A Robustly Optimized BERT Pretraining Approach“. 26. Juli 2019. arXiv: 1907.11692 [cs]. Preprint.
- [LLC21] Lin, J., Liu, Y. und Cleland-Huang, J. „Information Retrieval versus Deep Learning Approaches for Generating Traceability Links in Bilingual Projects“. In: *Empirical Software Engineering* 27.1 (22. Okt. 2021).
- [LM14] Le, Q. und Mikolov, T. „Distributed Representations of Sentences and Documents“. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML’14. Beijing, China: JMLR.org, 21. Juni 2014, S. II-1188–II-1196.

- [Loh+13] Lohar, S., Amornborvornwong, S., Zisman, A. und Cleland-Huang, J. „Improving Trace Accuracy Through Data-driven Configuration and Composition of Tracing Features“. In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, S. 378–388.
- [LOS06] Lucia, A. D., Oliveto, R. und Sgueglia, P. „Incremental Approach and User Feedbacks: A Silver Bullet for Traceability Recovery“. In: *2006 22nd IEEE International Conference on Software Maintenance*. 2006 22nd IEEE International Conference on Software Maintenance. Sep. 2006, S. 299–309.
- [Luh57] Luhn, H. P. „A Statistical Approach to Mechanized Encoding and Searching of Literary Information“. In: *IBM Journal of Research and Development* 1.4 (Okt. 1957), S. 309–317.
- [LWC22] Luo, Z., Wang, W. und Cen, C. „Improving Bug Localization with Effective Contrastive Learning Representation“. In: *IEEE Access* (2022), S. 32523–32533.
- [Mar08] Martin, R. C. „Clean Code: A Handbook of Agile Software Craftsmanship“. 1. Aufl. USA: Prentice Hall PTR, 2008. 448 S. ISBN: 978-0-13-235088-4.
- [MCN92] Mylopoulos, J., Chung, L. und Nixon, B. „Representing and Using Nonfunctional Requirements: A Process-Oriented Approach“. In: *IEEE Transactions on Software Engineering* 18.6 (Juni 1992), S. 483–497.
- [MEH18] Mills, C., Escobar-Avila, J. und Haiduc, S. „Automatic Traceability Maintenance via Machine Learning Classification“. In: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). Sep. 2018, S. 369–380.
- [Mek+17] Mekala, D., Gupta, V., Paranjape, B. und Karnick, H. „SCDV : Sparse Composite Document Vectors Using Soft Clustering over Distributional Representations“. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2017. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, S. 659–669.
- [MFN04] Mich, L., Franch, M. und Novi Inverardi, P. L. „Market Research for Requirements Analysis Using Linguistic Tools“. In: *Requirements Engineering* 9.2 (1. Mai 2004).
- [MHG10] McCandless, M., Hatcher, E. und Gospodnetic, O. „Lucene in Action, Second Edition: Covers Apache Lucene 3.0“. USA: Manning Publications Co., 2010. ISBN: 978-1-933988-17-7.
- [Mik+13a] Mikolov, T., Chen, K., Corrado, G. und Dean, J. „Efficient Estimation of Word Representations in Vector Space“. 6. Sep. 2013. arXiv: 1301.3781 [cs]. Preprint.

- [Mik+13b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. und Dean, J. „Distributed Representations of Words and Phrases and Their Compositionality“. In: *Advances in Neural Information Processing Systems*. Bd. 26. Curran Associates, Inc., 2013.
- [Mil+19] Mills, C., Escobar-Avila, J., Bhattacharya, A., Kondyukov, G., Chakraborty, S. und Haiduc, S. „Tracing with Less Data: Active Learning for Classification-Based Traceability Link Recovery“. In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). Sep. 2019, S. 103–113.
- [Mil95] Miller, G. A. „WordNet: A Lexical Database for English“. In: *Communications of the ACM* 38.11 (1. Nov. 1995), S. 39–41.
- [Mit97] Mitchell, T. M. „Machine Learning. 1997“. First. McGraw-Hill Science/Engineering/Math, 1997. ISBN: 0-07-042807-7.
- [MM03] Marcus, A. und Maletic, J. I. „Recovering Documentation-to-source-code Traceability Links Using Latent Semantic Indexing“. In: *Proceedings of the 25th International Conference on Software Engineering*. ICSE '03. Washington, DC, USA: IEEE Computer Society, 2003, S. 125–135.
- [MMS93] Marcus, M. P., Marcinkiewicz, M. A. und Santorini, B. „Building a Large Annotated Corpus of English: The Penn Treebank“. In: *Comput. Linguist.* 19.2 (Juni 1993), S. 313–330.
- [MN11] Mahmoud, A. und Niu, N. „Source Code Indexing for Automated Tracing“. In: *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*. ICSE11: International Conference on Software Engineering. Waikiki, Honolulu HI USA: ACM, 23. Mai 2011, S. 3–9.
- [MN15] Mahmoud, A. und Niu, N. „On the Role of Semantics in Automated Requirements Tracing“. In: *Requirements Engineering* 20.3 (1. Sep. 2015), S. 281–300.
- [Mor+20a] Moran, K., Palacio, D. N., Bernal-Cárdenas, C., McCrystal, D., Poshyvanyk, D., Shenefiel, C. und Johnson, J. *Comet-Data-Replication-Package*. 7. Juli 2020. URL: <https://gitlab.com/SEMERU-Code-Public/Data/icse20-comet-data-replication-package> (besucht am 20.04.2023).
- [Mor+20b] Moran, K., Palacio, D. N., Bernal-Cárdenas, C., McCrystal, D., Poshyvanyk, D., Shenefiel, C. und Johnson, J. „Improving the Effectiveness of Traceability Link Recovery Using Hierarchical Bayesian Networks“. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ICSE '20. New York, NY, USA: Association for Computing Machinery, 27. Juni 2020, S. 873–885.
- [Mor38] Morris, C. W. „Foundations of the Theory of Signs“. Monograph Collection (Matt - Pseudo), 1938.

- [MRN14] Moro, A., Raganato, A. und Navigli, R. „Entity Linking Meets Word Sense Disambiguation: A Unified Approach“. In: *Transactions of the Association for Computational Linguistics 2* (2014), S. 231–244.
- [MRS08] Manning, C. D., Raghavan, P. und Schütze, H. „Introduction to Information Retrieval“. Cambridge University Press, 2008. ISBN: 978-0-511-80907-1.
- [New+19] Newman, C. D., Decker, M. J., Alsuhaibani, R. S., Peruma, A., Kaushik, D. und Hill, E. „An Empirical Study of Abbreviations and Expansions in Software Artifacts“. In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). Sep. 2019, S. 269–279.
- [Nid15] Nida, E. A. „A Componential Analysis of Meaning: An Introduction to Semantic Structures“. De Gruyter Mouton, 3. Juni 2015. ISBN: 978-3-11-082869-6.
- [NJL17] Navarro-Almanza, R., Juarez-Ramirez, R. und Licea, G. „Towards Supporting Software Engineering Using Deep Learning: A Case of Software Requirements Classification“. In: *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*. 2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT). Okt. 2017, S. 116–120.
- [Oll72] Oller, J. W. „ON THE RELATION BETWEEN SYNTAX, SEMANTICS, AND PRAGMATICS“. In: 10.83 (1. Jan. 1972), S. 43–55.
- [Ope23] OpenAI. *GPT-4 Technical Report*. arXiv, 27. März 2023. arXiv: 2303.08774 [cs].
- [Ora] Oracle. *Documentation Comment Specification for the Standard Doclet (JDK 18)*. URL: <https://docs.oracle.com/en/java/javase/18/docs/specs/javadoc/doc-comment-spec.html> (besucht am 03.06.2022).
- [Pan+13] Panichella, A., McMillan, C., Moritz, E., Palmieri, D., Oliveto, R., Poshyva-nyk, D. und Lucia, A. D. „When and How Using Structural Information to Improve IR-Based Traceability Recovery“. In: *2013 17th European Conference on Software Maintenance and Reengineering*. 2013 17th European Conference on Software Maintenance and Reengineering. März 2013, S. 199–208.
- [PCW85] Parnas, D., Clements, P. und Weiss, D. „The Modular Structure of Complex Systems“. In: *IEEE Transactions on Software Engineering SE-11.3* (März 1985), S. 259–266.
- [Pep07] „Aspekte der Programmiermethodik“. In: *Programmieren lernen: Eine grundlegende Einführung mit Java*. Hrsg. von Pepper, P. eXamen.press. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 109–142.

- [Pet+18] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. und Zettlemoyer, L. „Deep Contextualized Word Representations“. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. NAACL-HLT 2018. New Orleans, Louisiana: Association for Computational Linguistics, Juni 2018, S. 2227–2237.
- [PGK05] Palmer, M., Gildea, D. und Kingsbury, P. „The Proposition Bank: An Annotated Corpus of Semantic Roles“. In: *Comput. Linguist.* 31.1 (März 2005), S. 71–106.
- [PMB19] Polisetty, S., Miranskyy, A. und Başar, A. „On Usefulness of the Deep-Learning-Based Bug Localization Models to Practitioners“. In: *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*. PROMISE’19. New York, NY, USA: Association for Computing Machinery, 18. Sep. 2019, S. 16–25.
- [Poh16] Pohl, K. „Requirements Engineering Fundamentals, 2nd Edition: A Study Guide for the Certified Professional for Requirements Engineering Exam - Foundation Level - IREB Compliant“. Rocky Nook, Inc., 30. Apr. 2016. 263 S. ISBN: 978-1-937538-84-2.
- [Pop02] Popper, K. R. „Conjectures and Refutations: The Growth of Scientific Knowledge“. Psychology Press, 2002. 614 S. ISBN: 978-0-415-28594-0.
- [Por80] Porter, M. „An Algorithm for Suffix Stripping“. In: *Program* 14.3 (1. Jan. 1980), S. 130–137.
- [Qi+22] Qi, B., Sun, H., Yuan, W., Zhang, H. und Meng, X. „DreamLoc: A Deep Relevance Matching-Based Framework for Bug Localization“. In: *IEEE Transactions on Reliability* 71.1 (März 2022), S. 235–249.
- [Rad+18] Radford, A., Narasimhan, K., Salimans, T. und Sutskever, I. *Improving Language Understanding by Generative Pre-Training*. 2018. Preprint.
- [Rat+18] Rath, M., Rendall, J., Guo, J. L. C., Cleland-Huang, J. und Mäder, P. „Traceability in the Wild: Automatically Augmenting Incomplete Trace Links“. In: *Proceedings of the 40th International Conference on Software Engineering*. ICSE ’18. New York, NY, USA: ACM, 2018, S. 834–845.
- [RC20] Rodriguez, D. V. und Carver, D. L. „Multi-Objective Information Retrieval-Based NSGA-II Optimization for Requirements Traceability Recovery“. In: *2020 IEEE International Conference on Electro Information Technology (EIT)*. 2020 IEEE International Conference on Electro Information Technology (EIT). Juli 2020, S. 271–280.
- [RCF21] Rodriguez, A. D., Cleland-Huang, J. und Falessi, D. „Leveraging Intermediate Artifacts to Improve Automated Trace Link Retrieval“. In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). Sep. 2021, S. 81–92.

- [RG19] Reimers, N. und Gurevych, I. „Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks“. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. EMNLP-IJCNLP 2019. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, S. 3982–3992.
- [Rie13] Rierson, L. „Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance“. CRC Press, Jan. 2013. ISBN: 978-1-4398-1368-3.
- [RMC14] Rahimi, M., Mirakhorli, M. und Cleland-Huang, J. „Automated Extraction and Visualization of Quality Concerns from Requirements Specifications“. In: *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. 2014 IEEE 22nd International Requirements Engineering Conference (RE). Aug. 2014, S. 253–262.
- [RTC00] RTCA/EUROCAE. *DO-178B/ED-12B: Software Considerations in Airborne Systems and Equipment Certification*. 2000.
- [RTG98] Rubner, Y., Tomasi, C. und Guibas, L. „A Metric for Distributions with Applications to Image Databases“. In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271). Jan. 1998, S. 59–66.
- [Sah+13] Saha, R. K., Lease, M., Khurshid, S. und Perry, D. E. „Improving Bug Localization Using Structured Information Retrieval“. In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). Nov. 2013, S. 345–355.
- [SAK17] Sisman, B., Akbar, S. A. und Kak, A. C. „Exploiting Spatial Code Proximity and Order for Improved Source Code Retrieval for Bug Localization“. In: *Journal of Software: Evolution and Process* 29.1 (2017).
- [San+20] Sanh, V., Debut, L., Chaumond, J. und Wolf, T. „DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter“. 29. Feb. 2020. arXiv: 1910.01108 [cs]. Preprint.
- [SB15] Sharma, R. und Biswas, K. K. „Functional Requirements Categorization Grounded Theory Approach“. In: *2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*. 2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE). Apr. 2015, S. 301–307.
- [SC12] Shin, Y. und Cleland-Huang, J. „A Comparative Evaluation of Two User Feedback Techniques for Requirements Trace Retrieval“. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. SAC 2012: ACM Symposium on Applied Computing. Trento Italy: ACM, 26. März 2012, S. 1069–1074.

- [SHC15] Shin, Y., Hayes, J. H. und Cleland-Huang, J. „Guidelines for Benchmarking Automated Software Traceability Techniques“. In: *2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability*. 2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability. Mai 2015, S. 61–67.
- [Shi+22] Shi, E., Wang, Y., Du, L., Chen, J., Han, S., Zhang, H., Zhang, D. und Sun, H. „On the Evaluation of Neural Code Summarization“. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE '22. New York, NY, USA: Association for Computing Machinery, 5. Juli 2022, S. 1597–1608.
- [SM05] Sayyad Shirabad, J. und Menzies, T. *The PROMISE Repository of Software Engineering Databases*. 2005.
- [Som12] Sommerville, I. „Software Engineering“. 9. Aktual. München Harlow Amsterdam: Pearson Studium ein Imprint von Pearson Deutschland, 3. März 2012. 850 S. ISBN: 978-3-86894-099-2.
- [Spa72] Sparck Jones, K. „A Statistical Interpretation of Term Specificity and Its Application in Retrieval“. In: *Journal of Documentation* (1972).
- [Stu08] Student. „The Probable Error of a Mean“. In: *Biometrika* 6.1 (1908), S. 1–25. JSTOR: 2331554.
- [SV20] Schlutter, A. und Vogelsang, A. „Trace Link Recovery Using Semantic Relation Graphs and Spreading Activation“. In: *2020 IEEE 28th International Requirements Engineering Conference (RE)*. 2020 IEEE 28th International Requirements Engineering Conference (RE). Aug. 2020, S. 20–31.
- [SV21] Schlutter, A. und Vogelsang, A. „Improving Trace Link Recovery Using Semantic Relation Graphs and Spreading Activation“. In: *Requirements Engineering: Foundation for Software Quality*. Hrsg. von Dalpiaz, F. und Spoletini, P. Cham: Springer International Publishing, 2021, S. 37–53.
- [SW13] Slankas, J. und Williams, L. „Automated Extraction of Non-Functional Requirements in Available Documentation“. In: *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*. 2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE). Mai 2013, S. 9–16.
- [TDP19] Tenney, I., Das, D. und Pavlick, E. „BERT Rediscovered the Classical NLP Pipeline“. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. ACL 2019. Florence, Italy: Association for Computational Linguistics, Juli 2019, S. 4593–4601.
- [TEF11] TEFSE. *6th International Workshop on Traceability in Emerging Forms of Software Engineering Challenge*. 2011. URL: <https://www.cs.wm.edu/semeru/tefse2011/Challenge.htm> (besucht am 20. 04. 2023).
- [Tha17] Thanaki, J. „Python Natural Language Processing“. Packt Publishing Ltd, 31. Juli 2017. 476 S. ISBN: 978-1-78728-552-1.

- [TMS03] Taylor, A., Marcus, M. und Santorini, B. „The Penn Treebank: An Overview“. In: *Treebanks: Building and Using Parsed Corpora*. Hrsg. von Abeillé, A. Text, Speech and Language Technology. Dordrecht: Springer Netherlands, 2003, S. 5–22.
- [Vas+17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. und Polosukhin, I. „Attention Is All You Need“. In: *Advances in Neural Information Processing Systems*. Bd. 30. Curran Associates, Inc., 2017.
- [Wan+18] Wang, W., Niu, N., Liu, H. und Niu, Z. „Enhancing Automated Requirements Traceability by Resolving Polysemy“. In: *2018 IEEE 26th International Requirements Engineering Conference (RE)*. 2018 IEEE 26th International Requirements Engineering Conference (RE). Aug. 2018, S. 40–51.
- [Wan+20] Wang, W., Zhang, Y., Sui, Y., Wan, Y., Zhao, Z., Wu, J., Yu, P. und Xu, G. „Reinforcement-Learning-Guided Source Code Summarization via Hierarchical Attention“. In: *IEEE Transactions on Software Engineering* (2020), S. 102–119.
- [Wan+21a] Wang, X., Wang, Y., Mi, F., Zhou, P., Wan, Y., Liu, X., Li, L., Wu, H., Liu, J. und Jiang, X. „SynCoBERT: Syntax-Guided Multi-Modal Contrastive Pre-Training for Code Representation“. 9. Sep. 2021. arXiv: 2108.04556 [cs]. Preprint.
- [Wan+21b] Wang, Y., Wang, W., Joty, S. und Hoi, S. C. „CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation“. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2021. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, S. 8696–8708.
- [Wei+19] Weigelt, S., Keim, J., Hey, T. und Tichy, W. F. „Unsupervised Multi-Topic Labeling for Spoken Utterances“. In: *2019 IEEE International Conference on Humanized Computing and Communication (HCC)*. 2019 IEEE International Conference on Humanized Computing and Communication (HCC). Sep. 2019, S. 38–45.
- [Wen+19] Wen, F., Nagy, C., Bavota, G. und Lanza, M. „A Large-Scale Empirical Study on Code-Comment Inconsistencies“. In: *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC). Mai 2019, S. 53–64.
- [Wil92] Wilcoxon, F. „Individual Comparisons by Ranking Methods“. In: *Breakthroughs in Statistics: Methodology and Distribution*. Hrsg. von Kotz, S. und Johnson, N. L. Springer Series in Statistics. New York, NY: Springer, 1992, S. 196–202.
- [WK22] White, R. und Krinke, J. „TCTracer: Establishing Test-to-Code Traceability Links Using Dynamic and Static Techniques“. In: *Empirical Software Engineering* 27.3 (17. März 2022).

- [Wol+20] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q. und Rush, A. M. „Transformers: State-of-the-art Natural Language Processing“. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Okt. 2020, S. 38–45.
- [WV16] Winkler, J. und Vogelsang, A. „Automatic Classification of Requirements Based on Convolutional Neural Networks“. In: *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW). Sep. 2016, S. 39–45.
- [WvP10] Winkler, S. und von Pilgrim, J. „A Survey of Traceability in Requirements Engineering and Model-Driven Development“. In: *Software & Systems Modeling* 9.4 (1. Sep. 2010), S. 529–565.
- [WWF20] Wang, Y., Wang, M. und Fujita, H. „Word Sense Disambiguation: A Comprehensive Knowledge Exploitation Framework“. In: *Knowledge-Based Systems* 190 (29. Feb. 2020).
- [Xia+17] Xiao, Y., Keung, J., Mi, Q. und Bennin, K. E. „Improving Bug Localization with an Enhanced Convolutional Neural Network“. In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. 2017 24th Asia-Pacific Software Engineering Conference (APSEC). Dez. 2017, S. 338–347.
- [Xia+19] Xiao, Y., Keung, J., Bennin, K. E. und Mi, Q. „Improving Bug Localization with Word Embedding and Enhanced Convolutional Neural Networks“. In: *Information and Software Technology* 105 (1. Jan. 2019), S. 17–29.
- [Yan+19] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R. und Le, Q. V. „XLNet: Generalized Autoregressive Pretraining for Language Understanding“. In: *Advances in Neural Information Processing Systems* 32. Hrsg. von Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F., Fox, E. und Garnett, R. Curran Associates, Inc., 2019, S. 5754–5764.
- [YBL16] Ye, X., Bunescu, R. und Liu, C. „Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine-Grained Benchmark, and Feature Evaluation“. In: *IEEE Transactions on Software Engineering* 42.4 (Apr. 2016), S. 379–402.
- [ZCS17] Zhao, T., Cao, Q. und Sun, Q. „An Improved Approach to Traceability Recovery Based on Word Embeddings“. In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. 2017 24th Asia-Pacific Software Engineering Conference (APSEC). Dez. 2017, S. 81–89.
- [Zha+20] Zhang, J., Xie, R., Ye, W., Zhang, Y. und Zhang, S. „Exploiting Code Knowledge Graph for Bug Localization via Bi-directional Attention“. In: *Proceedings of the 28th International Conference on Program Comprehension*. ICPC '20. New York, NY, USA: Association for Computing Machinery, 12. Sep. 2020, S. 219–229.

- [Zha+21] Zhang, M., Tao, C., Guo, H. und Huang, Z. „Recovering Semantic Traceability between Requirements and Source Code Using Feature Representation Techniques“. In: *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS). Dez. 2021, S. 873–882.
- [Zhu+15] Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A. und Fidler, S. „Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books“. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV '15. USA: IEEE Computer Society, 7. Dez. 2015, S. 19–27.
- [ZSC08] Zou, X., Settmi, R. und Cleland-Huang, J. „Evaluating the Use of Project Glossaries in Automated Trace Retrieval“. In: *Proceedings of the 2008 International Conference on Software Engineering Research & Practice, SERP 2008, July 14-17, 2008, Las Vegas Nevada, USA, 2 Volumes*. Hrsg. von Arabnia, H. R. und Reza, H. CSREA Press, 2008, S. 157–163.
- [ZSC10] Zou, X., Settmi, R. und Cleland-Huang, J. „Improving Automated Requirements Trace Retrieval: A Study of Term-Based Enhancement Methods“. In: *Empirical Software Engineering* 15.2 (1. Apr. 2010), S. 119–146.
- [ZZL12] Zhou, J., Zhang, H. und Lo, D. „Where Should the Bugs Be Fixed? More Accurate Information Retrieval-Based Bug Localization Based on Bug Reports“. In: *2012 34th International Conference on Software Engineering (ICSE)*. 2012 34th International Conference on Software Engineering (ICSE). Juni 2012, S. 14–24.

Anhang

A. Etikettensätze

Die folgenden Abschnitte enthalten die vollständigen Etikettensätze, wie sie oftmals zur Wortarterkennung (s. Abschnitt 2.8.3), dem syntaktischen Zerteilen (s. Abschnitt 2.8.6) sowie der Erkennung semantischer Rollen (s. Abschnitt 2.8.7) verwendet werden.

A.1. Wortarten

Etiketten für Wortarten basierend auf dem Etikettensatz der Penn Treebank [TMS03]:

Etikette	Bedeutung
CC	Nebenordnende Konjunktion
CD	Kardinalzahl
DT	Determinativ (Artikel)
EX	Existenzielles <i>there</i>
FW	Fremdwort
IN	Präposition or unterordnende Konjunktion
JJ	Adjektiv
JJR	Adjektiv, vergleichend
JJS	Adjektiv, superlativ
LS	Listeneintragsmarkierung
MD	Modalverb
NN	Nomen, Singular oder Kontinuativum
NNS	Nomen, Plural
NNP	Eigennamen, Singular
NNPS	Eigennamen, Plural
PDT	Prädeterminativ
POS	Possessivendung
PRP	Personalpronomen
PRP	Possessivpronomen
RB	Adverb
RBR	Adverb, vergleichend
RBS	Adverb, superlativ
RP	Partikel
SYM	Symbol
TO	Infinitivistisches <i>to</i>

Etikette	Bedeutung
UH	Interjektion
VB	Verb, Grundform
VBD	Verb, Präteritum
VBG	Verb, Gerundium oder Partizip I
VBN	Verb, Partizip II
VBP	Verb, 1. oder 2. Person Singular Präsens
VBZ	Verb, 3. Person Singular Präsens
WDT	<i>Wh</i> -Determinativ
WP	<i>Wh</i> -Pronomen
WP	Possessiv- <i>Wh</i> -Pronomen
WRB	<i>Wh</i> -Adverb

A.2. Syntaktisches Zerteilen

Etiketten für syntaktische Kategorien basierend auf dem Etikettensatz der Penn Treebank [TMS03]:

Etikette	Bedeutung
ROOT	Wurzel
ADJP	Adjektivphrase
ADVP	Adverbialphrase
NP	Nominalphrase
PP	Präpositionalphrase
S	Einfacher Deklarativsatz
SBAR	Nebensatz
SBARQ	Direkte Frage, eingeleitet durch ein <i>wh</i> -Element
SINV	Deklarativsatz mit Subjekt-Hilfsverb-Umkehrung
SQ	Ja-Nein-Fragen und Teilkonstituent von SBARQ, ausgeschlossen <i>wh</i> -Elemente
VP	Verbalphrase
WHADVP	<i>Wh</i> -Adverbialphrase
WHNP	<i>Wh</i> -Nominalphrase
WHPP	<i>Wh</i> -Präpositionalphrase
X	Konstituent unbekannter oder unklarer Kategorie
*	Implizites Subjekt eines Infinitives oder Imperatives
θ	Nebensatz mit Auslassung von <i>that</i>
T	Spur eines <i>wh</i> -Konstituenten

A.3. Semantische Rollen

Etiketten für semantische Rollen basierend auf dem Etikettensatz des CoNLL-2004 bzw. CoNLL-2005 Shared Task [CM04; CM05]:

Etikette	Bedeutung
V	Verb (Prädikat)
A0	Handelnder (Agent)
A1	1. verbbezogene Rolle (meist Patiens oder Thema)
A2	2. verbbezogene Rolle
A3	3. verbbezogene Rolle
A4	4. verbbezogene Rolle
A5	5. verbbezogene Rolle
AA	Weitere verbbezogene Rolle
AM	unspezifizierter Modifikator
AM-ADV	Universalmodifikator
AM-CAU	Ursache
AM-DIR	Richtung
AM-DIS	Diskursmarker
AM-EXT	Ausmaß
AM-LOC	Ort
AM-MNR	Art und Weise
AM-MOD	Modalverb
AM-NEG	Negationsmarker
AM-PNC	Absicht
AM-PRD	Aussage
AM-REC	Umkehrung
AM-TMP	Zeitausdruck
R-A0	Referenz zu A0
R-A1	Referenz zu A1
R-A2	Referenz zu A2
R-A3	Referenz zu A3
R-A4	Referenz zu A4
R-A5	Referenz zu A5
R-AA	Referenz zu AA
R-AM-ADV	Referenz zu AM-ADV
R-AM-CAU	Referenz zu AM-CAU
R-AM-DIR	Referenz zu AM-DIR
R-AM-DIS	Referenz zu AM-DIS
R-AM-EXT	Referenz zu AM-EXT

A. Etikettensätze

Etikette	Bedeutung
R-AM-LOC	Referenz zu AM-LOC
R-AM-MNR	Referenz zu AM-MNR
R-AM-MOD	Referenz zu AM-MOD
R-AM-NEG	Referenz zu AM-NEG
R-AM-PNC	Referenz zu AM-PNC
R-AM-PRD	Referenz zu AM-PRD
R-AM-REC	Referenz zu AM-REC
R-AM-TMP	Referenz zu AM-TMP

B. Stoppwortlisten

Die nachfolgenden Abschnitte listen die für natürliche Sprache und Quelltext verwendeten Stoppwörter auf.

B.1. Stoppwörter für natürliche Sprache

Stoppwörter der **englischen** Sprache wie sie von NLTK¹ verwendet werden:

i	me	my	myself	we	our
ours	ourselves	you	you're	you've	you'll
you'd	your	yours	yourself	yourselves	he
him	his	himself	she	she's	her
hers	herself	it	it's	its	itself
they	them	their	theirs	themselves	
what	which	who	whom	this	that
that'll	these	those	am	is	are
was	were	be	been	being	have
has	had	having	do	does	did
doing	a	an	the	and	but
if	or	because	as	until	while
of	at	by	for	with	about
against	between	into	through	during	before
after	above	below	to	from	up
down	in	out	on	off	over
under	again	further	then	once	here
there	when	where	why	how	all
any	both	each	few	more	most
other	some	such	no	nor	not
only	own	same	so	than	too
very	s	t	can	will	just
don	don't	should	should've	now	d
ll	m	o	re	ve	y
ain	aren	aren't	couldn	couldn't	didn
didn't	doesn	doesn't	hadn	hadn't	hasn

¹ <https://www.nltk.org/>, zuletzt besucht am 26.04.2023.

hasn't	haven	haven't	isn	isn't	ma
mightn	mightn't	mustn	mustn't	needn	needn't
shan	shan't	shouldn	shouldn't	wasn	wasn't
weren	weren't	won	won't	wouldn	wouldn't

Stoppwörter der **italienischen** Sprache wie sie von NLTK² verwendet werden:

ad	al	allo	ai	agli	all
agl	alla	alle	con	col	coi
da	dal	dallo	dai	dagli	dall
dagl	dalla	dalle	di	del	dello
dei	degli	dell	degl	della	delle
in	nel	nello	nei	negli	nell
negl	nella	nelle	su	sul	sullo
sui	sugli	sull	sugl	sulla	sulle
per	tra	contro	io	tu	lui
lei	noi	voi	loro	mio	mia
miei	mie	tuo	tua	tuoi	tue
suo	sua	suoi	sue	nostro	nostra
nostri	nostre	vostro	vostra	vostri	vostre
mi	ti	ci	vi	lo	la
li	le	gli	ne	il	un
uno	una	ma	ed	se	perché
anche	come	dov	dove	che	chi
cui	non	più	quale	quanto	quanti
quanta	quante	quello	quelli	quella	quelle
questo	questi	questa	queste	si	tutto
tutti	a	c	e	i	l
o	ho	hai	ha	abbiamo	avete
hanno	abbia	abbiate	abbiano	avrò	avrà
avrà	avremo	avrete	avranno	avrei	avresti
avrebbe	avremmo	avreste	avrebbero	avevo	avevi
aveva	avevamo	avevate	avevano	ebbi	avesti
ebbe	avemmo	aveste	ebbero	avessi	avesse
avessimo	avessero	avendo	avuto	avuta	avuti
avute	sono	sei	è	siamo	siete
sia	siate	siano	sarò	sarai	sarà
saremo	sarete	saranno	sarei	saresti	sarebbe
saremmo	sareste	sarebbero	ero	eri	era
eravamo	eravate	erano	fui	fosti	fu
fummo	foste	furono	fossi	fosse	fossimo
fossero	essendo	faccio	fai	facciamo	fanno

² <https://www.nltk.org/>, zuletzt besucht am 26.04.2023.

faccia	facciate	facciano	farò	farai	farà
faremo	farete	faranno	farei	faresti	farebbe
faremmo	fareste	farebbero	facevo	facevi	faceva
facevamo	facevate	facevano	feci	facesti	fece
facemmo	faceste	fecero	facessi	facesse	facessimo
facessero	facendo	sto	stai	sta	stiamo
stanno	stia	stiate	stiano	starò	starai
starà	staremo	starete	staranno	starei	staresti
starebbe	staremmo	stareste	starebbero	stavo	stavi
stava	stavamo	stavate	stavano	stetti	stesti
stette	stemmo	steste	stettero	stessi	stesse
stessimo	stessero	stando			

B.2. Quelltextstoppwörter

Programmiersprachenspezifische Stoppwörter und häufige Terme in der Programmierung von Software, welche durch **FTLR** entfernt werden:

abstract	array	assert	bean	boolean	break
byte	case	catch	char	character	class
continue	default	do	double	else	enum
exception	extends	final	finally	float	for
get	goto	hash	if	implements	import
instanceof	int	integer	interface	long	method
native	new	null	package	parameter	private
protected	public	return	servlet	set	short
static	strictfp	string	struct	super	switch
synchronized	this	throw	throws	transient	try
union	unsigned	void	volatile	while	

Für italienischen Quelltext werden zusätzlich die folgenden Übersetzungen der häufigen Terme entfernt:

classe eccezione metodo ottenere settare

C. Javadoc-Etiketten

Etiketten für Javadoc-Dokumentationskommentare [Ora]:

Etikette	Ausgabe	Anwendung in
@author	Beschreibt den Autor	Überblick, Modul, Paket, Typ
{@code}	Stellt Text in Quelltextzeichensatz dar, ohne HTML- oder Javadoc-Etiketten zu interpretieren	Überblick, Modul, Paket, Typ, Konstruktor, Methode, Attribut
@deprecated	Beschreibt veraltete Funktionalität, die nicht mehr verwendet werden sollte	Modul, Typ, Konstruktor, Methode, Attribut
{@docRoot}	Gibt den absoluten Pfad zum Hauptverzeichnis an	Überblick, Modul, Paket, Typ, Konstruktor, Methode, Attribut
@exception / @throws	Beschreibung einer Ausnahme, die von dieser Methode geworfen werden kann	Konstruktor, Methode
@hidden	Versteckt Element in der Dokumentation	Typ, Methode, Attribut
{@index}	Wort oder Phrase die in den Indexdateien auftauchen soll	Überblick, Modul, Paket, Typ, Konstruktor, Methode, Attribut
{@inheritDoc}	Kopiert die Beschreibung aus dem überschriebenen Objekt	Typ, Methode

Etikette	Ausgabe	Anwendung in
<code>{@link}</code>	Verknüpfung zu einem anderen Symbol	Überblick, Modul, Paket, Typ, Konstruktor, Methode, Attribut
<code>{@linkplain}</code>	Die Verknüpfung wird in Klarschrift statt in Quelltextzeichensatz angezeigt	Überblick, Modul, Paket, Typ, Konstruktor, Methode, Attribut
<code>{@literal}</code>	Kennzeichnet buchstabengetreuen Text und unterdrückt die Interpretierung von beinhalteten HTML- oder Javadoc-Tags	Überblick, Modul, Paket, Typ, Konstruktor, Methode, Attribut
<code>@param</code>	Parameterbeschreibung	Typ, Konstruktor, Methode
<code>@provides</code>	Dokumentiert Implementierung die das Modul anbietet	Modul
<code>@return</code>	Beschreibung des Rückgabewertes einer Methode	Methode
<code>@see</code>	Erzeugt eine Verknüpfung auf ein anderes Element der Dokumentation	Überblick, Modul, Paket, Typ, Konstruktor, Methode, Attribut
<code>@serial</code>	Beschreibt die serialisierten Daten eines serialisierbaren Objektes.	Paket, Typ, Attribut
<code>@serialData</code>	Dokumentiert Typen und Reihenfolge der Daten anhand einer Datenbeschreibung	Methode
<code>@serialField</code>	Dokumentiert ein Attribut eines serialisierbaren Objektes	Attribut

Etikette	Ausgabe	Anwendung in
@since	Gibt an, seit wann die Funktionalität existiert	Überblick, Modul, Paket, Typ, Konstruktor, Methode, Attribut
{@snippet}	Inkludiert Quelltextfragmenet in Dokumentation	Überblick, Modul, Paket, Typ, Konstruktor, Methode, Attribut
{@summary}	Kennzeichnet die Zusammenfassung einer API-Beschreibung	Überblick, Modul, Paket, Typ, Konstruktor, Methode, Attribut
{@systemProperty}	Kennzeichnet eine Systemeigenschaft	Modul, Paket, Typ, Konstruktor, Methode, Attribut
@uses	Kennzeichnet durch ein Modul verwendete Dienstgeber	Modul
{@value}	Gibt den Wert eines konstanten Feldes zurück	Überblick, Modul, Paket, Typ, Konstruktor, Methode, Attribut
@version	Erzeugt einen Versionseintrag	Überblick, Modul, Paket, Typ

D. Weitere Informationen zu den Datensätzen

Domänen und erste Veröffentlichungen der von TLR-Ansätzen verwendeten Datensätze:

Datensatz	Domäne	Erste Veröffentlichung
Anforderungen-zu-Quelltext		
LEDA	Algorithmik	[MM03]
Albergate	Tourismus	[Ant+02]
EasyClinic	Gesundheit	[LOS06]
IBS	Verwaltung	[Cle+05b]
EBT	Nachverfolgbarkeit	[CCC03]
LC	Lichtsteuerung	[Cle+05a]
SE450	Verkehr	[ZSC08]
eTour	Tourismus	[TEF11]
iTrust	Gesundheit	[MN11]
SMOS	Bildung	[Get+11]
eAnci	Verwaltung	[Get+11]
Dronology	Luft- und Raumfahrt	[CVB18]
LibEST	Vernetzung	[Mor+20b]
Anforderungen-zu-Anforderungen		
MODIS	Luft- und Raumfahrt	[HDO03]
CM1	Luft- und Raumfahrt	[HDS06]
InfusionPump	Gesundheit	[CoE]
Gantt	Projektmanagement	[HHD09]
WARC	Archivierung	[Kon+12]
CCHIT	Gesundheit	[SC12]

Beispiele für Anforderungen der verwendeten Datensätze

In diesem Abschnitt werden einzelne Anforderungen als Beispiele für Anforderungen aus den für die Evaluation verwendeten Datensätzen dargestellt.

Beispiel 1: eTour - UC57

Use case name: AdvancedSearch

Description: The tourist searching for a site using the potential offered by the Advanced Search.

Participating Actor: initialized by Tourist

Entry conditions: The Tourist has successfully authenticated to the system.

Flow of events User System:

1. Enable the advanced search feature from your personal area.
- 2 View the advanced search form.
- 3 Fill in the form of advanced search and submit.
- 4 Gets the position of relying on the tourist event of the use location and process the request.

Exit conditions: The system displays a list of results.

Interruption of the connection to the server ETOUR.

Quality requirements:

The system requirements into the transaction in more than 15 seconds.

Beispiel 2: iTrust - UC25

A user chooses to view physician satisfaction survey results. The user provides a zip code [E1] Or a hospital code and an (optional) physician type (from a pull-down list: see data format 6.2 - general, surgeon, heart specialist, pediatrician, OB/GYN). The patient is provided with the following for each physician of that type that practices in a zip code (based upon the address/zipcode provided in UC2) that match the first three digits of the provided zip code: Name, address, average number of minutes patients wait in waiting room, average number of minutes patients wait in examination room prior to seeing physician, average office visit satisfaction, average satisfaction with treatment/information and percentage of office visits for which satisfaction information is available.

Beispiel 3: SMOS - SMOS18

Nome: InserisciNuovoIndirizzo

Attori: Amministratore

Descrizione: Inserisce un nuovo indirizzo nell'archivio

Precondizioni:

-
-

L'utente è loggato al sistema come Amministratore L'utente ha già eseguito il caso d'uso "VisualizzaElencoIndirizzi" e il sistema sta visualizzando l'elenco degli indirizzi L'utente clicca sul pulsante "Nuovo Indirizzo"

Sequenza degli eventi

Utente

2. Compila il form 3. Clicca sul pulsante "Salva"

Sistema

1. Il sistema mostra il form da compilare con: nome indirizzo.

4. Effettua dei controlli sulla validità dei dati immessi e inserisce un nuovo indirizzo nell'archivio; nel caso in cui i dati inseriti non sono validi, attiva il caso d'uso "ErroreDati".

Postcondizioni:

-

L'utente ha inserito un indirizzo Viene notificato l'errore dati Connessione al server SMOS interrotta L'Amministratore interrompe l'operazione

Beispiel 4: eAnci - EA67

Nome caso d'uso

ModificaPrenotazioneColloquio

Attori partecipanti

Iniziato da Cittadino

Flusso di eventi

1. Il cittadino richiede la modifica della prenotazione del colloquio nel consultorio.

2. Il sistema visualizza un form per la modifica dei campi della prenotazione.

3. Il cittadino effettua le modifiche nel form conferma la prenotazione.

4. Il sistema modifica la prenotazione.

Condizione di entrata

▲ Questo estende il caso d'uso VisualizzazionePrenotazioneConsultorio considerando il caso in cui il cittadino voglia modificare la prenotazione scelta.

Condizioni di uscita ▲ Il sistema ha effettuato la modifica della prenotazione.

Requisiti di qualità ▲ Non previsti.

Beispiel 5: Albergate - F-GES-04

Requisito:

Configurazione delle opzioni di sistema.

Descrizione:

Alcuni parametri e dati che il sistema utilizza per svolgere le attività per cui è stato realizzato devono poter essere impostate dal gestore dell'albergo. Dovrà essere prevista perciò la possibilità di configurare il sistema in base alle esigenze dell'utente. Ciò che dovrà sicuramente comparire sono le seguenti impostazioni:

date di inizio e fine delle varie stagionalità

sconti e riduzioni da effettuare a neonati e bambini

durata massima per la quale una stanza può restare bloccata

supplemento per le varie stagionalità

costo di uno scatto telefonico

Input richiesto:

I dati di configurazione con le modifiche ritenute opportune dal gestore.

Output desiderato:

Aggiornamento configurazione di sistema.

Criterio di

accettazione:

I dati dovranno influire sulla gestione dell'albergo in base all'area di competenza. Es. se viene previsto uno sconto per i bambini tale sconto dovrà apparire chiaramente nel calcolo del conto finale e nella sua stampa.

Aspettative collegate:

Permettere una maggiore flessibilità al sistema.

Requisiti collegati:

F-PRE-03, F-SOG-03, F-SOG-04, F-GES-01

Beispiel 6: LibEST - RQ8

REQUIREMENT 8: HTTP-BASED CLIENT AUTHENTICATION

The EST server can optionally also request that the EST client submit a username/password using the HTTP Basic or Digest authentication methods (see Section 3.2.3). This approach is desirable if the EST client cannot be authenticated during the TLS handshake (see Section 3.3.2) or the EST server policy requires additional authentication information; see Section 3.2.3. In all cases, HTTP-based client authentication is only to be performed over a TLS-protected transport (see Section 3.3).

E. Weitere Evaluationsergebnisse

In den folgenden Abschnitten und Tabellen werden über die im Hauptteil der Arbeit enthaltenen Ergebnisse hinausgehende Ergebnisse aus den Evaluationen der einzelnen Beiträge dargestellt.

Tabelle E.1.: Vergleich der Ergebnisse der Merkmalsvarianten mit angepassten festen Schwellenwerten (Mehrheit: 0,580, Abschluss: 0,475). \rightleftharpoons markiert Varianten, die Aufrufabhängigkeiten nutzen, \bullet Varianten, die Methodenkommentare verwenden und \blacktriangledown Varianten, die Anwendungsfallvorlagen miteinbeziehen

Projekt	Maß	FTLR-Varianten							
		\rightleftharpoons	\bullet	\blacktriangledown	\rightleftharpoons \bullet	\rightleftharpoons \blacktriangledown	\bullet \blacktriangledown	\rightleftharpoons \bullet \blacktriangledown	
eTour	Präz.	.232	.235	.234	.356	.239	.365	.342	.350
	Ausb.	.753	.750	.763	.701	.773	.695	.708	.708
	F ₁	.355	.358	.358	.472	.365	.478	.461	.468
iTrust	Präz.	.104	.109	.105	.104	.108	.109	.105	.108
	Ausb.	.416	.399	.469	.416	.448	.399	.469	.448
	F ₁	.166	.171	.172	.166	.174	.171	.172	.174
SMOS	Präz.	.349	.348	.354	.383	.353	.381	.392	.393
	Ausb.	.380	.379	.425	.357	.425	.353	.388	.387
	F ₁	.364	.363	.387	.370	.386	.367	.390	.390
eAnci	Präz.	.218	.219	.189	.273	.189	.274	.232	.232
	Ausb.	.284	.286	.337	.257	.335	.257	.317	.316
	F ₁	.247	.248	.242	.265	.242	.265	.268	.268
Albergate	Präz.	.246	.246	.246	.227	.246	.227	.227	.227
	Ausb.	.528	.528	.528	.283	.528	.283	.283	.283
	F ₁	.335	.335	.335	.252	.335	.252	.252	.252
LibEST	Präz.	.422	.422	.409	.422	.409	.422	.409	.409
	Ausb.	.691	.691	.784	.691	.784	.691	.784	.784
	F ₁	.524	.524	.538	.524	.538	.524	.538	.538
	\emptyset F ₁	.332	.333	.339	.342	.340	.343	.347	.348

E.1. Vergleich mit Referenzimplementierungen

Tabelle E.2.: Vergleich der erzielten Ergebnisse der Referenzimplementierung, welche Kosinusähnlichkeit auf Elementebene anwendet (ECosS), mit **FTLR**

Projekt	Maß	📄			💬			🔄💬		
		OPT			OPT			OPT		
		FTLR	FTLR	ECosS	FTLR	FTLR	ECosS	FTLR	FTLR	ECosS
eTour	Präz.	.376	.505	.551	.373	.472	.466	.379	.479	.430
	Ausb.	.643	.597	.435	.633	.571	.422	.633	.562	.461
	F ₁	.475	.548	.486	.469	.517	.443	.474	.517	.445
	MAP	.528	.528	.499	.483	.483	.430	.483	.483	.422
iTrust	Präz.	.135	.193	.288	.142	.252	.358	.149	.190	.300
	Ausb.	.322	.241	.234	.374	.255	.220	.346	.322	.248
	F ₁	.190	.215	.258	.206	.253	.273	.209	.239	.272
	MAP	.227	.227	.194	.284	.284	.249	.271	.271	.232
SMOS	Präz.	.425	.313	.299	.432	.380	.282	.440	.369	.286
	Ausb.	.314	.590	.521	.340	.466	.591	.335	.484	.598
	F ₁	.361	.409	.380	.381	.419	.382	.381	.419	.387
	MAP	.438	.438	.379	.451	.451	.407	.451	.451	.406
eAnci	Präz.	.312	.311	.307	.273	.255	.251	.274	.266	.229
	Ausb.	.226	.272	.243	.254	.300	.224	.243	.287	.254
	F ₁	.262	.290	.271	.263	.276	.237	.258	.277	.241
	MAP	.151	.151	.150	.154	.154	.149	.153	.153	.149
Albergate	Präz.	.171	.284	.145	.171	.284	.145	.171	.284	.145
	Ausb.	.132	.509	.226	.132	.509	.226	.132	.509	.226
	F ₁	.149	.365	.176	.149	.365	.176	.149	.365	.176
	MAP	.406	.406	.230	.406	.406	.230	.406	.406	.230
LibEST	Präz.	.403	.421	.402	.431	.406	.526	.431	.406	.526
	Ausb.	.490	.799	.784	.623	.863	.642	.623	.863	.642
	F ₁	.442	.552	.532	.509	.552	.578	.509	.552	.578
	MAP	.559	.559	.584	.581	.581	.658	.581	.581	.658
	∅ F ₁	.313	.396	.351	.329	.397	.348	.330	.395	.350
	∅ MAP	.385	.385	.339	.393	.393	.354	.391	.391	.350

Tabelle E.3.: Vergleich der erzielten Ergebnisse der Referenzimplementierung, welche Kosinusähnlichkeit auf Artefaktebene anwendet (ACosS), mit **FTLR**

Projekt	Maß	🗨️			👇			🗨️👇		
		FTLR	OPT		FTLR	OPT		FTLR	OPT	
			FTLR	ACosS		FTLR	ACosS		FTLR	ACosS
eTour	Prüz.	.267	.323	.422	.376	.505	.434	.373	.472	.464
	Ausb.	.688	.623	.354	.643	.597	.377	.633	.571	.357
	F ₁	.385	.425	.385	.475	.548	.403	.469	.517	.404
	MAP	.349	.349	.369	.528	.528	.401	.483	.483	.377
iTrust	Prüz.	.142	.252	.224	.135	.193	.179	.142	.252	.224
	Ausb.	.374	.255	.287	.322	.241	.196	.374	.255	.287
	F ₁	.206	.253	.252	.190	.215	.187	.206	.253	.252
	MAP	.284	.284	.253	.227	.227	.177	.284	.284	.253
SMOS	Prüz.	.411	.405	.278	.425	.313	.288	.432	.380	.310
	Ausb.	.356	.406	.584	.314	.590	.549	.340	.466	.527
	F ₁	.382	.405	.377	.361	.409	.378	.381	.419	.390
	MAP	.418	.418	.362	.438	.438	.378	.451	.451	.412
eAnci	Prüz.	.204	.193	.191	.312	.311	.254	.273	.255	.168
	Ausb.	.291	.381	.220	.226	.272	.250	.254	.300	.335
	F ₁	.240	.256	.205	.262	.290	.252	.263	.276	.224
	MAP	.157	.157	.139	.151	.151	.141	.154	.154	.146
Albergate	Prüz.	.208	.254	.116	.171	.284	.169	.171	.284	.169
	Ausb.	.302	.547	.604	.132	.509	.396	.132	.509	.396
	F ₁	.246	.347	.195	.149	.365	.237	.149	.365	.237
	MAP	.447	.447	.269	.406	.406	.336	.406	.406	.336
LibEST	Prüz.	.431	.406	.512	.403	.421	.288	.431	.406	.559
	Ausb.	.623	.863	.623	.490	.799	.980	.623	.863	.583
	F ₁	.509	.552	.562	.442	.552	.445	.509	.552	.571
	MAP	.581	.581	.669	.559	.559	.485	.581	.581	.666
	∅ F ₁	.328	.373	.329	.313	.396	.317	.329	.397	.346
	∅ MAP	.373	.373	.343	.385	.385	.320	.393	.393	.365

Tabelle E.4.: Vergleich der erzielten Ergebnisse der Referenzimplementierung, welche die Wortüberferungs-
distanz auf Artefaktebene anwendet (AWMD), mit **FTLR**

Projekt	Maß	🗨️			🔍			🗨️🔍		
		OPT			OPT			OPT		
		FTLR	FTLR	AWMD	FTLR	FTLR	AWMD	FTLR	FTLR	AWMD
eTour	Präz.	.267	.323	.516	.376	.505	.496	.373	.472	.429
	Ausb.	.688	.623	.377	.643	.597	.367	.633	.571	.438
	F ₁	.385	.425	.435	.475	.548	.422	.469	.517	.433
	MAP	.349	.349	.480	.528	.528	.512	.483	.483	.482
iTrust	Präz.	.142	.252	.238	.135	.193	.170	.142	.252	.238
	Ausb.	.374	.255	.301	.322	.241	.311	.374	.255	.301
	F ₁	.206	.253	.266	.190	.215	.220	.206	.253	.266
	MAP	.284	.284	.283	.227	.227	.220	.284	.284	.283
SMOS	Präz.	.411	.405	.417	.425	.313	.375	.432	.380	.423
	Ausb.	.356	.406	.431	.314	.590	.493	.340	.466	.473
	F ₁	.382	.405	.424	.361	.409	.426	.381	.419	.447
	MAP	.418	.418	.469	.438	.438	.453	.451	.451	.476
eAnci	Präz.	.204	.193	.240	.312	.311	.298	.273	.255	.300
	Ausb.	.291	.381	.252	.226	.272	.291	.254	.300	.282
	F ₁	.240	.256	.246	.262	.290	.294	.263	.276	.291
	MAP	.157	.157	.149	.151	.151	.146	.154	.154	.146
Albergate	Präz.	.208	.254	.229	.171	.284	.189	.171	.284	.189
	Ausb.	.302	.547	.358	.132	.509	.264	.132	.509	.264
	F ₁	.246	.347	.279	.149	.365	.220	.149	.365	.220
	MAP	.447	.447	.287	.406	.406	.268	.406	.406	.268
LibEST	Präz.	.431	.406	.543	.403	.421	.325	.431	.406	.543
	Ausb.	.623	.863	.529	.490	.799	.917	.623	.863	.529
	F ₁	.509	.552	.536	.442	.552	.480	.509	.552	.536
	MAP	.581	.581	.662	.559	.559	.492	.581	.581	.662
	∅ F ₁	.328	.373	.364	.313	.396	.344	.329	.397	.366
	∅ MAP	.373	.373	.388	.385	.385	.348	.393	.393	.386

E.2. Evaluationsergebnisse mit Anforderungsfiltern

Tabelle E.5.: Vergleich der Anforderungsfiltervarianten mit den ungefilterten **FTLR**-Ergebnissen bei optimierten Schwellenwerten. \rightleftarrows markiert Varianten, die Aufrufabhängigkeiten und \bullet , die Methodenkommentare verwenden. NF bezeichnet Varianten, die Anforderungselemente ohne funktionale Aspekte und NB Varianten, die Nutzerbezogenes filtern

FTLR-Variante:					\bullet			\rightleftarrows		
Projekt	Filter	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁
eTour	Ohne	.331	.656	.440	.323	.623	.425	.325	.623	.427
	NF	.376	.607	.464	.354	.588	.442	.350	.601	.442
	NB	.377	.620	.469	.364	.584	.448	.366	.581	.449
	NF + NB	.378	.617	.469	.367	.584	.451	.369	.581	.451
iTrust	Ohne	.193	.241	.215	.252	.255	.253	.190	.322	.239
	NF	.234	.241	.238	.271	.252	.261	.220	.290	.250
	NB	.137	.224	.170	.182	.199	.190	.161	.210	.182
	NF + NB	.155	.231	.185	.182	.220	.199	.183	.206	.194
SMOS	Ohne	.288	.629	.396	.405	.406	.405	.380	.433	.405
	NF	.298	.620	.402	.369	.461	.410	.380	.448	.411
	NB	.330	.527	.405	.374	.455	.411	.375	.461	.414
	NF + NB	.330	.527	.406	.375	.455	.411	.376	.461	.414
eAnci	Ohne	.214	.333	.261	.193	.381	.256	.188	.395	.255
	NF	.245	.328	.281	.215	.363	.271	.216	.358	.270
	NB	.327	.268	.295	.299	.296	.298	.302	.287	.295
	NF + NB	.336	.259	.293	.299	.291	.295	.290	.295	.292
LibEST	Ohne	.421	.799	.552	.406	.863	.552	.406	.863	.552
	NF	.438	.750	.553	.476	.667	.555	.476	.667	.555
	NB	.383	.794	.517	.440	.725	.548	.440	.725	.548
	NF + NB	.375	.725	.494	.422	.681	.522	.422	.681	.522
∅	Ohne	.290	.532	.372	.316	.506	.378	.298	.527	.375
	NF	.318	.509	.388	.337	.466	.388	.328	.473	.386
	NB	.311	.487	.371	.332	.452	.379	.329	.453	.378
	NF + NB	.315	.472	.369	.329	.446	.375	.328	.445	.375

Tabelle E.6.: Vergleich der Anforderungsfiltervarianten auf Basis des Goldstandards mit den ungefilterten **FTLR**-Ergebnissen bei optimierten Schwellenwerten. \rightleftarrows markiert Varianten, die Aufrufabhängigkeiten einbeziehen und \bullet , die Methodenkommentare verwenden. NF bezeichnet Varianten, die Anforderungselemente ohne funktionale Aspekte und NB Varianten, die Nutzerbezogenes filtern

Projekt	FTLR-Variante: Filter				\bullet			$\bullet \rightleftarrows$		
		Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁
eTour	Ohne	.331	.656	.440	.323	.623	.425	.325	.623	.427
	NF	.500	.601	.546	.474	.568	.517	.472	.568	.515
	NB	.508	.604	.552	.492	.571	.529	.494	.565	.527
	NF + NB	.508	.604	.552	.492	.571	.529	.494	.565	.527
iTrust	Ohne	.193	.241	.215	.252	.255	.253	.190	.322	.239
	NF	.231	.238	.234	.214	.329	.259	.218	.290	.249
	NB	.144	.192	.165	.178	.185	.182	.194	.150	.169
	NF + NB	.156	.192	.172	.191	.178	.184	.183	.175	.179
SMOS	Ohne	.288	.629	.396	.405	.406	.405	.380	.433	.405
	NF	.367	.438	.399	.388	.442	.413	.401	.428	.414
	NB	.328	.545	.409	.407	.429	.418	.380	.466	.418
	NF + NB	.328	.545	.409	.406	.428	.417	.337	.549	.418
eAnci	Ohne	.214	.333	.261	.193	.381	.256	.188	.395	.255
	NF	.264	.323	.290	.227	.362	.279	.222	.369	.277
	NB	.330	.270	.297	.307	.286	.296	.339	.261	.295
	NF + NB	.386	.242	.297	.309	.286	.297	.341	.261	.296
LibEST	Ohne	.421	.799	.552	.406	.863	.552	.406	.863	.552
	NF	.433	.770	.554	.405	.873	.553	.405	.873	.553
	NB	.419	.789	.548	.411	.838	.552	.411	.838	.552
	NF + NB	.431	.765	.551	.435	.755	.552	.435	.755	.552
∅	Ohne	.290	.532	.372	.316	.506	.378	.298	.527	.375
	NF	.359	.474	.405	.342	.515	.404	.344	.506	.402
	NB	.346	.480	.394	.359	.462	.395	.364	.456	.392
	NF + NB	.362	.470	.396	.366	.444	.396	.358	.461	.394

Tabelle E.7.: Vergleich der Ergebnisse der Anforderungsfiltervarianten mit **FTLR** mit Anwendungsfallvorlagenfilter auf optimierten Schwellenwerten. \rightleftharpoons markiert Varianten, die Aufrufabhängigkeiten einbeziehen, \bullet , die Methodenkommentare verwenden und \blacktriangledown , die das Filtern von Vorlagenelemente nutzen. NF bezeichnet Varianten, die Anforderungselemente ohne funktionale Aspekte und NB Varianten, die Nutzerbezogenes filtern

FTLR-Variante:		\rightleftharpoons			\bullet			$\bullet\rightleftharpoons$		
Projekt	Filter	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁
eTour	\blacktriangledown	.508	.604	.552	.472	.571	.517	.479	.562	.517
	NF	.378	.614	.468	.354	.588	.442	.350	.601	.442
	NB	.381	.614	.470	.364	.584	.448	.366	.581	.449
	NF + NB	.383	.610	.471	.367	.584	.451	.369	.581	.451
iTrust	\blacktriangledown	.227	.196	.210	.252	.255	.253	.190	.322	.239
	NF	.253	.206	.227	.271	.252	.261	.220	.290	.250
	NB	.131	.224	.166	.182	.199	.190	.161	.210	.182
	NF + NB	.160	.203	.179	.182	.220	.199	.183	.206	.194
SMOS	\blacktriangledown	.334	.546	.415	.380	.466	.419	.369	.484	.419
	NF	.301	.603	.402	.369	.461	.410	.380	.448	.411
	NB	.334	.524	.408	.374	.455	.411	.375	.461	.414
	NF + NB	.334	.524	.408	.375	.455	.411	.376	.461	.414
eAnci	\blacktriangledown	.287	.296	.292	.255	.300	.276	.266	.287	.277
	NF	.263	.305	.282	.215	.363	.271	.216	.358	.270
	NB	.345	.256	.294	.299	.296	.298	.302	.287	.295
	NF + NB	.344	.250	.290	.299	.291	.295	.290	.295	.292
LibEST	\blacktriangledown	.421	.799	.552	.406	.863	.552	.406	.863	.552
	NF	.438	.750	.553	.476	.667	.555	.476	.667	.555
	NB	.383	.794	.517	.440	.725	.548	.440	.725	.548
	NF + NB	.375	.725	.494	.422	.681	.522	.422	.681	.522
∅	\blacktriangledown	.355	.488	.404	.353	.491	.403	.342	.503	.401
	NF	.327	.496	.386	.337	.466	.388	.328	.473	.386
	NB	.315	.482	.371	.332	.452	.379	.329	.453	.378
	NF + NB	.319	.463	.368	.329	.446	.375	.328	.445	.375

Tabelle E.8.: Vergleich der Ergebnisse der Anforderungsfiltervarianten auf Basis des Goldstandards mit **FTLR** mit Anwendungsfallvorlagenfilter auf optimierten Schwellenwerten. \rightleftharpoons markiert Varianten, die Aufrufabhängigkeiten einbeziehen, \bullet Varianten, die Methodenkommentare verwenden und \blacktriangledown welche, die das Filtern von Vorlagenelemente nutzen. NF bezeichnet Varianten, die Anforderungselemente ohne funktionale Aspekte und NB Varianten, die Nutzerbezogenes filtern

FTLR-Variante:		\rightleftharpoons			\bullet			$\bullet\rightleftharpoons$		
Projekt	Filter	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁	Präz.	Ausb.	F ₁
eTour	\blacktriangledown	.508	.604	.552	.472	.571	.517	.479	.562	.517
	NF	.508	.594	.548	.474	.568	.517	.472	.568	.515
	NB	.512	.601	.553	.492	.571	.529	.494	.565	.527
	NF + NB	.512	.601	.553	.492	.571	.529	.494	.565	.527
iTrust	\blacktriangledown	.227	.196	.210	.252	.255	.253	.190	.322	.239
	NF	.245	.206	.224	.214	.329	.259	.218	.290	.249
	NB	.132	.199	.159	.178	.185	.182	.194	.150	.169
	NF + NB	.151	.189	.168	.191	.178	.184	.183	.175	.179
SMOS	\blacktriangledown	.334	.546	.415	.380	.466	.419	.369	.484	.419
	NF	.316	.563	.405	.388	.442	.413	.401	.428	.414
	NB	.335	.545	.415	.407	.429	.418	.380	.466	.418
	NF + NB	.335	.545	.415	.406	.428	.417	.337	.549	.418
eAnci	\blacktriangledown	.287	.296	.292	.255	.300	.276	.266	.287	.277
	NF	.289	.300	.294	.227	.362	.279	.222	.369	.277
	NB	.339	.265	.297	.307	.286	.296	.339	.261	.295
	NF + NB	.339	.265	.297	.309	.286	.297	.341	.261	.296
LibEST	\blacktriangledown	.421	.799	.552	.406	.863	.552	.406	.863	.552
	NF	.433	.770	.554	.405	.873	.553	.405	.873	.553
	NB	.419	.789	.548	.411	.838	.552	.411	.838	.552
	NF + NB	.431	.765	.551	.435	.755	.552	.435	.755	.552
∅	\blacktriangledown	.355	.488	.404	.353	.491	.403	.342	.503	.401
	NF	.358	.487	.405	.342	.515	.404	.344	.506	.402
	NB	.347	.480	.394	.359	.462	.395	.364	.456	.392
	NF + NB	.354	.473	.397	.366	.444	.396	.358	.461	.394

E.3. Auswirkungen der Übersetzung auf FTLR

Tabelle E.9.: Vergleich der Auswirkung einer automatischen Übersetzung auf **FTLRs** Leistung mit *fastText*-Worteinbettungen und optimierten Schwellenwerten

Datensatz				🗨️			⚡			🗨️⚡		
	P.	A.	F ₁	P.	A.	F ₁	P.	A.	F ₁	P.	A.	F ₁
SMOS	.288	.629	.396	.405	.406	.405	.313	.590	.409	.380	.466	.419
SMOS _{übers.}	.280	.533	.367	.282	.533	.368	.347	.432	.385	.356	.441	.394
eAnci	.214	.333	.261	.193	.381	.256	.311	.272	.290	.255	.300	.276
eAnci _{übers.}	.252	.265	.258	.257	.250	.254	.459	.206	.285	.399	.208	.273

E.4. Evaluationsergebnisse mit UniXcoder

Tabelle E.10.: Vergleich von **FTLRs** Leistung mit *UniXcoder*. Varianten, die Kosinusähnlichkeit nutzen, sind mit KOS und Varianten, die die Wortüberferungsdistanz nutzen, mit WMD gekennzeichnet. UXC bezeichnet **FTLR** mit *UniXcoder* und vollständiger Vorverarbeitung und UXC* mit reduzierter Vorverarbeitung

Ansatz		eTour		iTrust		SMOS		eAnci		LibEST		∅			
		F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP	F ₁	MAP		
⇕	KOS	UXC	.415	.379	.206	.228	.327	.311	.287	.171	.556	.589	.358	.336	
		UXC*	.401	.394	.230	.227	.299	.329	.294	.177	.583	.652	.362	.356	
	WMD	UXC	.404	.390	.219	.214	.317	.316	.231	.155	.585	.619	.351	.339	
		UXC*	.407	.451	.122	.229	.312	.329	.236	.154	.574	.633	.330	.359	
☛	KOS	UXC	.415	.350	.228	.227	.335	.317	.260	.165	.567	.561	.361	.324	
		UXC*	.347	.339	.294	.271	.293	.318	.265	.160	.585	.664	.357	.350	
	⇕	WMD	UXC	.396	.347	.233	.246	.347	.337	.215	.146	.595	.588	.357	.333
			UXC*	.398	.431	.271	.291	.323	.337	.232	.148	.595	.647	.364	.371
☛	KOS	UXC	.520	.500	.206	.228	.359	.370	.347	.173	.556	.589	.398	.372	
		UXC*	.406	.416	.230	.227	.298	.340	.346	.187	.583	.652	.373	.364	
	⇕	WMD	UXC	.446	.486	.219	.214	.349	.395	.224	.155	.574	.504	.362	.351
			UXC*	.378	.468	.122	.229	.316	.354	.208	.144	.663	.561	.337	.351
☛	KOS	UXC	.483	.460	.228	.227	.374	.383	.317	.173	.567	.561	.394	.361	
		UXC*	.365	.356	.294	.271	.299	.331	.295	.162	.585	.664	.368	.357	
	⇕	WMD	UXC	.436	.438	.233	.246	.378	.425	.235	.153	.501	.455	.357	.343
			UXC*	.366	.471	.271	.291	.325	.366	.212	.132	.549	.545	.345	.361