Karlsruhe Institute of Technology

Three-Way Analysis for a Better Understanding of Word Embedding Models

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Ábel Elekes

Tag der mündlichen Prüfung: 11.01.2021

Erster Gutachter: Prof. Dr. Klemens Böhm
Zweiter Gutachter: Prof. Dr. Wolf-Tilo Balke

Abstract

Word embedding models have great value in a wide variety of problems in Computer Science, especially in Natural Language Processing. By focusing on the semantic contexts of words, they allow to understand relationships between text segments in more accurate, less biased ways. The motivation of this work is the fact that several aspects of word embedding models, such as how those 'relationships' should be interpreted and how far different embedding models and approaches are comparable, are still not sufficiently clear. We group these aspects into three categories: algorithmic, theoretic and application questions. In this work, we present a three-way analysis of word embedding models based on these categories. These analyses are built on top of each other. Without first understanding the algorithmic questions, we are not able to answer the theoretical ones. Similarly, to answer the questions regarding word embedding model applications, we need to understand their theoretical aspects better.

The first part of the three-way analysis investigates the training algorithm of embedding models. Previous literature used word embedding models without taking into consideration the fact that their similarity value distributions can be significantly different. Our first contribution is to show that models trained with different parameter settings can be significantly different in the sizes of their similarity values, but at the same time, the shape of their distribution is indeed fundamentally similar.

One big advantage of embedding models is that they can be trained on arbitrary text corpora. While the quality of word embedding models trained on full-text corpora is fairly well known, an assessment of models built on fragmented corpora is missing. To fill this void, in the second part of the algorithmic analysis, we describe experiments to examine how model quality changes when the training corpus is not full-text, but n-grams. The experiments quantify how much fragmentation (i.e., values of n) reduce the average quality of the corresponding word embedding models, on common word similarity and analogical reasoning test sets.

The second part of our analysis is regarding the similarity values of word embedding models from a theoretical point of view. We investigate questions such as: What do similarity values of those models actually mean? For example, if word A is more than 0.5-similar to word B, then are A and B always semantically similar?

We answer these and other equally important questions regarding similarity values. We identify meaningful similarity thresholds, i.e., similarity values and similarity list indices that separate relevant word pairs from irrelevant ones. Based on these thresholds, we

propose a similarity threshold aware evaluation method of word embedding models, one that does not compare the word pairs which fall below the calculated threshold. This ensures a more reliable comparison of word embedding models in the future.

Our final analysis addresses the applicability of word embedding models in down-stream applications. Because of their understanding of words one of the most intuitive use case for word embedding models is text classification, i.e., to assign predefined categories to text documents. Usually, the more labeled samples a text classifier is presented with, the better it will be able to predict categories of unknown samples. However, in most cases the majority of data is unlabeled and labeling it is a costly and time consuming process.

In this work, we explore the possibility to improve the quality of text classification despite the scarcity of labeled data. We present a lexical substitution approach for preprocessing that compensates the scarcity of labeled data. It is an orthogonal extension to virtually any existing text classification approach, to improve classification accuracy. Our approach replaces words unknown to the classifier with known ones for statistical robustness, based on the main contribution of this part: a novel semantic-distributional word distance measure that includes both semantic information extracted from word embedding models and distributional information extracted from the training data. This is the first time to use the combination of these two information for text classification.

Zusammenfassung

Word Embedding Models sind für eine Vielzahl von Problemen in der Informatik von großem Wert, insbesondere für die natürliche Sprachverarbeitung. Sie ermöglichen, indem sie sich auf die semantischen Kontexte von Wörtern konzentrieren, Beziehungen zwischen Textsegmenten genauer und mit weniger Verzerrung zu verstehen. Die Motivation dieser Arbeit ist die Tatsache, dass einige Aspekte von Word Embedding Models, zum Beispiel wie diese 'Beziehungen' interpretiert werden sollten und wie weit verschiedene embedding models vergleichbar sind, noch nicht klar genug erfasst worden sind. Wir gruppieren diese Aspekte in drei Kategorien: in algorithmische Fragen, theoretische Fragen und Anwendungsfragen. Auf diesen drei Kategorien basierend präsenieren wir in dieser Arbeit eine Drei-Wege-Bewertung von Word Embedding Models.

Der erste Bewertungssatz untersucht den Trainingsalgorithmus von Word Embedding Models. In der bisherigen Literatur wurden Word Embedding Models verwendet, ohne die Tatsache zu berücksichtigen, dass ihre Ähnlichkeitswertverteilungen erheblich unterschiedlich sein können. Unser erster Beitrag besteht darin, zu zeigen, dass Modelle, die mit unterschiedlichen Parametereinstellungen trainiert wurden, sich in der Größe ihrer Ähnlichkeitswerte erheblich unterscheiden können, obwohl gleichzeitig die Form ihrer Verteilung tatsächlich grundlegend ähnlich ist.

Ein großer Vorteil der Embedding Models besteht darin, dass sie auf beliebigen Textkorpora trainiert werden können. Während die Qualität von Word Embedding Models, die auf Volltextkorpora trainiert wurden, ziemlich bekannt ist, fehlt eine Bewertung von Modellen, die auf fragmentierten Korpora basieren. Um diese Lücke zu schließen, beschreiben wir im zweiten Teil des Abschnitts zur algorithmischen Bewertung Experimente, deren Ziel es ist zu untersuchen, wie sich die Modellqualität dann ändert, wenn der Trainingskorpus nicht Volltext, sondern n-Gramm ist. Die Experimente quantifizieren, um wie viel Fragmentierung (d. h. Werte von n) die durchschnittliche Qualität der entsprechenden Word Embedding Models auf der Basis gemeinsamer Wortähnlichkeit und analoger Argumentationstestsätze verringert.

Der zweite Bewertungssatz betrifft die Ähnlichkeitswerte von Word Embedding Models aus theoretischer Sicht. Wir untersuchen Fragen wie: Was bedeuten Ähnlichkeitswerte dieser Modelle tatsächlich? Wenn beispielsweise Wort A Wort B mehr als 0,5 ähnlich ist, sind A und B dann immer semantisch ähnlich?

Wir beantworten diese und andere ebenso wichtige Fragen zu Ähnlichkeitswerten. Wir identifizieren sinnvolle Ähnlichkeitsschwellen, d. h. Ähnlichkeitswerte und Ähnlichkeits-

listenindizes, die relevante Wortpaare von irrelevanten trennen. Basierend auf diesen Schwellenwerten schlagen wir eine Bewertungsmethode für Word Embedding Models vor, bei der die Wortpaare, die unter den berechneten Schwellenwert fallen, nicht verglichen werden. Dies gewährleistet in Zukunft einen zuverlässigeren Vergleich von Word Embedding Models.

Unsere abschließende Bewertung befasst sich mit der Anwendbarkeit von Word Embedding Models in nachgeschalteten Anwendungen. Aufgrund ihres Verständnisses von Wörtern ist die Textklassifizierung, d. h. das Zuweisen vordefinierter Kategorien zu Textdokumenten, einer der intuitivsten Anwendungsfälle für Word Embedding Models. Je mehr beschriftete Stichproben einem Textklassifizierer präsentiert werden, desto besser können normalerweise Kategorien unbekannter Stichproben vorhergesagt werden. In den meisten Fällen ist der Großteil der Daten jedoch unbeschriftet, und die Kennzeichnung ein kostspieliger und zeitaufwändiger Prozess.

In dieser Arbeit untersuchen wir die Möglichkeit, die Qualität der Textklassifizierung trotz des Mangels an gekennzeichneten Daten zu verbessern. Wir präsentieren einen lexikalischen Substitutionsansatz für die Vorverarbeitung, der die Knappheit markierter Daten kompensiert. Es ist eine orthogonale Erweiterung für praktisch jeden vorhandenen Textklassifizierungsansatz, um die Klassifizierungsgenauigkeit zu verbessern. Unser Ansatz ersetzt Wörter, die dem Klassifizierer unbekannt sind, durch bekannte Wörter für statistische Robustheit, basierend auf dem Hauptbeitrag dieses Teils: einem neuartigen semantisch verteilten Wortabstandsmaß, das sowohl semantische Informationen aus Word Embedding Models als auch Verteilungsinformationen aus den Trainingsdaten enthält . Dies ist das erste Mal, dass die Kombination dieser beiden Informationen für die Textklassifizierung verwendet wird.

Acknowledgments

This thesis is more than just the work of one person, and I would like to express my gratitude to everyone who has helped me along the way.

First of all, I would like to thank my supervisor, Prof. Klemens Böhm, for his advice and support. His confidence in me from the very beginning has allowed me to pursue a PhD in a foreign country. Furthermore, I'm grateful for the fundamental guidance as well as the scientific freedom he has given me over the last four years.

Likewise, I would like to thank Martin Schäler, who has contributed just as much to the supervision of my thesis. He has been a great source of inspiration throughout the years. In particular I would like to thank you for both the strong scientific and moral support.

I am very grateful to Prof. Wolf-Tilo Balke, for agreeing to act as an external supervisor for this thesis. Thank you very much for your interest in my research.

In addition to the direct support from my supervisors, I have also received a lot of support from my colleagues at our chair. In particular I would like to thank Saeed Taghizadeh and Elaheh Ordoni for the love they showed me.

I also would like to thank my family and all my friends for supporting me throughout this journey.

Contents

List of Figures

List of Tables

Part I.

Introduction and Fundamentals

1. Introduction

1.1. Motivation

Natural Language Processing (NLP) is a highly important field both in Computer Science (CS) and Artificial Intelligence (AI). Subtopics of NLP ranges from parsing natural language inputs, such as speech or optical characters, through understanding language semantics and syntax, to down-stream applications, such as machine translation or automatic summarization. NLP solutions have found their way into our everyday lives with applications, such as Google's search engine or translation service, or recently even more visibly with gadgets, such as Amazon's Alexa. Also, more important than ever, state-of-the-art applications, which track global pandemics or natural disasters are mostly NLP-based [36, 95]. These applications leverage the vast amount of textual data generated by billions of users on various platforms to predict, for example the spread of the Ebola virus or the epicenter of locust invasions, based on social media posts or Google search patterns.

Similarly, as images and acoustic waves are modeled in computer systems, we need to find a way to represent text data in order to process it automatically. For example, the sentence "The cat sat on the mat." cannot be understood directly by the computer. However, there are several methods to make it processable. The easiest way to represent a text document, i.e., sentence, paragraph or larger text segments, is through a sparse discrete vector $(i_{cat}, 1), (i_{the}, 2), \ldots$, where i_v denotes the index of word v in the vocabulary, and the number next to it is its frequency. This is called one-hot encoding. However, there are several disadvantages in the case of this simple model. For instance, it generates high dimensional vectors whose length depends on the size of the vocabulary, hence it changes when using different training data. The dimensionality is also usually very large. Because of the complexity of natural language, even for very small datasets the size of the vocabulary may be several thousand words. Nevertheless, one-hot encoding can be used to compare documents by similarity based on their word frequency distributions and it is a good baseline method to compare more advanced models to [1, 12, 21, 24].

For the computer to understand natural language the next step in granularity is to grasp the semantics not only of documents but distinct words. This means, we want to capture and quantitatively measure the similarity of words. Based on the idea of one-hot encoding of documents we intend to represent the words in a vector space. This is called the problem of word embedding, which concept was first introduced by Hinton in 1986 [44]. Word embedding models, sometimes named as word representation, is a collective name for a

set of language models and feature selection methods. Its main goal is to map ("embed") textual words or phrases into a fixed-size low-dimensional continuous space.

The main objective of such word embeddings is to encode the semantic and syntactic information of words, where semantic information mainly correlates with the meaning of words, while syntactic information refers to their structural roles in language [55, 108]. If one is able to quantify their similarity, there will be a good understanding of the actual meaning of a word, by knowing which words are similar to it.

In word embedding models, each word in the corpus is mapped to a d-dimensional vector. Using the example above, "cat" could be denoted as $[0.17, 0.72, 0.35, ...] \in R^d$ and "mat" can be expressed as $[0.4, 0.05, 0.93, ...] \in R^d$, where d is a pre-selected hyper-parameter. These vectors feature the similarity property, and for some models other properties as well, explained in the following. Similarity means that representations of similar words are close to each other in the vector space, based on a distance function, such as the cosine distance. For example, close to "apple" there are words related to the company, but also words related to the fruit. This attribute of an embedding model makes it useful in more complex tasks as well, since many word meaning phenomena, such as synonymy, priming or categorization, can be described in terms of semantic similarity.

Modern embedding models can do more than just predict semantic similarity. For example, some models feature analogy properties. This can be described by the following example: "man" is to "woman" is like "king" is to "queen". In the embedding space of these models, subtracting the vector representing "man" from the one representing "king" and adding the one representing "woman" results in a vector being close to the vector of "queen" [55, 85, 83, 51].

The question is how to obtain these embedding vectors? It is not straightforward as in the one-hot encoding case since there is nothing to count. The solution is based on one of the fundamental assumptions of NLP, namely the distributional hypothesis. It states that two words that occur in similar contexts in large corpora tend to have similar meanings [40]. In other words, formulated by linguist J.R. Firth in 1957 [31]:

"You shall know a word by the company it keeps."

Models based on this assumption are named distributional models. Distributional models represent a word through the contexts in which it has been observed. Adopting the taxonomy of Baroni et al. [8], one can discern between count-based [25, 13, 93] and prediction-based distributional models [10, 23, 22, 49, 84, 85]. All of these models have in common that two words are *semantically similar* if the vectors representing them are close according to some distance function.

The idea of count-based models is to count how many times a word appears in a particular context. Count based models represent a word as a point in high-dimensional space, where each dimension stands for a word, and a word's coordinates represent its context counts,

i.e., how many times the word appeared in another word's context. We can see that this basic count-based model is very similar to the one-hot encoding of documents. It has the same fundamental problems presented earlier as well, i.e., the number of dimensions is not fixed and grows very fast. This makes using these vectors without further processing for word similarity virtually impossible in practice. Nevertheless, these frequency count vectors are the base of count-based models. Count-based models use various processing methods, such as matrix factorization [93], to simplify the count matrix and make the word representations fit into a low dimensional vector space with pre-selected dimensionality.

Prediction-based models have recently received renewed popularity after Mikolov et al. presented new neural network-based models [84, 85]. Models from this family compute word vectors that are optimized in a prediction task, such as predicting the next word based on previous words or predicting a word given its context. In general, most prediction-based models try to optimize a loss function that aims at minimizing the discrepancy between prediction values and target values. In comparison to count-based models, which have to keep a large dictionary of the frequencies in memory, the training of such models scales very well even to huge corpora, while learning high-quality word vector representations.

It is still an open question whether embedding models in general are superior to traditional count-based models. Various researchers suggest that they are indeed better in various similarity and analogy detection tasks [8, 85]. But others have argued that this superiority is only a result of better parameter selection [68, 65, 42, 101]. Moreover, it has been shown that the most famous novel models in each category (the GloVe [93] and Word2Vec [84] models, respectively) are fundamentally very similar in their training algorithm, in contrast to their intuitive differences [67].

Despite the limited linguistic information distributional models contain, word embedding models have proven to be successful not only in elementary NLP tasks, such as similarity detection [84], word sense disambiguation [50], sentiment analysis [114], part-of-speech tagging [23], named entity recognition [92] or dependency parsing [59], but also complex down-stream NLP tasks can be implemented based on these real-valued vectors, such as social media sentiment analysis [120], irony detection [99], out-of-vocabulary word classification [75], or semantic shift detection [38, 76]and machine translation [27, 112, 72].

Apart from the linguistic applications, embedding models have been used successfully in various other fields of computer science, such as bioinformatics, most notably genomics [3, 37], image annotation [58], recommendation systems [41], or automated ontology enrichment [111].

Although, word embedding models have become instrumental tools in computer science, especially in NLP, several aspects of them are still not sufficiently clear. The ultimate motivation of this work is to understand these aspects better. We group these aspects in three categories: algorithmic, theoretic and application questions. Based on these groups in this work we present a three-way analysis of word embedding models. These analyses are built on top of each other, i.e., the algorithmic part serves as a foundation for the

theoretical analysis, while in the application part we use the results from both previous parts. In the following, we present each group separately.

Algorithmic analysis. The first part of our analysis is regarding the training algorithms of embedding models. It is known how these algorithms work in theory, but to fully understand them we have to evaluate their final product, the models themselves. For example, it is currently not known, and we need to evaluate the created models to understand, how the similarity values of the models change when trained with different parameter settings or using different training corpora.

There are existing works that evaluate how the quality of the models change on baseline NLP tasks using different training parameter settings [84, 85, 8]. However, not only they do not consider the size of the similarity values, which we deem highly important, but these evaluations are missing something else as well. Although it is one of the most important prerequisites when creating high-quality embedding models, the effect that the quality of the training corpora has on the quality of the models is not sufficiently clear.

One big advantage of embedding models is that they can be trained on arbitrary text corpora and hence they can be very useful in a wide range of specialized fields. Diverse corpora, such as Wikipedia, Google Books, or the ACM Digital Library offer great potential for linguistic analysis. For instance, one may be able to discover how language evolves over time based on the Google Books historical corpora, or reveal how language is used in different fields of sciences based on the ACM Digital Library. Such analyses ultimately strive for understanding our society by means of comprehensive empirical results.

One of the above mentioned corpus that many approaches use is the Google Books n-gram corpus [76, 87, 57, 76, 61, 60, 39]. This is mainly because of its size and its historic contexts. It is much larger than any other corpus openly available. It is also the largest currently available corpus with historic data and it is available in several languages. It incorporates over 5 million books from the previous centuries split into n-grams and their occurrence frequencies over time [81]. n-grams are text segments separated into pieces consisting of n words each. For example, "you are" is a 2-gram, "you are beautiful" is a 3-gram. The *fragmentation* of a corpus is the size of its n-grams. To illustrate, a corpus of 2-grams is highly fragmented, but for instance, one of 5-grams is moderately fragmented.

There are several advantages of storing textual data in n-grams. First of all, n-gram counts over time can be published even if the underlying full-text is subjected to copyright protection. Next, this format reduces the data volume significantly. The Google n-gram data set for each language is about 1 Terabyte, already an impressive size. However, the underlying full-text is much larger. This makes certain analyses impossible on full-text corpora. Therefore, it is important to know how good models built on n-gram corpora are.

While the quality of word embedding models trained on full-text corpora is fairly well-known [65, 8], an assessment of models built on fragmented corpora is missing [42]. The question that we evaluate in this work is, whether using n-gram corpora is suitable for

word embedding model training or not. We evaluate, whether n-gram representations are sufficient to extract word semantics, and which quality differences compared to full-text corpora one should expect.

Theoretical analysis. The second part of our analysis is regarding the most fundamental property of word embedding models, namely their similarity values. We have already seen that word embedding models are very good at various down-stream tasks, which make use of these values. However, these papers are only *using* the similarity attribute of the models, while the following questions remain open: What do similarity values from those models actually mean? For instance, are low values of similarity comparable to each other? To illustrate, if Word A is 0.2-similar to Word B and 0.1-similar to Word C on a [-1, 1] scale, should we say that A is more similar to B than to C, or does it not make any difference at these low levels of similarity? Are there 'natural' thresholds for similarity, such that values above (beneath) it represents a definite similarity (dissimilarity) of two words? For example, if A is more than 0.5-similar to B, then are A and B always semantically similar? How about the same questions with similarity lists, i.e., lists of words most similar to certain words, sorted by similarity? For instance, can we say that the 100 words most similar to an arbitrary word are always similar to this one, or words not in the top 500 are always dissimilar? When exactly is it meaningful to stick to the natural idea of taking the top N most similar words for a certain word and deem them similar?

In order to study and answer all these questions, we have to evaluate how different parameter settings (e.g.: the size of the corpus they are trained on, vocabulary size) influence the similarity values of the models. This is exactly what we do in the algorithmic analysis section, and we use the results here for better theoretical understanding of embedding models.

These questions are not just academic in nature; any study relying on comparisons of similarity values might lack validity if these questions remain open. As we will see, giving an answer to these questions have implications in the qualitative evaluation of word embedding models as well.

Application analysis. After understanding the algorithmic and theoretical questions regarding word embedding models better, we want to use them for down-stream applications. As we have previously listed, there are many applications in and even outside of the NLP domain where word embedding models are useful. Since they interpret words in one of the most straightforward and intuitive way, the utilization of word embedding models occupy a highly relevant field of NLP, namely text classification.

Text classification, i.e., to assign predefined categories to text documents [77] is an effective way to organize enormous number of documents [121, 4, 119, 66, 102]. It has other important applications, like spam filtering [80], sentiment analysis [91, 90], word sense disambiguation [110] or health prediction [97].

Nowadays text classification is mainly done by machine learning algorithms [102]. In general, machine learning-based classifiers build a model based on the pre-classified (labeled) documents they are presented with. Based on this model they are able to predict the categories of unknown samples. Usually, the more training data the classifier learns from, the better it will be in predicting the categories of previously unseen documents. This means that the success of text classification applications come not only from the goodness of the underlying model, but from the quantity and quality of the training data as well. However, since labeling is costly, in the majority of real world scenarios the number of pre-classified documents are relatively small. One way to overcome the scarcity of labeled data is to improve its quality.

It is the purpose of the application analysis part of this work to explore the possibility of improving the accuracy of text classification despite the scarcity of labeled data, by increasing the quality of the training data. We present a text preprocessing approach, which is based on how humans categorize text without first reading thousands of example documents, by taking knowledge on semantic relationships between words extracted from word embedding models into account. We show that our proposed preprocessing approach compensates the scarcity of labeled data by increasing its quality, and ultimately increases text classification accuracy.

1.2. Challenges

The complexity and variability of human language makes natural language processing a non-trivial task. As we have mentioned earlier even for a small dataset the one-hot encoding reaches thousands of dimensions. This phenomenon is one of the biggest fundamental challenges in NLP.

For example, assuming a language has 10,000 vocabulary terms and a sentence would have 10 words, then the solution space will be as large as $10,000^{10}$, a number which is too big to be processed by any computer. This is an intrinsic difficulty of language modeling: a word sequence that is used to test the model is likely to be different from all the sequences in the training set [10]. One might argue that with grammar tricks, such as stemming, lemmatization or using n-gram data, the solution space can be reduced significantly. It is true, that these ideas help, but even with the aid of them, we will run into combinatorial explosion every now and then when processing language. Natural language is just so dynamic and flexible that the grammars or n-grams bring too many exceptions and finally we have to make a trade-off between accuracy and scalability.

So far, we have talked about dimensionality problems, but there are several other language specific challenges, such as the grammar of different languages or synonyms. The first problem calls for algorithms tweaked for every specific language and using different corpora for different languages in order to build good models. The second issue has become a whole research sub-topic in NLP called word sense disambiguation. This means, that due to the subjectivity of languages, the meaning of words varies in different contexts.

As we have already mentioned, it is also not straightforward to find good training corpora for NLP models. Not only it should be grammatically correct, without containing many misspellings or missing parts, but it needs to be large as well due to complexity issues.

To overcome all these difficulties of handling such complex training data as natural language modern NLP systems, such as word embedding models, usually use neural networks for training. With all the obvious benefits, using neural networks bring along its own set of fundamental challenges. First of all, although they work very well in practice, neural networks are hard to profoundly understand. This means, since they usually contain millions of parameters, it is hard to look inside of them and see what really happens. They are more like a black box or oracle that produce good results, but we do not know how. Second, to train neural networks, it is very important to have sufficiently large training data, because of the huge number of trainable parameters in them. If this is not the case neural networks are very prone to overfitting the training data. This may be an issue for infrequent words even when the training corpus is large enough. Third, neural networks are hard to intuitively parameterize, meaning to set how deep it should be, i.e., how many many layers are optimal, and what kind of layer each one should be. Finally, training neural networks requires strong hardware, ideally GPU, which is expensive.

Apart from the general challenges of NLP and neural network approaches, there are specific challenges regarding our analysis objectives. We group them into algorithmic, theoretical and application challenges, explained in the following.

Algorithmic challenges. When it comes to algorithmic analysis an obvious challenge is the huge runtime necessary to build the word embedding models. It took more than three months on a modern machine to create the 424 models, which we used to evaluate the effects of the different parameters and corpora.

The evaluation of the similarity value distributions of models trained with different parameter settings is pretty straightforward, however, when it comes to qualitatively evaluating the models for the n-gram corpus experiments it is challenging for various reasons.

First, drawing *general* conclusions on the quality of embedding models only based on the performance of specific approaches, i.e., examining the extrinsic suitability of models, is error-prone [34, 101]. Consequently, to come to general conclusions one needs to investigate *general properties* of the embedding models itself, i.e., examine their intrinsic suitability. Properties of this kind are semantic similarity and analogy. For both properties, one can use well-known test sets that serve as comprehensive baselines, such as WordSim353 [30] or MEN [16] test sets for similarity and MSR [84] or Google [85] test sets for analogy.

Second, there are various parameters that influence how the model is trained. Using n-grams as training corpus leads the way to two new parameters, fragmentation, as discussed earlier, and minimum count, i.e., the minimum occurrence count of an n-gram in order to be considered when building the model. The latter is often used to filter error-prone n-grams from a corpus, e.g., spelling errors. While the effect of the other parameters on the models is known [65, 8], the effect of these new parameters is not. But this is important for scientists using the n-gram corpora, as more and more word embedding models are trained on such data. However, as word embedding models mostly rely on neural networks, it is hard to explain the effect of these parameters based on the training algorithms themselves, as we have explained earlier. We have to define meaningful experiments to quantify and compare the effects.

Third, the full-text, such as the Google Books corpus, is not openly available as reference. Hence, we need to examine how to compare results from other corpora, where the full-text is available, referring e.g., to well-known baselines as the Wikipedia corpus.

Theoretic challenges. The main question of our theoretic analysis is what the similarity values actually mean in word embedding models? In this section we take a closer look at the challenges of evaluating this question.

We have presented a list of questions regarding similarity values in the respective part of the motivation. These questions are easy to formulate, but much harder to systematically evaluate. Creating a pipeline for the evaluation is the main issue to handle. We have to

answer questions, such as: How to create good baseline test sets, how do they measure similarity, and how to evaluate a model on them?

To show the difficulty of how to create such test sets, think of the following linguistic challenge pointed out by Hill et al. [43] in this context: What is the definition of similarity? Are *cup* and *coffee* similar words or only *associated*, i.e., dissimilar? In general, does relatedness or associatedness imply similarity or not? They argue that word pairs that are only associated should only have moderately high similarity scores. This is in opposition to test sets such as WordSim353 or MEN where this is not the case, i.e., associated pairs have very high scores. Batchkarov et al. [9] also address the problem of creating good baseline test sets. They show that it is challenging even for human annotators to assign similarity scores to certain word pairs. For example, they show that the similarity scores for the *tiger-cat* pair range from 5 to 9 on a scale of ten in the WordSim353 test set. They also provide example word pairs where the similarity scores differ significantly when the pairs are contained in different test sets. They argue that this is the result of the different notions of similarity these test sets use.

Next, Avraham et al. [5] identified problems regarding the evaluation of the models. They argue that the use of the same rating scale for different types of relations and for unassociated pairs of words makes the evaluation biased. For example, they say that it is meaningless to compare the similarity value of *cat-pet* to *winter-season*, because they are unassociated, and models that rank the wrong word pair higher should not be punished. If *cat-pet* has a score of 0.7, and *winter-season* has one of 0.8 in a similarity test set, then an evaluation should not punish a model that ranks *cat-pet* higher. They also find it problematic how the conventional evaluation method measures the quality of a model [5]. It calculates the Spearman correlation of the ranking by the annotator and the model ranking, without considering the similarity values further. To illustrate, such an evaluation penalizes a model that misranks two low-similarity, unassociated pairs (e.g.: *cat-door, smart-tree*) just as much as one that misranks two objectively distinguishable pairs (e.g.: *singer-performer, singer-person*).

Having said this, the concept of similarity remains ambiguous in word embedding models, and understanding similarity values remains difficult as well, affecting several NLP tasks, especially when it comes to evaluate embedding models on these tasks.

Application challenges. As we have already mentioned, the lack of training data in NLP is generally considered a big problem. This is especially true when working on text classification problems. Classifiers trained with small data samples lack robust statistical information and tend to *overfit* the training data. Also, unknown documents very likely contain words not covered by the classification model, i.e., out-of-vocabulary (OOV) words.

Generating labeled training data for machine learning solutions in general are often significant and rarely discussed. E.g., the recent AI breakthroughs such as autonomous driving are based on enormous human tagging effort [15]. In the field of text classification, academic research focuses on improving the classification accuracy on annotated benchmark

datasets for comparability reasons. But, in practice, the generation of such a data set and the necessary size are important cost factors. Hence, the costs of generating large bodies of labeled data for all potentially relevant classes are a severe issue in many text classification use cases.

Next, the semantic relationships of word embedding models need to be considered with care in text classification. Namely, not only words with a similar meaning are semantically related, but also antonyms: Words like *'good'* and *'bad'*, which obviously are not interchangeable, tend to be very similar in word embedding models. Thus, approaches such as [75] that simply replace OOV words with the most similar labeled one, have little or even negative effect on classification accuracy.

1.3. Contribution

Now, we list the contributions of this work. Again, it is grouped in algorithmic, theoretical and application contributions.

Algorithmic Contribution. Our first contribution is that we evaluate how different parameter settings (for example, the size of the corpus they are trained on) influence the similarity values of word embedding models. We do so by systematically training various models with different settings and comparing the similarity-value distributions. In addition, we consider two other embedding model types that are not based on words, but rather on syllables and sentences respectively, to generalize our findings. We show that, except for a few marginal cases, all distributions have a very similar bell shape. We prove with statistical tests that most of the normalized distributions are almost identical even with the most extreme parameter settings, such as very large dictionaries or small dimensionality.

These evaluations show how parameters affect the similarity values, but not how they influence the quality of the models. Our next contribution is that we describe experiments to examine how model quality changes when the training corpus is not full-text, but n-grams. The experiments quantify how much fragmentation and different minimum count settings changes the average quality of the corresponding word embedding models, on common word similarity and analogical reasoning test sets. One can repeat these experiments using any corpus we used in this work (full-text or n-gram) or extend the experiments with any arbitrary training corpus or test set and utilize our results for comparison.

To show the usefulness and significance of the experiments and to give general recommendations on which n-gram corpus to use as well as creating a baseline for comparison, we conduct the experiments on two large full-text corpora.

To make our results more intuitive we answer the following explicit research questions during our analysis.

1. What is the smallest number n for which an n-gram corpus is good training data for word embedding models?
2. How sensitive is the quality of the models to the fragmentation and the minimum count parameter?
3. What is the actual reason for any quality loss of models trained with high fragmentation or a high minimum count parameter?

Our results for the baseline test sets indicate that minimum count values exceeding a corpus-size-dependent threshold drastically reduce the quality of the models. Fragmentation in turn brings down the quality only if the fragments are very small. Based on this, one

can conclude that n-gram corpora such as Google Books are valid training data for word embedding models.

Theoretical Contribution. Our next set of contribution is an examination on what similarity values in embedding models mean. One intention behind these experiments is to confirm that the meaning of similarity values of two terms is not sufficiently clear.

Our core contribution is the discovery that meaningful similarity threshold values exist, and we show that they can be found. We do so by calculating similarity-value and similarity-list aggregates based on WordNet [33] similarity as the baseline and evaluate the resulting similarity distributions of the models with statistical tests. It turns out that these thresholds are not general and should be calculated for every individual model using the method we present in this work. At this point, our analysis connects with the parameter evaluation of the models mentioned in the algorithmic analysis part: The evaluation shows that although altering the parameters changes the similarity value distributions, it does not change our method, since all distributions are fundamentally similar.

Based on these results, our next contribution is that we propose a similarity threshold aware evaluation method of word embedding models on similarity tasks, which does not compare the word pairs during evaluation that fall below the calculated threshold. Using well-known benchmark test sets, we arrive at two insights. First, there are pairs in these sets that fall below the threshold, i.e., that should not be part of the evaluation. Second, excluding these pairs from the benchmark does change the benchmark results to a noticeable extent. This ensures a more reliable comparison. This is an important step regarding the design of future word embedding models as well as for the improvement of existing evaluation methods.

Application Contribution. Our last set of contributions is regarding applications of word embedding models.

We present a lexical substitution approach for preprocessing training data, in order to compensate the scarcity of labeled documents. Our approach mimics how human annotators preprocess texts. Since it is a preprocessing method, it is generally applicable in combination with any text classification algorithm. It replaces words unknown to the classifier with known ones and substitutes semantically similar words for statistical robustness, based on the main contribution of this part: A novel *semantic-distributional word distance measure*. This is the first time when both semantic and distributional information is used in term substitution.

Our results show that even if there is plenty of labeled data for learning, classification accuracy increases using the preprocessed training data to train the classifiers. Nevertheless, the improvement in classification accuracy with our method is most significant when labeled data is scarce.

1.4. Structure

In the following parts of this work we present all the necessary prerequisites and our three-way analysis of word embedding models. We detail the structure of the work in the following.

In Chapter 2, we present the fundamentals and notation that we use throughout this work. First, in Section 2.1 we introduce word embedding models in general. We present their brief history, their variants, and most importantly the two most relevant word embedding models for this work, namely the Word2Vec and the Glove models. Second, in Section 2.2 we explain the fundamentals of text classification. This includes the notation in text classification, the basics of the dimensionality reduction techniques we use for our preprocessing method, and the introduction of the classifiers that we use later on in this work. Third, we introduce n-grams, and we explain how to train word embedding models on n-gram data. Finally, we present all the datasets we use in this work, including the training corpora that we use for training as well as the similarity, analogy and text classification test sets that we use for evaluation.

Next, in Chapters 3, 4 and 5 we present our three-way analysis of word embedding models. These are algorithmic, theoretical and application analysis, respectively.

In the algorithmic analysis chapter in Section 3.1 we investigate how various parameters of word embedding models influence their similarity values. We dedicate a subsection for each parameter. Next, in Section 3.2 we investigate how training corpus fragmentation affects the quality of the word embedding models. We show that n-gram corpora, such as the Google Books, is a valid training data for word embedding models.

In the theoretical analysis chapter, first, in Section 4.1, we present experiments that let us find similarity value thresholds in word embedding models. We show that these thresholds exist, however they are not general, but model specific. Second, in Section 4.2, we propose a similarity threshold aware evaluation method to quantify the quality of word embedding models. We argue that our method gives a fairer score for the models than the baseline evaluation method used in previous literature.

The final part of our three-way analysis is the application part in Chapter 5. In this chapter we present our novel preprocessing method, which combines both semantic and distributional information of words. We show that using our method helps increasing the classification accuracy in every tested text classification test set.

Finally, we conclude our work in Chapter 6. We also give an overlook on future research possibilities in this chapter.

2. Fundamentals and Notation

In this section we establish a framework of terminology and notations that will be used in later explanations and throughout the course of the entire work. First, we introduce the main subject of this work: word embedding models. Then, we give a general introduction to text classification, which is the use case scenario we focus on in the application analysis part. Finally, we introduce n-grams and the datasets we use throughout this work.

2.1. Word Embedding Models

Word embedding models represent words as vectors in a continuous low dimensional vector space. A particular property of this approach is that the angle between two word vectors is a measure of how similar the respective words are from a semantic point of view. Moreover, it is possible to answer analogy questions with word embedding models as well.

In the following, we first define word embedding models and their parameters in general. We then introduce the two most relevant models which we rely on in this work, namely the Word2vec and Glove models. Then, we discuss other types of embedding models and the software framework we have used to create the models in this work.

2.1.1. Background on Word Embedding Models

Formally, a word embedding model is a function F which takes a corpus C as input, such as a dump of the Wikipedia, generates a dictionary D based on the corpus and associates any word in the dictionary $v \in D$ with a d-dimensional vector $vec(v) \in R^d$. F is not deterministic, as it may use random values when initializing the word vectors. The *dimension size* parameter (d) sets the dimensionality of the vectors. It usually ranges between 50 and 1000 for word embedding models. The training, i.e., iteratively associating vectors with words in the dictionary, is based on word-context pairs $v \times c \in D \times D^{2 \times win}$ extracted from the corpus. There is a further parameter *epoch_nr* that states how many times the training algorithm passes through the corpus. If not stated otherwise, we will train models with five iterations. *win* is the *window size* parameter, which determines the *context* of a word. For example, a window size of 5 means that the context of a word is any other word in its sentence, and their distance is at most 5 words.

There are further parameters that affect the generation of the dictionary. One is the *minimum count* parameter (*min_cnt*). When creating the dictionary from the corpus, the model adds only words to the dictionary which appear at least *min_cnt* times in the corpus. An alternative is to set the *dictionary size* directly as a parameter (*dict_size*). This means that the model adds only words to the dictionary which are in the *dict_size* most frequent words of the corpus. In this work, we rely on the *dict_size* parameter, because we find it easier to handle in our experiments. With this variant, the corpus does not influence the size of the dictionary. Note, *dict_size* is not necessarily equal to the size of the dictionary $|D|$. For example, it is unequal when the number of distinct words in the corpus is smaller than *dict_size*. Additionally, there are model specific parameters (θ) which change minor details in the embedding algorithms.

The result of the training is a word embedding $\mathcal{W}$, with vocabulary

$$voc(\mathcal{W}) = \left\{v_{\mathcal{W}}^1, \ldots, v_{\mathcal{W}}^{dict\_size}\right\},$$

and corresponding word vectors

$$\left\{vec(v_{\mathcal{W}}^1), \ldots, vec(v_{\mathcal{W}}^{dict\_size})\right\}.$$

Each vector represents a word $v \in voc(\mathcal{W})$ in a d-dimensional real-valued vector space.

Semantic Similarity. The most important property of word embedding models is that vectors close to each other according to a distance function represent words that are semantically similar.

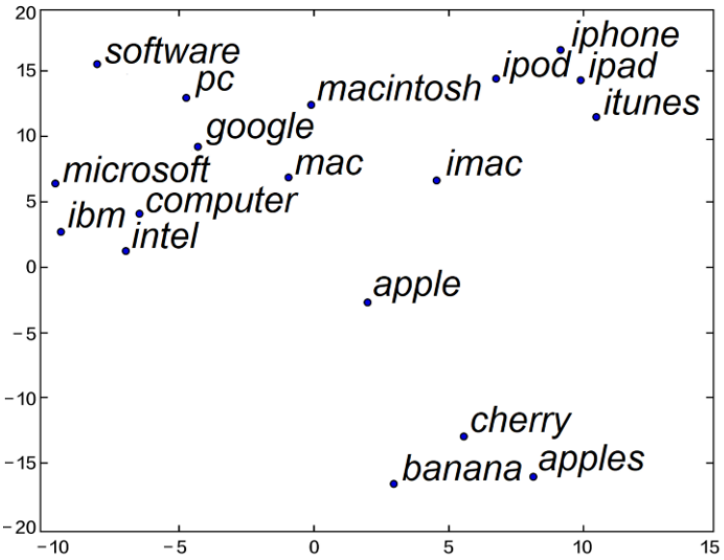

Figure 2.1.: A 2-dimensional projection of the vectors corresponding to word 'apple' and its 15 most similar words.

Figure 2.1 illustrates the similarity property of word embedding models. It is a two dimensional projection of the embedding space of a word embedding model. We can see, that the most similar words to 'apple' are words related to the Apple company as well as apple as a fruit[1].

Analogical Reasoning. Another property of word embedding models that we use in this work is called analogy, which is the process by which knowledge is transferred from one concept to another. For example, 'man is to king as woman is to queen' is an analogy. Besides the semantic similarity property, word embedding models are able to capture such analogies as well.

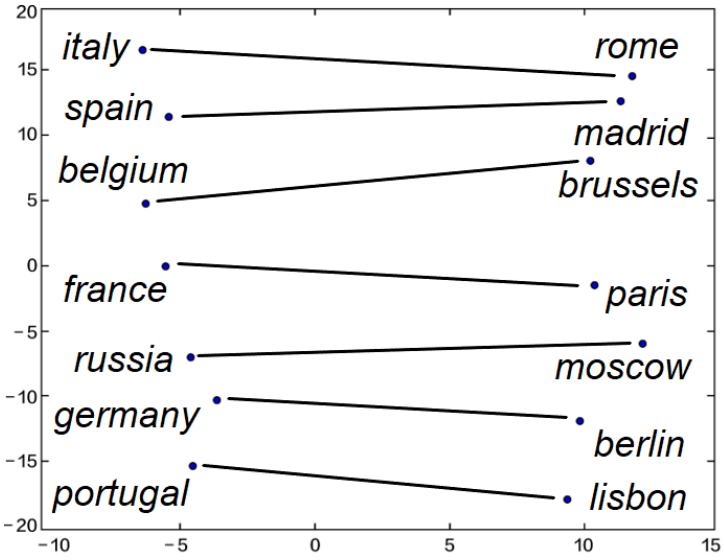

Figure 2.2.: A 2-dimensional projection of the vectors corresponding to words of countries and their capitals

Figure 2.2 is an illustration of this property. It shows, that the direction of the vectors between countries and their capitals are very similar. This allows us to solve analogical reasoning tasks with word embedding models using vector operations. For example, "*Chicago* is to *Illinois* as *Denver* is to *?*" can be solved by simply computing *Illinois* − *Chicago* + *Denver* in the embedding vector space and finding the closest word vector to the resulting vector (≈ *Colorado*).

[1] Every example in this section is an actual 2-dimensional projection of the embedding space of one of the models we have trained for this work.

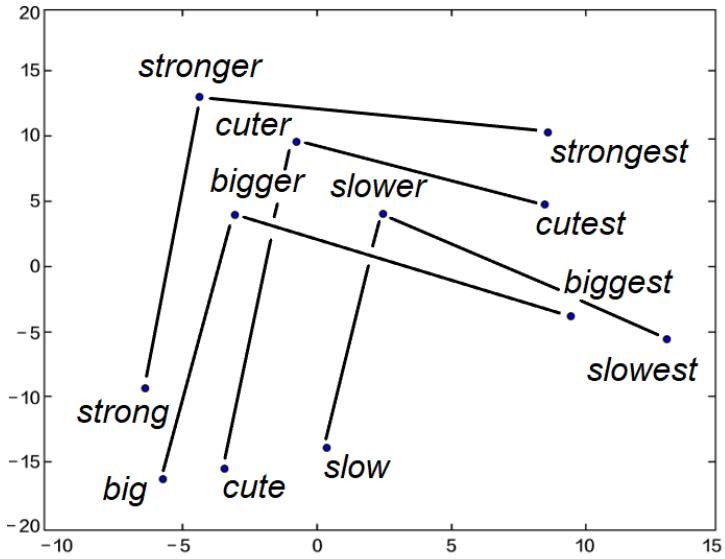

Figure 2.3.: A 2-dimensional projection of the vectors corresponding to adjectives and their comparative and superlative forms

Word embedding models able to capture not only semantic but syntatic analogy as well. Figure 2.3 illustrates how the comparative and superlative forms of different adjectives are captured in the embedding space.

Similarity Measure. As it is generally recommended in word embedding model literature, throughout this work, we use the cosine similarity of two word vectors as the similarity score of the respective words. The cosine similarity ranges from -1 (unrelated words) to 1 (identical words). Formally, the cosine similarity of the vectors corresponding to words v_1 and v_2 is the following.

$$\text{cos-sim}\left(vec(v_1), vec(v_2)\right) = \frac{vec(v_1) \cdot vec(v_2)}{\|vec(v_1)\| \cdot \|vec(v_2)\|} \in [-1, 1]\,.$$

Just as the cosine similarity expresses the relatedness between two words, the *cosine distance* can be employed to produce values for dissimilarity. The cosine distance is defined as follows:

$$\text{cos-dist} = 1 - \text{cos-sim} \in [0, 2].$$

Note that the cosine distance is not a *metric* since it does not satisfy the *triangle inequality*, i.e.,

$$\exists \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^d : \text{cos-dist}(\mathbf{x}, \mathbf{z}) > \text{cos-dist}(\mathbf{x}, \mathbf{y}) + \text{cos-dist}(\mathbf{y}, \mathbf{z}).$$

Whenever in this work we refer to a *distance* with respect to words or word vectors, we refer to the following measure that expresses their dissimilarity:

$$dist_{\mathcal{W}}(v_1, v_2) = \begin{cases} \frac{\text{cos-dist}(vec(v_1), vec(v_2))}{2}, & \text{if } v_1, v_2 \in voc(\mathcal{W}) \\ \infty, & \text{if } v_1 \notin voc(\mathcal{W}) \vee v_2 \notin voc(\mathcal{W}). \end{cases}$$

In the definition above, we normalized the cosine distance to $[0, 1]$, which will be useful later in this work.

2.1.2. Variants of Word Embedding Models

There are two main types of word embedding models: count-based and prediction-based models. On one hand Baroni et al. [8] claim that the prediction-based methods are generally better when comparing the quality of the produced embeddings. On the other hand, work by Levy et al. [68] suggests that the performance of both families is on-par if the hyperparameters are correctly chosen. It is not the aim of this work to compare the goodness of different word embedding models, but to understand their general properties better, hence we will work with both groups of models.

In the following we give a brief overlook on their history, then we describe the most important model from each group in detail, namely the Word2Vec and Glove models. Although, the fundamental ideas of Word2Vec and Glove are different, Levy et al. [67] have shown that the Word2Vec model implicitly factorizes a word-context pointwise mutual information matrix, which is a count-based approach to obtain the embeddings. This means that the objectives of the two models and sources of information are used quite similarly and, more importantly, they share the same parameter space. We refer to Shi and Liu's work for a further comparison [104].

2.1.2.1. Count-Based Models.

Count-based word embedding models use global statistics on the contexts of words (i.e., word-context counts) to derive the word vectors. This means they store the frequency of every word-context pair in a huge sparse matrix and use different matrix factorization techniques to extract word vectors from this matrix. A classic example of a word embedding model from the count-based family is the Singular Value Decomposition (SVD) of a word-context matrix, which has already been introduced back in 1990 [25]. Recently, Pennington et al. have presented a novel count-based model, the GloVe model [93], which has become highly popular. We introduce Glove in the following.

The Glove Model. Pennington et al.'s GloVe model [93] trains the word vectors by explicitly factorizing the log-count matrix of the underlying corpus, regarding the word-context pairs.

The advantage of GloVe is that, unlike Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence counts) to obtain word vectors. Instead of looking at bare word-word co-occurrence counts, GloVe compares the probability ratios of co-occurrences, i.e., how much more probable it is for word v_1 than another word v_2 to appear in the context of word v_3. The intuition is the observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning and the word meanings are captured by the ratios of co-occurrence probabilities rather than the probabilities themselves.

Figure 2.4 shows actual probabilities from a real worlds corpus[2].

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 2.4.: Glove probability examples

As we can see, the ratio is large if the target word is correlated with the word corresponding the numerator, but not with the denominator (for example, 'solid' is correlated to 'ice', but not with 'steam'). It is small the other way around ('gas' is correlated to 'steam' but not with 'ice'), and it is around 1 if it is correlated to neither or both words ('water' is correlated to both, while 'fashion' is correlated to neither of the words 'ice' and 'steam'). In this way, the ratio of probabilities encodes some form of meaning associated with the abstract concept of thermodynamic phase.

Formally, let X be the co-occurrence matrix. X_{ij} is the number of times word v_j appears in the context of word v_i. Let $X_i = \sum_k X_{ik}$ be the total number of words that appeared in the context of v_i. The probability of word v_j appearing in the context of word v_j is

$$P_{ij} = \frac{X_{ij}}{X_i}.$$

[2] The figure and example are from the Glove project website: https://nlp.stanford.edu/projects/glove/

Our goal is to express the probability ratios with a function F[3]:

$$F(v_i, v_j, v_k) \approx \frac{P_{ik}}{P_{jk}}$$

There are two requirements for F to consider. First, we know P_{ik}/P_{jk} is a scalar. Second, we should be able to perform arithmetic operations in the embedding space, such as the analogy calculations. Based on this, we can reformulate the equation:

$$F((v_i - v_j)^\intercal v_k) \approx \frac{P_{ik}}{P_{jk}}.$$

Now, to make the fraction on the right side disappear we take the logarithm of both sides. To be able to do this we assume that F is an exponential function.

$$(v_i - v_j)^\intercal v_k = v_i^\intercal v_k - v_j^\intercal v_k \approx log(P_{ik}) - log(P_{jk}).$$

Because of symmetry, this means:

$$v_i^\intercal v_k \approx log(P_{ik}) = log(X_{ik}) - log(X_i).$$

We also want to capture the fact that some words occur more or less often than others. We do this by adding bias b_i for each word v_i, and expressing X_i with the biases.

$$v_i^\intercal v_k + b_i + b_k \approx log(X_{ik}),$$

or,

$$v_i^\intercal v_k + b_i + b_k - log(X_{ik}) \approx 0.$$

The goal of the training is to create word vectors to minimize the squared error of the above equation, summarized over every word pair.

$$\sum_{i,j} (v_i^\intercal v_k + b_i + b_k - log(X_{ik}))^2.$$

However, there is a problem with this summarization: it weights all word pairs equally. This is not ideal, since not all word pairs are equally important. For example, infrequent

[3] Note, in the following equations we use the notation v_i instead of $vec(v_i)$ as the vector of word v_i for readability reasons.

word pairs tend to be error-prone, so we want to weight frequent word pairs more heavily. In contrast, very frequent word pairs, such as "I am" or "it is" should not be dominating the loss function neither. The authors of Glove propose the following weight function:

$$f(x) = \begin{cases} (x/x_{max})^{\alpha}, & \text{if } x < x_{max} \\ 1, & \text{otherwise.} \end{cases}$$

In the original publication x_{max} is set to 100 and $\alpha = 0.75$. Figure 2.5 shows the weight function $f(x)$[4].

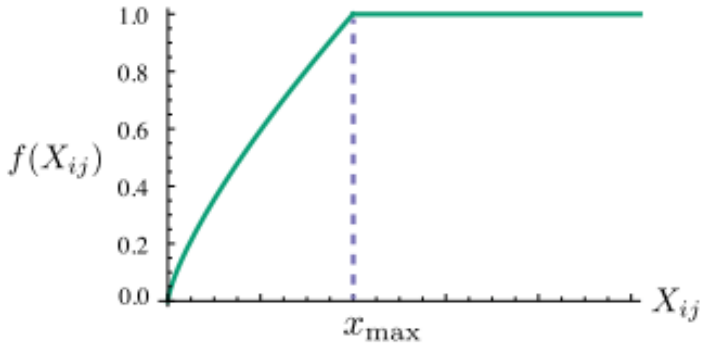

Figure 2.5.: Glove weight function

Finally, the loss function in Glove is as follows.

$$\sum_{i,j} f(X_{ij})(v_i^{\intercal} v_k + b_i + b_k - log(X_{ik}))^2.$$

For more details on the Glove model we refer to the original publication [93].

2.1.2.2. Prediction-Based Models.

Prediction-based models compute word vectors which are optimal in a prediction task, such as predicting a word given its context. They usually use neural networks for prediction. Neural networks in general are inspired from biological neural networks, hence the basic processing unit is called a *neuron*. Neural networks can be viewed as weighted graphs of interconnected neural units. Computations are performed by propagating data from the input units throughout the whole net. The result is returned at the output units. Learning is performed by iteratively adapting the weights between neural units while minimizing the output error with respect to a target function. This method is called back-propagation.

[4] The figure is from the original publication.

This means, the weights of the neural network are changed in a backwards order, such that the error between the actual output and the expected one decreases according to a loss function.

It has been shown that neural networks can be used as universal function approximators and therefore are appropriate for language prediction tasks [47].

The first prediction-based word embedding model was introduced by Bengio et al. [10]. It learns embeddings with a neural network for *language modeling*, i.e., predicting the next word of the text, knowing the previous several words. Prediction-based word embedding models gained a lot of popularity after the introduction of the Continous-Bag-of-Words (CBOW) and Skip-gram (SG) models by Mikolov et al., also called *Word2Vec* models [84]. These models try to predict the word given its context (CBOW) or the word context from a word (SG).

Peters et al. [94] introduced *Embeddings from Language Models* (ELMo). In ELMo each embedding represent not only a word, but a *word in its context*. To obtain an embedding for a word, the previous and the following part of the sentence are fed into a forward and backward neural language model, respectively. By making embeddings context-specific, ELMo deals better with synonyms.

In this work we work with the Word2Vec model from the prediction-based family. In the following we introduce this model in detail.

The Word2Vec Model. Mikolov et al.'s Word2Vec model [84, 85] uses a learning algorithm based on a shallow neural network. Word2Vec generates word vectors estimating the probability of a context given an input word (SG) or the other way around estimates the probability of a word given an input context (CBOW). The context words are the words surrounding the target word within a symmetrical window of a pre-defined size. The network is trained in an unsupervised fashion by using a possibly large and topically heterogeneous text corpus, such as the first billion words of *Wikipedia* (see Section 2.4.1).

Figure 2.6 shows the neural network used for training. The input and output layers have $dict\_size$ number of nodes, while the hidden layer has d, where d is the dimensionality of the embedding space. The output layer is a softmax layer.

Every input and output node represents a specific word in the vocabulary. During training the words are represented with one-hot encoding. For example, during CBOW training the input is 1 on the nodes which correspond to words which are in the context and 0 otherwise. On the output layer the expected output is 1 on the node corresponding to the target word and 0 otherwise.

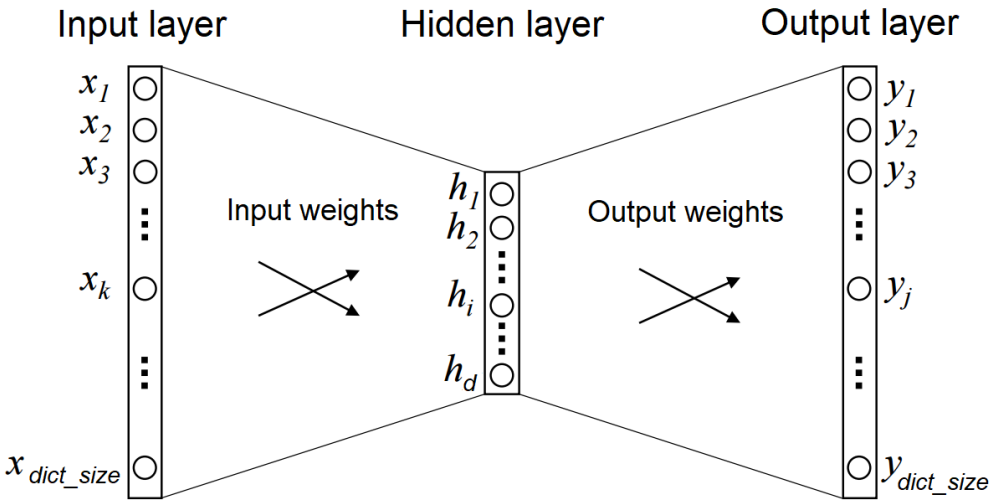

Figure 2.6.: The neural network used to train the Word2Vec models

The vector representation of a word is the weights on the edges of the neural network connecting the corresponding node to the hidden layer. There are d edges connecting each node to the hidden layer, hence the vectors are d-dimensional vectors. However, there are two sets of weights in the neural network, one that connects the input layer to the hidden layer (input weights), and the other which connects the hidden layer to the output layer (output weights). This means, there are two representation for each word. The final word vector is the average of the two vector representations.

To obtain word vectors with the required properties, i.e., similarity and analogy, we have to train the neural network. In general, the aim of the training is to optimize the weights of the neural network in a way that the prediction is as close to the expected outcome as possible. Formally, the loss function of the CBOW model is:

$$\frac{1}{T}\sum_{t=1}^{T} \log P(v_t \mid v_{tc}) = \frac{1}{T}\sum_{t=1}^{T} \log P(v_t \mid v_{t-n}, \cdots, v_{t-1}, v_{t+1}, \cdots, v_{t+n}) = \\ = \frac{1}{T}\sum_{t=1}^{T} \sum_{-n\leq j\leq n, j\neq 0} \log P(v_t \mid v_{t+j}),$$

where v_{tc} is the context of word v_t.

Similarly, for the SG model the loss function is

$$\frac{1}{T}\sum_{t=1}^{T} \log P(v_{tc} \mid v_t) = \frac{1}{T}\sum_{t=1}^{T} \sum_{-n\leq j\leq n, j\neq 0} \log P(v_{t+j} \mid v_t).$$

To calculate the probabilities (e.g., for the SG model) we use the softmax definition[5]:

$$P(v_{t+j} \mid v_t) = \frac{\exp(v_t^\top v'_{t+j})}{\sum_{v_i \in V} \exp(v_t^\top v'_i)}, \tag{2.1}$$

where v_i refers to the vector on the input weights and v'_i refers to the vector on the output weights of the neural network corresponding to word v_i.

Since, it is computationally very expensive to calculate these probabilities for each word in every iteration several softmax approximation methods have been developed. The two used in the original Word2Vec publications and in this work are the Hierarchical Softmax (hs) and the Negative Sampling (ns) methods.

Hierarchical Softmax. The hierarchical softmax method makes the calculation of the sum in Equation 2.1 faster with the help of a binary tree. It encodes the output softmax layer into a hierarchical tree structure. Each word is represented as a leaf and the inner nodes represent relative probabilities of their children nodes.

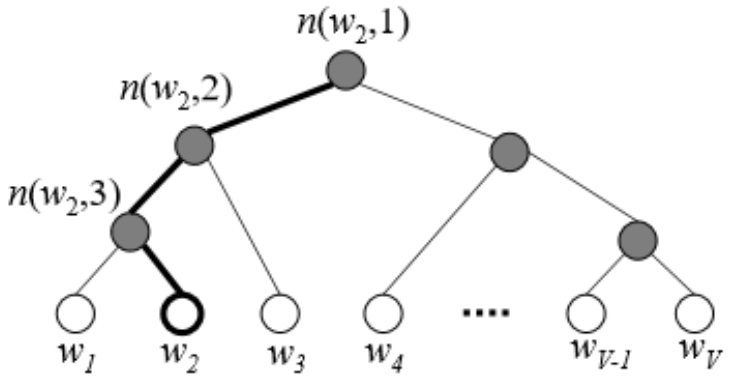

Figure 2.7.: Illustration of the hierarchical softmax binary tree

Figure 2.7 is an illustration of a hierarchical softmax tree. The idea is that each word has a unique path from the root of the tree to the leaf corresponding to the word. The probability of choosing the word equals to the probability of taking this unique path, which is the product of the relative probabilities in each inner node of the path. The relative probability at an inner node n can be calculated

$$P(\text{turn right}|v_i, n) = \sigma({v'_n}^\top v_i) = \frac{e^{{v'_n}^\top v_i}}{e^{{v'_n}^\top v_i} + 1},$$
$$P(\text{turn left}|v_i, n) = 1 - P(\text{turn right}|v_i, n),$$

[5] Note, we use the same notation v_i for the word and its corresponding vector in the following equations for readability reasons.

where σ is the sigmoid function. Such a binary tree structure reduces the of the estimation of the sum in the denominator of the probability equation 2.1 from $O(|V|)$ to $O(\log |V|)$ during training. However, a good tree structure is needed for the speed-up. For example, similar words should be close to each other in the tree structure.

Negative Sampling. The idea behind the negative sampling method is that during training we provide wrong inputs for which no outputs can be determined. Practically, negative examples are not context words, in contrast to positive examples which are context words. This means, negative examples should produce all zeros as output. The intuition is that a good model should be able to differentiate between real and fake input signals. Another idea in negative sampling, is that we sample the observations and do not use all of them when changing the weights of the neural network. This speeds up the computations dramatically.

For more details on the approximation methods we refer to the original publications [84, 85].

2.1.2.3. Alternative Embedding Models

Recently, alternative approaches based on the idea of word embeddings have been presented to model semantics. Bojanowski et al. have refined the Word2Vec models by additionally learning embeddings of subwords, instead of only full word embeddings [14]. A subword in this context is a chain of characters, for example, default 3 to 6 characters. In this way, the model unifies different grammatical forms or spelling mistakes and is expected to learn good embeddings for rare words. Additionally, it is also able to infer embeddings for words that have not been present during training. The model accomplishes this by averaging the vectors of the subwords in the target word. The approach has been published together with an efficient implementation *fastText*[6]. Since it is based on Word2Vec, both Skip-gram and Continuous-Bag-of-Words are available. The fastText models have been used in a variety of tasks, e.g., location prediction based on tweets [86] or review rating prediction from a text [103].

While fastText works on the character level, the other direction of granularity has been investigated as well: An approach called Paragraph Vector (Doc2vec) learns an embedding for a sequence of words [63]. Doc2vec is able to learn low-dimensional representations for arbitrary lengths of text, ranging from phrases up to multiple paragraphs. Doc2vec is widely used for text classification [6, 53] or to compute document similarities [62]. During training, each sentence/paragraph is annotated with a unique ID. The first type of model is called Distributed Memory model (DM). It is based on the idea of CBOW [63]. Given the sentence or paragraph ID and a few words in the current window, the model predicts the subsequent word. The second method is called Distributed-Bag-of-Words

[6] https://github.com/facebookresearch/fastText

(DBoW) [63]. The idea is similar to SG: given the paragraph ID the model tries to predict the words in a window.

Novel universal language models such as BERT [26] or ULMFiT [48] are not considered to be word embedding models, since strictly speaking they do not embed words into a vector space, hence they are not in the scope of this work.

2.1.3. Realization of Word Embedding Models

To build the models, we use the *gensim* software package [98] for the Word2Vec models and the GloVe toolkit[7] for the GloVe models. More specifically, we use these toolkits in Python. They allow querying any word in the model dictionary for its similarity with any other word. This means that for any word there is an indexed list containing every other word in the dictionary, sorted by similarity. For special models such as fasttext or Doc2Vec we have used their respective Python packages to build models.

In this work we use the terms *list index* and *position in the list* as synonyms. We differentiate between the *similarity values* of models and *similarity lists*. In the first case, we are only concerned with the similarity value of a word pair and not its position in those lists. In the second case, our interest is the opposite one. In our software settings similarity values are floating point numbers between -1 and 1, with 1 being the highest similarity, while similarity list indices are integers.

[7] http://nlp.stanford.edu/projects/glove/

2.2. Text Classification

In this section we introduce the notation and baseline algorithms of Text Classification. First, we give a general introduction to text classification, with the fundamental notation. Then, we introduce a preprocessing method called term substitution, which is the fundamental idea behind our method introduced in the application analysis section (see Section 5.2). Finally, we present the text classification algorithms that we use in this work.

2.2.1. Background on Text Classification

The goal of text classification is to assign categories to textual data, also called *documents*, according to their contents.

A document d, is a tuple of $|d|$ *words* such that $d = (v_1, \ldots, v_{|d|})$, where v_j refers to the jth word of the document. We define a set of documents $D = \{d_1, \ldots, d_{|D|}\}$. Unknown documents are $\tilde{d} = (\tilde{v}_1, \ldots, \tilde{v}_{|\tilde{d}|})$. The term "word" can also refer to a sequence of numbers or symbols, or a syntactically or semantically tagged word. The *domain of documents* $\mathcal{D}$ and the *domain of words* $\mathcal{V}$ define the sets of all possible documents and all possible words, respectively.

For a set of documents D we define the *vocabulary of* D as follows:

$$voc(D) = \left\{ v_j \mid d \in D, j = 1, \ldots, |d| \right\}.$$

An important issue to handle in text classification is Out-Of-Vocabulary (OOV) words. OOV words are words that are not part of the classifier's known vocabulary. Formally, these are the words:

$$\tilde{V} := \mathcal{V} \setminus V.$$

As a consequence, OOV words cannot be used for prediction, since they do not have any representation in the classifier's feature space.

Labels. A document set D is called a *labeled dataset* if there is a function $\omega : D \to \Omega$, that assigns a label to each document $d \in D$, where $\Omega = \left\{\omega_1, \omega_2, \ldots, \omega_{|\Omega|}\right\}$ is the set of classes defined for the current classification problem. The function ω is called a *labeling over* D. In this work, a dataset identified with D is a labeled dataset with labeling ω. *Unlabeled documents* or *unknown documents* are the documents $\tilde{d} \in \mathcal{D} \setminus D$. The subset of documents in a class $\omega \in \Omega$ is denoted as:

$$D_\omega := \{d \mid \omega(d) = \omega\}.$$

In this work, each document has exactly one class associated to it and we will only consider the case of *binary classification.* This means, the label of a document is either "+" (positive) or "−" (negative), i.e., $\Omega = \{-, +\}$. [8]

Word Frequencies. $N_{v,d}$ refers to the amount of occurrences of word $v \in \mathcal{V}$ in a document d. Formally,

$$N_{v,d} = \sum_{j=1}^{|d|} \mathbb{1}\{v_j = v\},$$

where $\mathbb{1}\{\cdot\}$ denotes the characteristic function.

Also, $N^D_{v,\omega}$ refers to the number of occurrences of word v in documents $d \in D$ with label $\omega(d) = \omega$. Formally,

$$N^D_{v,\omega} = \sum_{d \in D_\omega} N_{v,d}.$$

Similarly, the total amount of occurrences of v in a dataset D is denoted as:

$$N^D_v = \sum_{d \in D} N_{v,d}.$$

2.2.1.1. Bag-of-Words Model

One of the most used document representations method in text classification is the *Bag-of-Words (BoW)* representation [77, 52]. Two of the most successful classifiers in text classification are based on the BoW model, namely the Multinomial Naive Bayes and the linear Support Vector Machine with Naive Bayes classifiers. We will introduce these models in Section 2.2.3. The idea of the BoW representation is to use the statistical information yielded by words frequencies to predict the classes of unknown documents. In order to do this, one needs to transform the training textual data into a numerical representation. A document d can be represented as a BoW as follows:

[8] Multi-class problems — i.e., the scenarios in which there are multiple labels for a document or more than two classes — can be handled by solving several binary classification problems.

$$bow_V : \mathcal{D} \to \mathbb{N}^{|V|} =: \mathcal{B}_V,$$
$$bow_V(d) := \left(N_{d,v_1}, \ldots, N_{d,v_{|V|}})\right)^\intercal,$$

where $V = \{v_1, \ldots, v_{|V|}\} \subset \mathcal{V}$. In other words, a BoW is a vector of the size of the vocabulary V. The vector contains non-negative integers, such that the j-th element of the BoW vector indicates the frequency of word $v_j \in V$ in document d. $\mathcal{B}_V$ denotes the feature space defined by a BoW over a vocabulary V.

Next, we define the binary Bag-of-Words (bBow), as follows:

$$bbow_V : \mathcal{D} \to \{0,1\}^{|V|} := \mathcal{B}_V^b,$$
$$bbow_V(d) := (\mathbb{1}\{b_1 > 0\}, \ldots, \mathbb{1}\{b_{|V|} > 0\}).$$

In this case, the j-th element of a bBoW vector is 1 if v_j is present in the corresponding document and 0 otherwise. The bBoW feature space is denoted as $\mathcal{B}_V^b$.

Generally, in a BoW model the order of the words in a document is neglected and only the presence and the frequency of word occurrences is relevant. (In contrast, in n-gram representations the order is maintained as well. See Section 2.3.)

2.2.2. Dimensionality Reduction in Text Classification

One of the biggest issues in text classification is that because of the complexity of human language the dimensionality of the feature space quickly becomes very big. This means, even for relatively small datasets, the number of unique terms is usually large. Having a high dimensional feature space in text classification introduce three main issues, which are detrimental to classification accuracy. First, low frequency words, which are hard to train, because of their limited context. Second, OOV words, which are not in the vocabulary, hence cannot be used to prediction. Third, overfitting — i.e., when the classifier is falsely optimized to explain the training data instead of generalizing the classification problem [102]. In order to overcome these problems, various techniques, including the preprocessing method proposed in this work, have been developed.

One family of approaches is called dimensionality reduction techniques. The goal of dimensionality reductions is to reduce the feature space without losing information that is relevant to the underlying classification task. The dimensionality reduction technique that is used in this work, is called *Term substitution*. Term substitution is a preprocessing method, which aims at reducing feature dimensionality by transforming the set of terms V in the training data to a *synthetic* set of terms V' with $|V'| < |V|$ [102], i.e., it reduces the number of unique terms in the training data. In the following we explain how term substitution works in detail.

2.2.2.1. Term Substitution

We now introduce term substitution through an example, while addressing the question of how the preprocessing done by term substitution implies an alleviation of the above-mentioned challenges.

Increasing Statistical Robustness. In order to show how substitution works for text classification, we show an exemplary polarity task that we solve using the Multinomial Naive Bayes model (see Section 2.2.3). Table 2.1 show the distributions of some of the words from the dataset.

v	$N(v,+)$	$N(v,-)$	$\hat{P}(v\|+)$	$\hat{P}(v\|-)$
she	19	21	$2.6 \cdot 10^{-3}$	$2.9 \cdot 10^{-3}$
is	42	43	$5.7 \cdot 10^{-3}$	$5.9 \cdot 10^{-3}$
smart	9	3	$1.2 \cdot 10^{-3}$	$4.1 \cdot 10^{-4}$
intelligent	1	1	$1.4 \cdot 10^{-4}$	$1.4 \cdot 10^{-4}$

Table 2.1.: The distribution of words in an exemplary dataset

$N(v,+)$ and $N(v,-)$ are the number of occurrences of the word in the positive and negative classes, respectively. $\hat{P}(v|+)$ and $\hat{P}(v|-)$ are the probabilities of the word occurring in the positive and negative classes, respectively.

The task is to predict the polarity of the sentences “she is smart” and “she is intelligent”. Their true labels are positive. The sentences are represented by the following tuples:

$$d_1 = (\text{'she'}, \text{'is'}, \text{'smart'}),$$
$$d_2 = (\text{'she'}, \text{'is'}, \text{'intelligent'}).$$

With the assumption of prior class probabilities $\hat{P}(+) = \hat{P}(-) = 0.5$, the estimated posteriori probabilities of d_1 are :

$$\hat{P}(d_1|+) = \hat{P}(\text{'she'}|+) \cdot \hat{P}(\text{'is'}|+) \cdot \hat{P}(\text{'smart'}|+) = 17.8 \cdot 10^{-6}\%,$$
$$\hat{P}(d_1|-) = \hat{P}(\text{'she'}|-) \cdot \hat{P}(\text{'is'}|-) \cdot \hat{P}(\text{'smart'}|-) = 6.9 \cdot 10^{-6}\%,$$

$$\hat{P}(+|d_1) = \frac{\hat{P}(d_1|+) \cdot \hat{P}(+)}{\sum_{\omega' \in \{+,-\}} \hat{P}(d_1|\omega')} = 72.1\%,$$
$$\hat{P}(-|d_1) = \frac{\hat{P}(d_1|-) \cdot \hat{P}(-)}{\sum_{\omega' \in \{+,-\}} \hat{P}(d_1|\omega')} = 27.9\%.$$

Analogously, for d_2 the posterior probabilities are:

$$\hat{P}(d_1|+) = \hat{P}(\text{'she'}|+) \cdot \hat{P}(\text{'is'}|+) \cdot \hat{P}(\text{'intelligent'}|+) = 20.8 \cdot 10^{-7}\%,$$
$$\hat{P}(d_1|-) = \hat{P}(\text{'she'}|-) \cdot \hat{P}(\text{'is'}|-) \cdot \hat{P}(\text{'intelligent'}|-) = 23.9 \cdot 10^{-7}\%,$$

$$\hat{P}(+|d_2) = \frac{\hat{P}(d_2|+) \cdot \hat{P}(+)}{\sum\limits_{\omega' \in \{+,-\}} \hat{P}(d_2|\omega')} = 46.5\%,$$
$$\hat{P}(-|d_2) = \frac{\hat{P}(d_2|-) \cdot \hat{P}(-)}{\sum\limits_{\omega' \in \{+,-\}} \hat{P}(d_2|\omega')} = 53.5\%.$$

Since $\hat{P}(+|d_1) > \hat{P}(-|d_1)$, the naive bayes classifier correctly assigns d_1 to the positive class, however in the meantime incorrectly assigns d_2 to the negative class, since $\hat{P}(-|d_2) > \hat{P}(+|d_2)$. This happens because the words 'she' and 'is' occur slightly more often in the negative class. These general words are obviously not decisive for the classification task. The more important word 'intelligent', which indicates a positive polarity, is very infrequent and by chance it is evenly distributed over the classes, hence does not provide any discriminatory information. Consequently, the model's prediction for d_2 tends towards the negative class.

At this point term substitution comes in the picture. Formally, it can be described as follows. For a labeled dataset D with vocabulary $V = voc(D)$, for each $v \in V$ a set of words $T_v \subset \mathcal{V}$ that are semantically similar to v is created. Both the documents $d \in D$ and unknown samples $\tilde{d} \in \mathcal{D} \setminus D$ can then be preprocessed as follows:

$$\boldsymbol{\pi} : \mathcal{D} \to \mathcal{D},$$
$$\boldsymbol{\pi}(d) = (\pi(v_1), \ldots, \pi(v_{|d|})),$$
$$\pi(v_j) = \begin{cases} v, & \text{if } v_j \in T_v \\ v_j, & \text{otherwise} \end{cases}, \text{ for } j = 1, \ldots, |d|.$$

We now recompute the probabilities of the dataset, after preprocessing it using the substitutions given by the following dictionary:

$$T_{\text{smart}} = \{\text{'intelligent', 'bright', 'knowledgeable', 'clever', } \ldots\}.$$

During preprocessing the training data $\boldsymbol{\pi}$ substitutes the occurrences of 'intelligent' with 'smart'. After the substitutions, the word frequencies in the dataset become:

v	$N(v,+)$	$N(v,-)$	$\hat{P}(v\|+)$	$\hat{P}(v\|-)$
she	19	21	$2.6 \cdot 10^{-3}$	$2.9 \cdot 10^{-3}$
is	42	43	$5.7 \cdot 10^{-3}$	$5.9 \cdot 10^{-3}$
smart = intelligent	10	4	$1.3 \cdot 10^{-3}$	$5.5 \cdot 10^{-4}$

After the frequencies of the words ‘smart’ and ‘intelligent’ are merged, we compute the posteriors the same as we did for the first model:

d	$\hat{P}(+\|d)$	$\hat{P}(-\|d)$
$\boldsymbol{\pi}(d_1)$	71.2%	28.8%
$\boldsymbol{\pi}(d_2)$	71.2%	28.8%

We can see that using the preprocessed dataset, both documents are correctly classified. This simplified example demonstrates how text classification can be improved using semantic knowledge: substituting the word ‘intelligent’ with ‘smart’ is equivalent to utilizing the knowledge that these terms are semantically related. In the text classification algorithm, the statistical information provided by the more frequent term ‘smart’ inferred a more robust estimate of the statistics of the infrequent term ‘intelligent’. Using the combined word frequencies, the second model classified d_2 correctly.

Substitution of OOV Words. The smaller the training data is, the more probable it is to encounter words in the unknown documents that are not in the vocabulary of the classification model, i.e., OOV words. To show the negative effects that OOV words can have on classification accuracy, we continue the example presented previously.

Let us assume that the goal is to classify the unknown document “she is knowledgeable” (d_3). Again, we assume that the true label is positive ($\omega(d_3) = +$). Similarly, as for d_2 the posterior probability for the negative class is larger than for the positive class. This is because the word ‘knowledgeable’ is not in the vocabulary of the classification model, hence it cannot be incorporated in the probability estimation. This means, the posteriors only depend on the words ‘she’ and ‘is’:

$$\begin{aligned}
\hat{P}(d_3|+) &= P('she'|+) \cdot P('is'|+) = 14.8 \cdot 10^{-4}\%, \\
\hat{P}(d_3|-) &= P('she'|-) \cdot P('is'|-) = 17.1 \cdot 10^{-4}\%, \\
\hat{P}(\omega|d_3) &= 46.4\%, \\
\hat{P}(\omega|d_3) &= 53.6\%.
\end{aligned}$$

Again, if we use the list of semantically similar words of T_{smart} to preprocess the document we obtain the same sentence:

$$\boldsymbol{\pi}(d_3) = (\text{'she'}, \text{'is'}, \text{'smart'}).$$

Replacing the OOV word 'knowledgeable' with the known 'smart', the posterior probabilities will be the same as for d_1 and d_3, hence d_3 is correctly classified after preprocessing.

2.2.2.2. Equivalence of Lexical Substitution and Bag-of-Clusters

In this section, we show that the problem of word substitution is equivalent to finding clusters of similar words and employing the discovered clusters as the new features.

Let $V = voc(D)$ be the vocabulary of a document set D. A clustering algorithm yield K pairwise disjoint clusters of semantically similar words $\mathcal{C} = \{C_1, \ldots, C_K\}$ with $C_k \subset V$, for $k = 1, \ldots, K$, by using a dissimilarity measure over pairs of words $v, v' \in V$. We define the *Bag-of-Clusters* for document d over a clustering $\mathcal{C}$ as follows:

$$\begin{aligned} boc_{\mathcal{C}} &: \mathcal{D} \to \mathbb{N}^{|C|}, \\ boc_{\mathcal{C}}(d) &= (c_1(d), c_2(d), \ldots, c_K(d)), \\ c_k(d) &:= \sum_{v \in C_k} N_{v,d}. \end{aligned}$$

The k-th element of the feature vector in a BoC representation contains the combined counts of the words in cluster C_k. We can obtain the the same features by representing the document as a BoW, if in every document the occurrences of a term $v \in C_k$ is replaced by a fixed cluster member $v_k^* \in C_k$.

$$\begin{aligned} v_k^* &\in C_k, k = 1, \ldots, K, \\ V^* &:= \{v_k^* \mid k = 1, \ldots, K\}, \\ d &:= (v_1, \ldots, v_{|d|}) \in D, \\ \pi(v) &:= v_k^*, \text{if } v \in C_k, \\ \boldsymbol{\pi}(d) &:= (\pi(v_1), \pi(v_2), \ldots, \pi(v_{|d|})), \\ &\Rightarrow bow_{V^*}(\pi(d)) = boc_{\mathcal{C}}(d). \end{aligned}$$

This means, if the clusters $C_k \in \mathcal{C}$ consist of semantically similar words, representing a document as a Bag-of-Clusters over $\mathcal{C}$ is equivalent to performing lexical substitution.

2.2.2.3. Term Clustering

As we have just seen Term Clustering is equivalent to term substitution. It also aims at reducing the dimensionality and the statistical noise introduced by infrequent and OOV

terms. The goal is to find clusters of similar terms and use them, instead of single words, to represent documents as a Bag-of-Clusters (BoC). There are two main approaches how to cluster similar terms, which we explain in the following.

Semantic Clustering. Semantic clustering has been widely researched, especially since word embedding models gained much popularity. Ma et al. [75] used the cosine similarity between word vectors to cluster terms using K-Means. Their results suggest that certain values of K might yield a slightly better classification accuracy over the unprocessed training data. Wang et al. [117] uses word embedding-based clustering in combination with neural networks. They use the Euclidean distance in the embedding vector space to map similar phrases (n-grams) to the same neural units. They have evaluated their approach on short text classification tasks and reported slight improvements in classification accuracy over other approaches.

In numerous cases, text classification accuracy often has been worse after preprocessing the training data based on semantic clustering than using the original datasets. We see two reasons for this. The first one is the inability of word embedding models to distinguish antonyms and synonyms. For example, the words 'hot' and 'cold' are close to each other in the embedding space, because their context tends to be similar. Second, there is the problem of task-specific synonyms. Task-specific synonyms are words that are used synonymously with respect to the classification task. For example, the words 'manager' and 'CEO' should be in the same cluster for a general text classification task, but not when we want to distinguish changes in the *upper-management* of specific companies. The first problem could be addressed using more sophisticated methods to generate word embeddings [115] or to detect antonyms [88]. As for the second one, recently it has become common to use pre-trained word embeddings and to fine-tune them to integrate task-specific information [48, 26]. However, this does not work at all for smaller data sets. This is because fine-tuning changes only the embeddings of words seen in the training data. Hence it distorts the word structure of the pre-trained model when using small training data sets. This ultimately hurts its generalization performance [35, 127].

Distributional Clustering. In numerous cases, the decision of whether or not to substitute a word with another semantically similar word depends on the underlying classification task. For example, assume a text classification task in which the goal is to make a distinction between good and enthusiastic movie reviews. The word frequencies for a sample dataset are as follows:

Word	Positive Class	Negative Class
excellent	10	3
good	1	10

In this case, in spite of the semantic similarity of the two words they should not be substituted. This is because the terms are distributed differently among the classes, i.e.,

the *distributional information* strongly suggests that these two words are not task specific synonyms. Substituting 'smart' with 'excellent' would result in a loss of information[9]. This phenomenon can be observed in many classification tasks and should be handled with care.

In previous literature, Baker et al. [7] propose clustering terms using a distributional metric based on a variant of the Kullback-Leibler divergence. The suggested metric expresses the discriminative information loss given by clustering two terms together. The proposed clustering algorithm greedily clusters (see Section 2.2.2.4) terms with minimal information loss. The results of their experiment showed that feature dimensionality can be heavily reduced without losing much classification accuracy. Nonetheless, no improvement in comparison to unprocessed datasets were reported. Based on Baker et al.'s work, Slonim and Tishby [105] propose an improved clustering algorithm using the *Information Bottleneck Method*. The algorithm maximizes the mutual information of a word and its cluster with respect to their relative distribution over the categories. They also develop a clustering algorithm, in which the pairwise cluster-distances are updated as the clusters change. They show that their method can improve classification accuracy, however only when datasets are very small. For large datasets, the quality of classification decays.

Semantic-Distributional Clustering. Any existing term clustering approach refers to one of the previous groups. In other words, either they do not integrate language semantics, or they do not consider the distributional properties of the classification task. In Section 5.2 we present our preprocessing method which includes both kinds of information. As our experiments show, using semantic-distributional clustering is very suitable for text classification and is crucial for consistent improvements in text classification accuracy. It also complements the novel fine-tuning approaches well: In contrast to them, our method is most effective with small training data.

Moreover, there is another advantage of combining the two clustering methods, described in the following. Embedding models contain neural networks with millions of parameters. This makes understanding their clustering decisions just by evaluating the underlying model virtually impossible. In contrast, this is not the case for distributional methods where we can clearly interpret why two words are clustered together or not based on the distributional statistics. Such with the combination of the two approaches we are able to employ the robustness of embedding models while retaining the explainability of the method to the user.

[9] Note that this is only due to this specific classification task. In most other scenarios in turn, it would make sense to merge these words, i.e., consider them as task-specific synonyms.

2.2.2.4. Greedy Agglomarative Hierarchical Clustering

To cluster terms we need an algorithm which creates the actual clusters based on the distance measure between the words. In this work we use *Greedy Agglomarative Hierarchical Clustering* to find the word clusters. One advantage of using agglomerative clustering, is that it we can use any distance function to create the dissimilarity matrix. It works as follows.

For a set of documents D and vocabulary $V = \{v_1, \ldots, v_{|V|}\}$ we define a symmetrical *dissimilarity matrix* $\mathbf{W} \in \mathbb{R}^{|V|\times|V|}$. The elements of the matrix w_{ij} represent the dissimilarity between the term v_i and v_j according to a distance function $\Lambda_{\mathcal{W}}$. Since $\mathbf{W}$ is symmetrical $w_{ij} = \Lambda_{\mathcal{W}}(v_i, v_j) = \Lambda_{\mathcal{W}}(v_j, v_i) = w_{ji}$.

The greedy agglomerative hierarchical clustering is a bottom-up clustering algorithm. At the first (step $i = 0$) every word $v \in V$ forms its own cluster. At each step i the clusters are merged into bigger clusters. When the number of clusters equals a pre-defined number K, the algorithm stops. To merge two clusters or not in an iteration is based on the dissimilarity matrix $\mathbf{W}$ and a *linkage criterion* λ. At every step $i > 0$ the two closest clusters C, C' are merged:

$$C'' = \min_{C,C' \in \mathcal{C}_{i-1}} \lambda(C, C'),$$

where $\mathcal{C}_{i-1}$ denotes the clustering yielded at step $i - 1$.

There are various linkage functions, such as the minimum, the maximum or the average distance between words in the different clusters. In this work, we choose λ to be the maximum, also known as complete-linkage clustering. In contrast to the single-linkage, where two clusters are merged together based on their closest members, complete-linkage clustering avoids the so called chaining phenomenon. This means, using another linkage criterion, clusters that have many elements may contain elements that are very distant to each other, but because of successive elements that are close to each other they are merged. This may be useful in other applications, but in our case we make sure that every word fits in its cluster. Formally,

$$\lambda(C, C') = \max_{v_i \in C, v_j \in C'} w_{ij}.$$

The output of the algorithm is a clustering $\mathcal{C} = \{C_1, \ldots, C_K\}$ over V, where $C_k \subset V$ and $C_k \cap C_{k'} = \emptyset$, for every $k \neq k'$.

2.2.3. Text Classification Algorithms

In this section we give a brief overview on their history and introduce the most important models used in TC, namely the Multinomial Naive Bayes, the Support Vector Machine and the Neural Network-based models.

2.2.3.1. Naive Bayes Classifiers

Naive Bayes (NB) classifiers belong to the family of probabilistic classifiers. Various NB models have been used for text classification [29, 69]. These classifiers estimate the probability of a class ω given a document d ($P(\omega|d)$) and assign the document to the class with the highest probability [77]. All these models are characterised by the "naive" assumption that word occurrences are conditionally independent. Despite being based on such a vague assumption, classification performance of NB models show great performance [118].

In general, NB is usually outperformed by other statistical classifiers, such as SVM on most classification tasks, however, optimization techniques can make NB models competitive [56, 78]. Also, it has been shown that NB models often perform better on short text classification tasks than SVMs [118]. Moreover, NB models are fast both in terms of implementation and computation time, which makes them popular in practical applications.

In this work we use the Multinomial Naive Bayes (MNB) model, since it has been shown that it outperforms the other NB variants in almost all classification tasks [29].

The Multinomial Naive Bayes Classifier. Naive Bayes classifiers, such as the Multinomial Naive Bayes classifier, perform classification by estimating the probability of a class given a document. In MNB we assume that the frequencies of words have been generated by a multinomial distribution. We calculate these probabilities based on the *Bayesian rule*:

$$P(\omega|d) = \frac{P(d|\omega)P(\omega)}{P(d)} = \frac{P(d|\omega)P(\omega)}{\sum_{\omega' \in \Omega} P(d|\omega')P(\omega')}. \tag{2.2}$$

In Bayesian terminology $P(\omega|d)$ is called the *posterior*, while $P(\omega)$ is called the *prior* probability. The *likelihood* $P(d|\omega)$ is calculated using the probabilities of the words occurring in d:

$$P(d|\omega) = P(v_1, \ldots, v_{|d|}|\omega).$$

The "naive" assumption in NB models is that word occurrences are conditionally independent. Formally, we assume that:

$$P(v_1, \ldots, v_{|d|}|\omega) = \prod_{i=1}^{|d|} P(v_i|\omega). \tag{2.3}$$

To be able to predict the label of d we estimate the probabilities in equation 2.2.

First, we estimate the prior $P(\omega)$. It is the relative frequency of positive/negative samples in the labeled dataset:

$$\hat{P}(\omega) = \frac{|D_\omega|}{|D|}. \tag{2.4}$$

The likelihood of the model is approximated as follows:

$$\hat{P}(d|\omega) \coloneqq \prod_{v \in d} \frac{1 + N^D_{v,\omega}}{|V| + \sum_{v' \in V} N^D_{v',\omega}}. \tag{2.5}$$

The estimate of the probability of word v appearing in class ω ($\hat{P}(v|\omega)$) is calculated by the total count of the word in class ω divided by the counts of all words in that class. There are additional smoothing constants 1 in the numerator and $|V|$ in the denominator. These constants are used to avoid the product probability to be zero just because a word doesn't appear in ω. The rationale behind this smoothing, called *Laplacian Smoothing*, is that even if a word do not appear in a class ω, its true probability $P(v|\omega)$ is greater than 0 [77].

Finally, using equations 2.3, 2.4 and 2.5 we are able to estimate the posterior probability $P(d|\omega)$. Since $P(d)$ in Equation 2.2 does not depend on the class ω, it is enough to estimate the following term to later establish which posterior probability is larger.

$$\hat{P}(\omega|d) \propto \hat{P}(\omega)\hat{P}(d|\omega) = P(\omega) \prod_{i=1}^{|d|} P(v_i|\omega) =: \ell(d, \omega).$$

Since we only work with binary classification tasks in this work, we define the binary Multinomial Naive Bayes *classifier* in the following. The MNB classifier is a function c_{mnb} that assigns a document to the class for which the term $\ell(d, \omega)$ is greater. Formally,

$$c_{mnb} : \mathcal{D} \to \{+, -\},$$
$$c_{mnb}(d) = \begin{cases} +, & \text{if } \ell(d,+) > \ell(d,-) \\ -, & \text{if } \ell(d,-) > \ell(d,+) \\ \text{rand}(\{+,-\}), & \text{if } \ell(d,+) = \ell(d,-). \end{cases}$$

In the third case, when both classes are equally probable a class is randomly chosen by the classifier.

2.2.3.2. Support Vector Machine Classifiers

In general, Support Vector Machine (SVM) classifiers try to find hyperplanes that separate the samples of the different classes the best. In most text classification approaches the linear SVM version is used, i.e., there is no kernel function. Since, SVM models tend to be more robust than NB models, they have been generally preferred over NB classifiers in text classification, especially when dealing with long documents or with small numbers of training samples per class [118, 122, 28, 116].

In this work, we use the linear SVM with Naive Bayes features (NBSVM) classifier introduced by Wang et al. [118]. As opposed to traditional linear SVMs, the trained weights are processed after training, which makes NBSVM perform better than NB and SVM models in almost all classification tasks [118].

The Support Vector Machine with Naive Bayes Classifier. The NBSVM classifier uses the log-ratios of word frequencies as features in contrast to the baseline linear SVM approach, which uses the feature space of BoW to classify documents [77]. For dataset D, vocabulary $V = voc(D) = \{v_1, \ldots, v_{|V|}\}$, classes $\Omega = \{+, -\}$ and a document d, we define:

$$
\begin{aligned}
\mathbf{p}(d) &\coloneqq \left(\mathbb{1}\{N_{v_1,+} > 0\} + \alpha, \ldots, \mathbb{1}\{N_{v_{|V|},+} > 1\} + \alpha\right)^\intercal, \\
\mathbf{q}(d) &\coloneqq \left(\mathbb{1}\{N_{v_1,-} > 0\} + \alpha, \ldots, \mathbb{1}\{N_{v_{|V|},-} > 0\} + \alpha\right)^\intercal, \\
\mathbf{r}(d) &\coloneqq \log\left(\frac{\mathbf{p}(d)/\|\mathbf{p}(d)\|_1}{\mathbf{q}(d)/\|\mathbf{q}(d)\|_1}\right), \\
\mathbf{f}(d) &\coloneqq \mathbf{r}(d) \circ bbow(d; V),
\end{aligned}
$$

where $\circ$ denotes the elmentwise vector multiplication and α is the Laplacian Smoothing parameter. The vector representation $\mathbf{f}(d)$ of a document defined above is used to train the weights $\mathbf{w}$ and the bias b of each feature. To calculate the weights the NBSVM algorithm minimizes the following expression, based on the Euclidean distance:

$$
\mathbf{w}^\intercal\mathbf{w} + C\sum_{d\in D} \max\left(0, 1 - y(d)(\mathbf{w}^\intercal\mathbf{f}(d) + b)\right)^2,
$$

$$
y(d) \coloneqq \begin{cases} +1, & \text{if } \omega(d) = + \\ -1, & \text{if } \omega(d) = - \end{cases}
$$

After the regularization of the weights using parameter $\beta \in [0, 1]$, the classifier based on this model is a function c_{nbsvm} defined as follows:

$$\begin{aligned}
\hat{y} &: \mathcal{D} \rightarrow \{+1, -1\}, \\
\hat{y}(d) &= (\mathbf{w}'^{\mathsf{T}}\mathbf{f}(d) + b), \\
c_{nbsvm} &: \mathcal{D} \rightarrow \{+, -\}, \\
c_{nbsvm}(d) &= \begin{cases} + & \text{if } \hat{y}(d) = +1 \\ - & \text{if } \hat{y}(d) = -1 \\ rand(\{+, -\}) & \text{if } \hat{y} = 0. \end{cases}
\end{aligned}$$

Intuitively, the classifier decides based on whether a vector of a unknown document $\mathbf{f}(d)$ is on the left ($\hat{y}(d) = -1$) or on the right ($\hat{y}(d) = +1$) side of the hyperplane defined by the normal vector $\mathbf{w}'$ and the bias b.

2.2.3.3. Neural Networks

Neural network-based training algorithms appropriate for multilabel classification have been first applied by Zhang et al. in 2006 [125]. The authors reported improvements in classification accuracy on the *Reuters* dataset. Since then, the ever-increasing computational power and memory resources allowed methods using neural networks to become highly popular in text classification applications. Nowadays, almost every state-of-the-art approach uses neural networks for model building or prediction. However, in general, to train a neural network-based model capable of generalizing the underlying classification task, because of the large numbers of parameters, large scale datasets are required [126].

The neural network-based classifier which we use in this work is presented in [107]. At the time it was introduced it has produced state-of-the-art classification accuracy on various datasets. It is based on Recursive Autoencoders (RAEs).

Autoencoders are special types of neural networks that compress the input into a lower dimensional representation (encoding), then reconstruct the output from this representation (decoding). The output should be as similar as possible to the input, ideally identical. In this sense autoencoders are data compression methods. Figure 2.8 shows the general architecture of an autoencoder.

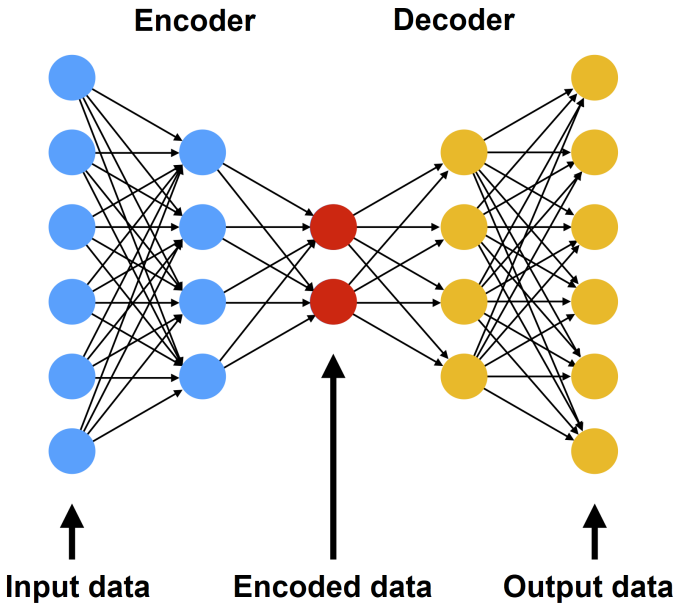

Figure 2.8.: General autoencoder architecture

The idea in Socher et al.'s work is that given a pre-trained word embedding model (in the original publication one based on the papers from Bengio et al. and Collobert et al. [10, 22]) it can be used as a representation of phrases and sentences as well. Figure 2.9 show the basic structure of a recursive autoencoder used in the publication[10].

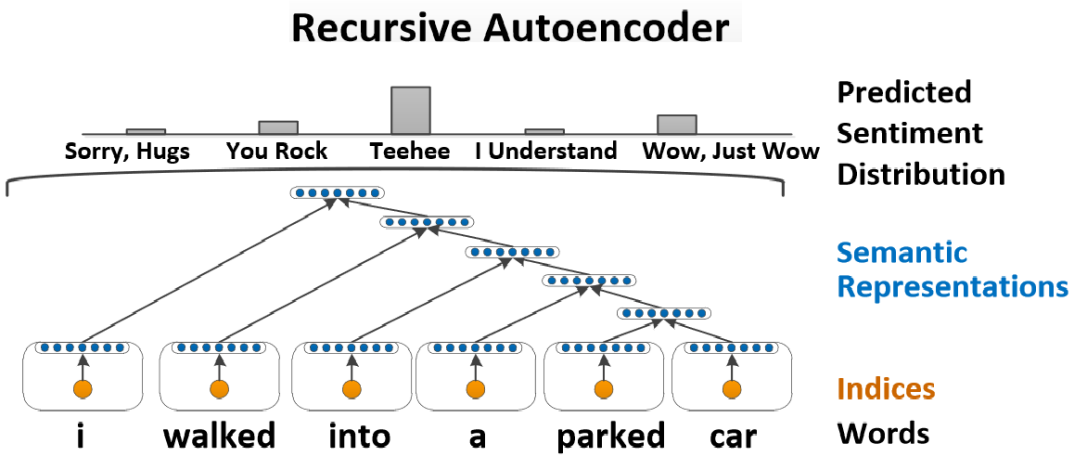

Figure 2.9.: Recursive autoencoder-based sentence embedding

[10] The figure is from the original publication.

As the figure shows the word representations in a sentence are recursively merged using the same autoencoder, until the full sentence has its own vector. In the meantime at every phase, the vectors are used as features to predict sentiment labels for the classification task. In general, the tree structure we can see in the figure is not given in advance. In Socher et al.'s work the tree is built in an unsupervised fashion with a greedy hierarchical algorithm, based on the autoencoders error. This means, at every step it merges the two vectors where the current reconstruction error is the smallest. Finally, each sentence has its own embedding vector, which can be used in different classification tasks.

For details we refer to the original publication [106].

2.3. N-grams

We have previously introduced BoW representations in Section 2.2.1.1. As we have already explained, in BoW representation the order in which words occur in a document is neglected. However, this order can often contain important information that are relevant to the classification task. For example, the sentences “no, it’s good" and “it’s no good" express the opposite sentiment, while their BoW representation is identical. Hence, a classifier operating on the BoW model would be unable to distinguish between the two sentences.

In spite of this, representing documents as BoW works well for many text classification tasks [77]. For example, the Multinomial Naive Bayes classifier estimates the posterior probability $\Pr(\omega|d)$ of a class ω given a document d by counting the occurrences of each word for each class (see Section 2.2.3), hence the statistics of word occurrences is every information it needs. In contrast to BoW, *n-grams* represent the local order of words as well.

2.3.1. Definition

Formally, an n-gram is a contiguous sequence of n words from a given document. Consider the following sentence:

I have three cats

The 1-grams are:

I	have	three	cats

The 2-grams are:

<s> I	I have	have three	three cats	cats </s>

The tokens “<s>” and “</s>” denote the start and the end of the sentence, respectively. To include n-grams representation of a dataset we concatenate each document with its n-grams as follows:

I	have	three	cats	<s> I	I have	have three	three cats	cats </s>

Formally we define the n-gram representation of a document as follows [118, 32]:

$$
\begin{aligned}
g_n &: \mathcal{D} \rightarrow \mathcal{D}, n \geq 2, \\
d &= (v_1, v_2, \ldots, v_{|d|}) \in \mathcal{D}, \\
g_n(d) &= (\underbrace{(< s >, v_1, \ldots, v_n)}_{\text{Tuple of } n \text{ contiguous words}} , \ldots, (v_{|d|-n+2}, \ldots, v_{|d|}, < /s >)),
\end{aligned}
$$

2.3.2. Building Word Embedding Models on N-grams

Distributional models conventionally are trained on full-text corpora by creating word-context pairs. The training corpus does not need to be a coherent text; it is sufficient if the sentences are meaningful. This means we can train word embedding models on n-gram corpora. When creating the context of a word, we treat each n-gram as if it was a sentence.[11]

Creating Fragmented Corpora. To create fragmented n-gram corpora from raw text, we use a simple method described in the following. With a sliding window of size n passing through the whole raw text, we collect all the n-grams which appear in the corpus and store them in a dictionary, together with their match count, i.e., how many times the n-gram occurs in the full-text corpus. This means that we create datasets similar to the Google Books dataset (see Section 2.4.1), but from other raw text such as the Wikipedia dump. For every fragmented corpus, we create different versions of it, by trimming n-grams from the corpora with regard to different minimum match count thresholds.

We store the n-gram training corpora in the Google n-gram format. This format comprises 4 values: the n-gram, the year, the book count (i.e., in how many books the *n*-gram has appeared in) and the match count. Figure 2.10 shows an example of a 5-grams from the Google Books dataset.

```
eaten nothing for two days      1988    7       7
eaten nothing for two days      1989    6       7
eaten nothing for two days      1990    2       2
eaten nothing for two days      1991    4       4
eaten nothing for two days      1992    8       8
eaten nothing for two days      1993    5       5
eaten nothing for two days      1994    5       6
eaten nothing for two days      1995    5       5
eaten nothing for two days      1996    11      11
eaten nothing for two days      1997    5       5
```

Figure 2.10.: Google Books n-gram example

[11] If there is a punctuation mark in the n-gram ending a sentence, it splits the n-gram into several sentences.

For our purpose, only the first and last values are relevant. When building a model with the n-gram versions, we deem every n-gram a sentence and use it as many times as it occurs in the raw text. We explain the impact of the window size and of the match count parameter in the following.

Window Size Parameter. The window size parameter (*win*) affect how the word embedding model is trained using differently fragmented corpus. For example, $win = 4$ is not meaningful when we work with 3-grams, because the maximum distance between two words in a 3-gram corpus is 2. The following examples illustrate how exactly the word-context pairs are generated on n-gram corpora depending on the size of the window.

Example 1. Let us look at the context of a specific word in a 5-gram corpus with $win = 4$. Let A B C D E F G H I be a segment of the raw text consisting of 9 words. In the raw text, the context of word E are words A, B, C, D, F, G, H, I. Now we create the 5-gram version of this segment and identify 5-grams which include word E. These are

(A B C D E); (B C D E F); (C D E F G); (D E F G H) and (E F G H I).

For word E on the 5-gram corpus, the contexts are words A, B, C, D; ...; words F, G, H, I. We can see that we have not lost any context words. But we also do not have all raw-text context words in one context, only fragmented into several ones.

Example 2. As extreme case, we consider a window size which is bigger than the size of the n-grams. In this setting, we naturally lose a lot of information. This is because distant words will not be in any n-gram at the same time. For example, look at the same text segment as in Example 1 with $win = 4$, but with a 3-gram variant of it. The raw-text context is the same as before for word E, but the fragmented contexts are words C, D; words D, F and words F, G.

Example 3. Another extreme case is when *win* is less than or equal to $\lfloor \frac{n-1}{2} \rfloor$. In this case at least one n-gram context will be the same as the full-text context. This means that no information is lost. However, there also are fragmented contexts in the n-gram variant which can influence the training and, hence, model quality.

We point out that bigger window sizes do not necessarily induce higher accuracy on various test sets, as explained by Levy et al. [68].

Match Count Parameter. Another parameter to consider when building the models on the n-gram corpora is the match count of the n-grams. The intuition behind including only higher match-count n-grams in the training data is that they may be more valid segments of the raw text, as they appear several times in the same order. For example, we exclude typos and meaningless combination of words. However, we naturally lose relevant information as well by pruning low match-count n-grams.

2.4. Datasets

In this section we introduce the different datasets that we use in this work. First, we present the corpora on which we train our word embedding models on. Then, we introduce the similarity and analogy as well as the text classification test sets.

All datasets were tokenized using NLTK's TwitterTokenizer, since it has a good recognition of tokens in written colloquial language. We did not exclude special characters or punctuation symbols and also didn't remove any stop-words in our datasets.

2.4.1. Training Corpus

Throughout this work we use three corpora to train word embedding models on.

- **1 Billion Word** [19]: This is one of the largest publicly available language-modeling benchmarks. The dataset is around 4 GB in size and contains almost 1 billion words in approximately 30 million English sentences. The sentences are shuffled, and the data is split into 100 disjoint partitions. This means that one such partition is 1% of the overall data [19].
- **Wikipedia**[12]: Another large publicly available corpus containing more than 3 billion words. Before training we shuffled the articles. For some parts of this work for comparability reasons we sampled the dump to contain approximately 1 billion words. We use a version of the dump downloaded on 01.11.2016.
- **Google Books** [81]: This is the largest currently available corpus with historic data which exists for several languages. The English version incorporates over 3 million books from the previous centuries split into n-grams with a size larger than 2 Terabytes. Figure 2.11 shows a screenshot of the Google n-gram viewer, a graphical interface that allows querying n-grams from the Google Books corpus.

All three corpora are good benchmark datasets for language modeling, with their huge size, large vocabulary and topical diversity [19, 124].

[12] https://dumps.wikimedia.org/enwiki/

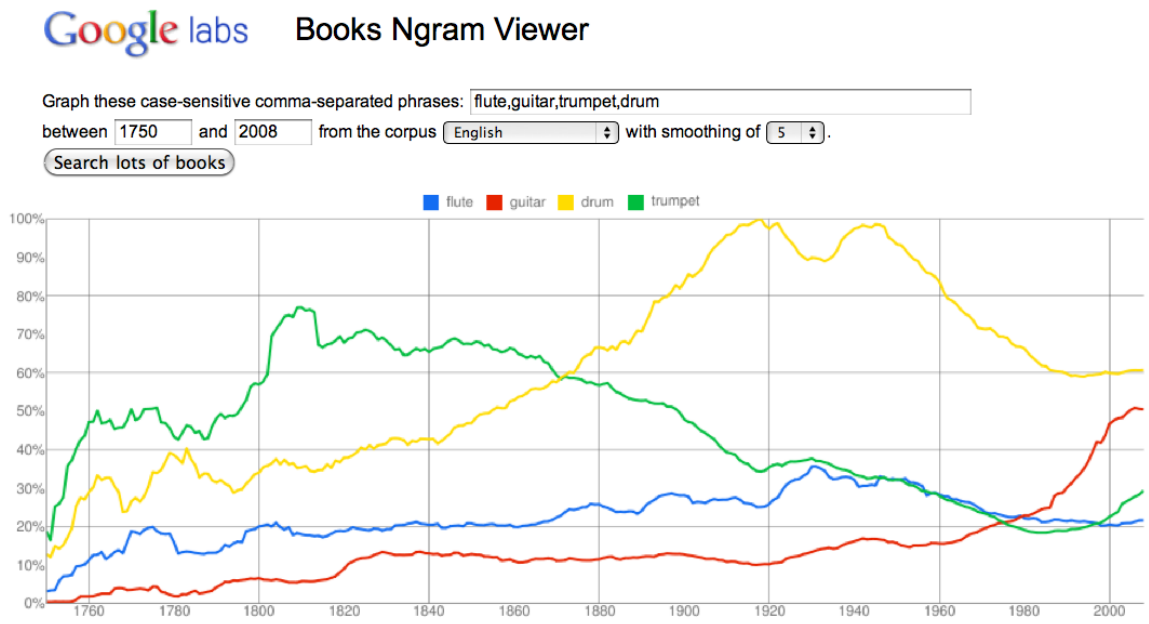

Figure 2.11.: The Google n-gram viewer

2.4.2. Similarity and Analogy Test Sets

In this section we present the similarity and analogical reasoning test sets that we use in this work.

Word Similarity. We use six test sets to evaluate word similarity. Every test set has the same three column format, which is a word pair and a corresponding similarity score.

- **WordSim353** [30]: 353 similar or related word pairs with average similarity scores given by human annotators.
- **MEN** [16]: 3000 pairs of words scored on a [0, 1] normalized semantic relatedness scale via ratings obtained by crowd sourcing on the Amazon's Mechanical Turk.
- **SimLex-999** [43]: 999 pairs of words that quantifies similarity, such that pairs related by association or relatedness have a low rating.
- **RG-65** [100]: 65 pairs of words, in a range from highly synonymous to totally unrelated words.
- **Rare Words** [74]: 2034 word pairs which are selected in a way to reflect words with low occurrence frequency.
- **Mechanical Turk** [96]: 771 word pairs obtained with Amazon's Mechanical Turk.

In general, we evaluate the models with the conventional baseline evaluation method [68, 8]: We rank the word pairs of an evaluation test set by their similarity scores, based on cosine distance of the word vectors. Then we measure Spearman's correlation between this ranking and the one based on human annotators. This number is the score of the model on the test set.

In Section 5.2, we present a similarity value threshold-based evaluation method to compensate the fundamental flaws of the baseline method.

Analogical Reasoning. We use two analogical reasoning test sets.

- **MSR** [84]: 8000 syntactic analogy questions, such as "big is to biggest as good is to best".
- **Google** [85]: 19544 questions, both syntactic and semantic question, such as "Paris is to France as Rome is to Italy".

The models answer the questions with the following formula:

$$\text{argmax}_{d \in D \setminus \{a,b,c\}} \cos(d, b - a + c).$$

Here $a, b, c, d \in D$ are the vectors of the corresponding word. The score of a model is the percentage of questions for which the result of the formula is the correct answer (d).

2.4.3. Text Classification Test Sets

In this section we describe the Text Classification datasets used for evaluation[13].

- ***Customer Reviews* (CR)**: Short product reviews in colloquial English. The task is to discriminate good and bad reviews.
- ***MPQA***: A collection of short 2-3 word phrases. The task is to classify them based on their polarity (positive, negative).
- ***Subjectivity* (Subj.)**: Positive/negative subjective reviews and plot summaries.
- ***Short Movie Reviews* (RT)**: Short movie reviews with one sentence per review.

[13] The datasets evaluated in this work were downloaded at `https://github.com/sidaw/nbsvm`

In Table 2.2 we show detailed statistics of the datasets. $|D|$ stands for the size of dataset, $|V_1|$ is number of words. $|O_1|$ stands for the overlap of the words with the word2vec vocabulary. ℓ is the average document length. $|D_+|$ and $|D_-|$ stands for the number of positive and negative samples, respectively.

Dataset	$\|D\|$	$\|V_1\|$	ℓ	$\|D_+\|$	$\|D_-\|$
MPQA	10,603	6,298	3	3,311	7,292
CR	3,772	6,596	20	2,406	1,366
RT	10,662	20,621	21	5,331	5,331
Subj.	10,000	23,187	24	5,000	5,000

Table 2.2.: Statistics of the text classification datasets

Part II.

Three-Way Analysis

3. Algorithmic Analysis of Word Embedding Models

In this first part of our three-way analysis of word embedding models we aim at understanding word embedding model algorithms better. We have introduced the word embedding models we employ throughout this work in Section 2.1. These models have been widely studied, however key elements of their behavior regarding their training algorithms are not sufficiently clear.

First, it is not known how the models trained with different parameter settings differ in their similarity values. This is highly important, since if their distribution significantly changes when using different parameter settings, we cannot utilize different models the same in down-stream scenarios.

Second, although it is known that in order to create high-quality embedding models a good training corpus is needed, it is not clear what can we consider a good quality corpus. For example, by the definition of the training algorithms, it is not needed for the corpus to be coherent text. This means, we can train word embedding models on n-grams. Hence, we can use the Google Books corpus, which is by far the biggest text corpus publicly available. However, it is not known yet whether such fragmented corpora is suitable for training word embedding models.

Our contribution in this chapter is to answer both questions presented above. First, we systematically evaluate how different parameter settings influence the similarity value distributions of the models. We show that, with the exception of few marginal cases, the shape of the distributions are very similar, i.e., they only differ in their mean and standard deviation values.

Second, we answer the question whether using n-gram corpora is suitable for word embedding model training, by quantifying the quality differences of models trained on fragmented and full-text corpora. We conduct the experiments on both the Wikipedia dump and Chelba et al.'s 1-Billion word datasets and their respective fragmented versions. We conclude that n-gram corpora such as Google Books are valid training data for word embedding models.

To be able to answer the algorithmic questions of this work, we trained over 400 word embedding models. We have also created different versions of full text training corpora to train the models on. All this took more than three months of computing time on a modern computer.

3.1. An Investigation of the Influence of the Various Parameters

As the first part of the algorithmic analysis, in this section, we investigate how the different parameters affect the similarity values of word embedding models. These insights are relevant to understand word embedding models in general. We also require these insights in the next Chapter 4 for our theoretic analysis.

At the end of the section, we investigate the similarity value distributions produced by the alternative models introduced in Section 2.1.2.3 as well. We show that their distributions tend to be very similar to the ones of the word models, but not in every case.

3.1.1. Investigation Objectives

In previous literature every word embedding model have been treated the same, without taking into consideration that their similarity value distributions, and hence their downstream applicability, may be considerably different. For example, if a text classification application scenario only merges word pairs which are at least 0.7 similar to each other, it may merge a big amount of word pairs in one model, but only a few pairs for another. To this end, we need to evaluate how different training algorithms and their parameters affect the similarity value distributions of word embedding models. This has not been done before in previous literature. We will show that similarities in embedding models can differ significantly when trained with different parameters. To be more precise, we show that their similarity value distributions have statistical characteristics such as different mean values or different highest similarity values which can be significantly different.

Although the similarity value distributions of the models can significantly differ in certain characteristics, we hypothesize that they are all similar in shape, with only their means and standard deviations depending on the parameters.

Hypothesis 1. While the learning algorithms and parameters influence the similarity value distributions of the models, these distributions are very similar in shape.

We plan to confirm this hypothesis as follows. First we normalize all distributions, so that they have 0 mean and 1 standard deviation. We then randomly draw 1000 values from all distributions and pairwise compare the samples by means of the two-sample Kolmogorov-Smirnov (K-S) test [54] with 99% confidence. This test checks if two samples are drawn from the same distribution.

For the overall understanding of the similarity values and lists in word embedding models, it is important to know how the model selection and the parameters affect the similarities. Our main contribution in this section is that we conduct the evaluation systematically for all the parameters and models introduced in Section 2.1.

3.1.2. Evaluation Setup

In this section, we work with Chelba et al.'s 1 Billion word dataset as training corpus (see Section 2.4.1). We train all our models using this corpus.

In the following sections, for every parameter, we present our results in the same way. In particular, we graph results in two figures. First there are similarity value distributions of the models. For these plots, we randomly select 10,000 words from the model dictionary and calculate the similarity values of *every other word* to them. Then we group the values in 0.01 intervals and count the number of values in each group. Thus, the x-axis represents the similarity values from [-1,1], the y-axis the share of the values per group.

The second figures contain the results from the similarity lists experiments. In these experiments, we randomly select 10,000 words $(v_1, v_2, \ldots, v_{10000})$ from the dictionary of the model . Their respective word vectors are $(vec(v_1), \ldots, vec(v_{10000}))$.

For each of these words, we compute the most similar one thousand words

$$(v_{i,1}, v_{i,2}, \ldots, v_{i,1000}) \text{ for } i \in \{1, \ldots, 10000\},$$

together with their respective similarity values, $t_{i,1}$, $t_{i,2}$, . . . , $t_{i,1000}$, where

$$t_{i,j} = cos - sim\left(vec(v_i), vec(v_{i,j})\right),$$

i.e., $t_{i,j}$ is the similarity value of words v_i and $v_{i,j}$. The list $v_{i,1}, v_{i,2}, \ldots, v_{i,1000}$ is sorted by the similarity values in descending order. Because of this sorting for every i , it holds that $t_{i,j_1} \geq t_{i,j_2}$, for any $j_1 < j_2$. We then calculate the average similarity value for every list index

$$avg\_sim\,(j) = mean\left(t_{\cdot,j}\right).$$

Finally, we plot the results with the x-axis being the list indices j and the y-axis the average similarities $(avg\_sim\,(j))$. Although the $avg\_sim\,()$ function is only defined for arguments that are natural numbers, the plots connect the points to arrive at a smooth curve, for better visibility.

At this point we are not trying to answer *why* different parameters affect the similarity values as they do; we are investigating *how* they affect the values. This means that we are not making qualitative statements, i.e., we are not concerned how parameters affect the quality of the models on different semantic tasks. We are not making any statement that any model is better or worse than the other one, but only how and to which extent they are different. In other words, we focus on the hypothesis from Section 3.1.1.

3.1.3. Model Selection

The first parameter whose effect we investigate is the model itself. We consider the three models already introduced, namely Word2Vec SG, Word2Vec CBOW and GloVe. We build all three models on the full 1 billion words dataset with the same parameter settings. As we have noted in Section 2.1, these models share the same parameter space. This means that we can use the exact same parameter setting for the models. The parameters we use are $d = 100$, $win = 5, dict\_size = 100,000$, the default settings for the Word2Vec models in the gensim package. These values have shown to be a good baseline setting for different semantic tasks [20, 43].

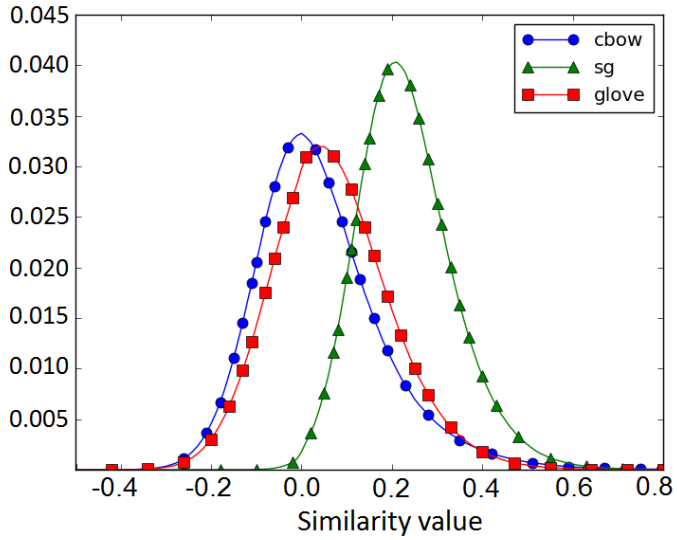

Figure 3.1.: Learning algorithms similarity value distributions

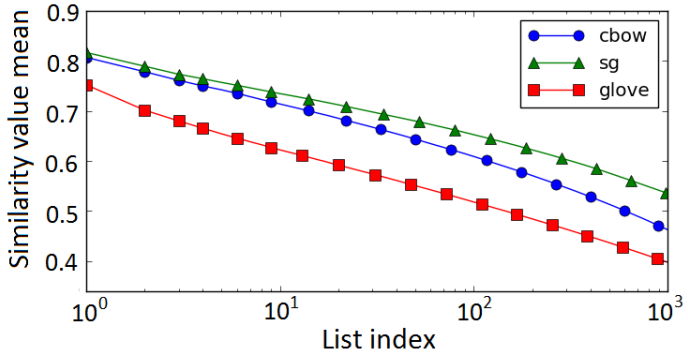

Figure 3.2.: Learning algorithms similarity values by list indices

Similarity Values . Figure 3.1 shows that the approaches visually differ much in their similarity values. The CBOW and GloVe models are almost identical, although GloVe has slightly higher values. But the SG algorithm generally yields higher values than the other

two, and only few pairs of words have negative similarities. This implies that, while words in the CBOW and GloVe model fill almost the entire space, the SG model learns word vectors positioned at a high-density area of the space, leaving the remainder of the space sparse. We test Hypothesis 1. by comparing the normalized distributions pairwise:

$$K\_S\_p\_value\left(sim\_dist_i, sim\_dist_j\right) > 0.01 \text{ for every } i, j \in \{cbow, sg, glove\}.$$

We conclude that the models are similar in their distributions.

Regarding Figure 3.2, although the GloVe model generally produces higher similarity values than CBOW, the values by list position are smaller than with both Word2Vec models. At the end of the top 1000 list, the values of the SG model are the highest ones.

Result Interpretation. Both results indicate that our hypothesis hold. That is, the distributions of the similarity values are indeed very similar, although at the same time they are visibly different in certain characteristics. This is important: It indicates a certain robustness of embedding models and generalizability of empirical results.

3.1.4. Dimensionality

When measuring similarity with the cosine distance, the dimensionality of the embedding model is a parameter that strongly affects its similarity values. In this section, we train every model with the Word2Vec CBOW algorithm with different dimensionalities on the full corpus, with $win = 5$, $dict\_size = 100,000$.

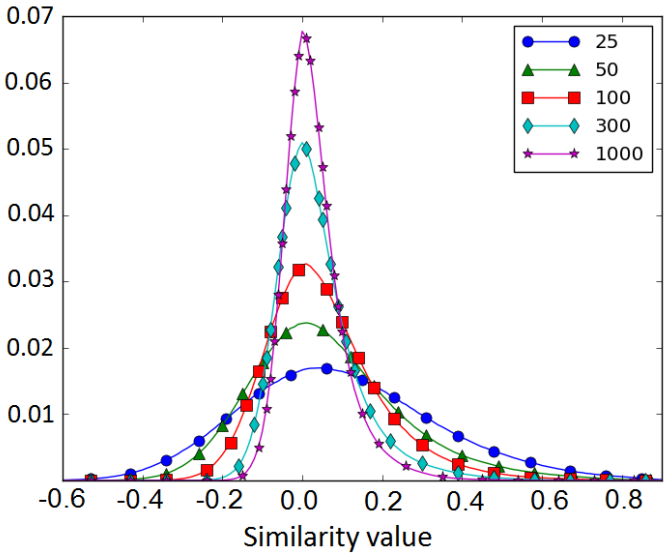

Figure 3.3.: Dimension size similarity value distributions

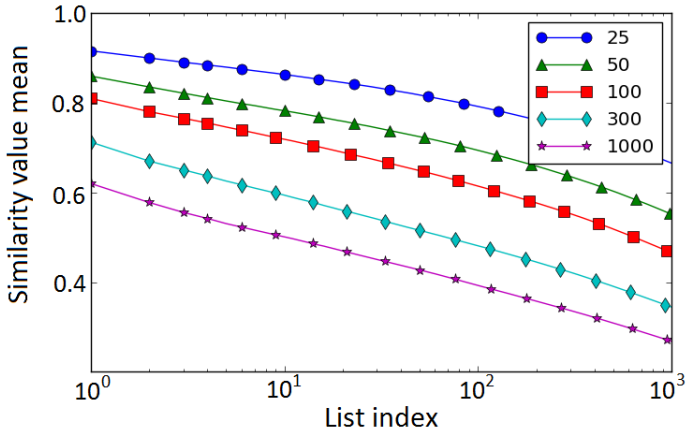

Figure 3.4.: Dimension size similarity values by list indices

Similarity Values. Figure 3.3 shows that the higher the dimensionalities of the model are, the narrower are the similarity distributions. We have expected this, as vector spaces with lower dimensionality are denser when filled with 100,000 words than those with higher dimensionality. This leads to closer words and higher similarity values. In contrast to the visibly different distributions, we again see that the distributions are similar, as the K-S test does not distinguish the normalized distributions, with 99% confidence.

Figure 3.4 is even more straightforward - the higher the dimensionality, the lower the similarity values in the similarity lists are.

Result Interpretation. The dimensionality parameter confirms our hypothesis in a manner that we deem clearer than the previous experiments. Namely, the models are fundamentally very similar and at the same time different. The average and highest similarity values are very different, but the distributions only differ in their standard deviations. This means that they are fundamentally very similar.

3.1.5. Dictionary Size

In this section, we evaluate how the dictionary size of the models affects their similarity values and lists. We train five models with different dictionary sizes with the Word2Vec CBOW algorithm on the full corpus, with $d = 100$, $win = 5$.

Similarity Values. Figure 3.5 shows that the dictionary size does not affect the similarity value distribution of the models up to a certain size. With very large dictionaries however, the numerous noise words (typos, unmeaningful words, contraction, etc.) have a very strong effect on the distribution. The same effect is visible in the dimensionality experiment, i.e., when considering many words in the dictionary, the 100 dimensional space is not

large enough for the models to distribute them sufficiently. This leads to wider similarity value distributions and even to an asymmetric distribution with the largest dictionary.

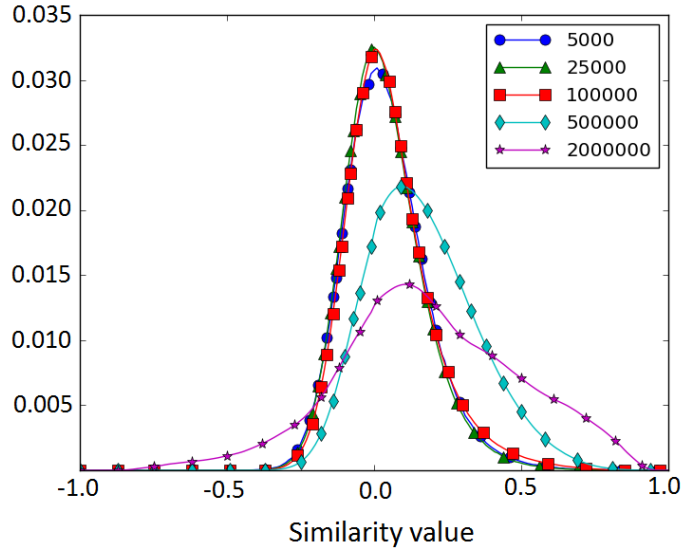

Figure 3.5.: Dictionary size similarity value distributions

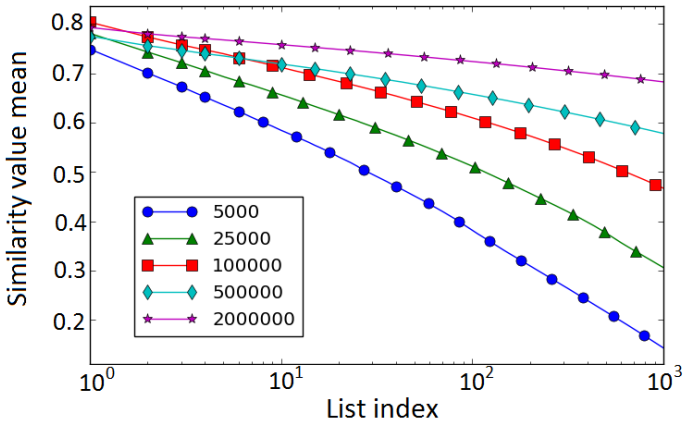

Figure 3.6.: Dictionary size similarity values by list indices

The K-S test confirms the similarity distribution of the 2 million word dictionary model to significantly differ from the others, as

$$K\_S\_p\_value\,(sim\_dist_{2M}, sim\_dist_i) < 0.01 \text{ for every } i \in \{5k, 25k, 100k, 500k\}.$$

Let us now look at the similarities of items with the same position in the different similarity lists in Figure 3.6. The smaller dictionary models naturally have consistently lower similarity values. This is because there are fewer words which are close to each other.

Result Interpretation. This is the only evaluation where one distribution does not have the bell shape observable in all other experiments. This is a consequence of an unreasonably large dictionary. The models are not able to successfully embed the words in the limited space. Apart from this, even in a 500 thousand word dictionary the hypothesis hold, as the distributions are similar.

3.1.6. Corpus

Now we investigate how the size of the corpus affects similarity values and lists. We compare five different models which are trained on differently sized parts of the 1 billion word benchmark dataset. Sampling is performed by retaining different percentages of the 1 billion words data used for the training. Every other parameter of the models is identical. We train them with the Word2Vec CBOW model, with $d = 100$, $win = 5, dict\_size = 100,000$.

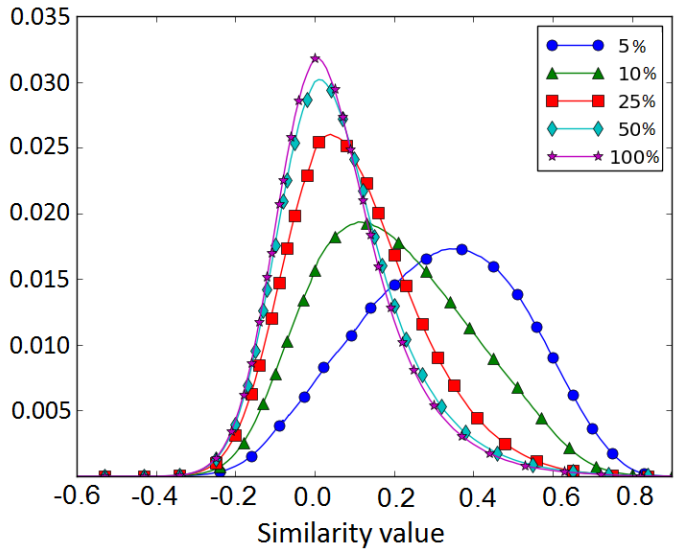

Figure 3.7.: Corpus size similarity value distributions

Similarity Values. According to Figure 3.7, the bigger the corpus, the narrower the distribution is. We can see that using 25% of the corpus is almost identical to using 50% , and very close to using the entire corpus for training. We test the normalized similarity distributions pairwise with the K-S test. Every p-value again is above 0.01. This means that the models are very similar.

Figure 3.8 shows that at the top 10 similar words there almost is no difference between the models. For higher indices, models trained on smaller corpora generally have higher similarity values, but the three models trained on bigger corpora are almost identical.

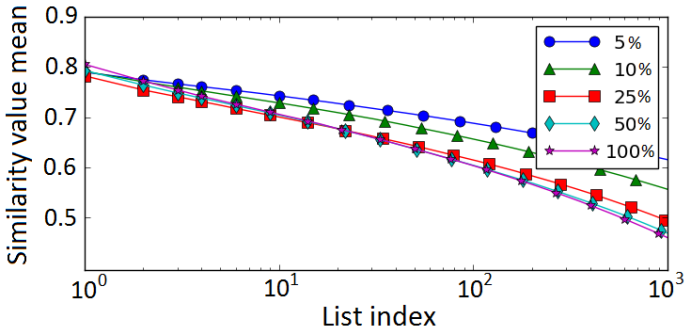

Figure 3.8.: Corpus size similarity values by list indices

Result Interpretation. We conclude that models trained on more than 1 GB (which is 25% of the full corpus in this case) of text data or approximately 250 million words have almost identical similarity value distributions. All distributions are similar, but visibly different at the same time, for smaller corpus sizes in particular. This confirms our hypothesis.

3.1.7. Window Size

In this section, we train every model with the Word2Vec CBOW algorithm on the full 1 billion word corpus, with $d = 100$, $dict\_size = 100,000$ and five different window sizes.

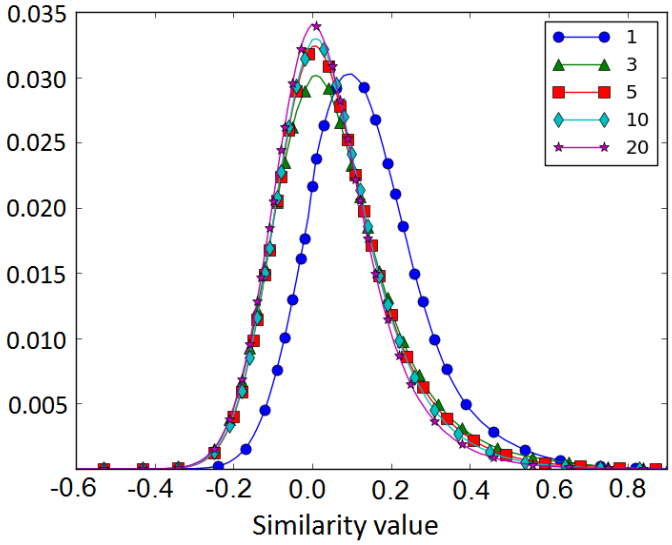

Figure 3.9.: Window size similarity value distributions

Similarity Values. Figure 3.9 shows that there is only a slight difference of similarity values between models trained with different window sizes. It is noteworthy that, when the window size is 1, the distribution has a higher mean. This implies that the model has an

area of higher density in the word vector space. The distributions are very similar without even normalizing them. The pairwise K-S test confirms this, as again every p-value is above 0.01. Consequently, the normalized distributions are almost identical.

The similarities corresponding to different positions in the similarity lists in Figure 3.10 tell us that the differences between the models are very small. Still we can see that the smaller the window, the higher the similarity values are.

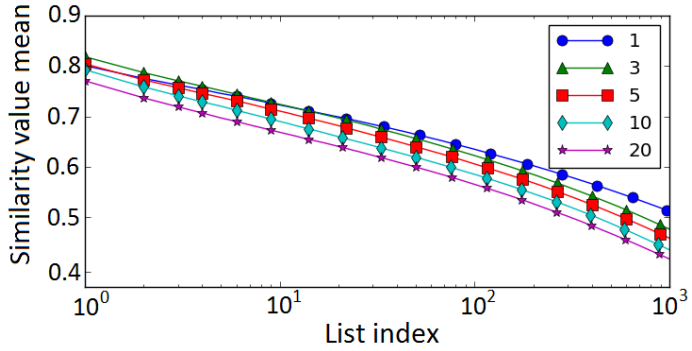

Figure 3.10.: Window size similarity values by list indices

Result Interpretation. These results are very similar to the ones for dimensionality, with both figures consistently changing with the parameters, only at a smaller scale in this current case. Only the smallest window size parameter, i.e., $win = 1$, interferes with the similarity distribution in an inconsistent manner, but it also changes the mean of the distribution.

3.1.8. Optimization Function

When training any model with neural networks, one has to choose an optimization function which approximates the gradient. In our case, for word embedding models, more precisely for Word2Vec models, as they are the ones using neural networks, there are two optimization functions used during training. First, there is Negative Sampling (*ns*), which we have used in this algorithmic analysis section so far, and, second, Hierarchical Softmax (*hs*) [84, 85]. We have explained the details on the differences between the functions in Section 2.1.2.2. So far we have used Negative Sampling, because it is the one which is closely related to matrix factorization, as we have mentioned earlier, and therefore to the GloVe model [67]. Next, even though the margin is small, it constantly outperforms the Hierarchical Softmax function on word similarity tasks [8]. However, in this section we are not concerned with the quality of the models, but their similarity values. Hence we train models differing only in their optimization functions and evaluate the similarity value distributions of the resulting models. We train every model with the Word2Vec CBOW algorithm on the full 1 billion word corpus, with $d = 100, win = 5, dict\_size = 100,000$.

Similarity Values. Figures 3.11 and 3.12 show that there are visible differences in the similarity value distributions between models trained with different optimization functions. Although the K-S test confirms that the distributions are very similar when normalized, it is noteworthy that the resulting p-value is only slightly higher than 0.01. This indicates a certain difference between the two distributions. Figure 3.12 shows that with Negative Sampling the model generally has higher similarity values at the top of the similarity lists.

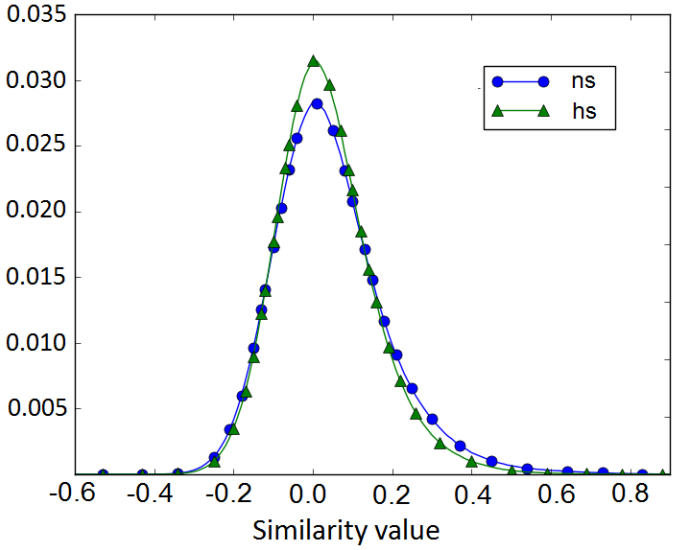

Figure 3.11.: Optimization function similarity value distributions

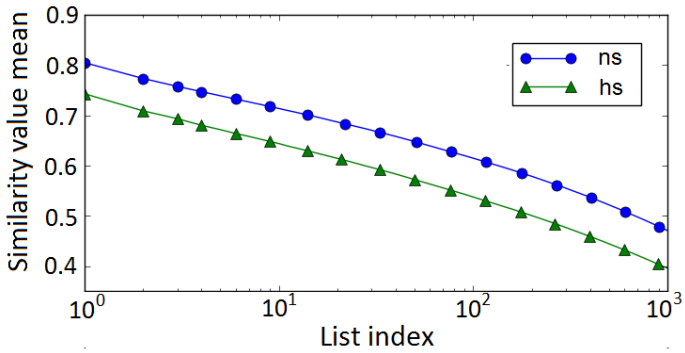

Figure 3.12.: Optimization function similarity values by list indices

Result Interpretation. The results show that changing the optimization function affects the similarity value distributions of the models. Although the distributions are quite similar, we can deduce a more significant structural difference from the low K-S test score than the visible differences would suggest. The similarity of the two distributions explains the small difference in the evaluation test set scores of models trained with different optimization functions, which we have referred to above.

3.1.9. Iteration Number

The training of a word embedding model has several iterations. In one iteration, the learning algorithm passes through the entire training corpus. For every word in the corpus, the algorithm updates the respective word vectors. So far in this algorithmic analysis chapter, we have trained the word embedding models with five iterations, which is the default value in our model building toolkit. In this section, we are interested in whether and how the iteration number affects the similarity value distributions of the word embedding models. We train every model with the Word2Vec CBOW algorithm on the full 1 billion word corpus, with $d = 100$, $win = 5, dict\_size = 100,000$, with five different iteration numbers.

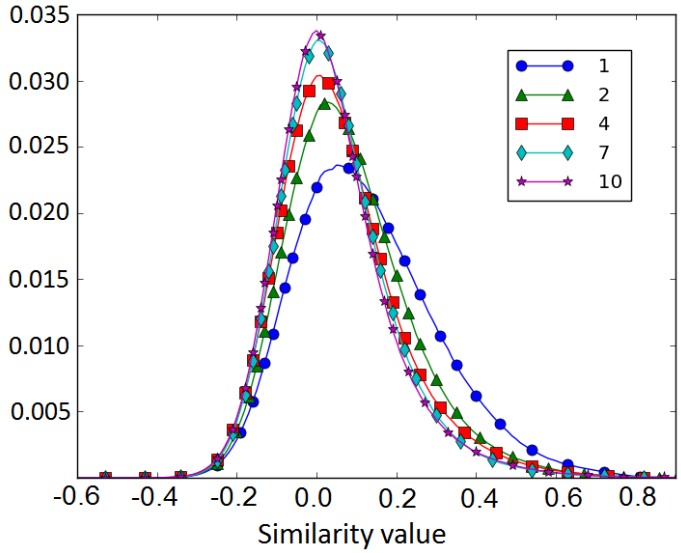

Figure 3.13.: Iteration number similarity value distributions

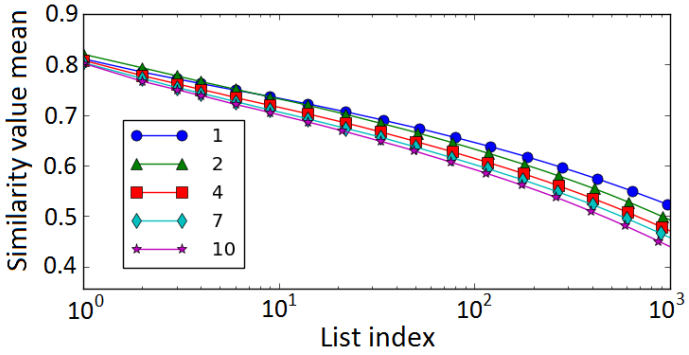

Figure 3.14.: Iteration number similarity values by list indices

Similarity Values. We can see in Figures 3.13 and 3.14 that the more we iterate through the corpus, the narrower the similarity value distributions and the lower the similarity

values at the top of the similarity lists become. However, the differences are quite small. For example, between seven and ten iterations there is almost no visible difference. The distributions are very similar without even normalizing them, and the K-S test confirms that the normalized distributions are almost identical.

Result Interpretation. These results are almost identical to the ones for the corpus size, with both figures consistently changing with the parameters. We conclude that the similarity value distributions change only slightly with more than five iterations. This confirms our original choice of the iteration number.

3.1.10. Generalization with Additional Embedding Models

Now we turn to the alternative models. We evaluate the similarity distributions produced by the fastText and Doc2Vec models. To do so in this section, we train the word embeddings as well as the fastText embeddings (*charvec*) on the Wikipedia dump. The Wikipedia articles are shuffled and trimmed to contain approximately 1 billion words. We then aggregate these charvec vectors to word vectors (*wordvec*). For Doc2Vec we learn two representations: First, for each *sentence* in the Wikipedia corpus, and second, for each Wikipedia *article*. We group the models by their learning algorithms, i.e., CBOW-like models and SG-like models. All models are trained with $d = 100$, $win = 5$. The distributions are calculated using the full dictionary of the models.

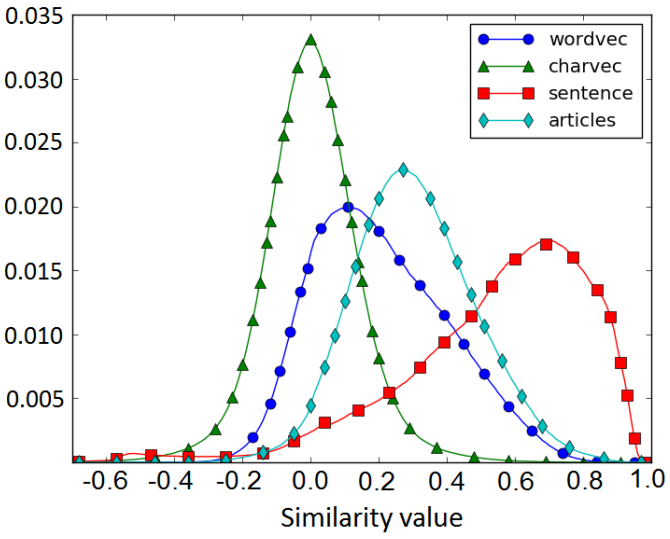

Figure 3.15.: CBOW models similarity value distributions

CBOW Similarity Values. Figure 3.15 shows the similarity value distributions of the CBOW models. We can see that their visual appearances are very different. It is interesting that

this is not the case in the next section for the SG models. Hence, we argue that the CBOW-like algorithms cause the distortion in the additional models, not the models themselves. Compared to the Word2Vec CBOW model, only the charvec model distribution is similar according to the K-S test, with 99% confidence, when normalized. In contrast, the other three models are very different - all three of their K-S test p-values are smaller than 10^{-4}. It is also noteworthy that the average similarity values are very different for the models. They are 0.0 for the charvec model, 0.19 for the wordvec model, 0.37 for the article and 0.53 for the sentence model. As explained before, the higher this average value is, the more concentrated the vectors are in one part of the space.

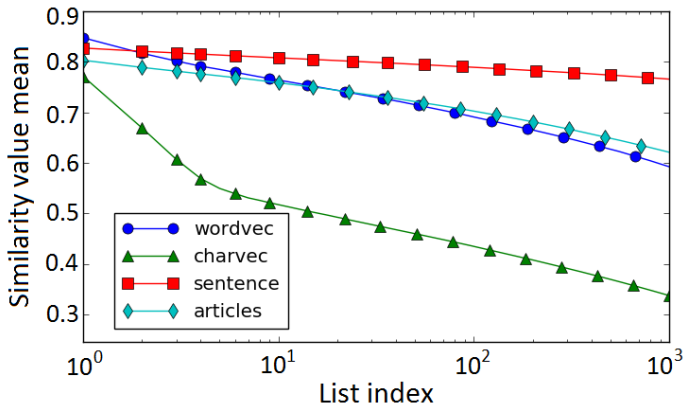

Figure 3.16.: CBOW models similarity values by list indices

At the top of the similarity lists, as seen in Figure 3.16, every model is very similar, except for the charvec model. The charvec models top similarity values are decreasing right away from the start. This is a pattern we have not seen in our evaluations yet.

Result Interpretation. The results show that different models have different similarity value distributions. Only the charvec model is visually similar to the original Word2Vec CBOW model. Even after normalization, all other models are significantly different, even though with a different margin.

SG Similarity Values. Figures 3.17 and 3.18 show the results for the SG models. We can see that, again, the distributions are visibly different, and, except for the charvec model, they also are not like their respective CBOW model distributions. According to the K-S test, the charvec and wordvec models are almost identical to the original SG model when normalized. Both the article and sentence models also are similar to the original SG model to some extent. Hence, their respective K-S test p-values are only slightly lower than 0.01. This means that, although we have found significant evidence that the distributions are different, there are certain similarities between these distributions and the original SG distribution. It is interesting to note that these model distributions are not distorted as they previously were in the CBOW versions. We can also see similarity value averages for

these models different from the ones for the respective CBOW models except, again, for the charvec model. The similarity values at the top of the similarity lists are very similar to the respective values of the CBOW models.

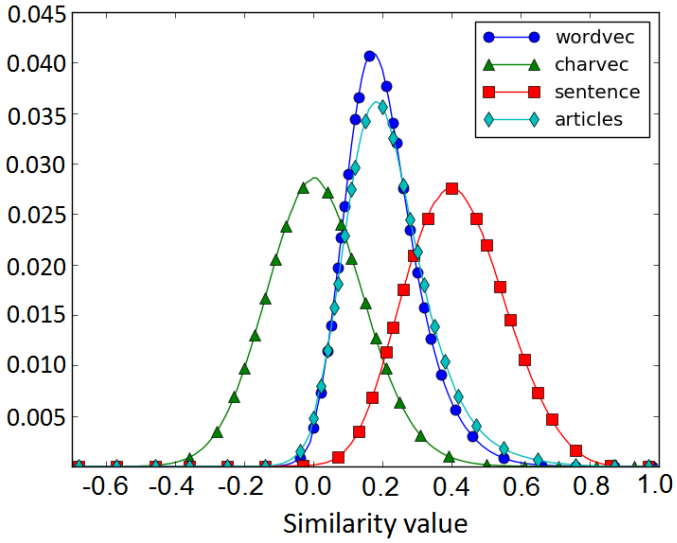

Figure 3.17.: SG models similarity value distributions

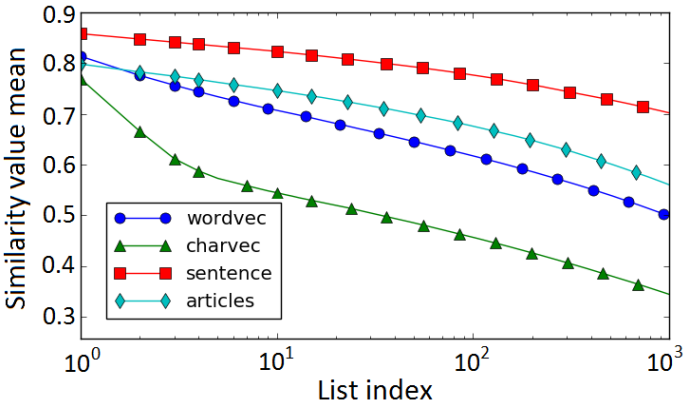

Figure 3.18.: SG models similarity values by list indices

Result Interpretation. The results show that the similarity value distributions of the models highly depend on the learning algorithm used, i.e., CBOW or SG. Only the charvec model is similar to the respective CBOW model. However, in contrast to the CBOW models, SG models are very similar to the original Word2Vec SG model. We conclude that the hypothesis also hold for the SG models, as their similarity value distributions are visibly different, but at the same time very similar when normalized.

3.1.11. Summarizing Parameter Effects

Our evaluations in this section have confirmed our hypothesis. We have shown that different algorithms and parameter settings indeed affect the value distributions of embedding models significantly, but at the same time they have the same abstract shape. All value distributions of the models are bell-shaped, except for one unrealistic setup of the original model and for several models not based on words.

To our knowledge, such systematic experiments have not been done for embedding models before. For systematic evaluations of the effect of parameters on the quality of word embedding models see Hill et al. [42], Altszyler et al. [2], Chiu et al. [20] and Lin et al. [71]. These studies evaluate how the corpus size, window size and dimensionality affect the results of the models on similarity and analogy tasks. We will show in Section 4.2 that all these evaluations suffer from one thread of validity: They do not take the size of the similarity values into consideration when comparing the similarity of two word pairs.

3.2. An Investigation of the Influence of Fragmented Corpora

As the second part of the algorithmic analysis in this section, we investigate how the training of word embedding models on fragmented corpora affects the quality of the models. First, we present our investigation objectives. These are intuitive research questions regarding the fragmentation and minimum count parameters of the training corpora. Then, we present our evaluation setup, where we describe experiments to measure how much fragmentation and minimum count settings reduce the quality of the corresponding word embedding models. Finally, we conduct these experiments on word embedding models trained on systematically different training corpora, and evaluate the results.

3.2.1. Investigation Objectives

In order to make our evaluation results more intuitive we present three research questions. Our objective in this section is to answer these questions by evaluating word embedding models trained on differently fragmented corpora.

Question 1. What is the smallest number n for which an n-gram corpus is good for the training of embedding models?

Rationale behind Question 1. The size of any n-gram corpus highly increases with large n. Hence, it is important to know the smallest value that is expected to still yield good results.

Question 2. How does the minimum count parameter affect the quality of the models? How does this result compare to the effect caused by the fragmentation?

Rationale behind Question 2. Having answered the first question, we will be able to quantify the effect of the fragmentation. However, it is necessary to study the effect of the second parameter as well, in order to quantify the applicability of n-grams for embedding comprehensively. In other words, we want to compare the effects of both parameters; we will be able to give recommendations for both parameters.

Question 3. How does the quality loss of models trained on fragmented corpora of size n or with high minimum count parameter manifest itself in the embedding models?

Rationale behind Question 3. By answering Questions 1 and 2, we are able to quantify the effect of both parameters. We hypothesize that the parameters affect the quality of the models differently, and that we are able to observe this in the word vectors themselves.

The rationale behind our hypothesis is that large minimum count values might eliminate various meaningful words from the vector space. Fragmentation in isolation however does not have the effect that a word is lost. Hence, a quality loss must manifest itself differently, which might be observable.

3.2.2. Evaluation Setup

In this section we present the setup of our experiments. First, we introduce the training corpora and evaluation test sets used to train and evaluate word embedding models. Then, we validate the goodness of our fragmentation method.

Full-Text Corpus Selection. We work with two full-text corpora: the 1 Billion word dataset and the Wikipedia dump sampled to contain approximately 1 billion words (see Section 2.4.1). For both corpora we create their respective n-gram versions in the Google Books format, with n=2,3,5,8, cf. Section 2.3. The fact that we use two of the largest available raw-text corpora reduces the possibility that different qualities of embedding models are due to the underlying corpus and not the fragmentation itself.

Baseline Test Sets. In this chapter we work with the similarity and analogical reasoning baseline test sets introduced in Section 2.4.2. We evaluate every model on each one of the datasets.

Validation of the Fragmentation Method. Intuitively we assume that the more often the training algorithm uses a word to establish its context in one iteration, the better the model becomes. Namely, the algorithm finds the final position of every word faster, i.e., convergence is faster. It follows from the definition of the fragmentation method that an n-gram corpus is n-times the size of the raw text it is created from. In other words, every word of the raw text appears n times as often in the fragmented one. When training the embedding models, every word is processed n times more often in every iteration when the fragmented corpus is the training data. The question is whether this characteristic of the corpus makes a comparison between models trained on them biased or not.

To answer this question, we train several models on the full-text corpora, with different numbers of iterations, and evaluate if they become better when the training algorithm uses more iterations. If this was the case, it would mean that fragmented corpora may have an advantage over the full-text. Figure 3.19 shows that this is not the case: With more than 5 iterations the model does not get better. This means that, with the same iteration number for any training, and with both the full-text and the fragmented versions as training data, the quality of the models trained on fragmented corpora is not higher. Therefore, in this chapter we train every model in 5 iterations.

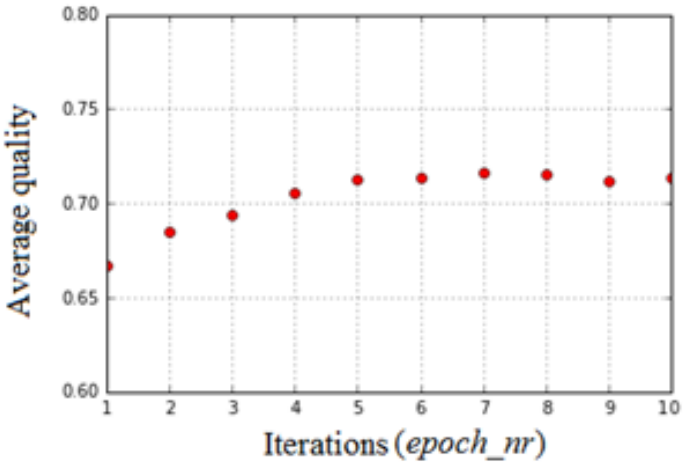

Figure 3.19.: The average score on all the evaluation test sets of the models trained with different numbers of iteration

3.2.3. Evaluation Results

In this section we evaluate and answer the questions we presented previously in Section 3.2.1. We dedicate a subsection to each question. In the first two sections, we give an overview of the results using the Wikipedia corpus. The results for the 1-Billion word corpus are almost identical. For brevity, we do not show all results for this corpus in this section, but it can be found in the supplementary material.

3.2.3.1. Answering Question 1: Minimal Meaningful N-gram Size

Question 1. What is the smallest number n for which an n-gram corpus is good for the training of word embedding models?

In order to answer Question 1. we quantify the influence of the training corpus fragmentation on the quality of the word embedding models. We use the full n-grammed versions of the corpora to train the models, i.e., we do not use a minimum count parameter in this section.

Results for the Wikipedia Corpus. Figure 3.20 shows the result for the models trained on the Wikipedia corpus.[1] The interpretation of the plots is as follows: We evaluate a specific model on every test set introduced in Section 2.4.2. We calculate the average scores for this model for both the similarity and analogy test sets. We do this for every trained model. We group the results by the window-size parameter of the models and plot the average values. So every plot shows the calculated average scores of such models which only differ

[1] Note that we use different scales for the first and the second two subplots.

in the fragmentation of their training corpus. As explained in Section 2.3, it is meaningless to train models on n-grammed corpora with window-size parameter $win > n$.

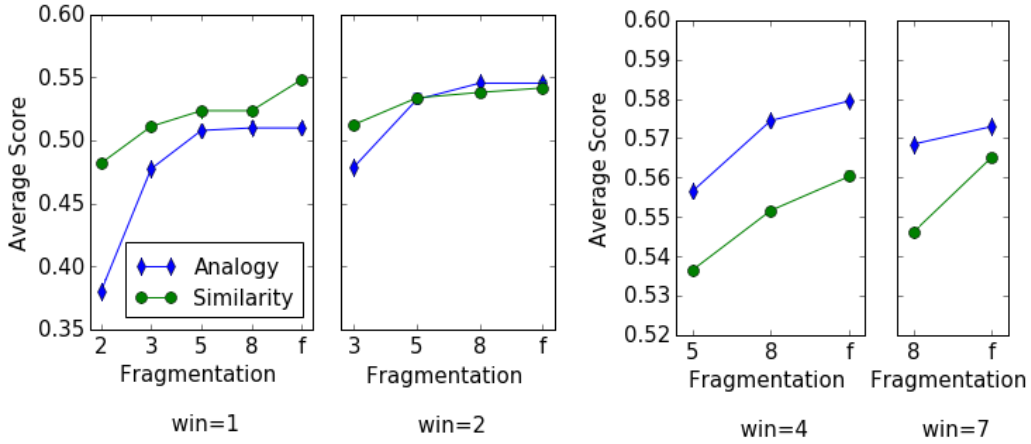

Figure 3.20.: The average score of models trained on differently fragmented Wikipedia corpora

The detailed results can be found in Tables 3.1 - 3.4. The numbers are the scores, defined in Section 2.4.2, of the models on different tasks. We use a naming convention for the models. A model `wiki_n_win` is trained on the n-grammed Wikipedia corpus, with *win* being the window size. f stands for the full corpus. If there is a third parameter in the name, it is the minimum count threshold (if there is no third parameter, it means the minimum count is 0, i.e., we use all the n-grams).

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_2_1	0.311	0.449	0.640	0.485	0.224	0.289	**0.645**	0.608
wiki_3_1	0.422	0.533	0.657	0.557	0.305	0.318	0.595	0.635
wiki_5_1	0.483	0.533	0.652	0.566	**0.341**	0.293	0.639	0.651
wiki_8_1	**0.485**	0.510	0.648	0.544	0.320	0.297	0.633	0.640
wiki_f_1	0.475	**0.545**	**0.676**	**0.612**	0.340	**0.350**	0.638	**0.675**

Table 3.1.: Models trained on differently fragmented Wikipedia corpora with $win = 1$

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_3_2	0.413	0.545	0.668	0.554	0.316	0.308	0.605	0.625
wiki_5_2	0.518	0.547	0.668	0.585	0.351	0.300	0.643	**0.656**
wiki_8_2	**0.540**	**0.551**	0.680	0.583	0.352	0.307	**0.654**	0.653
wiki_f_2	0.509	0.534	**0.694**	**0.613**	**0.355**	**0.316**	0.619	0.653

Table 3.2.: Models trained on differently fragmented Wikipedia corpora with $win = 2$

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_5_4	0.544	0.569	0.663	0.589	0.367	0.304	0.638	0.658
wiki_8_4	**0.579**	**0.570**	0.697	0.609	0.371	0.307	**0.657**	0.669
wiki_f_4	0.554	0.525	**0.731**	**0.620**	**0.377**	**0.315**	0.645	**0.674**

Table 3.3.: Models trained on differently fragmented Wikipedia corpora with $win = 4$

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_8_7	0.573	**0.564**	0.686	0.609	0.347	0.315	0.648	0.672
wiki_f_7	**0.588**	0.518	**0.746**	**0.651**	**0.366**	**0.317**	**0.656**	**0.697**

Table 3.4.: Models trained on differently fragmented Wikipedia corpora with $win = 7$

The figure and the tables reveal that fragmentation does influence the quality of the models significantly. For five out of the six the similarity test sets, (WordSim353, MEN, Simlex999, Rare Words, RG-65), fragmentation reduces the quality of the models for any value of the window-size parameter *win* almost linearly. However, the models trained on 5-gram versions are generally only slightly worse than the ones trained on the 8-gram corpora. For the Mechanical Turk test set, models do not necessarily get worse with fragmentation. The best results for different values of *win* come from the full-text corpora and the 8-gram variants, except for $win = 1$, where the best model is trained on 2-grams.

For the analogy test sets, the results are not as straightforward. Generally, the same observation holds as for the similarity test sets, namely that fragmentation reduces the quality of the models, however there are a few exceptions. As one already knows from earlier comparisons [68], models trained on full-text corpora have better/worse results on the Google/MSR test sets, with an increased window size parameter. This does not hold for the fragmented corpora. For the MSR analogy test set, the models get better when increasing the window size, until $win = 4$, and get worse only slightly with $win = 7$. It is true that the best models are trained on the 5 and 8-gram variants and the full-text corpora for any window size. For models with smaller *win* however, the results do not always get better when the corpus is less fragmented. For example, the very best model for the MSR test set is trained on 8-grams, not the full-text.

Generalization of Results. To generalize the results, we measure the overall average quality of the models trained on the differently fragmented corpora. Then we compare the results to ones computed on the full-text. To this end, we calculate the averages of the previous results, grouped by training corpus fragmentation. With this, we can see how much worse the fragmentation itself makes the models in general, not just for different window sizes separately. Table 3.5 shows the results. The total column is the average of the similarity and analogy columns. To make the resulting numbers more intuitive, we do this in relative terms, compared to the results with the full-text corpus. For example, on

analogical reasoning tasks, the models trained on 3-grams are 7.5% worse than the models trained on full-text with the same window size.

Formally, let $WS\_k\_(i,j), k \in \{1,\ldots,6\}$ and $AR\_l\_(i,j), l \in \{1,2\}$ be the scores of the model on the k^{th} word similarity and l^{th} analogical reasoning task trained on the i-gram Wikipedia corpus with window size j. f refers to the full-text corpus. For a specific task, WS_2 for example, we calculate the average decrease in quality caused by the fragmentation as follows. For every i we calculate $Diff\_WS\_2(i)$:

$$Diff\_WS\_2(i) = 1 - Average_j\left(\frac{WS\_2_{i,j}}{WS\_2_{f,j}}\right).$$

These numbers show how much worse the fragmented corpora are on the specific task on average. To reach our final results, shown in Table 3.5, we average these numbers for every i by the nature of the task (similarity, analogy) and for every task overall.

It is interesting that, on the analogy tasks, the models trained on 5 or 8-grams perform only slightly worse than the ones trained on full-text corpora. This can be a result of the greater number of word-context pairs these models consider during evaluation, as explained in the examples in Section 2.3.

Wikipedia	total	similarity	analogy
2-gram	-20.2%	-14.4%	-26.0%
3-gram	-6.9%	-6.4%	-7.5%
5-gram	-2.8%	-3.6%	-1.9%
8-gram	-2.5%	-3.4%	-1.6%

Table 3.5.: Average quality loss due to fragmentation compared to the full-text on the Wikipedia corpus

Result Interpretation. We conclude that word embedding models built on fragmented corpora are worse than models based on full-text, but the difference is not much. Word embedding models trained on 2-gram corpora are 20.2% worse overall than models trained on full-text, a significant drop. However, the 3-gram version is only 6.9% worse. The 5-gram version and the 8-gram version are almost tha same, they are only 2.8% and 2.5% worse, respectively. This answers Question 1., i.e., a 5-gram corpus is the most fragmented corpus which is almost as good as models trained on full-text, and the 3-gram version is also not much worse. This insight is of practical relevance as in the English version of the Google n-gram data, the 3-gram dataset is the largest one. This is due to the minimum count parameter of 40 used by Google, making the 4-gram and the 5-gram dataset sparse. This motivates studying the minimum count parameter further. A general takeaway for other researchers when training word embedding models on n-grams is that using at least 3-grams for training leads to good models and using at least 5-grams leads to almost identical models as training on full-text.

Insight Confirmation Using a Different Text Corpus. So far, our results only rely on one corpus. Even as Wikipedia is one of the largest and most frequently used corpora, it might be biased. To generalize, we verify our insights using another large corpus. Our procedure is the same as before. We only list aggregated results here, the full list of tables can be found in the supplementary material. We have calculated the numbers for all models trained on the 1-Billion word dataset and its fragmented versions. Table 3.6 shows that the numbers are generally very similar to the previous ones with Wikipedia. So the decline in quality does not depend on the underlying text corpus, but on the fragmentation.

1-Billion	total	similarity	analogy
2-gram	-19.3%	-15.0%	-23.6%
3-gram	-5.4%	-6.5%	-4.3%
5-gram	-1.8%	-2.9%	-0.8%
8-gram	-1.3%	-2.2%	-0.4%

Table 3.6.: Average quality loss due to fragmentation compared to the full-text on the 1-Billion word corpus

3.2.3.2. Answering Question 2: Effect of the Minimum Count Parameter

Question 2. How does the minimum count parameter affect the quality of the models? How does this result compare to the effect caused by the fragmentation?

In the following, we aim at quantifying the influence of the minimum count parameter and investigate whether there is an interaction with the fragmentation. Our procedure is the same as in the prior section. First, we aggregate the raw results obtained from the Wikipedia corpus and *all* models built on it. Then we draw first conclusions and finally verify the insights using the second corpus.

Results for the Wikipedia Corpus. Figure 3.21 shows the average quality of models trained on the same n-gram corpus with different minimum count parameter. The results indicate that an increase of this parameter usually leads to significantly worse models. However, there is one exception, where the minimum count parameter is 2 and the corpus is the 2-grammed version of Wikipedia. For this case the models actually gets slightly better on the analogy task using the minimum count threshold. The reason is that we do not lose too many 2-grams with the thresholding in this case, and those which we do lose may bias the model on the analogy tasks. However, on the similarity tasks the models get slightly worse, which means we lose meaningful training data as well. The reduction in quality is even more severe with n-gram corpora with a big value of n, such as 5 or 8-grams. We can see this in exemplary Tables 3.7 and 3.8. This is because these corpora have fewer high match count n-grams than the more fragmented 2 or 3-gram corpora.

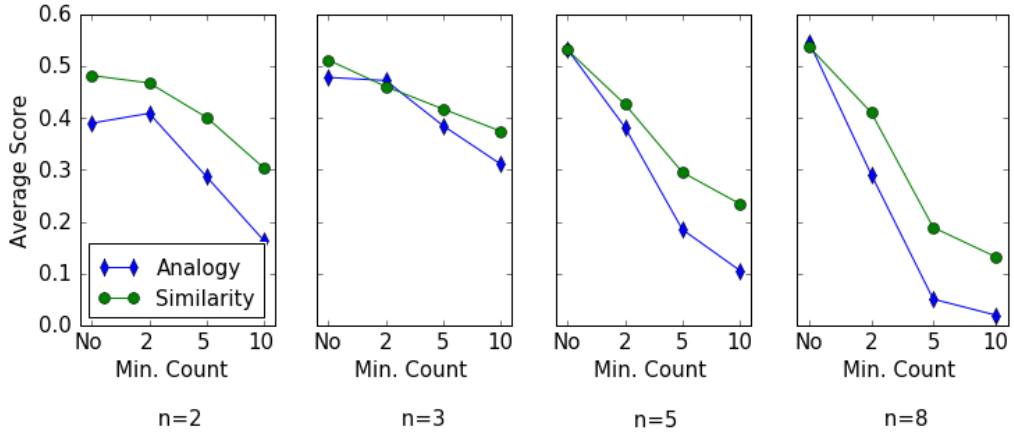

Figure 3.21.: The average scores of models trained with different minimum count parameter on differently fragmented Wikipedia corpora

We see that for the n-gram corpora with a large n, even the smallest minimum count threshold 2 brings down the quality of the models by much, which is not the case for smaller n values (especially for the analogy task). For higher minimum count parameter values, models have significantly lower quality. Detailed evaluation results can be found in the supplementary material.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_5_2_0	**0.518**	**0.547**	**0.668**	**0.585**	**0.351**	**0.300**	**0.643**	**0.656**
wiki_5_2_2	0.358	0.426	0.581	0.537	0.154	0.237	0.544	0.558
wiki_5_2_5	0.155	0.236	0.331	0.398	0.053	0.148	0.476	0.382
wiki_5_2_10	0.087	0.138	0.337	0.284	0.045	0.087	0.427	0.290

Table 3.7.: Models trained on the 5-gram Wikipedia corpora, $win = 2$, with different minimum count parameter

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_2_1_0	**0.311**	**0.449**	**0.640**	0.485	**0.224**	**0.289**	**0.645**	**0.608**
wiki_2_1_2	0.307	0.431	0.624	**0.490**	0.196	0.274	0.625	0.596
wiki_2_1_5	0.203	0.370	0.489	0.415	0.128	0.266	0.571	0.537
wiki_2_1_10	0.113	0.213	0.404	0.254	0.090	0.235	0.414	0.422

Table 3.8.: Models trained on the 2-gram Wikipedia corpora, $win = 1$, with different minimum count parameter

Summing up, for corpora of a size such as the Wikipedia dump, any threshold for the minimum count of the n-grams significantly reduces the quality of the word embedding models. One exception is when the corpus is highly fragmented with the smallest minimum count parameter.

Generalization of the Results. We generalize the aforementioned observations by computing the average quality loss for the models, just as in Section 3.2.3.1. The numbers in Table 3.9 are for the Wikipedia dataset. With the smallest minimum count threshold already, the models get significantly worse. On average, even for the smallest parameter value of 2, model quality decreases by 23.6%. For the 1-Billion word dataset, the results again differ only slightly. This again confirms our hypothesis that the quality differences are not corpus-dependent.

Min. count	total	similarity	analogy
2	-23.6%	-19.2%	-28.0%
5	-56.5%	-47.9%	-65.1%
10	-72.3%	-64.0%	-78.6%

Table 3.9.: Average quality loss caused by the minimum count parameter parameter on Wikipedia

Implications for the Google N-gram Corpus. So far, the question how to transfer the results from the Wikipedia and the 1-Billion corpus to the Google n-gram corpus remains open. We aim to answer whether we lose any meaningful information in the Google 5-gram corpus, because of the, at first sight, large threshold value of 40. We assume that even for such comprehensive text corpora, including the Wikipedia or the 1-Billion corpus, all the extracted n-grams are contained in the Google n-gram data as well, despite its large threshold value. We verify this hypothesis by comparing the number of existing 5-grams in the Google n-gram corpus (1.4 Billion) with those in the full Wikipedia (1.25 Billion). A systematic analysis of the data reveals that more than 99% of the 5-grams included in the Wikipedia corpus is included in the Google corpus as well. The ones which are not usually are typos or contain words which have not been present in the language until 2008 (the last year in the Google dataset). This holds for all n-gram corpora. This means that we do not lose any relevant information if we train our models on the Google n-gram dataset, despite its high minimum count threshold value. So it is a suitable training corpus for word embedding models.

Result Interpretation. To conclude, we can now answer Question 2. We see that the minimum count parameter reduces model quality. This conclusion depends on the size of the corpus . For smaller corpora, the effect will be even more pronounced. For such sizes of the training data we do not recommend to use any minimum count threshold when training word embedding models. In combination with the results from Section 3.2.3.1, we conclude that the Google Books dataset is valid training data for word embedding models. In general, one can expect good results using the 5-grams as training data, but anything above 2-grams could be used.

3.2.3.3. Answering Question 3: The Reason for the Quality Loss

Question 3. How does the quality loss of models trained on fragmented corpora of size n or with high minimum count parameter manifest itself in the embedding models?

Regarding Question 3, we start with an explanation why the increase of the minimum count parameter decreases the quality of the word embedding models. We have observed that in almost every case this has been a consequence of certain infrequent words of the evaluation test sets not occurring in sufficiently many n-grams. When analyzing the task specific results, we have seen that result quality on the rare words test set drops instantly even with the smallest threshold value. For other test sets on the other hand (WordSim353, RG-65 for instance) which include highly frequent words almost exclusively, results are not much worse. Low-frequency words, such as *clergyman* or *incommensurable*, are not even included in the dictionary of models with any minimum count parameter. Other more frequent words, such as *submariner* or *uncompetitive*, are included in some models, but not in those trained using 5 or 10 as minimum count. In summary, the reduced model quality generally is a consequence of the less frequent words not being trained sufficiently or even not at all. Therefore, they do not appear in the dictionaries of the model.

Corpus	2-gram	3-gram	5-gram	8-gram	Full-text
Avg. movement	0.018	0.016	0.011	0.010	0.006

Table 3.10.: Average movement in cosine distance of the word vectors with one extra iteration

The fragmentation of the corpora causes a quality loss for a different reason. Every word of the evaluation test sets is included in the dictionary of every model, but fragmentation causes a mix-up of the word vectors, cf. Section 2.3. As explained, each word is trained several times when fragmented corpora are used, and most of the time the context of the word, as considered by the algorithm during training, is not the full context. One can perceive this as changing every word vector several times in every iteration, but never to the exactly right direction (as the full context would do), but to suboptimal directions corresponding to the restricted contexts. To quantify this effect, we have measured the average movement of a word vector when we iterate through the training data one extra time, after training the models. See Table 3.10; the numbers are average cosine distances. The lower the corpus quality is, the more the vectors move in the additional iteration. As expected, the numbers are small compared to the average word vector distances: The average distance of a word and its closest neighbor is around 0.2 in a 100-dimensional model, see Section 3.1. The results seem to confirm our intuition that, with bad corpora, vectors move in suboptimal directions to a higher extent, ultimately resulting in worse models.

4. Theoretical Analysis of Word Embedding Models

As the second part of our three-way analysis of word embedding models, in this chapter we aim at understanding word embedding models better from a theoretical point of view.

Our first evaluation in this chapter is regarding the most fundamental property of word embedding models: their similarity values. The main question of our theoretic analysis is what the similarity values actually mean in word embedding models? Previous works have used the similarity values, but never answered questions such as: 'Are word-pairs with low values of similarity comparable to each other?' or 'When is it meaningful intuitively taking the top N most similar words for a certain word and deem them similar?' Without fully understanding such question any study relying on comparisons of similarity values might miss important observations and may lack validity.

To answer such questions, we need to evaluate how similarity values behave in different word embedding models, trained with different parameter settings. This is exactly what we did in the previous chapter in Section 3.1. Based on the results, in this chapter we identify meaningful similarity thresholds for both similarity values and similarity lists. These thresholds are not general, but they should be calculated for every individual model. As we have seen previously, all models are fundamentally very similar in their similarity value distributions. Hence, we are able to use the same threshold identifying method we present in the following sections for every individual model.

The fact that it is not always meaningful to compare two word pairs by their similarity values may cause implications in the qualitative evaluation of word embedding models as well. As our second contribution in this chapter, we propose a similarity threshold aware evaluation method of word embedding models. We evaluate the differences between our and the baseline method used in previous literature. We show that excluding incomparable word pairs from the benchmark test sets does change the results to a visible extent.

4.1. Similarity Value Thresholds of Word Embedding Models

The core contribution of this section is to find meaningful thresholds both for similarity values and for similarity lists for a given word embedding model. This means, we intend to be able to make statements such as "In this model above 0.4 similarity every word pair should be considered similar." or "In this model the top 25 most similar word for a given word are always similar to it."

The results of the algorithmic analysis chapter clearly indicates, that it is not possible to find general value and list thresholds that are reasonable for all embedding models, only for individual ones. We present two intuitive examples as well to affirm this statement in the following.

Example 1. Think of two models, Model A with an average similarity between two words of 0.0, and Model B with an average of 0.1. This means that the similarity value is negative for roughly half of the pairs in Model A and for roughly 1% of the pairs in Model B. If one now assumed that a negative similarity value implied dissimilarity between the words of the pair, this assumption would have a highly different meaning for the two models.

Example 2. Again think of two models. The highest similarity score of a word pair is 0.9 in Model A and 0.6 in B. Saying that a pair with a similarity above 0.7 is definitively similar could be meaningful in Model A, but makes less sense in B. This is because there is no word pair with such a similarity value in this model.

These examples show that general similarity thresholds do not exist. Instead, in this section, we propose a method to find meaningful similarity value thresholds for a given model and baseline (e.g., WordNet). Then, we examine the validity of this method using various models.

4.1.1. Investigation Objectives

Reviewing various approaches [30, 16] has revealed that their evaluations compare similarity values and list indices without taking their size into account. This means that they deem, say, two word pairs with similarity values 0.8 and 0.7 just as different as ones with values -0.2 and -0.1. But there is no examination of the distribution of the similarity values of word vectors indicating that this is reasonable. In fact, it turns out that a more differentiated perspective is required. From Section 3.1, we already know the distribution characteristics of the similarity values of the word vectors, for example their average and highest similarities. But we do not yet know how vector similarity corresponds to word similarity, such as similarity values in WordNet[33].

WordNet is a large lexical database of English. Words are grouped into synonym groups that are connected using well defined semantic or lexical relations, such as word A is "type of" word B. The whole database contains more than 117,000 synonym groups.

In this section, we examine experimentally whether meaningful thresholds for similarity values exist. To make our investigation intuitive we present explicit research questions in the following. We answer these questions in the later parts of this section.

Question 1. Are low values of similarity comparable to each other?

Question 2. If Words A and B have a higher similarity value than A and C (say 0.2 for A and B, 0.1 for A and C), is A more similar to B than to C?

Question 3. Does being in the top 100 list of most similar words always imply similarity, or does not being in the top 500 list always imply dissimilarity?

Question 4. What are meaningful threshold values, and how to find them?

4.1.2. Evaluation Setup

Our procedure is similar to the one in Section 3.1. The main difference is that we compare the results to a baseline, WordNet in this case. We conduct two series of experiments, one for similarity values and one for lists. In both cases, we calculate word pair similarity aggregates, one grouped by values, the other one grouped by list indices, based on WordNet similarity scores. We do so in order to understand to what extent similarity values are meaningful in embedding models. We use the Leacock and Chodorow (LCH) [64] similarity measure in WordNet for the evaluation. We have chosen this measure because, according to the taxonomy of [82], it is not corpus-based, but knowledge-based. This means that it does not use any external resource or corpus, but only the WordNet ontology itself[82]. It also is a popular, highly researched measure and has proven to be a useful baseline for semantic similarity[18, 82, 17]. The LCH measure scores are on a [0, 3.64] scale, with a score of 3.64 corresponding to identical words. For more information on similarity measures in WordNet see Meng et al. [79].

We have calculated the similarity value distribution of the LCH measure just as we did in Section 3.1 for the embedding models. Figure 4.1 shows the distribution. For the sake of completeness, we compare the normalized LCH distribution to the CBOW and SG model distributions with the K-S test, as we have done in Section 3.1. We find that the LCH distribution can be distinguished from the distributions of the word embedding models with 99% confidence, i.e., they are not very similar.

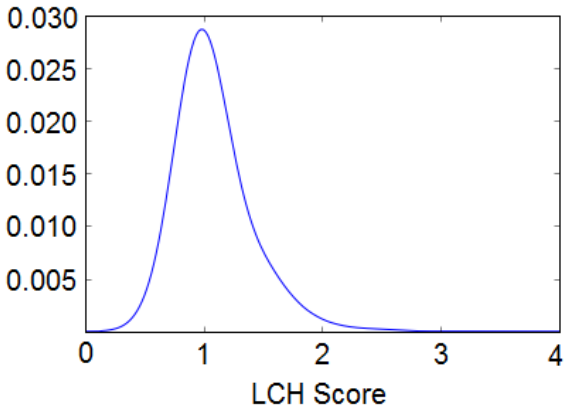

Figure 4.1.: LCH score distribution

We have implemented our experiments with WordNet using the NLTK python toolkit [109]. In all our experiments in this section, the baseline similarity measure (LCH) is replaceable. This means that one simply can rerun any experiment with a more specific, say, corpus-based similarity measure, as well as with another model.

The model we use in this section is trained with the CBOW algorithm on the full 1 billion word corpus, with $d = 100$, $win = 5, dict\_size = 100,000$, the default model and parameter settings in the gensim Word2Vec toolkit.

Similarity Value and List Experiments. For the first experiment, we compute the similarity values of every word to *any* other word in the dictionary: $p_{i,j}$ is a word pair containing words v_i and v_j for $i, j \in \{1, \ldots, 100000\}$, $t_{i,j}$ is their similarity. We now group these word pairs by their similarity value in 0.01 intervals: $G_{-1.0}, G_{-0.99}, \ldots, G_{0.0}, G_{0.01}, \ldots, G_{1.0}$ are these groups. To illustrate, $G_{0.05}$ contains all word pairs $p_{i,j}$ for which $0.04 < t_{i,j} \leq 0.05$ holds. Then we calculate the average similarity with the LCH measure in each group:

$$avg\_sim\,(G_k) = average\left(LCH\_dist\left(p_{i,j}\right)\right), \text{ where } p_{i,j} \in G_k.$$

In the second experiment, we create the full similarity lists for every word in the dictionary, $v_{i,1}, v_{i,2}, \ldots, v_{i,100000}$, i.e., for every $i \in \{1, \ldots, 100000\}$. We create groups of word pairs $(G_1, \ldots, G_{100000})$. G_k contains the pair $\left(v_i, v_{i,k}\right)$ for every $i \in \{1, \ldots, 100000\}$. We then calculate the average similarity for every group with the LCH measure with the same formula as above for the similarity value groups.

For both experiments, if a word is not in the WordNet dictionary, we remove all word pairs including it from the groups, in order to make the aggregation unbiased. We observe that the standard deviations are relatively high in the groups: In the similarity value groups, it is between 0.25 and 0.55, in the similarity-list groups between 0.25 and 0.6. We will return to this observation when discussing the outcomes of the experiments.

Figures 4.2 and 4.3 show the similarity averages on different scales for the similarity value and list groups, respectively. In the similarity value-distribution experiments, we evaluate the results only for values between -0.4 and 0.8. This is because the small number of word pairs with similarity values outside of this interval makes the data in these ranges noisy. This is in line with our parameter evaluation results. Namely, we can see from the graphs in Section 3.1 that the vast majority of word pairs has similarity values in this range for the model used in this section.

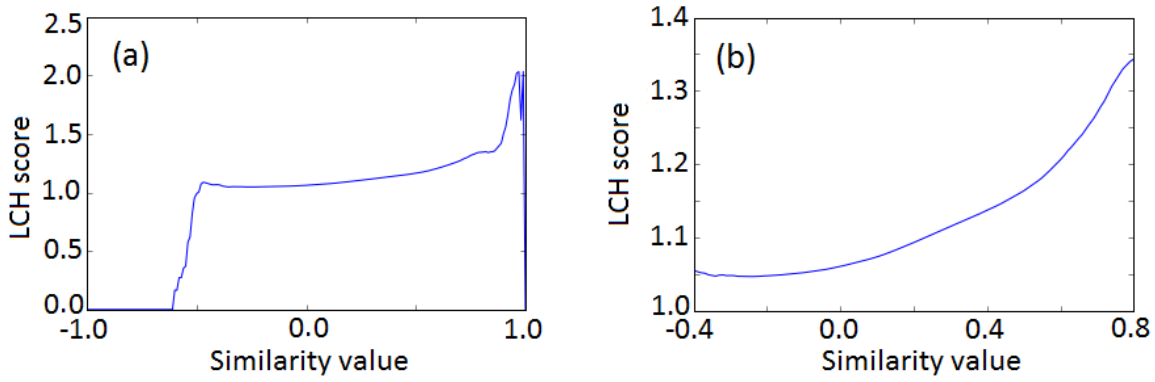

Figure 4.2.: LCH score aggregates by similarity values

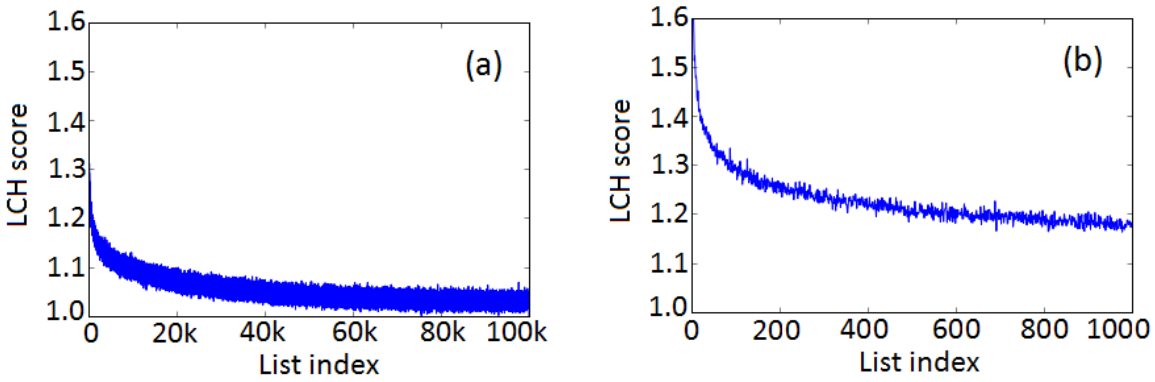

Figure 4.3.: LCH score aggregates by list indices

To find meaningful threshold values, we check the figures of the averages of the similarity value distributions for patterns that could imply meaningful values. We do so in two steps. Our first step is an intuitive naive inspection of the figures; the second step is a statistical analysis of the graphs. We now discuss these steps.

4.1.3. A Naive Approach to Find Similarity Thresholds

In this section, we describe a naive visual approach to inspect the similarity value and list figures. The naive inspection is important, because the statistics-based approaches to find similarity thresholds described later follow the same intuition as described in this section.

When analyzing the results visually, we hope to find horizontal segments in the result graph or other phenomena such as breaks, i.e., flat segments of the graph followed by a steep incline or decline, which might stand for certain properties of the models. A horizontal segment, for example, would mean that there is no difference in similarity between the values forming this line. To illustrate further, imagine that in Figure 4.3b there would be horizontal between list indices 800 and 1000. Then, we could interpret this as follows: There is no general difference in similarity between a word being the 800th or the 1000th most similar word to a given word. Thus, it is meaningless to differentiate between words at these similarities. The same would follow for the similarity value distribution if there was a horizontal segment there. Other phenomena such as a break in the figure would imply a general change in similarity. For example, if there was a break, we could interpret it as a threshold between relevant and irrelevant similarity values at first sight. However, as is observable in Figures 4.2 and 4.3, such a naive approach does not yield any useful result in our case. This is because there are no obvious horizontal segments or breaks in the graphs.

4.1.4. Towards Meaningful Threshold Values

The previous step has not identified any obvious patterns pointing to intuitive threshold values for similarity. Hence, we now strive for a statistically sound derivation of meaningful threshold values, in contrast to a mere visual inspection.

4.1.4.1. Confidence-based Threshold Identification.

The general idea is examining the results of our experiments with statistical tests. We test the hypothesis that two populations have equal means, without assuming that they have equal variance. Here, these populations are the LCH scores of two groups of word pairs. Formally, such a group (LCH_G_k) is as follows: $LCH\_G_k = \{LCH\_dist\left(p_{i,j}\right) : p_{i,j} \in G_k\}$, where G_k is either a similarity value or a similarity list group, as introduced in Section 3.1. We use Welch's unequal variances t-test [11] for our experiments, a widely used two-sample statistical test for this problem. So the answers to the research questions from the introduction are statistical in nature, i.e., we will give answers with a certain confidence, such as 99% , based on Welch tests.

Our tests are as follows: We compare two groups (LCH_G_k , LCH_G_l), as introduced above, with the Welch test. The groups are obtained by similarity values (Experiment 1) or by similarity list indices (Experiment 2). The null hypothesis in a Welch test is that the two groups have equal means. One either rejects the null hypothesis at a confidence level chosen apriori (99% in our case), or there is not enough evidence to do so. In case of a rejection, we conclude that there is a significant difference between the two groups in terms of similarity. I.e., the group with the higher LCH mean contains significantly more similar word pairs.

4.1.4.2. Experimental Results for Similarity Values.

For the similarity value group evaluation, we first test the exemplary questions asked in the introduction. We then investigate generally to what extent similarity value groups are different.

Question 1. Are low values of similarity comparable to each other?

Question 2. If Words A and B have a higher similarity value than A and C (say 0.2 for A and B, 0.1 for A and C), is A more similar to B than to C?

For Question 2., we test the following null hypothesis: The aggregated LCH scores have the same mean values for the word pairs with a 0.10 and with a 0.20 similarity value. - The number computed on *our corpus* is as follows:

$$welch\_test\_p\_value\,(LCH\_G_{0.10}, LCH\_G_{0.20}) = 5.19e^{-9} < 0.01.$$

So we conclude with 99% confidence that the hypothesis is false. We infer that the word pairs with 0.20 similarity values tend to be more similar to each other than the pairs with 0.10 similarity values. In other words, to answer Question 1., even at these low levels of similarity, differences in value have a meaning.

We now turn to the systematic experiment concerning this model and generalize the findings in Section 4.1.5. For every group, we search for the next successive group, i.e., having a higher index, which significantly differs in its LCH scores with 99% confidence (cf. Figure 4.4). We explain the interpretation of the values with the following example: For the -0.30 similarity value group (x-axis), the next successive group which significantly differs in similarity is the -0.17 similarity value group (y-axis). Starting from the -0.18 (x-axis) group, every successive group has a significantly higher LCH score mean than the previous one. On the other hand, there is a bend in the plot at -0.18. It means that, at low values of similarity, i.e., below -0.18, there is no significant evidence that a higher similarity value group implies a higher LCH score. Hence, we conclude that below the -0.18 similarity value there is no significant difference between the groups.

Another way to understand these values is as follows: Somewhat naturally, we assume that the -0.40 similarity value group contains dissimilar word pairs. This is because it is the group with the pairs with the smallest similarity values. For this group, we calculate the next group having a significantly higher LCH mean score, this is the -0.18 similarity value group. This means that between -0.40 and -0.18 there is no significant difference in LCH scores between the groups. Based on our assumption that the -0.40 group contains dissimilar word pairs, we conclude that the word pairs with similarity values between -0.40 and -0.18 are dissimilar.

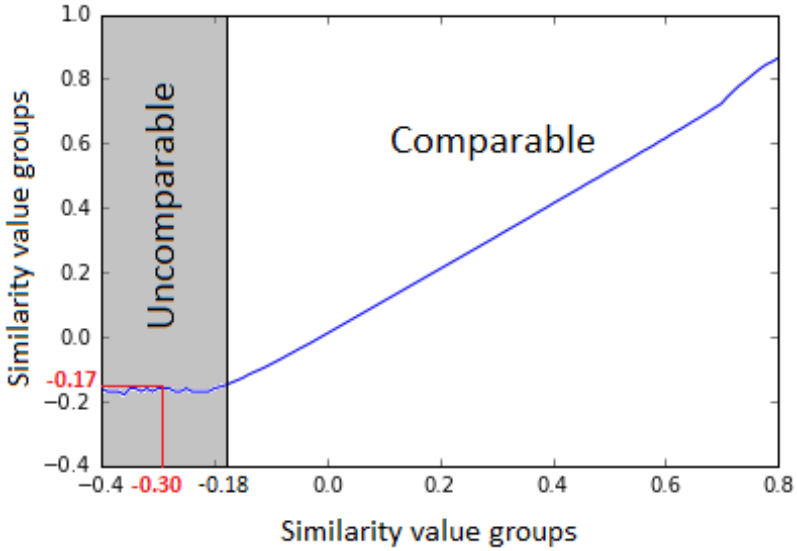

Figure 4.4.: Groups with significant differences in LCH mean scores by similarity values

Because of the relatively high standard deviation in the groups, we cannot conclude that all word pairs in these groups are dissimilar, but we can say that the groups do not differ significantly. For higher similarity values, i.e., above -0.18, every group is significantly different, as we have seen. This means that any increase in similarity, even if it is only 0.01, implies a higher similarity of the word pairs. Again, we cannot say this for every specific word pair, because of the high deviation, but only in general terms, for the groups as a whole.

Overall, we conclude that the similarity value groups are significantly different from each other above -0.18 and not different below this value. This also can be seen visually, as there is a specific bend in Figure 4.4 at -0.18 on the x-axis.

4.1.4.3. Experimental Results for Similarity Lists.

We now investigate the exemplary questions asked in the introduction with similarity lists.

Question 3. Does being in the top 100 list of most similar words always imply similarity, or does not being in the top 500 list always imply dissimilarity?

Question 4. What are meaningful threshold values, and how to find them?

We answer these questions with the following experiments. Our experiments with similarity lists actually are the same as just before, but with the word pairs being grouped by

list indices. Figure 4.3 shows that there is a long almost horizontal noisy stripe of LCH averages. We are making the same tests for the index groups ($G_k, k \in \{1, \ldots, 100000\}$) as we have with the similarity value groups, again with 99% confidence. For every index group, we search the next group with higher index with a significantly different LCH similarity mean using that test. The figure shows the following: At the smallest indices, even small differences in the indices imply significantly different mean score. But as the indices increase, the bigger the differences have to be between groups to yield a significant difference in the mean.

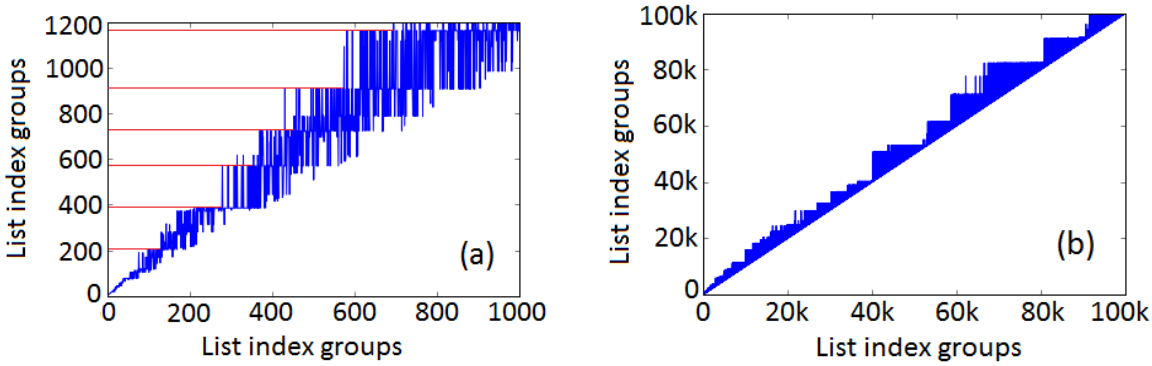

Figure 4.5.: Groups with significant differences in LCH mean scores by similarity indices

Figures 4.5a-b show that there are certain indices which generally identify the significant differences. These indices correspond to groups with particularly high LCH mean scores, and because of this, they are significantly different from many lower index groups. The horizontal lines in Figure 4.5a identify them.

Just as we have done with the similarity values, we assume that the last group of word pairs, i.e., pairs consisting of a word and its least similar word, are dissimilar. We test two items:

- Which is the last of these groups that is significantly different from the very last group regarding LCH score? Formally, what is the highest index (i) so that, for every $j > i$, $welch\_test\_p\_value\left(LCH\_G_{100000}, LCH\_G_j\right) > 0.01$ holds?
- What is the first group that is not significantly different from the last group? Formally, what is the lowest index (i) so that $welch\_test\_p\_value\left(LCH\_G_{100000}, LCH\_G_j\right) < 0.01$ holds, for every $j < i$?

The answers to these questions are the 31,584$^{\text{th}}$ group and the 6,094$^{\text{th}}$ group, respectively, for the *specific model* we are working with in this section. Namely, the horizontal line in Figure 26 is the one separating the groups whose LCH mean scores are significantly higher than the one of the last group from the rest. We conclude that indices higher than 31,584 are statistically not different from the last group. Based on our assumption, our interpretation is that they contain dissimilar word pairs. On the other side, all groups with indices below 6,094 have a higher LCH mean than the last group. This means that they all contain significantly more similar word pairs. Again this does not mean that all the

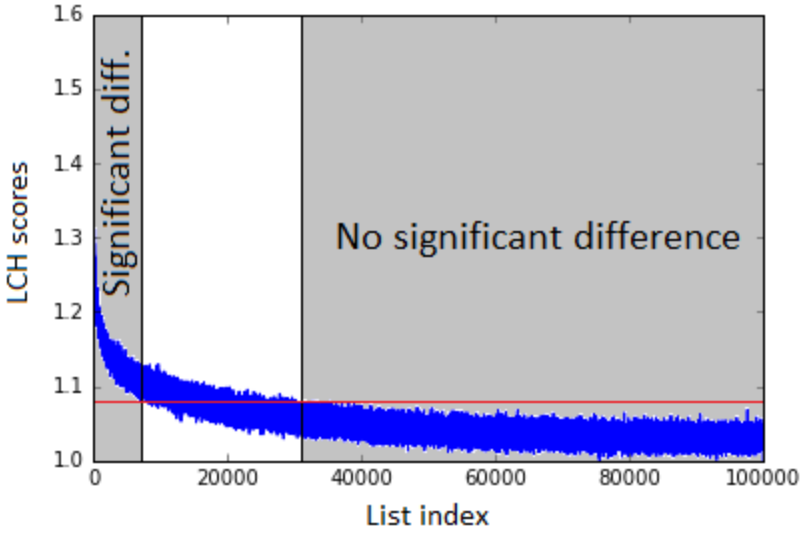

Figure 4.6.: Meaningful list indices

word pairs in these groups are dissimilar or similar, respectively, but that the groups differ significantly.

Implications and External Validity. The experimental results indicate that a confidence-based comparison based on statistical tests identifies large ranges of steady similarity values as well as large ranges of list positions where the similarity of word pairs is meaningful. However, the results so far are specific to the model and text corpus used. In the next section we generalize our insights with further models trained on different corpora to find meaningful similarity values.

4.1.5. Generalization with Additional Corpora

The results from the prior subsubsection indicate that our approach to identify meaningful similarity values with a statistical test is promising. The results in Sections 4.1.4.2 and 4.1.4.3 are already interesting for practitioners, as the corpus, embedding model (with these parameters), and the baseline are widely used. We now show that our approach yields meaningful results with other corpora as well.

Rationale Behind the Experiments. With the model algorithm (e.g., SG or GloVe) and the parameters changing, the similarity values and lists change as well, cf. Section 3.1.3. This means that one must adjust the specific numbers that identify ranges where similarity is meaningful for any other model. To show that the procedure we propose is generally relevant, we train two other models with different underlying corpora, but with the same model and parameter setting. To make the results of the experiments comparable, we use corpora of the same size as before. If the results (i.e., the plots) will be highly similar

to those from Section 3.1, we will claim that our method to find meaningful similarity thresholds or list sizes is valid in general.

Experimental Results. The first dataset we train a model on is the full a Wikipedia dump. The second model is trained on a 5-gram corpus extracted from the Google Books n-gram dataset [81]. We have extracted the 5-grams, shuffled them, and trimmed the data to have the same size as our original 1 billion word dataset. We note that working with 5-grams as the underlying corpus is slightly different from working with full-text corpora. This is because of the limited size of the 5-grams, i.e., all the sentences considered by the learning algorithm only have a length of 5 (see Section 2.3). We conduct the same experiments with the models trained on these corpora as in Section 4.1.4.2 to achieve comparable results.

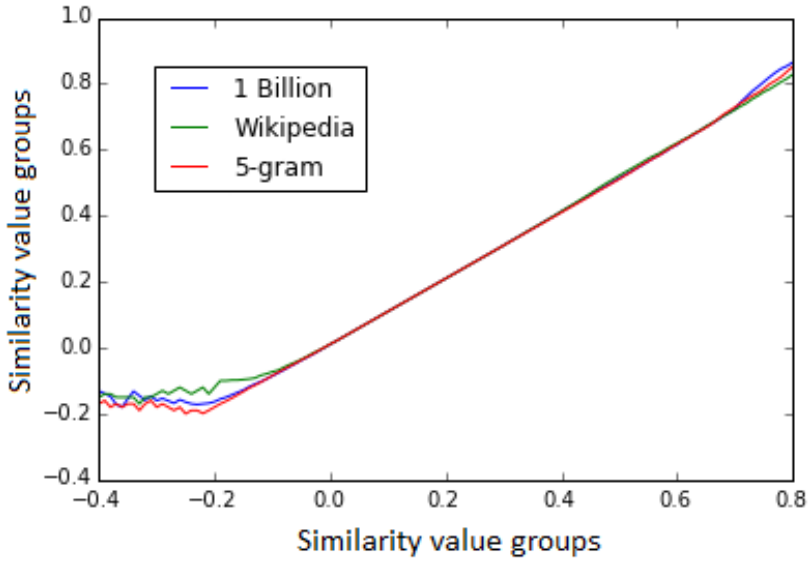

Figure 4.7.: Significantly different groups by similarity value

Figure 4.7 shows that the results are almost identical to the ones in Section 4.1.4.3. The structure of the figures and even the values are very similar. For all three models, the similarity values which are not meaningful are between -0.4 and approximately -0.2.

As for the similarity lists, Figure 4.8 shows that they are very similar, but they naturally differ in the actual values. We also test the two models regarding the same questions we have asked earlier, namely: What is the last group which is significantly different in LCH similarity score from the last group overall? What is the first group that is not significantly different from the last group? The results are 28,570 and 5,889, respectively, for the model trained on the Wikipedia corpus and 35,402 and 6,408, respectively, for the model trained on the 5-grams. These numbers also are very much like the ones calculated before.

All this shows that our approach to derive those threshold values is independent of the underlying corpus. The approach is applicable on any kind of corpus, and only the model selection and its parameters influence the resulting numbers.

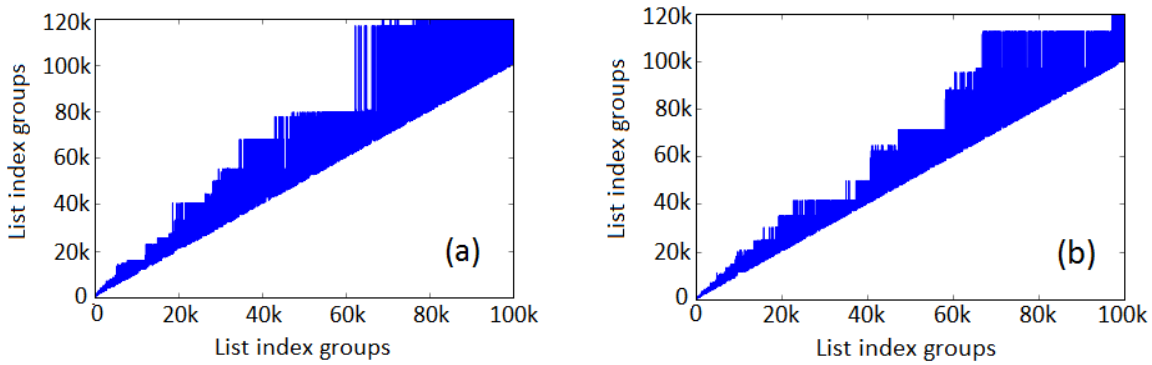

Figure 4.8.: Significantly different groups by list indices for the models trained on 5-grams (a), Wikipedia (b)

4.2. Similarity Threshold Aware Evaluation Method

In this section, we introduce two evaluation methods for word embedding models. The first one is the baseline method, which is also the commonly used approach in previous literature. The second one is our new similarity threshold aware evaluation method. We investigate how our method affects results on different word similarity test sets with various models. Generally, the objective of our method is not to achieve higher scores than using the baseline method, but to allow for a more reliable evaluation of embedding models on test sets.

4.2.1. Investigation Objectives

The results of Section 4.1 so far indicate that meaningful ranges of similarity values exist. More specifically, for these values it is meaningful to compare two word pairs with different similarity values and to conclude that higher values imply greater semantic similarity. In contrast, the values outside of these regions are either very noisy, because of the lack of word pairs with the respective values, or indistinguishable in terms of similarity.

Existing evaluation methods compare word pair similarities on the full scale of similarity values and lists. Based on our results so far, we propose that the comparison should only be done at certain ranges of similarities. One can determine these ranges using the experiments proposed in Section 4.1. In particular, we propose that only values should be compared which significantly differ in mean similarity scores, cf. Figure 4.4. For example, when evaluating the model in this section one should only compare word pair similarity values when the values are above -0.18. It is also noteworthy that every 0.01 difference in this range implies a significantly different similarity. For the list indices, similar conclusions are feasible. For example, with the model of this section we recommend comparing only indices below approximately 31,500.

With other models, these values and indices could be different, but the method of calculating them and the implications are the same. This means that for any embedding model we propose to calculate these values first, to improve any evaluation. We call this method similarity threshold aware evaluation.

Generally, we expect observable, but minor changes in the correlation scores. Nevertheless, as this evaluation method optimization affects all embedding models, and improvements reported are often only small, even minor differences are of practical relevance. To quantify the effect of our method, we focus on the following research questions.

Question 5. Do word pairs exist in the evaluation test sets which fall below the calculated threshold?

The thresholds may be very low, as we have seen previously. Hence, it may even turn out that the similarity test sets do not feature such word pairs at all, or only some of them are affected. If this was the case, our similarity threshold aware evaluation method would not increase the reliability of the evaluation results.

Question 6. Does our similarity threshold aware evaluation method observably change the evaluation scores, compared to the baseline method?

It is not obvious whether the scores will change with our method. Therefore, we aim at quantifying the effect as well its direction, i.e., whether the scores tend to increase or decrease. We conduct systematic experiments to answer this question.

4.2.2. Evaluation Setup

In this subsection, we describe test sets, embedding models and evaluation methods we use.

Evaluation Test Sets and Embedding Models. We use the six test sets that we have introduced in Section 2.4.2 to evaluate word similarity. Each test set has the same format. They contain word pairs with similarity scores, assigned by human annotators. The smallest test set contains 65, the biggest 3000 word pairs. The test sets, again, are Finkelstein et al.'s WordSim353 test set; Bruni et al.'s MEN test set; Hill et al.'s SimLex-999 test set; Rubinstein et al.'s RG-65 test set, Radinsky et al.'s Mechanical Turk test set and Luong et al.'s Rare Words test set.

We train three different word embedding models (CBOW, SG, GloVe) with the same parameter settings ($dict\_size = 100,000$, $d = 100, win = 5$) on the full 1 billion word corpus. We evaluate the embedding models with two evaluation methods.

Baseline Method. To evaluate a model with the baseline method, we do the following for each similarity test set:

1. We create two ranked lists of the word pairs both sorted by similarity value. Both lists share the same word pairs.

    a) The first list is sorted according to the scores provided in the similarity test set, i.e., created by human annotators.

    b) The second list is sorted according to the similarity values computed using the embedding model.

2. We calculate the Spearman's correlation between both rankings. This indicates how well the embedding model similarities reflect the ground truth.

Using this method has two benefits. First, the values of the scores do not have to be the same as we compare the ranks, not explicitly the values. Usually, human annotators use a point system, e.g., from 0 to 10, where 10 indicates maximum similarity. By contrast, cosine similarity values are distributed between [-1,1], and a value of 1 indicates the highest similarity. Second, the embedding model reflects the similarity test set well if the rank of most word pairs in both lists is *very similar*, but it does not necessarily have to be the same. Spearman's correlation expresses this well. The score can interpreted as follows. A correlation of 1 states that the ranked lists are identical. A value close to 1 means that the embedding models reflects the similarity test set very well. A value close to zero indicates that there is no connection between the similarity values and the ground truth.

Similarity Threshold Aware Method. It is important to note that the objective of our similarity threshold aware evaluation method is not to increase the value of the Spearman's correlation compared to the baseline method. The objective is to return a *more reliable* correlation score. Therefore, the score might be higher, lower, or even remain the same. The explanation why our method results in a more reliable score is justified based on statistics. The intuition is the following. The difference of our and the baseline method is how we compute the similarity threshold. Our new method removes every word pair from both lists where the cosine distance computed on the embedding models is below the threshold. This is because differences in the similarity and the resulting ranks for these word pairs are not reliable, as shown based on the statistical tests in the prior section[1]. Hence, they might change the correlation in an unpredictable way. If the order of these word pairs accidentally is similar to their order in the test set, the originally computed correlation score, i.e., the one computed with the baseline method, is too high. In the opposite case, the score is too low. Finally, in case the order is randomized, the score does not change at all, but the new result still is more reliable. Consequently, we can perceive such word pairs which fall below the threshold as noise, making the score less reliable. This is why they have to be excluded from the evaluation.

4.2.3. Evaluation Results

In this section we present our evaluation results of the comparison between the baseline and our similarity threshold aware evaluation methods.

Model Thresholds. First we calculate the similarity value thresholds for all three models. The calculation itself is quite complicated, as described in Section 4.1, but the output has a very simple structure, i.e., a floating point number for every model. It represents the similarity value threshold. The actual numbers are presented in Table 4.1.

[1] Note, we find all these word pairs at the end of the model's list as they have low similarity values.

Model	Threshold
CBOW	-0.18
SG	0.20
GloVe	-0.15

Table 4.1.: Similarity value thresholds for different models

We see that the SG model has a significantly higher threshold value than the other models. This is expected, as the SG model generally has much higher similarity values on average, cf. Figure 3.1.

Test Set Evaluation Results. For every model we create two tables. In the first tables, we count for every test set how many word pairs fall below the similarity value threshold. The first row shows how many word pairs exist in the full test set. The second row shows how many are above the threshold, and the third one how many are below.

The second tables show the evaluation results for the models. In the first row the numbers represent the results with the baseline method, i.e., with all word pairs in the test sets. The second row shows the results only for the word pairs which are comparable, i.e., are above the threshold.

	rg-65	ws353	rare	simlex	mturk	men
Total	65	353	2034	999	287	3000
Comparable	64	349	2028	997	287	2983
Not Comp. %	1.5	1.1	0.3	0.2	0	0.6

Table 4.2.: CBOW word pair counts

	rg-65	ws353	rare	simlex	mturk	men
Baseline	0.54	0.53	0.32	0.31	0.57	0.64
Ours	0.55	0.52	0.32	0.31	0.57	0.63

Table 4.3.: CBOW evaluation results

Tables 4.2 and 4.3 show that there are word pairs in the test sets for the CBOW model which fall below the threshold. However, there are only a few of them. Consequently, the new evaluation method changes the evaluation results only slightly.

	rg-65	ws353	rare	simlex	mturk	men
Total	65	353	2034	999	287	3000
Comparable	59	318	1838	960	262	2672
Not Comp. %	9.2	9.9	9.6	3.9	8.7	10.7

Table 4.4.: SG word pair counts

	rg-65	ws353	rare	simlex	mturk	men
Baseline	0.59	0.6	0.37	0.33	0.6	0.7
Ours	0.59	0.55	0.33	0.28	0.63	0.64

Table 4.5.: SG evaluation results

The tables for the SG model, Tables 4.4 and 4.5, show that much more word pairs fall below the threshold for this model, compared to the CBOW model. For most test sets, the evaluation results change significantly. We also see that the evaluation scores for different test sets are behaving rather inconsistently with the different test sets. For instance, the score is significantly better on the ws353 or men test sets, it is worse on the mechanical turk test set, and it is the same on the rg-65.

	rg-65	ws353	rare	simlex	mturk	men
Total	65	353	2034	999	287	3000
Comparable	62	350	2023	989	287	2991
Not Comp. %	4.5	0.8	0.5	1.0	0	0.3

Table 4.6.: Glove word pair counts

	rg-65	ws353	rare	simlex	mturk	men
Baseline	0.59	0.43	0.21	0.27	0.43	0.64
Ours	0.6	0.42	0.21	0.27	0.43	0.64

Table 4.7.: Glove evaluation results

The tables for the GloVe model, Tables 4.6 and 4.7, are very similar to the ones for the CBOW model. Very few word pairs fall below the threshold. This leads to very small changes in the evaluation scores.

Evaluation Results - Summary. Having all these results, we can answer the research questions presented previously.

Question 5. Do word pairs exist in the evaluation test sets which fall below the calculated threshold?

Our results confirm that such word pairs exist in almost every model and test set combination. As expected, their number is model dependent and rather low. For the CBOW and GloVe models, less than 1% of the word pairs usually fall below the respective thresholds. For both models, the rg-65 test set yields a particularly high number (1.5% and 4.5% respectively), most likely due to the small size of this test set. On the other hand, many word pairs fall below the models threshold for the SG model, the percentage ranging between 4 and 11% .

Question 6. Does our similarity threshold aware evaluation method observably change the evaluation scores, compared to the baseline method?

The answer to this question again is highly model dependent. As hypothesized earlier, the change is observable, but not significant, with the exception of the SG model. This is mainly because of the sheer number of word pairs being incomparable for the different models. We do not see a consistent increase or decrease in the evaluation scores. This further confirms that our method correctly removes the noise introduced by word pairs falling below the similarity threshold. We conclude that word pairs whose similarity is below the model threshold should be excluded from the evaluation test sets.

5. Application Analysis of Word Embedding Models

In the previous two parts of our three-way analysis of word embedding models we have evaluated algorithmic and theoretical questions regarding word embedding models. Now, we turn our attention to down-stream applications. The NLP problem we study in this chapter is one of the most intuitive use case for word embedding models, namely text classification.

In general, text classification accuracy heavily relies on the number of labeled samples a classifier is presented with. However, since labeling is usually done by humans, it is costly and time consuming. For example, many highly researched AI solutions such as autonomous driving are based on enormous human tagging effort. This means, in most cases, labeled data is scarce. In this chapter, we present an approach that uses exogenous information on both word semantics and distribution to compensate the scarcity of labeled data.

Our approach is a preprocessing step that increases training data quality and ultimately increases text classification accuracy. It compensates all three general issues presented by the shortage of labeled training data, namely task-specific synonyms, OOV words and overfitting. Our approach uses term substitution based on the core idea of this chapter: A novel semantic-distributional word distance measure. The novelty of the approach is using a combination of both semantic and distributional information of words. Since, we only preprocess the training data, our method is generally applicable to any text classification algorithm.

The results show that our preprocessing method helps improving text classification accuracy for each classifier we work with. The improvements are especially visible when the training data is small.

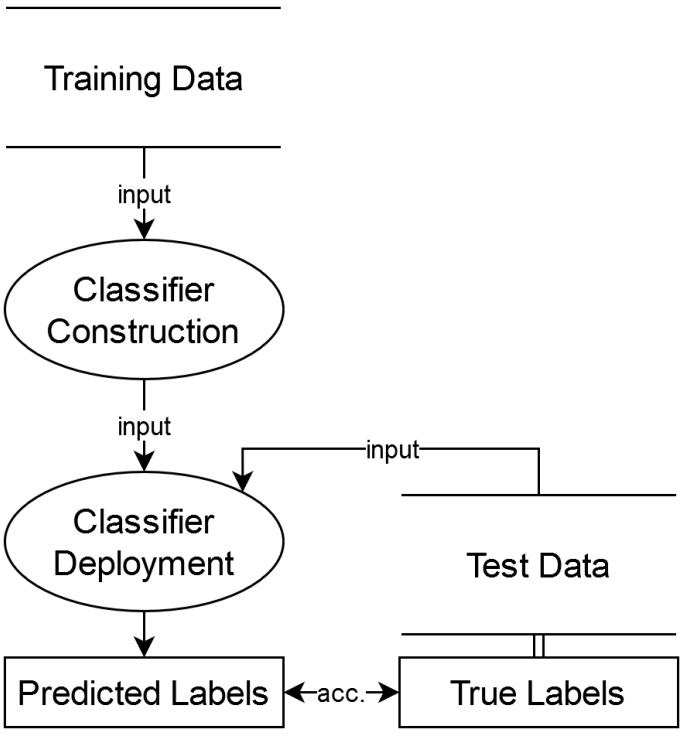

Figure 5.1.: General text classification flowchart

5.1. Intuition

In general, a text classification procedure follows the following scenario. There is a training data set, which we use to train the classifier. Next, we deploy the classifier to predict labels on a test data set, and evaluate the accuracy of the predictions, comparing the predicted labels to the true labels. Figure 5.1 shows this simple flowchart. This, and all other flowcharts in this chapter follows the Yourdon/DeMarco notation standard. Boxes represent input/output data, ellipses represent functions and stripes represent databases. Arrows represent the flow between the states or data.

Our preprocessing approach, that we present in this chapter, affects both the training and test data, but no other parts of the flowchart. This means, we are able to train any classifier after the preprocessing as well that we used to train the original training data.

As we have discussed before, because of the high complexity of natural language, classifiers trained with too few data samples, have several reoccurring issues. The lack of robust statistical information makes the number of infrequent and OOV words high, which makes the classification error-prone. Also, because of the lack of training samples classifiers tend to overfit the data, which is detrimental to classification accuracy.

The intuition behind our method is to improve the ability of text classification algorithms to generalize. We do so by transforming the input data so that task-specific similar words are merged. All words in a group are mapped onto one representative word (see Section

2.2). With this, unseen words can be handled, and random effects due to words with few occurrences in the training data are reduced.

At real world companies, in a lot of cases, usually human domain experts create the clusters of task specific synonyms. In the following, we illustrate this scenario with a simplified example to give the reader intuition of the procedure.

5.1.1. Manual Clustering by Human Domain Experts

In spite of their effectiveness, it is noteworthy that most text classification algorithms have no notion of the semantics behind the words contained in documents they classify: a word is usually represented as a dimension in a vector space with hundreds of thousands of other dimensions (see Section 2.2.1.1). For example, in probabilistic classifiers, such as Naive Bayes, it is the word statistics generated by the large number of pre-classified documents that provide the information relevant to the underlying classification task.

By contrast, the way how humans categorize text is heavily based on the knowledge about semantic relationships between words. It is this internal representation of language that allows humans to classify text without first reading thousands of example documents.

During manual clustering, depending on the classification task, the domain expert preemptively creates various sets of task specific synonyms.

For example, consider a sentence-classification task, namely that we want to detect changes in company management. A classifier might fail to classify “The corporation announced a new CEO for 2017” as positive, even if it has seen the sentence “The company announced a new executive for 2017” in the training data, because it is unaware of synonyms. For the recognition of text snippets that are about change in management, the domain expert creates the following dictionaries:

- T_{manager}: CEO, CFO, leader, …
- T_{company}: corporation, enterprise, business, concern, venture, …

The words in the dictionaries might indicate that a sentence refers to change in management personnel for a company. While an automated semantic similarity-based algorithm might falsely include the word ”coach” in the ”manager” or ”team” in the ”company” dictionary, a human annotator knows that in this scenario they should not be included, since they most likely relate to football, not to the industry.

Whenever a document in the training dataset contains a word that appears in one of the dictionaries, we substitute it with the word that represents it. To illustrate, consider the following sentence: “The **corporation** announced a new **CEO** for 2017”. In this case, the words in this sentence are substituted as follows: “The **company** announced a new **manager** for 2017”.

The dictionaries are also used to substitute OOV words with words that the classifier knows. If a test document contains an OOV word, we substitute it by looking up whether it occurs in one of the dictionaries.

Figure 5.2 show the flowchart of the manual preprocessing method.

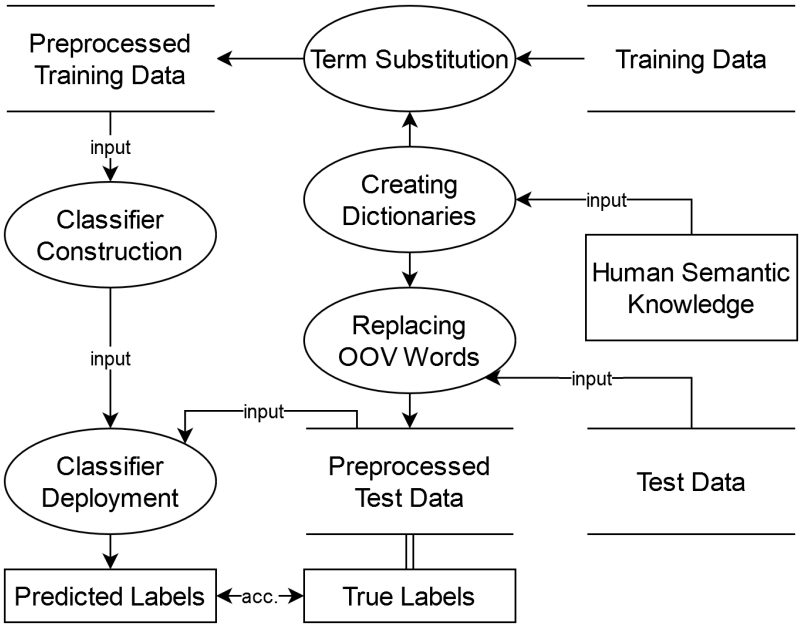

Figure 5.2.: Preprocessing method based on human semantic knowledge

In spite of its beneficial effects, preprocessing the dataset manually is a time consuming and costly process. In the following sections we show how to automate the lexical substitution process otherwise done by a human domain expert, by finding groups of semantically similar words using our word distance measure.

5.1.2. Automating the Clustering

In this section we explain why a purely semantic clustering-based approach is not sufficient by showing the importance of word frequencies. We also present how to automate the dictionary-based preprocessing method using our novel distance measure.

Naive Approach: Using only Semantic Distances for Clustering. An intuitive approach which we have implemented first to find groups of similar words would be to use a clustering algorithm based on the distance measure in a word embedding space [75]. However, text classification accuracy on many datasets became worse than using the original, unprocessed datasets. This shows that semantics alone are not enough to reduce the feature dimensionality while improving the text classification accuracy.

Hence, we investigated whether there is any important information about the terms that is not taken into consideration when using clustering based on distances in word embeddings only. Analyzing the resulting clusters, we have found that the *word frequencies*, i.e., information on the distribution of words among the positive and negative classes, play an important role in classification effectiveness.

Importance of Word Frequencies. To show why the approach from the previous section, i.e., where we only use semantic distances for clustering, often is not beneficial for classification accuracy, we computed actual clusters of words over the vocabulary of the *MPQA* dataset using an agglomerative clustering algorithm (see Sections 2.4.3 and 2.2.2.4 for the dataset and the clustering algorithm, respectively). The aim of the MPQA dataset is to facilitate distinctions between negative and positive expressions. Every expression in the MPQA training data is labeled either as negative or positive.

Table 5.1 shows several clusters found. The numbers in the tables represent the total frequency of the words in the positive and negative classes, respectively. We can see that the groups contain terms that are generally semantically similar, for example, *Cluster 5* contains words about governmental forms.

Cluster 1	+	-	**Cluster 2**	+	-	**Cluster 3**	+	-
clearly	4	8	absurdly	0	1	*illegitimate*	0	12
deeply	0	5	admittedly	1	0	*legitimate*	35	9
genuinely	2	0	amazingly	1	0	ruse	0	1
greatly	2	4	awfully	0	1	sham	0	2
massively	0	2	especially	3	4			
undoubtedly	2	1	extremely	2	12			
Cluster 4	+	-	**Cluster 5**	+	-	**Cluster 6**	+	-
abhorrent	0	1	anarchical	0	1	anger	0	11
despicable	0	1	despotic	0	1	disapproval	0	6
disgraceful	0	1	dictatorial	0	5	discontent	0	7
inexcusable	0	1	hegemony	1	4	disgust	0	2
inhuman	0	7	imperialism	2	1	displeasure	0	3
inhumane	0	15	imperialist	0	4	disquiet	0	1

Table 5.1.: Clusters yielded by the naive approach

However, we can see that in *Cluster 3*, the words “illegitimate” and “legitimate” were clustered together despite being antonyms and indicators of the negative and positive class, respectively. Clustering antonyms together is intuitively detrimental for classification accuracy. Nevertheless, the similarity of words in the word embedding vector space depends on how probable it is that they appear in the same context, hence word vectors of antonyms are usually located close to each other [73].

The naive approach neglects the word frequencies in the different classes, i.e., negative and positive expressions, which are a clear indicator, in this case, that these words should *not* be assigned to the same cluster. It is obvious that illegitimate and legitimate are significantly differently distributed among the positive and negative samples – a strong indication that they are not used synonymously in this task.

Cluster 1	**+**	**-**	**Cluster 2**			**Cluster 3**	**+**	**-**
illegality	0	1	genuine	1	0	abhorrent	0	1
illegitimacy	0	1	*legitimate*	35	9	affront	0	3
illegitimate	0	12	logical	3	3	barbaric	0	2
			rational	1	0	despicable	0	1
			realistic	4	0	disgraceful	0	1
						inhumane	0	15
Cluster 4	**+**	**-**	**Cluster 5**	**+**	**-**	**Cluster 6**	**+**	**-**
anger	0	11	anarchical	0	1	absurdly	0	1
disapproval	0	6	capitalism	0	1	amazingly	1	0
displeasure	0	3	dictatorial	0	5	ridiculously	0	1
frustration	0	6	hegemony	1	4			
impatience	0	2	imperialism	2	1			
irritation	0	2	imperialist	0	4			

Table 5.2.: Clusters yielded using our preprocessing method

In this work, we propose to integrate the information of the word frequencies into the classification algorithm. Table 5.2 shows the resulting clustering when using the distance measure introduced in this work. Since our distance measure also utilizes the distributional information of terms, “illegitimate” and “legitimate” now are in different clusters.

In addition to separating words with highly different word frequencies, our distance metric considers the statistical robustness of words as well. This is important since the distributional information tend to be unreliable for low frequency words. For example, if a word appears only a few times in the positive and never in the negative class, it should not be used as a strong indicator for positive documents.

For example, consider *Cluster 5* in Table 5.2: The words “imperialism” and “imperialist” are in the same cluster, but they are distributed differently among the classes: “imperialism” appears more often in the positive class than in the negative class, while “imperialist” is present only in the negative category. This is because their frequencies are very low and thus unreliable estimates of the true word probabilities. In such cases our distance metric is more tolerant and will not separate the words in question. This illustrates the strength of our approach: Due to the low counts for both words, their distributional difference is not significant enough and does not outweigh the low semantic distance.

5.2. The Semantic-Distributional Distance Measure

In this section we describe our novel semantic-distributional distance measure. The derivation of our measure consists of the following steps.

1. Modeling of word occurrences as being generated by a Bernoulli process.
2. Interpretation of assigning words to the same cluster as a probabilistic hypothesis.
3. Using *Bayesian Hypothesis Testing* to assess the plausibility of the probabilistic hypothesis.
4. Incorporating semantic information by using Bayesian priors.

5.2.1. Word occurrences as a Bernoulli Process

To derive the distributional dissimilarity between pairs of words, it is common to view word occurrences as being generated by a probabilistic process. A probabilistic model commonly used to describe word occurrences distributed over a dichotomy of classes is the *Bernoulli process* [52]. Suppose that a word $v \in V$ appears n times in a labeled dataset D. X is a random variable representing the occurrences of v in the positive class. Assuming an underlying Bernoulli process, the probability that, for n total occurrences, v appears k times in the positive class is:

$$B(n, k, \theta) := \Pr(X = k;\ \theta) = \binom{n}{k} \theta^k (1-\theta)^{n-k}$$

where $\theta \in [0, 1]$ is the distribution parameter. For a random variable drawn from a Bernoulli process with n trials, we use the notation:

$$X \sim B(n, \theta)$$

5.2.2. Merging Words as Probabilistic Events

We now perceive the decision when to assign two words to the same cluster as a probabilistic event. Ultimately, this implies modeling the decision of how useful it is to the underlying classification task to substitute a word v_1 with a word v_2 or vice versa. From now on, instead of speaking of "substituting words", we speak of "merging words", since the substitution direction is irrelevant.

We now model the decision process (*merging* or *not merging*) probabilistically. To do so, we associate either decision with the probabilities of two hypotheses H_0, H_1. They represent the plausibility of merging or not merging respectively.

Hypothesis Formulation. Let X_i, X_j be the random variables that represent the occurrences of $v_i, v_j \in V$. Let n_i, n_j be the total number of occurrences observed for v_i, v_j respectively.

Hypothesis H_0. (No Merge) In H_0, we assume that the observed word occurrences X_i, X_j are generated by two distinct latent, independent Bernoulli processes, i.e.,

$$H_0 : X_i \sim B(n_i, \theta_1), X_j \sim B(n_j, \theta_2), \theta_1 \neq \theta_2$$

In this scenario, we do not assume any relatedness of X_i, X_j, i.e., we do not merge v_1 and v_2.

Hypothesis H_1. (Merge) The probability of the second hypothesis H_1 should express how well we can explain the observed data if we assume that a common, latent generative process generates the occurrences of both words X_i, X_j:

$$H_1 : X_i \sim B(n_i, \theta_1), X_j \sim B(n_j, \theta_2), \theta_1 = \theta_2$$

High probabilities indicate that v_i and v_j are generated by similar processes and can be merged without losing any discriminatory information.

The goal now is to calculate the probabilities of the competing hypotheses H_0, H_1, given the observed data. We do this in Section 5.2.3 using *Bayesian Hypothesis Testing*. But before applying such methods, we establish estimates for the parameters (θ) of the underlying Bernoulli processes.

Maximum Likelihood Estimation. When there is no further knowledge about the generative process, a commonly used method to estimate the parameters of the underlying distribution is Maximum Likelihood Estimation (MLE). As shown in [89], in case of a Bernoulli process with n trials and k successes, the MLE is

$$\hat{\theta}_{MLE} = \arg\max_{\theta'} B(n, k, \theta') = \frac{k}{n}$$

Let k_i, k_j be the numbers of occurrences of v_i, v_j in the positive class. For H_0, the MLEs for each Bernoulli process generating X_i are the following ones:

$$H_0 : \hat{\theta}_{H_0,i} = \frac{k_i}{n_i}$$

For the second hypothesis, we assume that the independent observations X_i, X_j have the same underlying distribution parameter. Assuming that this hypothesis is true, the MLE for H_1 is

$$H_1 : \hat{\theta}_{H_1} = \hat{\theta}_{H_1,1} = \hat{\theta}_{H_1,2} = \frac{k_i + k_j}{n_i + n_j}$$

In the next section we describe how to use *Bayesian Hypothesis Testing* to assess the probability of either hypothesis.

5.2.3. Bayesian Hypothesis Testing

Bayesian Hypothesis Testing is a tool to estimate the probability of competing models. In our context, we want to compare how well the occurrences of two words v_i, v_j in a document set D is explained before and after merging these, with decisions being represented by H_0 and H_1. We start with the proposition that either H_0 or H_1 are true, i.e.,

$$\Pr(H_0 \vee H_1|D) = \Pr(H_0|D) + \Pr(H_1|D) = 1 \tag{5.1}$$

The probability of H_0, i.e., two different Bernoulli processes generate the words, can be expressed using the Bayesian Rule:

$$\Pr(H_0|D) = \frac{\Pr(D|H_0)\Pr(H_0)}{\Pr(H_0)\Pr(D|H_0) + \Pr(H_1)\Pr(D|H_1)} \tag{5.2}$$

$$\Pr(H_1|D) \overset{(5.1)}{=} 1 - \Pr(H_0|D)$$

Note that, since Hypotheses H_0 and H_1 are complementary, computing the probability of either one is sufficient. Without loss of generality, we now only use $\Pr(H_0|D)$ since it also is a dissimilarity measure, i.e., higher probabilities mean higher distributional incompatibility. We first simplify $\Pr(H_0|D)$ as follows:

$$\begin{aligned}\Pr(H_0|D) &= \frac{\Pr(D|H_0)\Pr(H_0)}{\Pr(H_0)\Pr(D|H_0) + \Pr(H_1)\Pr(D|H_1)} \\ &= \frac{1}{1 + \underbrace{\frac{\Pr(H_1)}{\Pr(H_0)}\frac{\Pr(D|H_1)}{\Pr(D|H_0)}}_{=:\kappa}} = (1+\kappa)^{-1}\end{aligned}$$

First we derive an estimate for κ and then we will insert it back in the estimate for $\Pr(H_0|D)$ at the end of our argumentation. κ is called the *Bayes Factor* and is often used as an alternative to the probability to express the plausibility of H_0 over H_1. From now on, we will refer to $\Pr(H_0|D)$ as *semantic-distributional dissimilarity* or *semantic-distributional distance*.

5.2.4. Estimation of the Semantic-Distributional Distance Measure

In this section, we derive an estimate for each probability of the Bayesian model, i.e., each probability on the right side of Equations 5.2. We will denote probability estimates with $\hat{}$ as follows:

$$\hat{P}(H_0|D) \coloneqq (1+\hat{\kappa})^{-1}; \qquad \hat{\kappa} \coloneqq \frac{\hat{P}(H_1)\hat{P}(D|H_1)}{\hat{P}(H_0)\hat{P}(D|H_0)}$$

Moreover, k_i, n_i, k_j, n_j denote the word frequencies in the positive class and in the whole dataset of v_i and v_j respectively.

Estimation of model likelihoods. We proceed by using *Bayesian Hypothesis Testing* to derive estimates for the probability $\Pr(H_0|D)$. To do so, we estimate the likelihoods $\Pr(D|H_i)$, $i = 0, 1$. We make the following assumptions:

1. Word occurrences follow a Bernoulli distribution with the probability-density function $B(n, k, \theta)$
2. For word co-occurrences we assume conditional independence (c. i.).

These are common, realistic assumption in NLP research [46] [45]. We parametrize the Bernoulli distribution with the Maximum likelihood estimates from before. For $\Pr(D|H_0)$, the estimate is:[1]

$$\begin{aligned}
&\hat{P}\left(X_i = k_i, X_j = k_j \mid \theta_1 = \hat{\theta}_{H_0,1}, \theta_2 = \hat{\theta}_{H_0,2}\right) \\
&\overset{\text{c.i.}}{=} B(n_i, k_i, \theta_{H_0,1}) \cdot B(n_j, k_j, \theta_{H_0,2}) \\
&= \binom{n_i}{k_i} \hat{\theta}_{H_0,1}^{k_i} \left(1-\hat{\theta}_{H_0,1}\right)^{n_i-k_i} \cdot \\
&\quad \binom{n_j}{k_j} \hat{\theta}_{H_0,2}^{k_j} \left(1-\hat{\theta}_{H_0,2}\right)^{n_j-k_j}
\end{aligned}$$

Analogously, for $\Pr(D|H_1)$ the estimate is

$$\hat{P}(D|H_1) = \binom{n_i}{k_i}\binom{n_j}{k_j} \hat{\theta}_{H_1}^{k_i+k_j} \left(1-\hat{\theta}_{H_1}\right)^{n_i+n_j-k_i-k_j}$$

We then insert $\hat{P}(D|H_i)$, $i = 0, 1$, in the estimate $\hat{\kappa}$ for κ

$$\hat{\kappa} = \frac{\hat{P}(H_1)}{\hat{P}(H_0)} \cdot \frac{\hat{\theta}_{H_1}^{k_i+k_j} \left(1-\hat{\theta}_{H_1}\right)^{n_i+n_j-k_i-k_j}}{\hat{\theta}_{H_0,1}^{k_i} \left(1-\hat{\theta}_{H_0,1}\right)^{n_i-k_i} \hat{\theta}_{H_0,2}^{k_j} \left(1-\hat{\theta}_{H_0,2}\right)^{n_j-k_j}}$$

[1] We consider only the probabilities for v_i and v_j, since all other word probabilities are equal for both hypotheses and do not affect the calculation of $\hat{\kappa}$

Integration of semantic knowledge in the priors. The only probabilities left for estimation are the priors $\Pr(H_0)$, $\Pr(H_1)$. In Bayesian models, the prior probabilities are often used as an interface to incorporate prior, expert knowledge in the models. When there is no further knowledge about the prior probabilities $\Pr(H_0)$ and $\Pr(H_1)$, a common assumption is that every hypothesis is equally probable, i.e., $\hat{P}(H_0) = \hat{P}(H_1) = 0.5$ [123]. However, we can use the information provided by word-embedding models to approximate the priors. The calculation of word vectors implies predicting the probability of a word appearing in a given context. The cosine similarity of two word vectors can be interpreted as an approximation of the probability that two words v_i, v_j appear in similar contexts [85, 84]. It ranges from -1 (unrelated words) to 1 (identical words). If the cosine similarity expresses the relatedness between two words, the *cosine distance* values can be used for dissimilarity. For an embedding $\mathcal{W}$ we define the following distance measure:

$$dist_{\mathcal{W}}(v, v') = \frac{1 - \text{cos-sim}(vec_{\mathcal{W}}(v), vec_{\mathcal{W}}(v'))}{2} \in [0, 1],$$

where $vec_{\mathcal{W}}(v)$ represents the vector corresponding to v in embedding $\mathcal{W}$.

For words farther away in the embedding vector space we favor the first hypothesis, i.e., the words should not be merged. We therefore introduce a parameter $\alpha \in [0, 1]$ that specifies how much the cosine distance lets the priors deviate from 0.5.

$$\hat{P}(H_0; \alpha) \coloneqq \alpha \cdot \left(\frac{1}{2} - dist_{\mathcal{W}}(v_i, v_j)\right) + \frac{1}{2}$$
$$\hat{P}(H_1; \alpha) \coloneqq 1 - \hat{P}(H_0; \alpha)$$

With parameter α included in the equations, we can tune our preprocessing procedure. For $\alpha = 0$, the cosine distance is completely neglected, and the hypotheses are assumed to be equally probable. For $\alpha > 0$, the priors deviate from 0.5 proportionally to $\alpha \cdot dist(v_i, v_j)$. In Section 5.4 we calculate the classification accuracy with different values of α to find the best one for each dataset.

After inserting the above priors in $\hat{\kappa}$, we obtain:

$$\hat{\kappa} = \kappa(\alpha) = \frac{\frac{1}{2} - \alpha \cdot \left(\frac{1}{2} - dist_{\mathcal{W}}(v_i, v_j)\right)}{\alpha \cdot \left(\frac{1}{2} - dist_{\mathcal{W}}(v_i, v_j)\right) + \frac{1}{2}} \cdot \frac{\hat{P}(D|H_1)}{\hat{P}(D|H_0)}$$

Finally, we define the *semantic-distributional distance* that incorporates both distributional and semantic information on the words:

$$\Lambda_{\alpha,\mathcal{W}}(v_i, v_j) \coloneqq \hat{P}(H_0|D) = (1 + \hat{\kappa}(\alpha))^{-1}$$

We note that $\Lambda_{\alpha,\mathcal{W}}(v_i, v_j)$ is defined using the cosine distance of word vectors in an embedding $\mathcal{W}$. Therefore, in order to produce valid distances, v_i and v_j must be part of the vocabulary $voc(\mathcal{W})$.

5.2.4.1. Correction Term

Although the cosine similarity of two word vectors can be interpreted as an approximation of the probability of both words appearing in similar contexts, the method could be improved by learning a function that maps the cosine distance onto a more accurate probability estimation for words being synonyms. However, since we found the cosine similarity part to be too pessimistic, we add a correction term allowing for more weight in the final measure. An additional parameter $\beta \in [0, 1]$ controls the influence of this term:

$$\Lambda_{\alpha,\beta,\mathcal{W}}(v_i, v_j) = \beta\Lambda_{\alpha,\mathcal{W}}(v_i, v_j) + (1 - \beta)dist_{\mathcal{W}}(v_i, v_j)$$

5.3. Implementation

In this section we present the complete text classification process. The first step is to preprocess the training data. The preprocessing of a dataset involves finding clusters of words using our semantic-distributional distance measure and then represent each document as a BoC. Subsequently a classification model with the preprocessed training dataset is built. Prior to prediction, unknown documents that contain OOV words are preprocessed by substituting unknown words with vocabulary contained in the training dataset. In the following, we provide descriptions, including pseudo-codes, for each part, namely the training data preprocessing, the test data preprocessing and the classification building and prediction parts.

5.3.1. Preprocessing Training Data

Figure 5.3 shows the flowchart, while Algorithm 1 shows the algorithm how we preprocess the training data prior building a classification model.

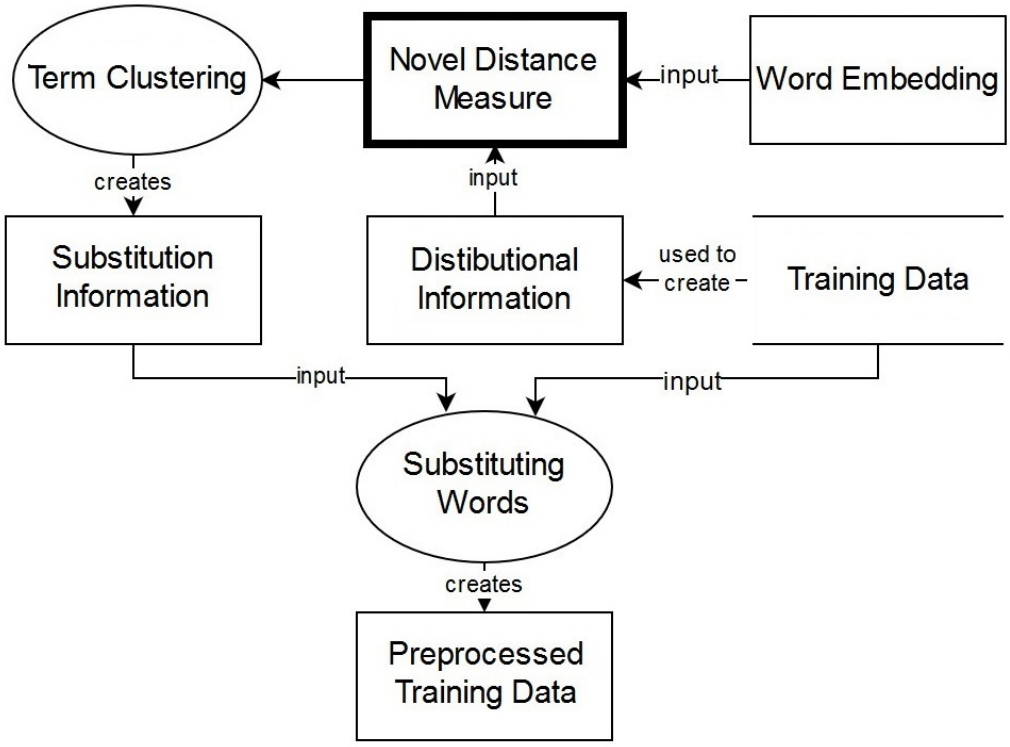

Figure 5.3.: The training data preprocessing method

The training data preprocessing has two main inputs: the training data that we want to preprocess and a word embedding model. The combination of the distributional information extracted from the training data and the semantic information of the word embedding

model let us construct our semantic-distributional distance measure, based on the three additional parameters α, β and K. Our distance measure is defined for each word pair that appears in the training data and the pretrained word embedding vocabulary. A small value stands for a high probability of the two words being task-specific synonyms. Based on this measure we cluster the words and substitute them in the training data with their corresponding cluster resulting the preprocessed training data.

Algorithm 1 Algorithm structure for preprocessing training data

```
1: function PreprocessTrainingData(D, 𝒲, α, β, K)
2:     V := voc(D)
3:     C := ∅
4:     W := pairwise − distances(V, Λ_{α,β,𝒲})
5:     C := complete − linkage − cluster(W, K)
6:     F ← bag − of − clusters(D, C)        ▷ The final feature matrix used for training
7:     return (C, V, F)
8: end function
```

In the pseudo-code, the preprocessing step accepts a dataset D, metric parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and a clustering parameter $\boldsymbol{K}$. As we have already explained, $\boldsymbol{\alpha}, \boldsymbol{\beta}$ regulate how much the semantic information is weighed against the distributional information in the distance measure. After obtaining the vocabulary we calculate the dissimilarity matrix based on our distance measure. Using complete-linkage clustering and parameter K we obtain the K word clusters from the dissimilarity matrix. All clustering are accumulated in C and finally used to represent the document as a bag-of-clusters. The matrix $\mathbf{F}$ is the resulting feature matrix of the type $\mathbb{N}^{|D|\times|C|}$ (number documents × number of clusters) that is later used for training the classifier. Each row of the matrix represents a document and each column the number of occurrences of the members of the corresponding cluster in C.

5.3.2. Preprocessing Test Data

Prior to prediction, unknown documents in the test data $\tilde{d} \in \mathcal{D} \setminus D$ need to mapped to feature vectors $\tilde{\mathbf{F}} \in \mathbb{N}^{|C|}$. Also, they might contain words not known by the classifier $\tilde{v} \in \mathcal{V} \setminus V$. Hence, a mapping procedure to known vocabulary has to be performed. We substitute every word $\tilde{v} \in \mathcal{V} \setminus V$ with the closest word $v \in V$ with respect to $dist_{\mathcal{W}}$. If the closest word is further away than θ_d we discard it. This procedure can be formally described as a preprocessing function τ as follows:

$$\tilde{d} = (\tilde{v}_1, \ldots, \tilde{v}_{|\tilde{d}|}) \in \mathcal{D} \setminus D,$$
$$\boldsymbol{\tau}(\tilde{d}) = (\tau(\tilde{v}_1), \ldots, \tau(\tilde{v}_{|\tilde{d}|})),$$
$$\tau(\tilde{v}) = \begin{cases} \tilde{v}, & \text{if } \tilde{v} \in V \\ v', & \text{if } v' = \arg\min_{v'' \in V} dist(\tilde{v}, v'') \text{ and } dist(v', \tilde{v}) \leq \theta_d \\ \epsilon, & \text{otherwise.} \end{cases}$$

Where ϵ is the *empty word* and simply denotes that a word is removed from the document in case there is no suitable substitute satisfying the constraint given by the distance threshold.

Algorithm 2 illustrates the algorithm for preprocessing unknown documents as described.

Algorithm 2 Preprocessing of unknown documents

```
1:  function PreprocessUnknownDocuments(D̃, V, θ_d, C)
2:      Ṽ := voc(D̃)
3:      Ṽ := Ṽ \ V
4:      NN := nearest − neighbor(Ṽ, V, 𝒲, dist) ⊂ Ṽ × V × ([0, 1] ∪ {∞})
5:      S := {(ṽ, v) | (ṽ, v, d) ∈ NN ∧ d ≤ θ_d}
6:      R := {ṽ | (ṽ, v, d) ∈ NN ∧ d > θ_d}
7:      D̃ ← substitute(D̃, S)
8:      D̃ ← remove(D̃, R)
9:      F̃ := bag − of − clusters(D̃, C)
10:     return F̃
11: end function
```

The algorithms input is a set of unknown documents $\tilde{D}$, distance threshold $\boldsymbol{\theta}_d$ and a clustering C. We then calculate the nearest neighbors of the OOV words $\tilde{V}$. Based on the word embeddings $\mathcal{W}$ and the distance measure $dist_{\mathcal{W}}$, the function $nearest - neighbor$ returns triples of the form (OOV word $\tilde{v}$, nearest known word v, cosine distance between v and $\tilde{v}$). From this set we filter out the substitutions S that satisfy the threshold $\boldsymbol{\theta}_d$ and the ones which do not R. The string functions $substitute$ and $remove$ are used to substitute or remove the words from the document set, respectively. Finally, the BoC representation of the preprocessed unknown document set is computed and returned as feature matrix $\tilde{\mathbf{F}}$.

5.3.3. Training and Prediction

After defining how to preprocess both the training and test data with the proposed method, we show how the preprocessed data can be incorporated in a classification process.

The feature matrix returned by PreprocessTrainingData is used for building a classification model M. Then, each unknown document fetched from the test data $\tilde{D}$. First, an

Algorithm 3 Build classification model and predict class of unknown documents

1: **function** BuildModelAndPredict($D, \mathcal{W}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{K}, \boldsymbol{\theta}_d$)
2: $(C, V, \mathbf{F}) :=$ PreprocessTrainingData($D, \mathcal{W}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{K}$)
3: $M = build - classification - model(\mathbf{F}, L)$
4: **while** Fetch unknown documents from test data $\tilde{D}$ **do**
5: $\tilde{\mathbf{F}} =$ PreprocessUnknownDocuments($\tilde{D}, V, \boldsymbol{\theta}_d, C$)
6: $p = M.predict(\tilde{\mathbf{F}})$
7: **end while**
8: **end function**

unknown document is preprocessed as described in the previous section, then the classifier predicts its label.

5.4. Evaluation Setup

In this section we evaluate our method on different classification tasks. Accuracy is measured for different classifiers trained on unprocessed and preprocessed training data. We show that classifiers trained on our preprocessed data consistently outperform the ones trained on the original datasets.

Datasets. In order to evaluate the effectiveness of the proposed method, we conducted experiments on 4 short text datasets that have been used in experiments of previous research. The datasets are the Customer Reviews (CR), MPQA, Subjectivity (Subj.) and Short Movie Reviews (RT). See Section 2.4.3 for more details.

We subdivide each dataset as follows: The *test sets* consist of 1000 samples held out from each dataset for later testing. To show the effectiveness of our method on different training-set sizes, parameter tuning and classifier training is performed with training sets of varying sizes: 500, 1000, 1500, 8500 for *MPQA*, *Rt10k* and *Subj* and 500, 1000, 1500, 2600 for *CR*. For each size, we sample five training sets randomly, using stratified sampling. For each of these sets, a 10-fold cross-validation is performed to find the best parameter combination, i.e., the combination that yields the highest average accuracy over all folds. The classifier is then trained on the same dataset that is used for the cross-validation and tested on the held-out test set (five times for each sample size and classifier combination).

Classifiers. We use three classifiers for our evaluations. Our goal is to show that our preprocessing method increases the accuracy with various classifiers built on top of the preprocessed training data. To show this, we deploy the same classifiers used in previous studies [118]. We only change the respective training data to our preprocessed version. We use two baseline text classification methods, the Multinomial Naive Bayes (MNB) and the Naive Bayes Support Vector Machine (NBSVM) classifiers. We parameterize them as recommended in [118]. The third classifier we evaluate is presented in [107]. See Section 2.2.3 for details regarding the classifiers.

Word Embedding Model. We use Google's pretrained *Word2Vec* model for the evaluation. It was trained on a part of Google's News dataset, which contains around 100 billion words. The final model consists 3, 000, 000 word vectors of dimensionality 300. [2]

Term Clustering. The term clusters are computed with the built-in agglomerative hierarchical clustering algorithm in Matlab [3]. Its advantage over, say, K-Means is that it allows to operate on a dissimilarity matrix based on any distance function designed by the user.

[2] `https://code.google.com/archive/p/word2vec/`
[3] https://www.mathworks.com/help/stats/cluster.html

In our case, this is the semantic-distributional distance. However, any other algorithm that supports custom distance metrics could have been used (e.g., DBSCAN).

Parameter Search. During parameter tuning, we search on all combinations of the following parameter values,

$$\begin{aligned}\boldsymbol{K} &= \{0, 0.25, 0.5, 0.75, 0.9\}\,,\\ \boldsymbol{\alpha} &= \{0, 0.25, 0.5, 0.75, 1\}\,,\\ \boldsymbol{\beta} &= \{0, 0.1, 0.3, 0.5, 0.7, 1\}\,,\\ \boldsymbol{\theta} &= \{0, 0.5, 2\}\,.\end{aligned}$$

We exclude all combinations with $\beta = 0$ and $\alpha \neq 0$ (see Section 5.2.4.1). Instead of the absolute number of clusters, we use K as a fraction of terms the resulting preprocessed dataset is reduced to. E.g., $K = 0.25$ means that the number of terms are reduced to $0.25 \cdot |V|$, where $|V|$ is the number of terms in the dataset.

5.5. Evaluation Results

Figure 5.4 shows our evaluation results with three different classifiers. The horizontal axis represents the size of the training set. The vertical axis represents the difference between classification accuracy *with* lexical substitution and *without*, i.e., a positive value indicates improvement of our method over unprocessed datasets.[4] The red values (left bar) represent the accuracy difference using only semantic clustering, i.e., not using the distributional information of the training data. The blue values (right bar) correspond to the runs using our novel distance measure. The dots indicate the mean accuracy difference over five runs, the thick error bars stand for the corresponding 25 percentile mark, i.e., the accuracy difference of 3 of 5 runs. The thin lines extend to the minimum/maximum. Note, for brevity, we refer in the following only to the achieved accuracy. This is valid since considering precision, recall, or F-score results in the same conclusions. Nevertheless, we present these numbers in the supplementary material.

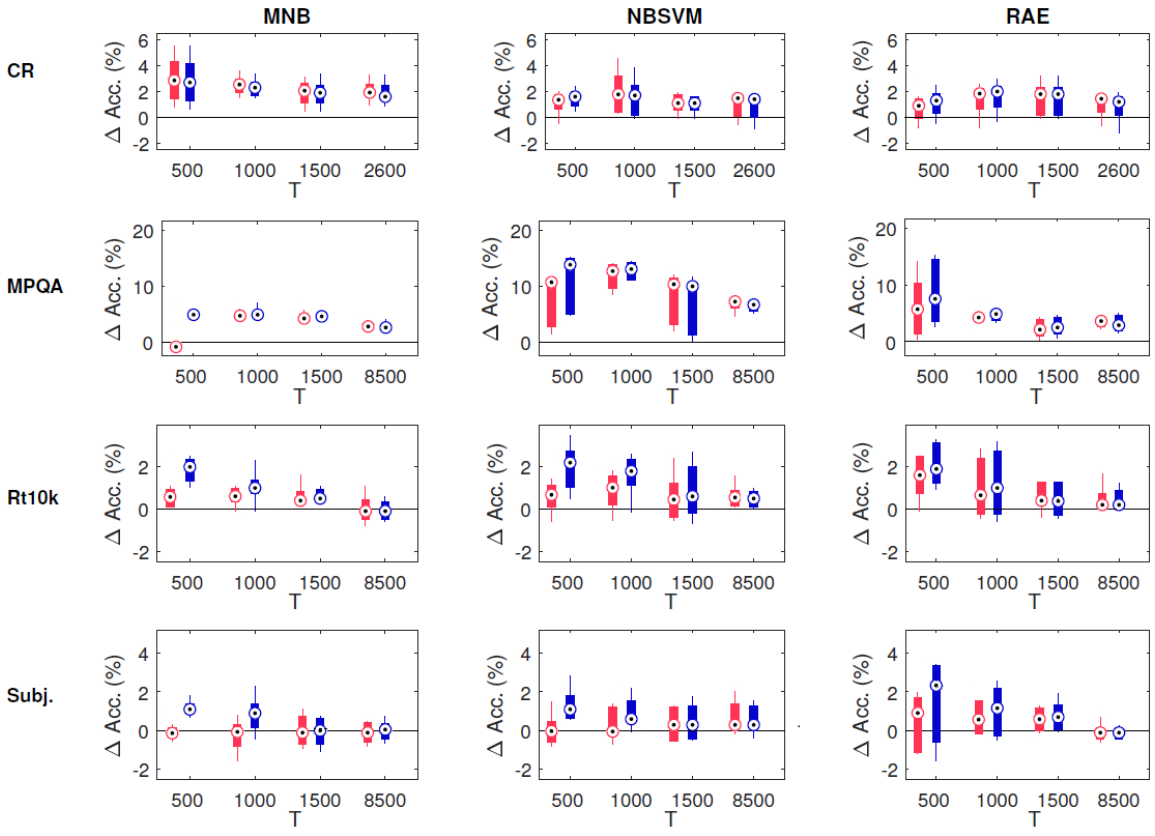

Figure 5.4.: Classification accuracy for each dataset using three different classifiers

We report on the results achieved with the best parameter combination, as described previously. The accuracies used to calculate the difference in performance are the mean accuracies measured over five runs of our algorithm. The algorithm is run with 4 different

[4] Note, that the value ranges are different between the rows.

training sizes. We can observe average improvements in all datasets, except for the *Subj* dataset classified with RAE and the *Rt10k* dataset classified with MNB, both trained with 8500 samples. More precisely, of all 240 scenarios tested[5], in 197 cases (82 %) our method improves classification accuracy, in 5 cases (2 %) no change is observed, and in 38 cases (16 %) the performance slightly deteriorates. In 173 cases (72 %) distributional information improves classification accuracy, in 11 cases (5 %) there is no change, and in 56 cases (23 %) the accuracy is worse. In 46 cases (19 %) distributional information has a higher added value in terms of classification accuracy than semantic information. These cases are mostly ones on the Subjectivity dataset and on smaller training sets.

As expected, the largest improvements are observed in the smaller training sets. The mean classification accuracy increases for *all* tested dataset and classifier combinations when the sample size is at most 1000 samples. This confirms the hypothesis that an exogenous knowledge base can improve classification when there is a lack of training samples. Next, even with more complex classifiers such as the RAE [107], our method could be used to facilitate the training of models which generalize and, hence, perform better.

We also observe higher improvements with growing training sets, e.g., for CR classified with RAE and NBSVM the biggest average quality jump is observed in training sets with 1000 samples. There seems to be an optimal training-set size, so that the external knowledge brought from word embeddings has the most beneficial effects. This is because distributional information becomes robust enough for the substitutions to be most efficient, compared to classification without word clusters.

[5] 4 Datasets $\times$ 4 Training-Set sizes $\times$ 5 Runs $\times$ 3 Classifiers = 240

Part III.

Conclusions

6. Conclusions and Future Work

6.1. Conclusions

The main motivation of this work is to understand different aspects of word embedding models better, which were previously not clear. In order to do this, we have presented a three-way analysis of word embedding models. Before the analysis chapters, in Chapter 2, we have introduced all the necessary background information, notation and previous works, which were necessary for the remainder of the work. This includes an introduction of word embedding models in general, text classification fundamentals, introduction to n-grams and the datasets used throughout the work. Then, we went through the different analysis aspects, which we categorized into algorithmic, theoretical and application groups. We have dedicated a chapter for each group.

First, in the algorithmic chapter (Chapter 3), we have evaluated the effect of the different parameter settings on the similarity values of word embedding models. More precisely, we train models with systematically different parameter settings and compare their similarity value distributions. To generalize our findings, we have extended our scope of models with considering two novel models that embed syllables and sentences, respectively, instead of words. The results show that the same value can represent different grades of similarity in different models, but at the same time almost all distributions have a very similar bell shape. We prove with statistical tests that most of the normalized distributions are almost identical even with the most extreme parameter settings, such as very large dictionaries or small dimensionality.

These first evaluations have showed how parameters affect the similarity values, but not how they influence the quality of the models. In contrast, in the next part of our algorithmic analysis, we have described experiments that allow us to answer which differences in quality one can expect when training word embedding models on fragmented corpora, such as the Google n-gram corpus, compared to full-text. With these experiments we are able to quantify how much fragmentation and different minimum count settings changes the average quality of the respective word embedding models. Using our experiments, we give recommendations on which n-gram versions to use for word embedding model training. We have also answered important research questions, such as 'How sensitive are the models to the fragmentation and the minimum count parameter?' and 'What is the reason for the quality loss of models trained with high fragmentation or a high minimum count parameter?'.

An in-depth evaluation of the results confirmed that one generally can expect good quality for n-grams with $n \geq 3$. In addition, we have showed that the minimum count parameter is highly corpus size dependent and should not be used for corpora with size similar to or smaller than the Wikipedia dump. Finally, our results have showed that the fragmentation and the minimum count parameter introduce different kinds of error. Based on this, we conclude that n-gram corpora such as Google Books are valid training data for word embedding models. In summary, our evaluation results indicate that one can train high-quality embedding models with n-grams if some (mild) prerequisites hold. This is particularly true for the Google n-gram corpus, which is a good corpus to this end.

Although, in the algorithmic analysis chapter we have showed how the similarity value distributions of word embedding models change when trained with different parameter settings, the notion of similarity and the meaning of similarity values remained ambiguous. In the theoretical analysis chapter (Chapter 4), we have studied when exactly such values are meaningful in word embedding models.

Our core finding is that meaningful similarity threshold values exist, and they can be found for each specific word embedding model by calculating similarity-value and similarity-list aggregates. Since these thresholds are not general, they should be calculated for every individual model using the evaluation method we have presented. We have also shown that these insights are corpus-independent.

Based on these results, we propose a new similarity-threshold aware evaluation method of word embedding models, built on top of the baseline method, which does not compare the word pairs during evaluation that fall below the calculated threshold. We have compared the baseline method and our similarity-threshold aware evaluation method with several models on well-known benchmark test sets. We show that our method indeed can affect the evaluation results significantly. We conclude that our method ensures a more reliable comparison of word embedding models, which also helps the design of such models in the future.

In the last, application analysis chapter (Chapter 5), we have presented a new approach to short text classification using a lexical substitution based preprocessing method that employs word embedding models. Since it is a preprocessing method, it is an orthogonal extension of any text classification algorithm. Our method mimics how human annotators preprocess texts: It replaces words unknown to the classifier with known ones and substitutes semantically similar words for statistical robustness, in order to compensate the scarcity of labeled documents. The main contribution of this chapter is the definition of a semantic-distributional word-distance measure. This is the first time when the combination of semantic and distributional information is used in term substitution. Our results show that classification accuracy increases using the preprocessed training data to train the classifiers in every case, but most significantly when labeled data is scarce.

6.2. Future Research Directions

Our algorithmic and theoretical analysis results strengthened our intention to generalize our threshold computation method in the future. We deem such evaluations relevant for any application scenario, where *any* attribute is measured by a score. We hypothesize that such patterns appearing in our evaluations, such as highly different, but when normalized similar distributions or meaningful and meaningless value intervals, appear in various other models as well. We not only aim at generalizations within word embedding models or NLP, but for any field of scientific research. We intend to find use cases where different, not only similarity, scores can be compared in order to find meaningful threshold values. As a matter of fact, we have already published a work [113] with highly similar fundamental ideas, in another, completely unrelated field of computer science, namely trajectory clustering in moving object databases.

A trajectory of a moving object is a sequence of GPS points. Finding similar trajectories is a fundamental task in this field. Classical models face several limitations, most notably scalability. To overcome these limitations, the authors of [70] adopted a similar idea to word embedding models to create similarity preserving embeddings of trajectories. They have named their model the t2vec (trajectory to vector) model.

Our evaluation in [113] investigates similar questions as we have done in this work regarding word embedding models: What do similarity values coming from this new embedding model mean? How do the parameters of the embedding model change the meaning of similarity values? For example, is it possible that in one model two trajectories which are 0.5-similar should be considered similar, and in another model trained with different parameter settings the same value implies dissimilarity?

In order to answer these questions, we have used similar methodology to evaluate the meaningfulness of the deep trajectory similarity values in t2vec, as we have done in this work with word embedding models. This means, we evaluate how different parameter settings affect the similarity values of the t2vec model. We conclude that the t2vec model is robust regarding parameterization, by showing that the similarity-value distributions are fundamentally very similar between models trained with different parameters.

Regarding other research scenarios of ours, presented in this work, the possible future works are more straightforward. As for the question whether it is suitable to train any kind of natural language models on fragmented text, one needs to evaluate their quality compared to full text versions and quantify the differences. Since the Google Books corpus is one of the largest language data publicly available it may be very important for any novel model to be trainable on n-grams, producing results on par or only slightly inferior to the full text version.

Also, since our preprocessing method is an orthogonal extension to virtually any text classification algorithm, one can employ it with any novel method with a good possibility of increasing its accuracy.

Bibliography

[1] Hussain Alkharusi. Categorical variables in regression analysis: A comparison of dummy and effect coding. *International Journal of Education*, 4(2):202, 2012.

[2] Edgar Altszyler, Mariano Sigman, Sidarta Ribeiro, and Diego Fernández Slezak. Comparative study of lsa vs word2vec embeddings in small corpora: a case study in dreams database. *arXiv preprint arXiv:1610.01520*, 2016.

[3] Ehsaneddin Asgari and Mohammad RK Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PloS one*, 10(11), 2015.

[4] Henri Avancini, Andreas Rauber, Fabrizio Sebastiani, and TU Wien. *Organizing digital libraries by automated text categorization.* na, 2004.

[5] Oded Avraham and Yoav Goldberg. Improving reliability of word similarity evaluation by redesigning annotation task and performance measure. *arXiv preprint arXiv:1611.03641*, 2016.

[6] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. Deep learning for hate speech detection in tweets. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 759–760, 2017.

[7] L Douglas Baker and Andrew Kachites McCallum. Distributional clustering of words for text classification. In *ACM SIGIR*. ACM, 1998.

[8] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, 2014.

[9] Miroslav Batchkarov, Thomas Kober, Jeremy Reffin, Julie Weeds, and David Weir. A critique of word similarity as a method for evaluating distributional semantic models. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 7–12. Association for Computational Linguistics, 2016.

[10] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[11] Welch L. Bernard. The generalization ofstudent's' problem when several different population variances are involved. 34:28–35, 1947.

[12] Kenneth J Berry, Paul W Mielke Jr, and Hariharan K Iyer. Factorial designs and dummy coding. *Perceptual and motor skills*, 87(3):919–927, 1998.

[13] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[14] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[15] Tim Bradshaw. Self-driving cars prove to be labour-intensive for humans. `https://www.ft.com/content/36933cfc-620c-11e7-91a7-502f7ee26895`, 07 2017. [Online; accessed 14-Jan-2019].

[16] Elia Bruni, Gemma Boleda, Marco Baroni, and Nam-Khanh Tran. Distributional semantics in technicolor. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 136–145. Association for Computational Linguistics, 2012.

[17] Alexander Budanitsky and Graeme Hirst. Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and Other Lexical Resources*, volume 2, pages 2–2, 2001.

[18] Alexander Budanitsky and Graeme Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1):13–47, 2006.

[19] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

[20] Billy Chiu, Anna Korhonen, and Sampo Pyysalo. Intrinsic evaluation of word vectors fails to predict extrinsic performance. In *Proceedings of the 1st workshop on evaluating vector-space representations for NLP*, pages 1–6, 2016.

[21] Patricia Cohen, Stephen G West, and Leona S Aiken. *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology Press, 2014.

[22] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[23] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537, 2011.

[24] Matthew J Davis. Contrast coding in multiple regression analysis: Strengths, weaknesses, and utility of popular coding structures. *Journal of Data Science*, 8(1):61–73, 2010.

[25] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[27] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1370–1380, 2014.

[28] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 148–155. ACM, 1998.

[29] Susana Eyheramendy, David Lewis, and David Madigan. On the naive bayes model for text categorization. *Artificial Intelligence and Statistics*, 2003.

[30] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414, 2001.

[31] John R Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, 1957.

[32] Johannes Furnkranz. A study using n-gram features for text categorization. *Austrian Research Institute for Artifical Intelligence*, 3(1998):1–10, 1998.

[33] Miller A. George. Wordnet: a lexical database for english. 38:39–41, 1995.

[34] A. Gladkova and A. Drozd. Intrinsic evaluations of word embeddings: What can we do better? In *RepEval*. ACL, 2016.

[35] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[36] Google. Google flu trends. `https://www.google.org/flutrends/about/`, 2017. [Online; accessed 03-June-2017].

[37] Maryam Habibi, Leon Weber, Mariana Neves, David Luis Wiegandt, and Ulf Leser. Deep learning with word embeddings improves biomedical named entity recognition.

Bioinformatics, 33(14):i37–i48, 2017.

[38] William L Hamilton, Jure Leskovec, and Dan Jurafsky. Cultural shift or linguistic drift? comparing two computational measures of semantic change. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing. Conference on Empirical Methods in Natural Language Processing*, volume 2016, page 2116. NIH Public Access, 2016.

[39] William L Hamilton, Jure Leskovec, and Dan Jurafsky. Diachronic word embeddings reveal statistical laws of semantic change. *arXiv preprint arXiv:1605.09096*, 2016.

[40] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[41] Balázs Hidasi, Alexandros Karatzoglou, Oren Sar-Shalom, Sander Dieleman, Bracha Shapira, and Domonkos Tikk. Dlrs 2017: Second workshop on deep learning for recommender systems. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 370–371, 2017.

[42] F. Hill et al. Not all neural embeddings are born equal. *CoRR*, abs/1410.0718, 2014.

[43] F. Hill, R. Reichart, and A. Korhonen. Simlex-999: Evaluating semantic models with genuine similarity estimation. *Computer Linguistics*, 41(4):665–695, December 2015.

[44] Geoffrey E Hinton et al. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA, 1986.

[45] Thomas Hofmann. Probabilistic latent semantic analysis. In *UAI*, 1999.

[46] Thomas Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Machine learning*, 42(1):177–196, 2001.

[47] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[48] Jeremy Howard and Sebastian Ruder. 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)*, volume 1, pages 328–339, 2018.

[49] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics, 2012.

[50] Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. Embeddings for word sense disambiguation: An evaluation study. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 897–907, 2016.

[51] S. Jansen. Word and phrase translation with word2vec. *CoRR*, abs/1705.03127, 2017.

[52] Thorsten Joachims. Text Categorization with Suport Vector Machines: Learning with Many Relevant Features. *ECML*, 1998.

[53] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[54] Jr and J. Massey Frank. The kolmogorov-smirnov test for goodness of fit. 46:68–78, 1951.

[55] Erk Katrin. Vector space models of word meaning and phrase meaning: A survey. 6:635–653, 2012.

[56] Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim, and Sung Hyon Myaeng. Some effective techniques for naive bayes text classification. *IEEE transactions on knowledge and data engineering*, 18(11):1457–1466, 2006.

[57] Yoon Kim, Yi-I Chiu, Kentaro Hanaki, Darshan Hegde, and Slav Petrov. Temporal analysis of language through neural language models. *arXiv preprint arXiv:1405.3515*, 2014.

[58] Benjamin Klein, Guy Lev, Gil Sadeh, and Lior Wolf. Fisher vectors derived from hybrid gaussian-laplacian mixture models for image annotation. *arXiv preprint arXiv:1411.7399*, 2014.

[59] Hiroya Komatsu, Ran Tian, Naoaki Okazaki, and Kentaro Inui. Reducing lexical features in parsing by word embeddings. In *Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation*, pages 106–113, 2015.

[60] Vivek Kulkarni, Rami Al-Rfou, Bryan Perozzi, and Steven Skiena. Statistically significant detection of linguistic change. In *Proceedings of the 24th International Conference on World Wide Web*, pages 625–635, 2015.

[61] Vivek Kulkarni, Bryan Perozzi, and Steven Skiena. Freshman or fresher? quantifying the geographic variation of language in online social media. In *ICWSM*, pages 615–618, 2016.

[62] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pages 957–966, 2015.

[63] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[64] Claudia Leacock and Martin Chodorow. Combining local context and wordnet similarity for word sense identification. *WordNet: An electronic lexical database*, 49(2):265–283, 1998.

[65] R. Lebret and R. Collobert. *Rehabilitation of Count-Based Models for Word Vector Representations*, pages 417–429. Springer, 2015.

[66] Anton Leuski. Evaluating document clustering for interactive information retrieval. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 33–40. ACM, 2001.

[67] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *NIPS*. MIT Press, 2014.

[68] O. Levy, Y. Goldberg, and I. Dagan. Improving distributional similarity with lessons learned from word embeddings. *TACL*, 3, 2015.

[69] David D. Lewis, Claire Nedellec, and Celine Rouveirol. Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval. *Machine Learning: ECML-98*, pages 4—-15, 1998.

[70] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S Jensen, and Wei Wei. Deep representation learning for trajectory similarity computation.

[71] Chu-Cheng Lin, Waleed Ammar, Chris Dyer, and Lori Levin. Unsupervised pos induction with word embeddings. *arXiv preprint arXiv:1503.06760*, 2015.

[72] Shujie Liu, Nan Yang, Mu Li, and Ming Zhou. A recursive recurrent neural network for statistical machine translation. 2014.

[73] Ang Lu, Weiran Wang, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Deep multilingual correlation for improved word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 250–256, 2015.

[74] Minh-Thang Luong, Richard Socher, and Christopher D Manning. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, 2013.

[75] L. Ma and Y. Zhang. Using word2vec to process big text data. In *Proc. Int'l. Conf. on Big Data (Big Data)*, pages 2895–2897. IEEE, 2015.

[76] Carlos Martinez-Ortiz, Tom Kenter, Melvin Wevers, Pim Huijnen, Jaap Verheul, and Joris Van Eijnatten. Design and implementation of shico: Visualising shifting concepts over time. In *HistoInformatics 2016*, volume 1632, pages 11–19, 2016.

[77] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI*, 1998.

[78] Marcelo Mendoza. A new term-weighting scheme for naïve Bayes text categorization. *International Journal of Web Information Systems*, 8(1):55–72, 2012.

[79] Lingling Meng, Runqing Huang, and Junzhong Gu. A review of semantic similarity measures in wordnet. *International Journal of Hybrid Information Technology*, 6(1):1–12, 2013.

[80] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam filtering with naive bayes-which naive bayes? In *CEAS*, volume 17, pages 28–69, 2006.

[81] Jean-Baptiste Michel, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K Gray, Joseph P Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, Jon Orwant, et al. Quantitative analysis of culture using millions of digitized books. *science*, 331(6014):176–182, 2011.

[82] Rada Mihalcea, Courtney Corley, Carlo Strapparava, et al. Corpus-based and knowledge-based measures of text semantic similarity. In *Aaai*, volume 6, pages 775–780, 2006.

[83] T. Mikolov, Q. Le, and I. Sutskever. Exploiting similarities among languages for machine translation. *CoRR*, abs/1309.4168, 2013.

[84] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[85] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[86] Yasuhide Miura, Motoki Taniguchi, Tomoki Taniguchi, and Tomoko Ohkuma. A simple scalable neural networks based model for geolocation prediction in twitter. In *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*, pages 235–239, 2016.

[87] SPFGH Moen and Tapio Salakoski2 Sophia Ananiadou. Distributional semantics resources for biomedical text processing. *Proceedings of LBM*, pages 39–44, 2013.

[88] Nikola Mrkšić, Diarmuid O Séaghdha, Blaise Thomson, Milica Gašić, Lina Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. Counter-fitting word vectors to linguistic constraints. *arXiv preprint arXiv:1603.00892*, 2016.

[89] In Jae Myung. Tutorial on maximum likelihood estimation. *Journal of mathematical Psychology*, 47(1):90–100, 2003.

[90] Bo Pang, Lillian Lee, et al. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135, 2008.

[91] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[92] Alexandre Passos, Vineet Kumar, and Andrew McCallum. Lexicon infused phrase embeddings for named entity resolution. *arXiv preprint arXiv:1404.5367*, 2014.

[93] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[94] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[95] Global Pulse. Global Pulse Projects. `http://www.unglobalpulse.org/projects`, 2017. [Online; accessed 02-June-2017].

[96] Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. A word at a time: computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th international conference on World wide web*, pages 337–346, 2011.

[97] Sandip Ray, Markus Britschgi, Charles Herbert, Yoshiko Takeda-Uchimura, Adam Boxer, Kaj Blennow, Leah F Friedman, Douglas R Galasko, Marek Jutel, Anna Karydas, et al. Classification and prediction of clinical alzheimer's diagnosis based on plasma signaling proteins. *Nature medicine*, 13(11):1359–1362, 2007.

[98] Radim Rehurek and Petr Sojka. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

[99] Antonio Reyes, Paolo Rosso, and Davide Buscaldi. From humor recognition to irony detection: The figurative language of social media. *Data & Knowledge Engineering*, 74:1–12, 2012.

[100] Herbert Rubenstein and John B Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, 1965.

[101] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 298–307, 2015.

[102] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[103] Sungyong Seo, Jing Huang, Hao Yang, and Yan Liu. Representation learning of users and items for review rating prediction using attention-based convolutional neural network. In *International Workshop on Machine Learning Methods for Recommender Systems*, 2017.

[104] Tianze Shi and Zhiyuan Liu. Linking glove with word2vec. *arXiv preprint arXiv:1411.5595*, 2014.

[105] Noam Slonim and Naftali Tishby. The power of word clusters for text classification. In *ECIR*, 2001.

[106] Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1201–1211. Association for Computational Linguistics, 2012.

[107] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the conference on empirical methods in natural language processing*, pages 151–161. Association for Computational Linguistics, 2011.

[108] Clark Stephen. Vector space models of lexical meaning. pages 493–522, 2013.

[109] Bird Steven. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72, 2006.

[110] Mark Stevenson and Yorick Wilks. Word sense disambiguation. *The Oxford Handbook of Comp. Linguistics*, pages 249–265, 2003.

[111] S Subhashree, Rajeev Irny, and P Sreenivasa Kumar. Review of approaches for linked data ontology enrichment. In *International Conference on Distributed Computing and Internet Technology*, pages 27–49. Springer, 2018.

[112] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[113] Saeed Taghizadeh, Abel Elekes, Martin Schäler, and Klemens Böhm. How meaningful are similarities in deep trajectory representations? *Information Systems*, page 101452, 2019.

[114] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1555–1565, 2014.

[115] Bruna Thalenberg. Distinguishing antonyms from synonyms in vector space models of semantics.

[116] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[117] Peng Wang, Bo Xu, Jiaming Xu, Guanhua Tian, Cheng-Lin Liu, and Hongwei Hao. Semantic expansion using word embedding clustering and convolutional neural network for improving short text classification. *Neurocomputing*, 174:806–814, 2016.

[118] Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *ACL*, 2012.

[119] Tao Wang and Bipin C Desai. An approach for text categorization in digital library. In *Database Engineering and Applications Symposium, 2007. IDEAS 2007. 11th International*, pages 21–27. IEEE, 2007.

[120] William Yang Wang and Diyi Yang. That's so annoying!!!: A lexical and frame-semantic embedding based data augmentation approach to automatic categorization of annoying behaviors using# petpeeve tweets. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2557–2563, 2015.

[121] Ian H Witten, Katherine J Don, Michael Dewsnip, and Valentin Tablan. Text mining in a digital library. 2004.

[122] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '99*, pages 42–49, 1999.

[123] Shelemyahu Zacks. *The theory of statistical inference*, volume 34. 1971.

[124] T. Zesch and I. Gurevych. Analysis of the wikipedia category graph for nlp applications. In *NAACL-HLT*, pages 1–8. ACL, 2007.

[125] Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE transactions on Knowledge and Data Engineering*, 18(10):1338–1351, 2006.

[126] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

[127] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

Supplemental Material: Three-Way Analysis for a Better Understanding of Word Embedding Models

Algorithmic Analysis Supplementary Material

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_2_1	0.299	0.444	0.642	0.470	0.225	0.287	0.648	0.589
wiki_3_1	0.426	0.527	0.662	0.557	0.304	0.321	0.600	0.632
wiki_5_1	0.479	0.546	0.671	0.574	0.334	0.301	0.623	0.646
wiki_8_1	0.491	0.514	0.653	0.534	0.311	0.307	0.641	0.636
wiki_f_1	0.489	0.552	0.684	0.607	0.353	0.357	0.627	0.651

Table S1.: Models trained on differently fragmented 1 Billion corpora with $win = 1$.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_3_2	0.404	0.537	0.671	0.567	0.326	0.323	0.589	0.626
wiki_5_2	0.523	0.561	0.657	0.569	0.360	0.297	0.663	0.642
wiki_8_2	0.551	0.554	0.676	0.565	0.358	0.303	0.657	0.641
wiki_f_2	0.523	0.538	0.688	0.608	0.364	0.322	0.633	0.653

Table S2.: Models trained on differently fragmented 1 Billion corpora with $win = 2$.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_3_4	0.521	0.544	0.656	0.587	0.365	0.318	0.645	0.648
wiki_8_4	0.561	0.570	0.688	0.617	0.377	0.299	0.649	0.671
wiki_f_4	0.555	0.509	0.719	0.615	0.377	0.306	0.654	0.666

Table S3.: Models trained on differently fragmented 1 Billion corpora with $win = 4$.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_8_7	0.573	0.547	0.662	0.631	0.355	0.307	0.646	0.663
wiki_f_7	0.581	0.507	0.722	0.634	0.366	0.323	0.644	0.680

Table S4.: Models trained on differently fragmented 1 Billion corpora with $win = 7$.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_2_1_0	0.341	0.449	0.640	0.485	0.224	0.289	0.645	0.608
wiki_2_1_2	0.347	0.431	0.624	0.490	0.196	0.274	0.625	0.596
wiki_2_1_5	0.203	0.370	0.489	0.415	0.128	0.266	0.571	0.537
wiki_2_1_10	0.113	0.213	0.404	0.254	0.090	0.235	0.414	0.422

Table S5.: Models trained on the 2-gram Wikipedia corpora, $win = 1$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_3_1_0	0.422	0.533	0.657	0.557	0.305	0.318	0.595	0.635
wiki_3_1_2	0.412	0.531	0.623	0.552	0.297	0.306	0.588	0.635
wiki_3_1_5	0.312	0.441	0.542	0.439	0.187	0.221	0.501	0.511
wiki_3_1_10	0.258	0.354	0.520	0.437	0.134	0.156	0.448	0.387

Table S6.: Models trained on the 3-gram Wikipedia corpora, $win = 1$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_5_1_0	0.483	0.533	0.652	0.566	0.341	0.293	0.639	0.651
wiki_5_1_2	0.357	0.401	0.567	0.470	0.134	0.201	0.544	0.529
wiki_5_1_5	0.122	0.239	0.410	0.420	0.057	0.141	0.453	0.410
wiki_5_1_10	0.079	0.135	0.369	0.344	0.034	0.087	0.320	0.359

Table S7.: Models trained on the 5-gram Wikipedia corpora, $win = 1$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_8_1_0	0.485	0.510	0.648	0.544	0.320	0.297	0.633	0.640
wiki_8_1_2	0.276	0.280	0.551	0.409	0.116	0.209	0.540	0.492
wiki_8_1_5	0.027	0.091	0.339	0.223	0.034	0.098	0.278	0.349
wiki_8_1_10	0.012	0.030	0.256	0.201	0.030	0.056	0.209	0.207

Table S8.: Models trained on the 8-gram Wikipedia corpora, $win = 1$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_3_2_0	0.413	0.545	0.668	0.554	0.316	0.308	0.605	0.625
wiki_3_2_2	0.407	0.545	0.644	0.561	0.311	0.300	0.594	0.621
wiki_3_2_5	0.330	0.451	0.509	0.425	0.201	0.243	0.521	0.538
wiki_3_2_10	0.269	0.359	0.509	0.402	0.142	0.178	0.421	0.399

Table S9.: Models trained on the 3-gram Wikipedia corpora, $win = 2$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_5_2_0	0.518	0.547	0.668	0.585	0.351	0.300	0.643	0.656
wiki_5_2_2	0.358	0.426	0.581	0.537	0.154	0.237	0.544	0.558
wiki_5_2_5	0.155	0.236	0.331	0.398	0.053	0.148	0.476	0.382
wiki_5_2_10	0.087	0.138	0.337	0.284	0.045	0.087	0.427	0.290

Table S10.: Models trained on the 5-gram Wikipedia corpora, $win = 2$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_8_2_0	0.540	0.551	0.680	0.583	0.352	0.307	0.654	0.653
wiki_8_2_2	0.290	0.309	0.571	0.445	0.134	0.204	0.556	0.519
wiki_8_2_5	0.037	0.084	0.344	0.259	0.023	0.112	0.289	0.341
wiki_8_2_10	0.012	0.030	0.256	0.201	0.023	0.056	0.240	0.199

Table S11.: Models trained on the 8-gram Wikipedia corpora, $win = 2$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_5_4_0	0.544	0.569	0.663	0.589	0.367	0.304	0.638	0.658
wiki_5_4_2	0.379	0.440	0.565	0.537	0.171	0.228	0.525	0.570
wiki_5_4_5	0.167	0.230	0.343	0.420	0.073	0.156	0.469	0.391
wiki_5_4_10	0.102	0.152	0.351	0.277	0.061	0.102	0.429	0.309

Table S12.: Models trained on the 5-gram Wikipedia corpora, $win = 4$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_8_4_0	0.579	0.570	0.697	0.609	0.371	0.307	0.657	0.669
wiki_8_4_2	0.302	0.321	0.579	0.457	0.141	0.197	0.560	0.534
wiki_8_4_5	0.055	0.101	0.339	0.271	0.031	0.134	0.271	0.355
wiki_8_4_10	0.020	0.042	0.281	0.212	0.014	0.067	0.250	0.205

Table S13.: Models trained on the 8-gram Wikipedia corpora, $win = 4$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
wiki_8_7_0	0.573	0.564	0.686	0.609	0.347	0.315	0.648	0.672
wiki_8_7_2	0.276	0.295	0.577	0.425	0.145	0.231	0.556	0.531
wiki_8_7_5	0.023	0.096	0.354	0.277	0.034	0.87	0.299	0.329
wiki_8_7_10	0.012	0.042	0.260	0.189	0.012	0.029	0.261	0.189

Table S14.: Models trained on the 8-gram Wikipedia corpora, $win = 7$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
1b_2_1_0	0.302	0.435	0.623	0.490	0.241	0.299	0.638	0.612
1b_2_1_2	0.302	0.420	0.618	0.469	0.207	0.281	0.611	0.601
1b_2_1_5	0.212	0.376	0.474	0.398	0.128	0.266	0.571	0.537
1b_2_1_10	0.113	0.213	0.404	0.254	0.101	0.242	0.407	0.434

Table S15.: Models trained on the 2-gram 1 Billion corpora, $win = 1$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
1b_3_1_0	0.425	0.523	0.634	0.561	0.312	0.309	0.601	0.632
1b_3_1_2	0.416	0.511	0.623	0.540	0.301	0.291	0.574	0.620
1b_3_1_5	0.299	0.420	0.534	0.451	0.175	0.219	0.496	0.517
1b_3_1_10	0.245	0.348	0.509	0.444	0.144	0.161	0.433	0.390

Table S16.: Models trained on the 3-gram 1 Billion corpora, $win = 1$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
1b_5_1_0	0.473	0.531	0.640	0.556	0.340	0.267	0.623	0.636
1b_5_1_2	0.345	0.405	0.558	0.476	0.122	0.199	0.539	0.536
1b_5_1_5	0.108	0.232	0.416	0.415	0.071	0.134	0.467	0.421
1b_5_1_10	0.061	0.144	0.372	0.355	0.042	0.079	0.307	0.373

Table S17.: Models trained on the 5-gram 1 Billion corpora, $win = 1$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
1b_8_1_0	0.466	0.518	0.661	0.539	0.337	0.303	0.621	0.627
1b_8_1_2	0.267	0.270	0.556	0.400	0.123	0.221	0.531	0.481
1b_8_1_5	0.035	0.082	0.351	0.210	0.036	0.099	0.269	0.335
1b_8_1_10	0.019	0.022	0.244	0.184	0.030	0.067	0.202	0.193

Table S18.: Models trained on the 8-gram 1 Billion corpora, $win = 1$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
1b_3_2_0	0.413	0.524	0.659	0.557	0.316	0.307	0.612	0.638
1b_3_2_2	0.405	0.534	0.619	0.537	0.311	0.300	0.594	0.621
1b_3_2_5	0.302	0.466	0.506	0.439	0.214	0.227	0.525	0.544
1b_3_2_10	0.255	0.354	0.481	0.389	0.156	0.188	0.421	0.402

Table S19.: Models trained on the 3-gram 1 Billion corpora, $win = 2$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
1b_5_2_0	0.499	0.516	0.651	0.586	0.371	0.321	0.609	0.636
1b_5_2_2	0.334	0.408	0.591	0.506	0.161	0.250	0.516	0.533
1b_5_2_5	0.117	0.209	0.340	0.396	0.077	0.168	0.435	0.351
1b_5_2_10	0.070	0.123	0.303	0.291	0.055	0.084	0.402	0.301

Table S20.: Models trained on the 5-gram 1 Billion corpora, $win = 2$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
1b_8_2_0	0.530	0.555	0.647	0.549	0.362	0.311	0.638	0.640
1b_8_2_2	0.278	0.312	0.575	0.437	0.137	0.189	0.523	0.508
1b_8_2_5	0.032	0.069	0.323	0.238	0.023	0.102	0.269	0.338
1b_8_2_10	0.008	0.035	0.243	0.188	0.023	0.067	0.225	0.167

Table S21.: Models trained on the 8-gram 1 Billion corpora, $win = 2$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
1b_5_4_0	0.532	0.572	0.654	0.571	0.371	0.298	0.623	0.648
1b_5_4_2	0.367	0.422	0.569	0.541	0.174	0.232	0.509	0.566
1b_5_4_5	0.171	0.217	0.327	0.411	0.080	0.161	0.454	0.393
1b_5_4_10	0.106	0.145	0.356	0.270	0.067	0.98	0.431	0.300

Table S22.: Models trained on the 5-gram 1 Billion corpora, $win = 4$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
1b_8_4_0	0.581	0.571	0.687	0.602	0.372	0.300	0.641	0.643
1b_8_4_2	0.308	0.298	0.542	0.455	0.146	0.191	0.565	0.529
1b_8_4_5	0.057	0.089	0.331	0.251	0.048	0.135	0.262	0.371
1b_8_4_10	0.025	0.032	0.267	0.234	0.008	0.054	0.234	0.194

Table S23.: Models trained on the 8-gram 1 Billion corpora, $win = 4$, with different minimum count parameter.

model	google	msr	rg-65	ws353	rare	simlex	mturk	men
1b_8_7_0	0.574	0.543	0.674	0.614	0.344	0.324	0.652	0.657
1b_8_7_2	0.277	0.259	0.534	0.409	0.161	0.224	0.544	0.541
1b_8_7_5	0.023	0.071	0.321	0.236	0.044	0.68	0.275	0.306
1b_8_7_10	0.016	0.036	0.234	0.149	0.008	0.034	0.251	0.180

Table S24.: Models trained on the 8-gram 1 Billion corpora, $win = 7$, with different minimum count parameter.

Application Analysis Supplementary Material

Dataset		Accuracy		Precision		Recall		F-score		Δ Acc.	Δ Prec.	Δ Recall	Δ F
		Orig.	Proc.	Orig.	Proc.	Orig.	Proc.	Orig.	Proc.				
CR	500	71.9	74.3	77.2	74.0	31.8	44.8	45.0	55.8	**2.41**	-3.21	**12.98**	**10.76**
CR	1000	75.8	77.9	74.8	74.7	49.2	58.8	59.3	65.8	**2.10**	**-0.05**	**9.67**	**6.51**
CR	1500	76.5	78.2	70.0	72.9	56.6	63.3	62.6	67.8	**1.77**	**2.96**	**6.63**	**5.16**
CR	2600	79.1	81.0	74.9	73.7	63.5	66.6	68.8	70.0	**1.92**	-1.22	**3.04**	**1.20**
MPQA	500	73.7	77.8	75.7	82.4	93.3	87.7	83.6	85.0	**4.12**	**6.76**	-5.59	**1.43**
MPQA	1000	76.7	80.4	78.0	83.8	93.9	90.1	85.2	86.8	**3.74**	**5.72**	-3.77	**1.59**
MPQA	1500	78.4	82.4	80.3	84.4	90.6	92.5	85.2	88.3	**4.01**	**4.12**	**1.82**	**3.10**
MPQA	8500	86.0	88.8	88.4	91.6	92.6	92.9	90.5	92.2	**2.80**	**3.20**	**0.28**	**1.78**
Rt10k	500	65.5	67.4	66.6	67.2	59.9	60.1	64.0	64.5	**1.95**	**0.63**	**0.20**	**0.52**
Rt10k	1000	66.8	68.3	70.3	70.5	60.7	65.4	65.1	67.8	**1.51**	**0.17**	**4.70**	**2.69**
Rt10k	1500	69.7	70.5	72.0	73.1	66.5	70.1	69.2	71.5	**0.89**	**1.03**	**3.52**	**2.35**
Rt10k	8500	75.7	75.6	77.5	77.6	74.0	73.6	75.7	75.7	-0.09	**0.14**	-0.29	-0.07
Subj.	500	85.7	86.1	86.5	87.8	84.9	84.5	85.7	86.1	**1.31**	**1.35**	-0.39	**0.46**
Subj.	1000	86.9	88.0	90.0	90.0	85.9	86.1	87.9	88.0	**1.14**	**0.02**	**0.20**	**0.11**
Subj.	1500	89.8	89.7	92.2	91.3	87.5	88.3	89.8	89.8	-0.16	-0.87	**0.78**	-0.01
Subj.	8500	92.2	92.2	94.1	94.2	89.8	90.0	91.9	92.0	0.00	**0.10**	**0.09**	**0.09**

Table S25.: Results on the sampled datasets using the Multinomial Naive Bayes classifier.

Dataset		Accuracy		Precision		Recall		F-score		Δ Acc.	Δ Prec.	Δ Recall	Δ F
		Orig.	Proc.	Orig.	Proc.	Orig.	Proc.	Orig.	Proc.				
CR	500	73.7	75.9	70.2	69.8	47.5	58.8	56.7	63.9	**2.22**	-0.37	**11.33**	**7.20**
CR	1000	76.0	77.9	70.9	71.7	57.2	63.0	63.3	67.1	**1.92**	**0.81**	**5.80**	**3.76**
CR	1500	77.6	78.7	71.3	70.8	63.8	70.2	67.3	70.5	**1.17**	-0.54	**6.35**	**3.11**
CR	2600	78.9	80.8	72.7	73.6	66.9	73.2	69.6	73.4	**1.90**	**0.94**	**6.35**	**3.77**
MPQA	500	61.9	76.9	80.6	85.2	61.6	82.0	69.8	83.6	**15.08**	**4.57**	**20.39**	**13.73**
MPQA	1000	68.3	80.4	84.1	87.8	68.7	84.4	75.6	86.0	**12.11**	**3.69**	**15.64**	**10.41**
MPQA	1500	72.5	81.2	87.1	88.8	72.3	84.4	79.0	86.5	**8.73**	**1.77**	**12.01**	**7.51**
MPQA	8500	81.8	87.7	92.5	93.7	84.2	88.8	88.2	91.2	**5.93**	**1.18**	**4.61**	**3.03**
Rt10k	500	62.0	64.6	64.3	67.9	57.5	59.5	60.7	63.2	**2.67**	**3.39**	**1.96**	**2.42**
Rt10k	1000	65.9	68.0	67.9	69.8	63.2	65.9	65.5	67.8	**2.11**	**1.92**	**2.74**	**2.36**
Rt10k	1500	70.4	71.5	70.5	72.2	68.9	71.8	69.7	72.0	**1.14**	**1.70**	**2.94**	**2.33**
Rt10k	8500	77.4	77.8	77.9	78.8	74.8	74.8	77.2	76.7	**0.41**	**0.99**	0.00	**0.46**
Subj.	500	83.6	85.5	84.8	86.7	82.8	84.5	83.8	85.6	**1.95**	**1.98**	**1.76**	**1.87**
Subj.	1000	85.3	86.9	87.0	88.2	83.8	85.9	85.3	87.0	**1.65**	**1.16**	**2.15**	**1.67**
Subj.	1500	87.7	88.1	89.0	89.1	86.2	86.5	87.2	87.4	**0.24**	**0.23**	**0.28**	**0.21**
Subj.	8500	91.6	91.6	93.3	92.3	90.0	91.2	91.6	91.7	**0.07**	-1.03	**1.17**	**0.10**

Table S26.: Results on the sampled datasets using the Naive Bayes Support Vector Machine classifier.

Dataset		Accuracy		Precision		Recall		F-score		Δ Acc.	Δ Prec.	Δ Recall	Δ F
		Orig.	Proc.	Orig.	Proc.	Orig.	Proc.	Orig.	Proc.				
CR	500	74,1	75,8	77,2	74,0	40,8	42,8	48,4	49,2	**1,71**	-3,21	**2,06**	**0,81**
CR	1000	77,7	79,7	74,8	74,7	50,2	51,3	59,9	60,4	**1,99**	-0,05	**1,06**	**0,50**
CR	1500	79,4	81,3	70,0	72,9	61,8	65,3	65,4	68,2	**1,87**	**2,96**	**3,52**	**2,85**
CR	2600	79,1	80,6	74,9	73,7	69,9	72,8	72,7	73,8	**1,53**	-1,22	**2,91**	**1,13**
MPQA	500	73,7	77,5	75,7	78,4	92,7	94,1	84,7	87,2	**3,79**	**2,76**	**1,39**	**2,44**
MPQA	1000	78,3	83,4	78,0	81,8	91,6	94,7	84,7	89,9	**3,08**	**3,72**	**3,07**	**5,16**
MPQA	1500	80,4	83,2	80,3	83,4	92,6	96,5	87,8	91,1	**2,77**	**3,12**	**3,88**	**3,30**
MPQA	8500	84,5	87,9	84,4	87,6	90,8	93,0	89,4	91,8	**3,42**	**3,20**	**2,22**	**2,32**
Rt10k	500	65,1	67,1	68,6	67,2	63,4	69,0	66,0	68,4	**1,97**	-1,43	**5,63**	**2,44**
Rt10k	1000	70,2	71,4	70,3	70,5	59,4	63,0	64,8	66,6	**1,2**	**0,17**	**3,58**	**1,86**
Rt10k	1500	69,1	69,5	72,0	73,1	65,8	65,2	69,0	69,4	**0,38**	**1,03**	-0,59	**0,41**
Rt10k	8500	79,1	79,2	77,5	77,6	82,3	82,1	79,9	80,1	**0,1**	**0,14**	-0,19	**0,16**
Subj.	500	87,1	89,3	86,5	87,8	95,3	95,6	90,2	90,8	**2,15**	**1,35**	**0,36**	**0,69**
Subj.	1000	88,7	90,1	90,0	90,0	92,9	97,9	91,4	93,3	**1,37**	**0,02**	**5,03**	**1,93**
Subj.	1500	90,3	91,1	92,2	91,3	90,6	93,7	91,6	92,3	**0,76**	-0,87	**3,11**	**0,65**
Subj.	8500	95,7	95,7	94,1	94,9	93,0	93,0	93,8	93,8	-0,04	**0,80**	-0,01	-0,01

Table S27.: Results on the sampled datasets using the Recursive Auto-Encoder classifier.