Interaktive Diagnose von verrauschten Inspektionsdaten durch subsymbolische Lernverfahren

Robert Suna

Juli 2000

Robert Suna
suna@suna.de

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
von der Fakultät für Informatik
der Universität Karlsruhe (Technische Hochschule)
genehmigte Dissertation

Erster Gutachter:	Prof. Dr. R. Dillmann
Zweiter Gutachter:	Prof. Dr. H. Steusloff

Tag der mündlichen Prüfung: 21.07.2000

Vorwort

Die vorliegende Arbeit entstand am Forschungszentrum Informatik an der Universität Karlsruhe in der Gruppe Interaktive Diagnose- und Servicesysteme bei Herr Prof. Dr.-Ing. R. Dillmann.

Ich möchte mich zunächst bei Herr Prof. Dr.-Ing. R. Dillmann für den gewährten wissenschaftlichen Freiraum in der Forschung bedanken als auch für die Unterstützung, die er mir insbesondere in der Endphase zukommen ließ und zum positiven Gelingen meiner Arbeit beitrug. Herrn Prof. Dr.rer.nat. H. Steusloff möchte ich für das ausgeprägte Interesse an meiner Arbeit danken sowie für seine Anregungen auf dem Gebiet der Diagnosesysteme.

Bei der Bewältigung von Industrieprojekten bleibt es manchmal nicht aus, daß die Forschungsarbeit in den Hintergrund tritt. Durch fachliche Diskussionen und immerwehrende Unterstützung hat mir der Abtelungsleiter der Gruppe Interaktive Diagnose- und Servicesysteme Dr.rer.nat. K. Berns durch einige Täler mit hindurchgeholfen.

An dieser Stelle möchte ich mich auch bei meinen Kolegen Dipl.-Inform. M. Eberl, Dipl.-Inform. W. Ilg, Dipl.-Inform. P. Feucht, Dipl.-Inform. M. Deck, Dipl.-Inform. K. U. Scholl, Dipl.-Ing. V. Kepplin und Dipl.-Inform. J. M. Zöllner für die fruchtbaren Disskusionen und die gute Zusammenarbeit bedanken.

Viele Studenten halfen mir durch ihre Diplom- und Studienarbeiten. Durch sie war ich gefordert, Zusammhänge zu erklären, Problemstellungen abzugrenzen und Lösungsansätze zu überdenken. Insbesondere bedanken möchte ich mich für diese Zusammenarbeit bei Dipl.-Inform. P. Eisenhauer, Dipl.-Inform. J. Gizzi, Dipl.-Inform. K. Heimannsfeld und cand.-Inform. K. Klier.

Für die langjährige gute Zusammenarbeit danke ich Herr Dr.-Ing. K. Germerdong und Dr.rer.nat K. Reber von der Firma PII, die durch ihre Unterstützung den Einsatz des in dieser Arbeit entwickelten interaktiven Diagnosesystems in der Produktion bei PII ermöglichten.

Einen abschliesenden Dank möchte ich an meine Mutter Dr.Med. J. Sunova richten, die mir mein Studium ermöglicht hat und nicht zuletzt an meine Gefährtin und Frau Dipl.-Inform. B. C. Kopp für ihre Unterstützung und den gewährten Rückhalt beim Durchschreiten mancher Täler.

Karlsruhe, im Juli 2000 Robert Suna

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Anwendungsbeispiel: Pipeline–Inspektion . . . 2
- 1.2 Probleme beim Aufbau eines fallbasierten Diagnosesystems . . . 4
- 1.3 Ziele und Beitrag der Arbeit . . . 5
- 1.4 Gliederung der Arbeit . . . 6

2 Architektur subsymbolischer Diagnosesysteme **9**
- 2.1 Grundlegendes über Diagnosesysteme . . . 9
  - 2.1.1 Lebenszyklus eines Diagnosesystems . . . 11
  - 2.1.2 Anforderungen an den Diagnosesystemkern . . . 12
  - 2.1.3 Wissensakquisition . . . 15
  - 2.1.4 Anwendung . . . 17
- 2.2 Fehlerquellen bei Diagnosesystemen . . . 18
- 2.3 Beispiele eingesetzter Diagnosesysteme . . . 19
  - 2.3.1 Motor-Diagnose . . . 20
  - 2.3.2 Getriebe-Diagnose . . . 20
  - 2.3.3 ZN-Face . . . 20
  - 2.3.4 ZN-LISA . . . 21
  - 2.3.5 Qualitätskontrolle von Kassettenlaufwerken . . . 21
  - 2.3.6 DIAMOND . . . 22
- 2.4 Schlußfolgerungen für Interaktive Diagnosesysteme . . . 22

3 Architekturansatz für ein lernendes Diagnosesystem **25**
- 3.1 Motivation eines Interaktiv Lernenden Diagnosesystems . . . 25
- 3.2 Profil der zur ILD–Diagnostik geeigneten Eingabedaten . . . 27
- 3.3 Anforderungen an die Architektur eines Interaktiv Lernenden Diagnosesystems . . . 27
- 3.4 Komponenten eines Interaktiv Lernenden Diagnosesystems . . . 29
  - 3.4.1 Vorverarbeitung der Rohdaten . . . 30
  - 3.4.2 Lernkomponente . . . 31
  - 3.4.3 Wissensverwaltung . . . 33
  - 3.4.4 Monitorkomponente . . . 33
  - 3.4.5 Interaktionskomponente . . . 33
- 3.5 Resümee der ILD – Architektur . . . 34

4 Prinzipien der Vorverarbeitung verrauschter mehrdimensionaler Inspektionsdaten **37**
- 4.1 Applikationsbeispiel: Ultraschallbasierte Inspektion von Pipelines . . . 38

4.2 Charakterisierung der Meßdaten und der Applikation 40
4.3 Formatierung der Meßdaten 41
4.4 Vorverarbeitung der Meßdaten 43
4.5 Bestimmung von Meßdatenregionen 44
4.6 Extraktion von Merkmalen 47
4.6.1 Beispiele für Merkmale 49
4.6.2 Einlernen von Merkmalen 52
4.7 Ermittlung der Charakteristik der Merkmale 52
4.7.1 Fraktale-Dimension 53
4.7.2 Informationstheoretische Betrachtung 55
4.8 Merkmalsselektion 56
4.8.1 Hauptkomponentenanalyse 57
4.8.2 Mutual Information Feature Selection (MIFS) 57
4.8.3 Maximum Mutual Information Projection Feature Extractor - (MMIP) 58
4.8.4 Separated Mutual Information Feature Extractor (SMIFE) . . . 58
4.8.5 Vergleich der Merkmalsselektionsverfahren 59
4.9 Resümee 59

5 Lernkomponente 61
5.1 Auswahl der Lernverfahren 62
5.1.1 Lokale Lernverfahren 63
5.1.2 Globale Lernverfahren 64
5.1.3 Hierarchische Ansätze 65
5.1.4 Beurteilung der Lernarchitekturen 67
5.2 Architektur der Lernkomponente 68
5.3 Lernen mit dem *Dynamische–Regionen*-Algorithmus 69
5.3.1 Die Idee des *Dynamische–Regionen*-Algorithmus 70
5.3.2 Propagierung 70
5.3.3 Lernen 71
5.3.4 Integration von Regelwissen 79
5.3.5 Aufwandsabschätzung 79
5.3.6 Bewertung 80
5.4 Synergese durch *Hybride Knoten* 83
5.4.1 Heuristik für Randregionen 86
5.4.2 Hybride Lernarchitekturen 88
5.4.3 Phasenlernen mit *Hybriden Knoten* 90
5.4.4 Evaluierung 91
5.5 Resümee 92

6 Interaktion zum Überwachen und Einlernen der Lernkomponente 95
6.1 Anforderung an die Interaktionskomponenten 96
6.2 Konzept der Interaktion 97
6.3 Interaktive Datenvisualisierung 99
6.4 Interaktive Akquisition von Anomalieklassen 102
6.4.1 Integration von Regelwissen 102
6.4.2 Interaktive Akquisition von Defektklassen 103
6.4.3 Interaktive Suche nach Anomaliesonderfällen 105

6.4.4 Automatisches Sammeln von Beispielsdaten 107
6.5 Verwaltung von Beispielswissen 109
6.6 Monitorkomponente 112
6.7 Resümee 113

7 Anwendung der Interaktiven Diagnose 115
7.1 Ultraschallbasierte Diagnose von Pipelines mit ***NeuroPipe I*** 115
7.2 Interaktive Diagnose von Längsnähten 120
7.3 Magnetstreuflußbasierte Diagnose von Pipelines mit ***NeuroPipe II*** . . 122
7.4 Resümee 124

8 Zusammenfassung und Ausblick 127

Literaturverzeichnis 130

A Entwicklungssystem *CMS* 139
A.1 Das Hybridkonzept 140
A.1.1 Die hybride Lernarchitektur 140
A.1.2 Die einzelnen Komponenten des Hybridobjekts 140
A.1.3 Steuerungskonzept für hybride Lernarchitekturen 145
A.2 Das Konzept der Neuronalen Netze 150

B *CMS*-Datenbank 153

C Generische graphische Schnittstelle *Iface* 157
C.1 Kurze Einführung in ***Iface*** 157
C.2 Herkömmliche Verfahren der Anwendungsanbindung 159
C.3 Architektur von ***Iface*** 160
C.3.1 Modellierung der Oberfläche 160
C.3.2 Entwurfskonzept unter ***Iface*** 164
C.3.3 Veränderung der Darstellung in ***Iface*** 164
C.4 Konfigurationseditor 168
C.4.1 Ziele eines Oberflächeneditors unter ***Iface*** 169
C.4.2 Veränderbarkeit der Oberfläche 170
C.4.3 Plazierung und Konfiguration von ***Iface***-Widgets 170
C.4.4 Einstellung der Hilfeoptionen 170
C.4.5 Vorteile des Editors als Oberflächengestalter 170
C.5 Makroverarbeitung 171
C.5.1 Voraussetzung für eine Makroverarbeitung unter ***Iface*** 171
C.5.2 Beeinflussung des Makroablaufs 173
C.6 Hilfesystem 173
C.6.1 Layout des Hilfesystems 174
C.6.2 Vorteile des ***Iface***-Hilfesystems 174

D Klassifikatorperformanz 177
D.1 Fehler 177
D.2 Kosten und Risiken 178
D.3 Repräsentative Beispiele 180
D.4 Schätzung des wahren Fehlers 181

D.4.1 Resampling 184
D.4.2 Vergleich der Schätzverfahren 185

E Datensätze **187**
E.1 defects1 187
E.2 lweld 187

Kapitel 1

Einleitung

Das Leben eines Menschen ist davon geprägt, Vorgänge und Ereignisse der Umwelt in Zusammenhang zu bringen und einzuordnen. Schon Säuglinge versuchen ihre Umwelt zu begreifen und dadurch verschiedene Erfahrungen in einem Kontext zu erfassen. Dazu gehört auch die Zuordnung von Eigenschaften zu bestimmten Gegenständen oder Erfahrungen, so wie z.B. ob Schokolade süß ist, oder es als angenehm empfunden wird, in den Schlaf gesungen zu werden. Diese Einordnung der Lebenseindrücke in Zusammenhänge, wobei die Eigenschaften der Eindrücke ebenfalls berücksichtigt werden, bildet Assoziationen. Bereits Aristoteles (350 v. Chr.) kannte diese Form der Bildung von Assoziationen.

Die einfachste Art der Bildung von Assoziationen wird in der Lernpsychologie als Konditionierung bezeichnet (vgl. [Ede93, Sch97]). Dabei werden die Reize mit den zu erwartenden Folgen verknüpft[1]. Diesen Prozeß kann man auch als eine Form der Klassifikation auffassen. Aber der Mensch klassifiziert diese Eindrücke nicht nur, sondern er benutzt die Erfahrungen, um in neuen Situationen unbekannte Eindrücke anhand bereits bekannter Eigenschaften besser einschätzen zu können und Verhaltensmuster zu erwägen. Dies stellt eine erste Form der Diagnostik dar. Allerdings kann aufgrund zu vieler Informationen über diverse Eigenschaften und den dazugehörigen Eindrücken, die Fähigkeit, eine Situation oder einen Sachverhalt richtig einzuschätzen, erschwert oder verhindert werden.

Insbesondere die Überwachung und Interpretation großer Informationsmengen bei technischen Applikationen stellt hohe Anforderungen an die Konzentration und Ausdauer des Menschen die bei Nichterfüllung zu gravierenden Fehleinschätzungen führen können. Aus diesem Grund ist es wünschenswert, für diese Überwachungs- bzw. Diagnoseaufgaben eine automatische Unterstützung zu erhalten. Computer können dabei unterstützend wirken, indem sie dem Benutzer, Software-Werkzeuge zur Verfügung stellen, die diese Arbeit erleichtern, oder sogar komplexere Systeme bereitstellen, die einen Großteil der Diagnoseaufgabe selbstständig durchführen können.

Die heutigen Rechner werden von Seiten des Benutzers zunehmend als BlackBox empfunden deren genaue Arbeitsweise bzw. Funktionsweise unbekannt ist, lediglich Arbeitskomponenten sind erkennbar. Bei der Diagnose läßt sich diese BlackBox grob in zwei Teile untergliedern: in eine Automatisierungskomponente, die den Ablauf steuert bzw. Zustandskenngrößen über die Applikation zur Verfügung stellt, und in eine Systemüberwachungskomponente. Letztere ist für das Einlesen und Aufbereiten,

[1] Anschaulich wird die Konditionierung in dem bekannten Hundeversuch des russischen Psychologen Iwan Petrowich Pawlow demonstriert.

die Visualisierung und Protokollierung sowie für die Kontrolle der Systemparameter zuständig.

Durch die zunehmende Komplexität der zu überwachenden Systeme steigt auch die Komplexität und die Anzahl der zu überwachenden Parameter. Deshalb wird in jüngster Zeit in zunehmendem Maße versucht, die durch die Systemüberwachung gesammelten Daten automatisch auszuwerten, um so möglichst schnell und zuverlässig aus der großen Anzahl von Parametern, eine Aussage über den Zustand des überwachten Systems zu erhalten.

Diese Art der Systemüberwachung mit gleichzeitiger Bewertung des Systemzustands wird im nachfolgenden als "Diagnosesystem" bezeichnet (vgl. [FPB96]). Um eine solche Diagnose durchführen zu können, ist es unerläßlich, das Wissen über die zu diagnostizierende Applikation aufzubereiten und dem Diagnosesystem zugänglich zu machen. Das Problem besteht darin, das vorhandene Modellwissen aus den verschiedenen Quellen, wie beispielsweise den physikalischen Zusammenhängen, Konstruktionsplänen und Wirkungszusammenhängen, in Regeln und Algorithmen umzusetzen, damit sie dem Computer zugänglich gemacht werden können. Während die aufgeführten Wissensquellen Modelle bieten, die es in eine computergerechte Repräsentation zu konvertieren gilt, ist dagegen das fallbasierte Wissen der Experten nur schwer erfaßbar. Es lassen sich gewöhnlich keine oder nur sehr rudimentäre Modelle dieses Wissen erstellen, d.h. es können widersprüchliche Aussagen koexistieren, und die Aussagenmenge selbst ist unvollständig.

Es ist aber unerläßlich, gerade bei Applikationen die nur durch wenige oder überhaupt keine Regeln beschrieben werden können, eine rechnergestützte Diagnose durchzuführen, um auf diese Weise eine gleichbleibende Qualität und Nachvollziehbarkeit der Diagnosen zu gewährleisten.

Um einen kurzen Einblick in die Komplexität und die Schwierigkeiten beim Entwurf solcher Diagnosesysteme zu geben, wird nachfolgend ein Beispiel aus der Industriepraxis angeführt.

1.1 Anwendungsbeispiel: Pipeline–Inspektion

In der Erdölindustrie stellt die Wartung und Instandhaltung der Gas- und Öl-Pipelines ein besonderes Problem dar. Dabei geht es insbesondere darum, festzustellen, ob sich die Pipelines in einem ordnungsgemäßen Zustand befinden, so daß weder Korrosion noch Materialschäden zur Zerstörung der Rohre oder zu Umweltschäden führen.

Mit sogenannten Molchen (siehe Abb. 1.1) werden mittels Ultraschall- oder Magnetstreuflußmeßverfahren die Beschaffenheit der Rohrwände von Gas- und Öl-Pipelines gemessen.

Die Auflösung der Sensoren variiert mit den unterschiedlichen Molchtypen. Bei einem auf Ultraschallmeßverfahren basierendem Molch[2] (siehe Abbildung 4.2) wird eine Messung pro 24mm^2 der Rohrinnenwand durchgeführt. Dabei werden pro Messung zwei Werte, nämlich die Laufzeit für das erste und zweite Echo mit jeweils einem Byte kodiert, aufgezeichnet. Es werden mit bis zu 512-Ultraschallsensoren bestückte Molche dieses Typs verwendet, die mit einer Frequenz von 400kHz feuern. Damit ergibt sich eine Auflösung von ca. 0.225mm für die Wanddicke. Die Messungen werden in einem durchschnittlichen Abstand von 3mm in längsrichtung der Pipeline durchgeführt. Dies

[2]Der UltraScan–Molch der Firma Pipetronix GmbH (PTX).

Abbildung 1.1: Ein Molch des UltraScan–Typs beim Einschleusen in eine Ölpipeline.

ergibt für eine 100km lange Pipeline mit einem Durchmesser von 28Zoll = 71cm ein Datenvolumen von ca. 20GByte.

All diese Daten müssen von Auswertern manuell gesichtet werden. Zu diesem Zweck werden sie in eine C-Scan[3] Darstellung konvertiert und dem Benutzer am Bildschirm präsentiert. Die auftretenden Defekte sind zu klassifizieren und auszumessen. Zu jedem Defekt wird eine Beschreibung erstellt. Um einen Kilometer einer Pipeline mit durchschnittlicher Datenqualität zu inspizieren, benötigt ein erfahrener Auswerter ca. einen Arbeitstag.

Für die Klassifikation der kritischen Regionen (Anomalien) stehen dem Auswerter keine konkreten Regeln zur Verfügung, sondern die Entscheidungen werden aufgrund von Erfahrungswissen getroffen, da die Anomalien keine einheitlichen geometrischen Formen besitzen und durch Rauschen in der Messung mit falschen, unplausiblen oder nicht vorhandenen Werten verfälscht werden können.

Diese Auswertungen sind demzufolge sehr zeitintensiv und teilweise subjektiv. Durch die fehlende Abwechslung wirken sie auf den Menschen ermüdend, der mit zunehmender Ermüdung zu Fehlern neigt. Gerade aber in einem Bereich wie der Sicherheitsüberprüfung von Pipelines sollten jedoch keine Defekte übersehen oder falsch interpretiert werden. Dies führt dazu, ein Diagnosesystem zu entwerfen, das dem Menschen bei diesen Auswertungen unterstützt. Um ein solches System aufzubauen, müssen jedoch einige Probleme berücksichtigt werden. Diese werden im nächsten Abschnitt eingehend behandelt.

[3] Zweidimensionale Falschfarbendarstellung, bei der die gemessene Materialdicke durch unterschiedliche Farben repräsentiert wird.

1.2 Probleme beim Aufbau eines fallbasierten Diagnosesystems

Eine der ältesten und auch am weit verbreitetsten Methoden, um technische Sachverhalte zu diagnostizieren, besteht in der Anwendung von Regeln. Die Diagnose von Prozessen bzw. Systemen, über deren Verhalten jedoch nur wenige Regeln verfügbar sind, erweist sich ohne die Hilfe von Modellwissen als äußerst schwierig. Oftmals wird die Diagnose eines solchen Problems von einem Menschen dann mit Hilfe der ihm vorliegenden Erfahrungen durchgeführt. Um ein derartiges Erfahrungswissen benutzen zu können, muß eine ausreichende Menge von Beispielen existieren, aufgrund derer, eine Diagnose getroffen werden kann. Aus diesem Grund kommt für die automatische Diagnose solcher Probleme ein fallbasierter Ansatz in Betracht. Solche Systeme beruhen auf der Vorgabe von Beispielen, die als Repräsentanten von Klassen interpretiert werden.

Um die fallbasierte Methodik verwenden zu können, ist eine gute und effiziente Wissensakquisition aufgrund repräsentativer Beispielsmengen erforderlich.

Hierzu müssen genügend repräsentative Daten zur Verfügung stehen. Darunter ist eine hinreichende Anzahl von Beispielen zu verstehen, die alle möglichen auftretenden Kombinationen der Daten enthält. In der Praxis ist diese Anforderung allerdings nur sehr schwer zu realisieren, da die Beispiele durch oft nicht direkt beeinflußbare Variablen beeinflußt werden. Wüßte man andererseits, wie die Variablen zusammenhängen, und wie sie zu verändern sind, könnte ein Modell erstellt werden. Für den automatischen Wissenserwerb ist es also eine grundlegende Voraussetzung, eine ausreichende Vielfalt bzgl. der Beispiele bzw. eine repräsentative Beispielsmenge zu ermitteln.

Erschwerend kommt hinzu, daß diese repräsentative Beispielsmenge a priori nicht vollständig bekannt ist, und die Daten darüber hinaus mit Rauschen behaftet sein können, wobei das Rauschen aus verschiedenen zu berücksichtigenden Ursachen herrühren kann. Es stellt sich also die Frage nach einer adäquaten Wissenserwerbs-Methode. Bei der Entwicklung von Diagnosesystemen werden häufig repräsentative Beispiele übersehen, weil sie vom Benutzer nicht als solche angesehen werden, oder aber für die Bewältigung der Aufgabe werden durch den Benutzer unnötig viele irrelevante Beispiele berücksichtigt (vgl. [Sun94b]). Dies ist deshalb in hohem Maße problematisch, da davon die erfolgreiche Realisation des Diagnosesystems stark abhängt.

Des Weiteren werden oft Methoden für Diagnosesysteme verwendet, die eine große Anzahl von Parametern benötigen (vgl. [Leo96]). Zu deren Identifikation wird erneut Expertenwissen benötigt. Diese Aspekte müssen bei einem Systementwurf berücksichtigt werden. Beispielsweise ließe sich das dadurch vermeiden, daß das System nicht parameterbehaftete Methoden benutzt. Ansonsten muß ein solches System in der Lage sein, die notwendigen Parameter selbständig zu ermitteln. Gleichzeitig muß auch darauf geachtet werden, daß der Zustand des Systems bzw. die Güte der erzielten Aussagen für den Benutzer jederzeit nachvollziehbar sind.

Es ist daher sinnvoll, die gesamte Interaktion zwischen dem Benutzer und dem Diagnosesystem in entsprechender Form anzulegen. Außerdem darf der Gesichtspunkt der Entlastung des Benutzers bei dem Aufbau des Diagnosesystems nicht aus den Augen verloren werden.

In der Praxis wird in der Regel versucht, anstelle applikationsunabhängiger Lösungen gezielt für eine bestimmte Anwendung eine maßgeschneiderte Lösung zu finden.

Diese kann dann auch im Allgemeinem nur für diese spezifische Anwendung eingesetzt werden.

Aus den zuvor beschriebenen Problemen ergeben sich deshalb bestimmte Erfordernisse für ein applikationsunabhängiges Diagnosesystem, die als Grundlage für die vorliegende Arbeit dienen. Deren Ziele werden in folgenden näher erläutert.

1.3 Ziele und Beitrag der Arbeit

Das Ziel der vorliegenden Arbeit besteht in der Untersuchung und Entwicklung eines interaktiven Diagnosesystems für subsymbolische Inspektionsdaten. Dabei liegt der Schwerpunkt auf der interaktiven Lernkomponente des Diagnosesystems, mit deren Hilfe der Benutzer den Lernvorgang unterstützen kann und somit sein Wissen durch das System direkt adaptiert wird.

Die in dieser Arbeit betrachteten Diagnoseaufgaben weisen Merkmale auf, aus denen sich Anforderungen für das Diagnosesystem ergeben. Im wesentlichen werden die Anforderungen an das Diagnosesystem durch die Eigenschaften der Daten und durch das abzubildende Anwendungswissen selbst gebildet.

Die Eingangsdaten des Diagnosesystems unterliegen dabei bestimmten Restriktionen. Es handelt sich dabei um sehr große Datenmengen für die a priori in der Regel keine repräsentativen Beispielsdatensätze vorliegen. Zudem sollen die Daten in digitaler Form präsent und zeitinvariant sein. Die Qualität der Daten kann in großem Maße differieren. Es ist außerdem davon auszugehen, daß ein großer Anteil der Daten verrauscht ist. Die Ursachen für die Verrauschung können unterschiedlicher Natur sein, beispielsweise können die Rauschanteile von Sensoren herrühren, durch Meßungenauigkeiten oder aber auch durch Effekte innerhalb der Übertragungsleitung entstanden sein.

Aber auch das abzubildende Diagnosewissen beeinflußt das Diagnosesystem. Für die Diagnose der Daten existieren entweder überhaupt keine expliziten Regeln, oder es können nur wenige Regeln aus der Anwendung zur Klassifikation abgeleitet werden. Darüber hinaus sind auf dem Applikationsgebiet Experten tätig, die auf Erfahrungen mit diesen Inspektionsdaten zurückgreifen können. Ferner besitzen diese Experten eine intuitive Vorstellung und Erfahrungswissen über das zu untersuchenden Diagnoseproblem.

Aus diesen Randbedingungen gilt es nun, unabhängig von einer bestimmten Anwendung, aber für diese gesamte Klasse des Applikationsgebiets eine methodische Vorgehensweise zur Entwicklung eines Diagnosesystems auszuarbeiten, so daß die Auswertung der vorliegenden Meßdatenmengen zu einem wesentlichen Teil durch das Diagnosesystem übernommen werden kann. Da nur wenige oder gar keine Regeln über die zu behandelnde Aufgabe bekannt sind, andererseits aber ausreichend Daten zu diesem Problem existieren, eignet sich zur Lösung dieser Diagnoseaufgabe ein beispielbasierter Ansatz. Aus diesem Grund ist das Lernparadigma[4] gegenüber einer expliziten Kodierung des Wissens in Form von Algorithmen vorzuziehen, da zu einem modellbasierten Lösungsansatz das notwendige Modellwissen fehlt.

Außerdem muß auch der Anteil der verrauschten Daten im Systemansatz berücksichtigt werden. Da in dieser Arbeit ausschließlich numerische Signalwerte behandelt werden, erscheint ein subsymbolisches Verfahren vielversprechend.

[4] Lernen statt Programmieren.

Abschließend ist noch ein Problem ganz anderer Natur in Betracht zu ziehen: die Wahl der richtigen Lernbeispiele. Im allgemeinen wird bei der Anwendung von Lernverfahren in der Weise vorgegangen, daß im ersten Schritt Lernbeispiele verschiedenster Art gesammelt werden. Diese werden dann im zweiten Schritt verwendet, um durch ein Lernverfahren die Wissensrepräsentation zu adaptieren. Dadurch entsteht aber der Nachteil, daß die Auswahl der Lernbeispiele durch den Menschen, also in subjektiver Form aus dessen Sicht heraus vorgenommen wird, was es zu vermeiden gilt.

Im Rahmen der vorliegenden Arbeit wird ausgehend von den genannten Randbedingungen eine Diagnosearchitektur vorgestellt und diskutiert, als auch ein Diagnosesystemkern entwickelt, der in der Lage ist, verrauschte Inspektionsdaten zu verarbeiten.

Ein wesentlicher Vorteil dieser speziellen Architektur liegt darin, daß Diagnoseaufgaben ohne die Hilfe von Regelwissen durch die Interaktion mit dem Benutzer als Lehrer erlernt werden können. Darüber hinaus werden die an der Diagnosearchitektur beteiligten Komponenten eingehend untersucht, ebenso wie die zugrundeliegenden Diagnoseproblemstellungen und deren Lösungen im Hinblick auf die Inspektionsdiagnose.

Um die Wissensakquisition für den Benutzer zu erleichtern, wurde ein weiterer Aspekt, den der transparenten Repräsentationsdarstellung innerhalb des Diagnoseprozesses, die es dem Benutzer ermöglicht, in einfacher Art und Weise die Vorgehensweise des Systems nachzuvollziehen sowie Widersprüche in den Systemaussagen leichter zu entdecken, berücksichtigt. Aus diesem Grund wird die Wissensakquisition durch unterschiedliche Interaktionsmöglichkeiten sowie durch eine automatische Beispielsuche unterstützt. Ein weiterer Aspekt, der sich aus der zu beachtenden Bedienerfreundlichkeit ergibt, führt dazu, ein weitgehend parameterloses Verfahren zu entwickeln, um auf diese Weise die Anzahl der einzustellenden Parameter möglichst gering zu halten und den Parametersuchraum zu verkleinern.

Die vorliegende Arbeit beschäftigt sich mit einer Klasse von Diagnosesystemen, die große verrauschte sensorische Datenmengen verarbeitet. Dabei geht der Ansatz davon aus, daß kein oder nur sehr eingeschränktes explizites Modellwissen vorhanden ist. Der Aufbau des Systems wird durch die interaktive Einbeziehung des Anwendungsexperten als Lehrer in den Lernprozess des Diagnosesystems erreicht. Sowohl die Parameterlosigkeit und die sofortige Adaption neuen Wissens durch die Lernkomponente während der Interaktion, als auch die Einschätzung der Güte der Aussagen des Systems, bietet einem mit den Problemen des Lernvorgangs nicht vertrauten Benutzer die Möglichkeit, auf einfache Art und Weise ein Diagnosesystem für seine Bedürfnisse anzupassen.

1.4 Gliederung der Arbeit

Nach einer Einführung der dieser Arbeit zugrundeliegenden Problematik der Entwicklung von Diagnosesystemen im Kapitel 1 wird diese in den nachfolgenden Kapiteln näher untersucht (vgl. Abbildung 1.2).

Dazu werden zunächst in Kapitel 2 Interaktive Diagnosesysteme erläutert, wobei der Lebenszyklus eines solchen Diagnosesystems dargestellt wird. Außerdem werden einige ausgewählte Diagnosesysteme vorgestellt und systematisch miteinander verglichen, um so die Vor- und Nachteile der unterschiedlichen Ansätze herauszuarbeiten. Daraus werden dann Schlußfolgerungen für die in dieser Arbeit vorgestellte Interaktive Lernende Diagnose (ILD) Architektur gezogen, die dann im Kapitel 3 eingehender erläutert wird.

Im dritten Kapitel wird das Profil beschrieben, das für die ILD geeignete Applikationen aufweisen muß, das dann dazu verwendet wird, die Anforderungen an ein Interaktiv Lernendes Diagnosesystem festzulegen. Es werden die Architektur des ILD-Systems und seine Komponenten vorgestellt. Zum Abschluß dieses Kapitels werden dann die Anforderungen an die einzelnen Komponenten eines ILD-Systems spezifiziert, die dann in den nachfolgenden Kapiteln eingehender untersucht werden.

Dementsprechend widmet sich Kapitel 4 den Prinzipien der Vorverarbeitung. Dabei wird näher auf die Signalvorverarbeitung und die Problematik bei der Extraktion von Merkmalen eingegangen. Die Eingangsdaten werden dazu in unterschiedlich dimensionale Merkmalstypen aufgeteilt. Insbesondere werden die Merkmalsbewertung und die Merkmalsselektion untersucht, wobei darauf geachtet wird, daß die analytische Betrachtung des Merkmals von der Klassifikationsmethode unabhängig ist. Nach den Prinzipien der Vorverarbeitung werden in den Kapiteln 5 und 6 die Lernkomponente und die Interaktion eingehend untersucht. Zusammen mit der experimentellen Verifikation im Kapitel 7 bilden diese drei Kapitel den Schwerpunkt der vorliegenden Arbeit.

In Kapitel 5 wird die eigentliche Lernkomponente vorgestellt. Kapitel 5 gliedert sich dabei in zwei Teile. Im ersten Teil wird die Untersuchung der Klassifikatorperformanz mit ihren Methoden und Fehlerabschätzungen, die Lernarchitektur mit ihren Verfahren sowie parameterloses Lernen mit Hilfe des *Dynamic Bounds* vorgestellt. Der zweite Teil widmet sich dann der Synergese mittels *Hybrider Knoten.*

Kapitel 6 beschäftigt sich dann mit dem Problem der Interaktion. Dabei wird im wesentlichen die Präsentation der Daten und die Wissenseingabe an sich untersucht. Darüber hinaus wird auf das Sammeln von Beispielen sowie die Korrekturfähigkeit des Systems eingegangen.

Die Interaktiv Lernende Diagnose wird im Rahmen dieser Arbeit auch experimentell verifiziert. Die Ergebnisse dieser Verifikation werden in Kapitel 7 dargestellt, die dann auch mit in die abschließende Betrachtung einfließen.

Den Abschluß dieser Arbeit bildet Kapitel 8 mit einer Zusammenfassung der vorliegenden Untersuchung sowie der Darstellung noch offener Fragen, die im Zusammenhang mit der vorliegenden Untersuchung aufgeworfen wurden. Eine Skizze weiterführender Ideen schließt die Arbeit ab.

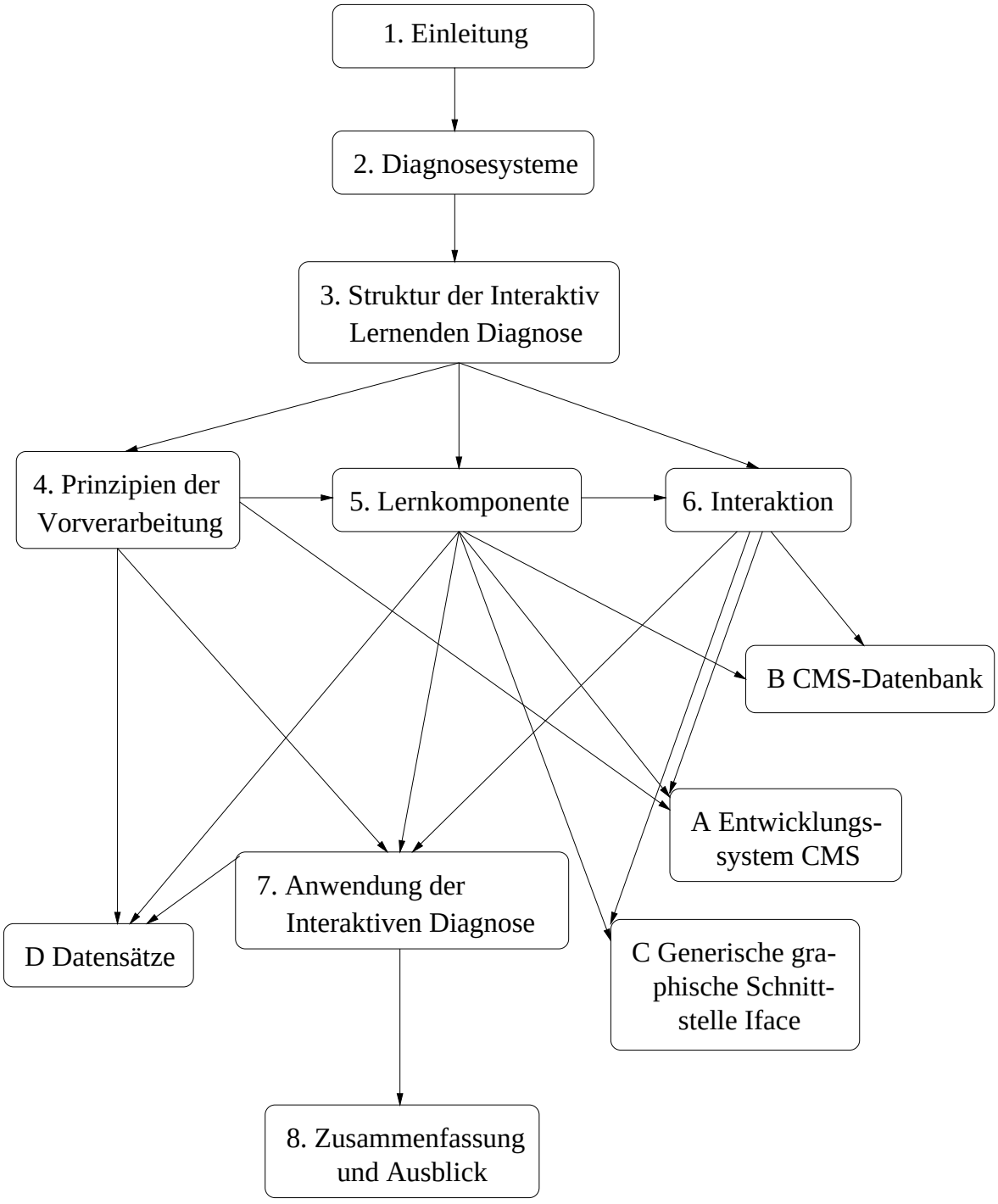


Abbildung 1.2: Kapitelübersicht

Kapitel 2

Architektur subsymbolischer Diagnosesysteme

Am bekanntesten ist der Begriff Diagnose aus dem Bereich der Medizin. Dort werden die Ursachen für verschiedene Krankheiten anhand ihrer Symptome festgestellt. Die Verknüpfung der Symptome zu einer oder möglicherweise mehreren Ursachen (Krankheiten) wird dann als Diagnose bezeichnet.

Bei technischen Systemen wird im Bereich der Fehlerdiagnose auf die gleiche Art und Weise verfahren. Im Gegensatz zur Medizin wird bei der technischen Diagnostik jedoch häufig nicht von Symptomen, sondern von Merkmalen gesprochen. Die Relation zwischen Merkmalen und Ursachen, deren Ergebnis jeweils einen Erklärungsversuch für die beobachteten Merkmale bietet, wird hier ebenfalls als Diagnose bezeichnet (vgl. [Pup88]).

Es liegt in der Natur der Diagnostik, daß die Messung Rauschen in die Merkmale einbringt oder aber nicht immer alle Merkmale bekannt sind. Zur Bearbeitung derartiger Daten eignen sich im wesentlichen subsymbolische Verfahren, da eine der wesentlichen Eigenschaften subsymbolischer Verfahren die Fehlertoleranz und Robustheit gegenüber verrauschten Daten darstellt. Aus diesem Grund werden im folgenden lediglich subsymbolische Diagnosesysteme betrachtet. Diese werden im weiteren kurz als Diagnosesysteme bezeichnet.

Wurde die Diagnose gestellt, schließt sich daran dann eine geeignete Maßnahme zur Fehlerbehebung an, die aber im Rahmen dieser Arbeit nicht im Vordergrund steht.

Die folgenden Abschnitte behandeln die grundlegenden Konzepte eines subsymbolischen Diagnosesytems von Inspektionsdaten, die möglichen Fehlerquellen bei der Erstellung eines Diagnosesystems und eine Auswahl von Systemen, die mit der in dieser Arbeit betrachteten Fragestellung verwandt sind. Daraus werden Schlußfolgerungen für die Architektur der Interaktiv Lernenden Diagnose (ILD) gezogen.

2.1 Grundlegendes über Diagnosesysteme

Bei technischen Systemen versteht man unter der Diagnose Lösungsprozesse, die den nachfolgenden Bedingungen genügen (vgl. [Pup88]):

Definition 2.1 (Diagnose)

- *Das zu untersuchende Problem besteht aus zwei explizit gegebenen disjunkten Mengen: der Menge der Merkmale M und der Menge der Klassen K (Ursachen).*

- *Die Merkmale $m \in M$ sind unter Umständen unvollständig, d.h. nicht alle Merkmale, die für eine Klasse ausschlaggebend sind, werden erfaßt.*
- *Die Lösung D (Diagnose) kann aus mehreren Klassen $k \in K$ bestehen.*

Unter Diagnosesystemen versteht man Systeme, die aus den vorhandenen Merkmalen, wie beispielsweise der Spannung, Frequenz oder Standardabweichung, den Zustand des beobachteten Objekts ableiten. Dabei werden online- und offline-Systeme unterschieden. Die online Diagnose ist direkt in dem Prozeß des Objektes eingebunden, wie zum Beispiel bei der Überwachung der Funktionen einer Maschine (siehe [Qua94, Der87, QRK91]) und ist somit in der Lage, z.B. vor Verschleiß zu warnen oder im Notfall sogar den entsprechenden Prozeß zu stoppen. Die offline Diagnose dagegen operiert auf gesammelten Daten und kann in den Prozeß nicht mehr unmittelbar eingreifen (vgl. [SB96b]).

In beiden Fällen sind nach der Diagnose eines Problems, geeignete Maßnahmen zu dessen Behebung vorzuschlagen oder sogar bereits einzuleiten (siehe auch Abb. 2.1). Diese Maßnahmen sind also von vornherein in den Aufbau eines Diagnosesystems einzuplanen.

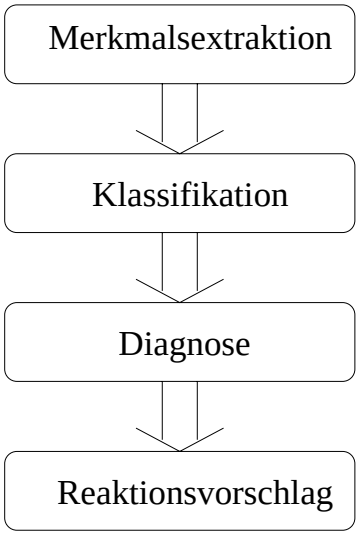


Abbildung 2.1: Schritte eines Diagnosesystems. Zuerst werden aus den Rohdaten relevante Segmente extrahiert. Die direkt oder durch eine Merkmalsextraktion gewonnenen Merkmale werden klassifiziert und in einer Diagnose zusammengefaßt. Als letzter Schritt schließt sich dann der Reaktionsvorschlag an.

Ein wesentlicher Aspekt beim Aufbau und der Anwendung von Diagnosesystemen besteht in der Art der Wissensakquisition. Das Wissen kann in Form von Regeln oder Modellen vorgegeben oder aus Beispielen erlernt werden (Näheres siehe Abschnitt 2.1.3). Der in dieser Arbeit verfolgte Ansatz geht davon aus, daß ein Applikationsexperte während der Lernphase dem Diagnosesystem bei Nachfragen direkt und gezielt Informationen zur Verfügung stellt.

Außerdem muß noch festgelegt werden, um was für eine Art der Diagnose es sich handelt. Bei der Diagnostik unterscheidet man diesbezüglich zwei wesentliche Ansätze, nämlich die Strukturdiagnose, die sich mit der Verifikation struktureller Zusammenhänge wie beispielsweise der Gerätekonfiguration befaßt, und der Parameterdiagnose,

welche die Meßwerte einer Applikation überwacht. Die Parameterdiagnose gliedert sich weiterhin in die fallbasierte und die modellbasierte Diagnose (vgl. dazu Kapitel 2.1.2.1). Da, wie bereits in Abschnitt 1.3 "Ziele und Beitrag der Arbeit" erläutert wurde, die in dieser Arbeit zu untersuchenden Probleme aus einer großen Menge von Fallbeispielen mit sehr wenigen Applikationsregeln bestehen, ist es sinnvoll, aus diesem Grund eine fallbasierte Diagnose zu wählen. Des Weiteren ist von verrauschten Daten auszugehen. Darüber hinaus sind die vorhandenen Merkmale größten Teils numerischer Natur, womit sich ein subsymbolischer Ansatz geradezu anbietet. Als Grundlage der Daten dienen einzelne Werte ohne Zeitabhängigkeit, so daß es sich hier um einen zeitdiskreten Ansatz handelt (vgl. hierzu auch Abb. 2.2).

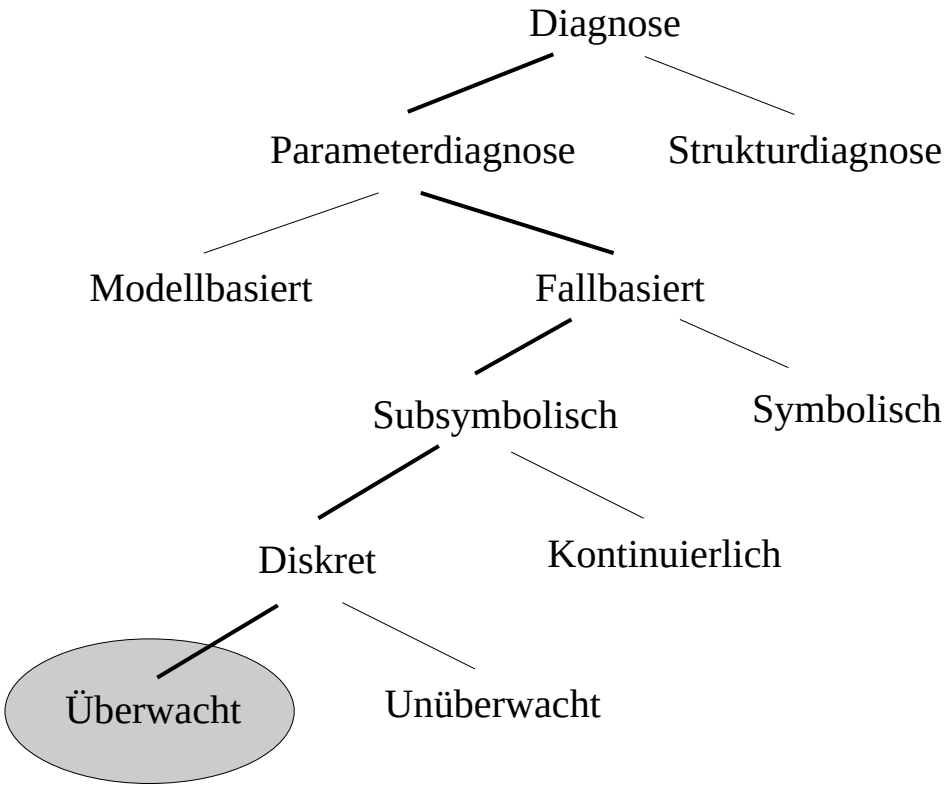


Abbildung 2.2: Taxonomie der in dieser Arbeit betrachteten Diagnosesysteme, in Abhängigkeit der verwendeten Wissensrepräsentation. Die vorliegende Arbeit beschäftigt sich mit dem eingekreisten Teilgebiet der Diagnose.

2.1.1 Lebenszyklus eines Diagnosesystems

In der Softwaretechnik wird die Erstellung, Wartung und Weiterentwicklung von Programmen bzw. Programmsystemen unter dem Begriff *Software Life Cycle* zusammengefaßt (vgl. [Jös93]). Für Diagnosesysteme läßt sich ein ähnlicher Lebenszyklus angeben. Ein Diagnosesystem durchläuft während seiner Erstellung und Anwendung mehrere Phasen (siehe Abb. 2.3). Dabei lassen sich drei Hauptphasen erkennen: die Diagnosesystemkern-Erstellung, die Wissensakquisition und die Anwendung selbst (vgl. [FPB96]). Diese drei Hauptphasen sind durch verschiedene Aufgabenbereiche gekennzeichnet:

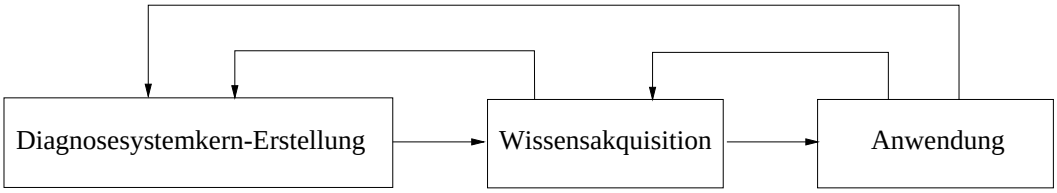


Abbildung 2.3: Lebenszyklus eines Diagnosesystems

Diagnosesystemkern-Erstellung: Implementierung der Algorithmen, die das Verarbeiten von Merkmalen, deren Zuordnung zu den Klassen sowie der Struktur der Wissensbasis und der Benutzerschnittstellen ermöglichen (siehe auch Abschnitt 2.1.2). Bei Expertensystemen wird dieser Diagnosesystemkern auch Shell genannt (vgl. [Pup88, Sun94a]).

Wissensakquisition: In dieser Phase wird alles erforderliche Wissen des Experten in den Diagnosesystemkern übertragen (vgl. dazu Abschnitt 2.1.3). Ist die Integration von Wissen nicht möglich, müssen neue Anforderungen an den Diagnosesystemkern gestellt werden, die durch eine Erweiterung des Kerns zu realisieren sind.

Anwendung: Der Benutzer verwendet das Diagnosesystem und ermittelt ggf. die Schwachstellen des Systems. Das dabei gewonnene neue Wissen kann in einer neuerlichen Wissensakquisition-Phase integriert werden.

Die Phasen verlaufen dabei verschachtelt, d.h. die nachfolgenden Phasen können immer wieder in vorherige Phasen zurückspringen, um somit eine Verbesserung des Diagnosesystems herbeizuführen. Die Phasen unterscheiden sich, wie bereits erwähnt, in ihren Aufgaben. Darüber hinaus differieren sie aber auch in dem Maße, in dem sie bereits Applikationswissen verwenden. Aus diesem Grund wird das System ohne Applikationswissen *Diagnosesystemkern* genannt, und erst das mit Wissen angereicherte System wird als *Diagnosesystem* bezeichnet.

Die Anforderung an die einzelnen Phasen des Lebenszyklus eines Diagnosesystems werden nun im folgenden näher spezifiziert.

2.1.2 Anforderungen an den Diagnosesystemkern

Die primären Anforderungen an einen Diagnosesystemkern gliedern sich in drei Hauptbereiche: die Wissensrepräsentation, das Klassifikationssystem und die Dialogschnittstellen (vgl. [PR93]).

An die jeweiligen Bereiche werden folgende tiefergehenden Anforderungen gestellt:

Wissensrepräsentation:

- Das Wissen muß in einer Wissensbasis repräsentiert werden. Wissensfragmente unterschiedlicher Form und Semantik müssen integrierbar sein.
- Dem Anwender bleibt die interne Repräsentation der Daten verborgen. Die Interpretation erfolgt in dem jeweiligen Kontext, aber für den Benutzer auf leicht nachvollziehbare und transparente Weise.

- Die Wissensbasis befindet sich immer in einem konsistenten Zustand, und darf nicht durch fehlerhafte Bedienung zerstört werden können.
- Es muß möglich sein, neues Wissen einfach hinzuzufügen bzw. zu entfernen.

Klassifikationssystem:

- Das Klassifikationssystem muß sich anhand vergangener Erfahrung adaptieren. Dieses Lernen soll problemangepaßt sein und möglichst ohne Eingriffe des Benutzers durchgeführt werden.
- Das System soll auch bei verwandten Problemen einsetzbar sein.
- Das Klassifikationssystem sollte auch bei verrauschten Daten gute Ergebnisse erzielen.
- Der Benutzer soll eine Prognose über die Güte/Zuverlässigkeit der durchgeführten Klassifikationen erhalten.

Dialogschnittstellen:

- Der Benutzer soll durch Parametrisierung, eine Vorauswahl/Filterung treffen, die es ermöglicht, eine schnellere Wissensakquisition durchzuführen.
- Es muß auch möglich sein, fehlerhafte Eingaben rückgängig zu machen.
- Dem Benutzer muß die Möglichkeit gegeben werden, Wissen und Hypothesen in unterschiedlicher Form eingeben zu können. Dem System gegenüber können Erläuterungen in Form von Erklärungen, Alternativen, Unsicherheiten oder Risiken gegeben werden
- Die Kommunikation mit dem System soll möglichst komfortabel und leicht für den Benutzer konzipiert sein. Es sollten Dialogkomponenten für unterschiedliche Eingabearten existieren.
- Das Vokabular des Systems muß dem des Benutzers entsprechen.
- Der Benutzer soll mit Hilfe der Dialogschnittstellen Fragen bzw. Statusmeldungen des Systems abrufen können.

Diese Anforderungen an den Diagnosesystemkern werden im folgenden ausführlicher betrachtet.

2.1.2.1 Wissensrepräsentation

Auf dem Gebiet der Wissensrepräsentation wurden im Rahmen der Entwicklung im Bereich der Künstlichen Intelligenz (KI) viele verschiedene Ansätze untersucht, beispielsweise Semantische Netze, Frames, Petrinetze, Anweisungstabellen, genetische Algorithmen, Neuronale Netze und viele andere (siehe [Qua94]).

Es hat sich gezeigt, daß keine Methode für sich allein allen Problemen gerecht werden kann. Aus diesem Grund werden die Methoden für spezielle Aufgabengebiete ausgewählt, wobei in letzter Zeit diese Methoden immer häufiger miteinander verknüpft werden, um immer komplexere Probleme lösen zu können.

Unabhängig von der gewählten Repräsentation, kann die Erstellung einer Diagnose als Vergleich zwischen der Wissensrepräsentation und den Merkmalen des Objekts aufgefaßt werden (siehe Abb. 2.4). Dabei werden also aktuell vorgegebene Werte mit dem

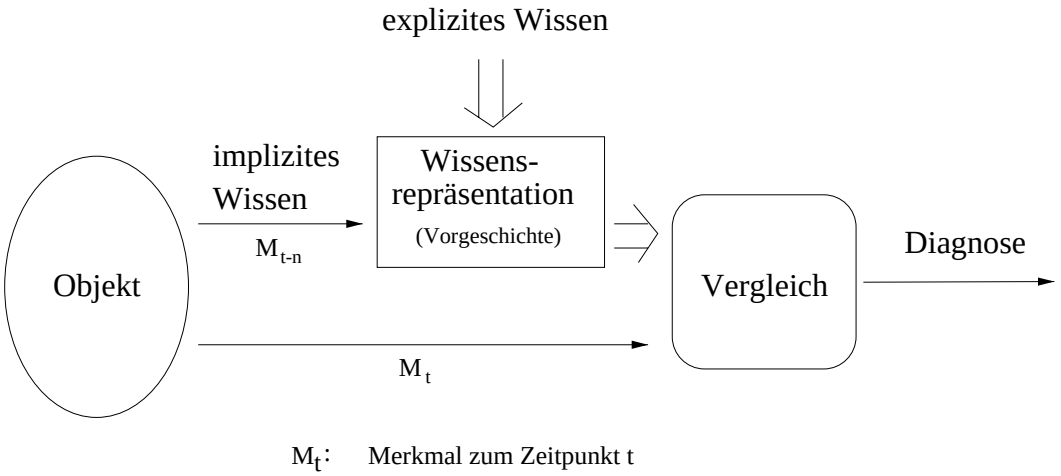


Abbildung 2.4: Die Diagnose wird durch den Vergleich zwischen der Wissensrepräsentation und den Merkmalen des Objekts erstellt.

in der Wissensbasis gespeicherten Wissen verglichen. Das Ergebnis dieses Vergleichs liefert dann die Diagnose.

Die Art des Diagnosesystems hängt stark von dem zu diagnostizierenden Objekt, der Art des Wissens über das Objekt und den zur Verfügung stehenden Daten ab. Anhand der unterschiedlichen Wissensquellen lassen sich folgende Klassen von Wissensrepräsentationen feststellen:

Modellbasiert: Mit Hilfe physikalischer Modelle oder Expertenheuristiken werden Regeln bzw. Modelle aufgestellt, die als Wissensbasis für die Diagnose dienen. Von vornherein existiert bereits eine Vorstellung/Wissen über die Objekte, die es zu kodieren gilt. In Abbildung 2.4 wird dieses Vorwissen als explizites Wissen dargestellt.

Fallbasiert: Anhand von Beispielen wird eine Fall-Wissensbasis aufgebaut. Die Wissensrepräsentation wird anhand dieser Beispiele gebildet, in dem induktive Verfahren zur Ableitung von Regeln, oder subsymbolische Verfahren (Neuronale Netze, genetische Algorithmen, etc.) eingesetzt werden. Diese Art der Repräsentation wird dann gewählt, wenn nur wenige oder keine Kenntnisse über die Zusammenhänge innerhalb eines Problems vorhanden sind. In Abbildung 2.4 wird es als sog. implizites Wissen dargestellt.

Die Art der gewählten Wissensrepräsentation beeinflußt wesentlich die Ausprägung des Klassifikationssystems.

2.1.2.2 Klassifikationssystem

Die Art des Klassifikationssystems hängt sowohl von dem Diagnoseproblem als auch von der Wissensrepräsentation ab, und kann bezüglich der folgenden Eigenschaften noch weiter unterteilt werden:

- Vollständigkeit des Modells bzw. der Anzahl der vorhandenen Regeln,
- Anzahl der zur Verfügung stehenden Beispiele,

- Qualität und Grad der Verrauschtheit der Daten,
- Datencharakteristik: symbolische bzw. numerische Repräsentation,
- Art der zu untersuchenden Daten: Diagnose von Zeitreihen oder diskreten Ereignissen,
- Adaptivität des Klassifikationssystems an neue Informationen.

In der Literatur sind die unterschiedlichsten Methoden zur Klassifikation bekannt (vgl. dazu [DH73, Bis95, FPB96, Qua94]).

Wie in Abschnitt 1.3 motiviert, sind die für diese Arbeit interessanten Methoden diejenigen, die subsymbolische (numerische), verrauschte und diskrete Daten verarbeiten können. Deswegen sind diejenigen Klassifikationssysteme für diese Arbeit von Interesse, die ohne Regeln auskommen können, aber dennoch in der Lage sind, vorhandene Regeln zu integrieren und eine schnelle Adaption an neue Informationen zu gewährleisten.

Aus diesem Grund sind auch diverse Anforderungen von der Wissensakquisition zu erfüllen, die im folgenden Abschnitt eingehender beleuchtet werden.

2.1.3 Wissensakquisition

Die Akquisition von Wissen wird beim Knowledge Engineering nach [KL90] in drei logische Phasen aufgeteilt:

Wissenserhebung: Die Daten werden erfaßt, in dem sie aus den unterschiedlichen Wissensquellen (siehe auch nächste Seite) extrahiert werden.

Interpretation: Es erfolgt eine mentale Konzeption des Wissens in ein Modell, bzw. beim modellbasiertem Ansatz wird in dieser Phase die Modellierung durchgeführt.

Operationalisierung: In dieser Phase erfolgt die tatsächliche Kodierung des Wissens in die Wissensbasis.

Die Arten der Wissenserhebung lassen sich grob, wie in Abbildung 2.5 dargestellt, in Erstellen/Eingeben des Wissens, automatisches Ableiten und Lernen einteilen (vgl. [Zöl95, FPB96]). Das so gewonnene Wissen muß in der Interpretationsphase in die richtige Operationalisierung umgesetzt werden. Beim automatischen Ableiten und Lernen ist die Interpretationsphase bereits mit der Operationalisierung verschmolzen, da die Umsetzung von der Wissenserhebung bis zur Operationalisierung durch die Methoden vorgegeben ist. Bei der Erstellen/Eingeben-Methodik ist eine Interpretation des gesammelten Wissens durch den Diagnoseexperten notwendig.

Diese unterschiedlichen Methoden sind auch mit unterschiedlichem Aufwand für den Diagnoseexperten und der notwendigen Komplexität des zugrunde liegenden Diagnosesystemkerns verbunden. Ein wichtiger Faktor für die Güte und Akzeptanz eines Diagnosesystems besteht in dem Aufwand, der notwendig ist, um das vorhandene Wissen dem System zur Verfügung zu stellen (siehe Abschnitt 2.5).

Demzufolge ist bei der Planung eines Diagnosesystems die Art der Wissensakquisition (manuell/automatisch) gegenüber dem Entwicklungsaufwand für den Diagnosesystemkern abzuwägen (siehe Abb. 2.6), denn im allgemeinen gilt: je mehr Aufgaben der

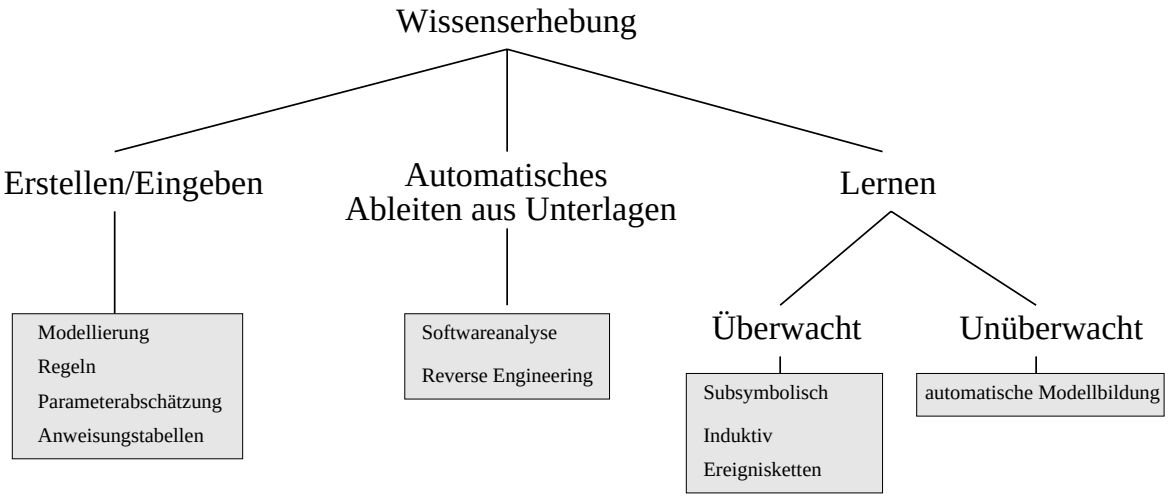


Abbildung 2.5: Unterschiedliche Arten der Wissenserhebung mit möglichen Operationalisierungsmethoden.

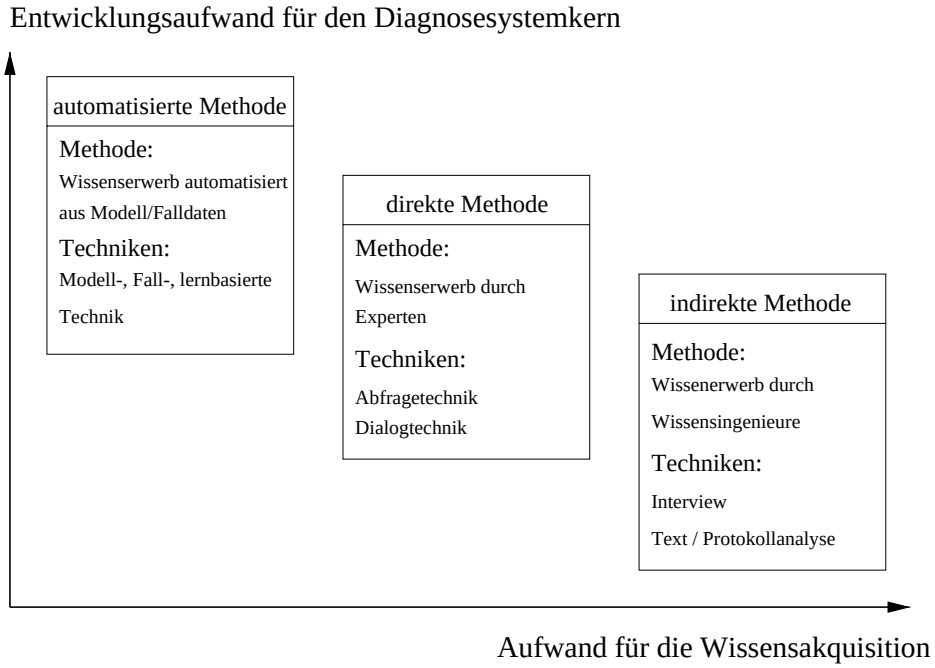


Abbildung 2.6: Manueller Aufwand bei der Wissensakquisition in Abhängigkeit von der Komplexität des Diagnosesystemkerns (aus [Zöl95])

Diagnosesystemkern automatisch übernimmt, desto höher bemißt sich der Aufwand für dessen Konstruktion.

Dies bedeutet, daß bei gleichbleibender Qualität des Diagnosesystems der Aufwand zwischen dem Erstellungsaufwand für den Diagnosesystemkern und dem Wissensakquisitionsaufwand verlagert werden kann.

Dabei ist allerdings zu berücksichtigen, daß mit steigender Anzahl der Beteiligten sich auch die Zahl der möglichen Fehlerquellen erhöht (siehe Abschnitt 2.2). Dies ist gerade bei der indirekten Methode (vgl. dazu Abb. 2.6) der Fall. Aus diesem Grund ist die direkte oder gar die automatisierte Methode der indirekten vorzuziehen.

Die Entscheidung, welche Art der Wissensakquisition angewandt werden kann, hängt im wesentlichen von den vorhandenen Wissensquellen ab.

Wissensquellen

Das Wissen, das für die Bildung eines Diagnosesystems verwendet werden kann, stammt aus den Domänen des strukturellen, funktionalen und heuristischen Wissens über die Applikation. Diese Quellen können mit den unterschiedlichen Methoden der Wissenserhebung erschlossen werden (vgl. Abb. 2.5). Hierbei kommt der Erstellungsaufwand eines Diagnosesystems zum Tragen. Denn im allgemeinen gilt: je automatischer eine Wissensquelle bearbeitet werden kann, desto höher ist der Erstellungsaufwand des betreffenden Systems, und desto geringer sind die zu erwartenden Fehldiagnosen, die im wesentlichen durch den Menschen verursacht werden (siehe Abschnitt 2.2).

Die obengenannten Wissensarten zeichnen sich durch spezielle Eigenschaften aus:

Strukturelles Wissen: beschränkt sich auf die strukturellen Aspekte der Anwendung, wie beispielsweise:

- Anlagenbau,
- Unternehmensstruktur.

Funktionales Wissen: beschreibt die Funktionalität der Komponenten und ihre dynamischen Eigenschaften wie

- Ablaufpläne,
- physikalische Zusammenhänge.

Heuristisches Wissen: beschreibt anhand besonderer Beispiele oder heuristisch fundierter Erkenntnisse Zusammenhänge im Applikationsbereich anhand von

- Fallbeispielen,
- Erfahrungswissen.

Die für diese Arbeit interessanten Quellen beschränken sich im wesentlichen auf das heuristische Wissen, da andere Wissensquellen für diese Klasse von Applikationen nicht verfügbar sind. Es wird aber auch aufgezeigt, wie das vorhandene strukturelle und funktionale Wissen eingebracht werden kann (vgl. Abschnitt 3.4). Die daraus resultierende Wissensakquisitionsmethode ist zwischen der automatisierten und der direkten anzusiedeln (siehe Abb. 2.6).

2.1.4 Anwendung

Am Schluß des Lebenszyklus eines Diagnosesystems steht die Anwendung des Diagnosesystems selbst. Diese bezieht sich immer auf eine ganz spezielle Problematik, für die das Diagnosesystem originär erstellt wurde. Im nächsten Abschnitt werden zur Veranschaulichung einige in der Industrie verwendete Diagnosesysteme näher beschrieben, um so auch nochmals die Eigenschaften zur Zeit existierender Diagnosesysteme aufzuzeigen.

2.2 Fehlerquellen bei Diagnosesystemen

Die Erstellung von Diagnosesystemen erfordert eine sehr hohe Sorgfalt. Nach einer Untersuchung über die Fehlerquellen bei der Entwicklung von wissensbasierten Systemen (siehe [Zöl95]), hängt die Qualität eines Diagnosesystems im wesentlichen von der Wissensakquisition und dem Wissen der Experten ab (siehe Abb. 2.7). Dabei hat sich gezeigt, daß Fehler, die dem Experten bei der Erstellung des Diagnosesystems unterlaufen (sog. Expertenfehler), wie z.B. falsche Zuordnung der Merkmale oder eine inkorrekte Wissensakquisition, ca. 50% der auftauchenden Probleme verursachen. Die restlichen Fehlerquellen (siehe dazu auch Abb. 2.7) hängen im wesentlichen von den eingesetzten Geldmitteln bzw. der Schulung der Mitarbeiter ab.

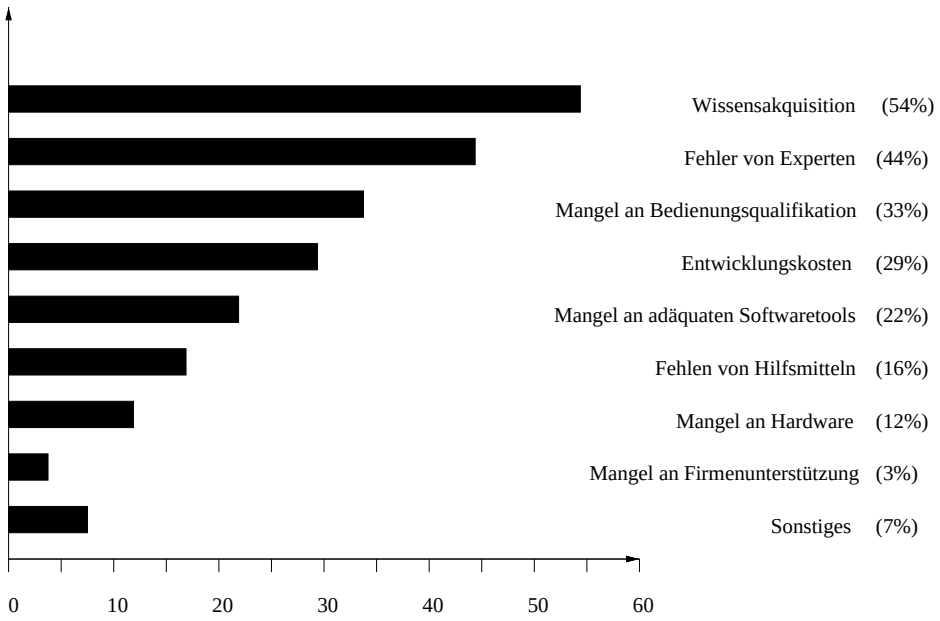


Abbildung 2.7: Fehlerquellen bei der Erstellung diverser Diagnosesysteme (aus [Zöl95]).

Sowohl die inkorrekte Wissensakquisition als auch Expertenfehler beeinflussen aber auch in großem Maße die Zuverlässigkeit und Effektivität des Diagnosesystems. Aus diesen Gründen müssen die Eingaben der Experten, die interaktiv mit dem System arbeiten, überprüft werden.

Die Erkennung und Korrektur von Fehlern sollte so früh wie möglich erfolgen, da die Verschleppung eines Defektes immer schwerwiegendere Auswirkungen nach sich zieht. Dies wird in Abb. 2.8 verdeutlicht.

Dabei unterscheidet man vier Arten von Fehlern, wobei sich vom Fehler hin zum Schaden die Auswirkungen sukzessive verschlimmern.

Bedeutet ein Fehler dabei lediglich noch eine Abweichung von dem gewünschten Verhalten (sog. Soll-Verhalten), so beeinträchtigt eine Störung bereits die Funktionalität des Systems. Dennoch wirkt sich eine Störung nur begrenzt aus. Beseitigt man diese Störung nicht, so führt dies letztendlich zum Ausfall der entsprechenden Funktion. Wird der Ausfall ebenfalls ignoriert, kann daraus ein Schaden resultieren, der die Zerstörung einzelner Komponenten nach sich zieht, die schließlich auch die Unbrauchbarkeit der eigentlichen Applikation verursachen.

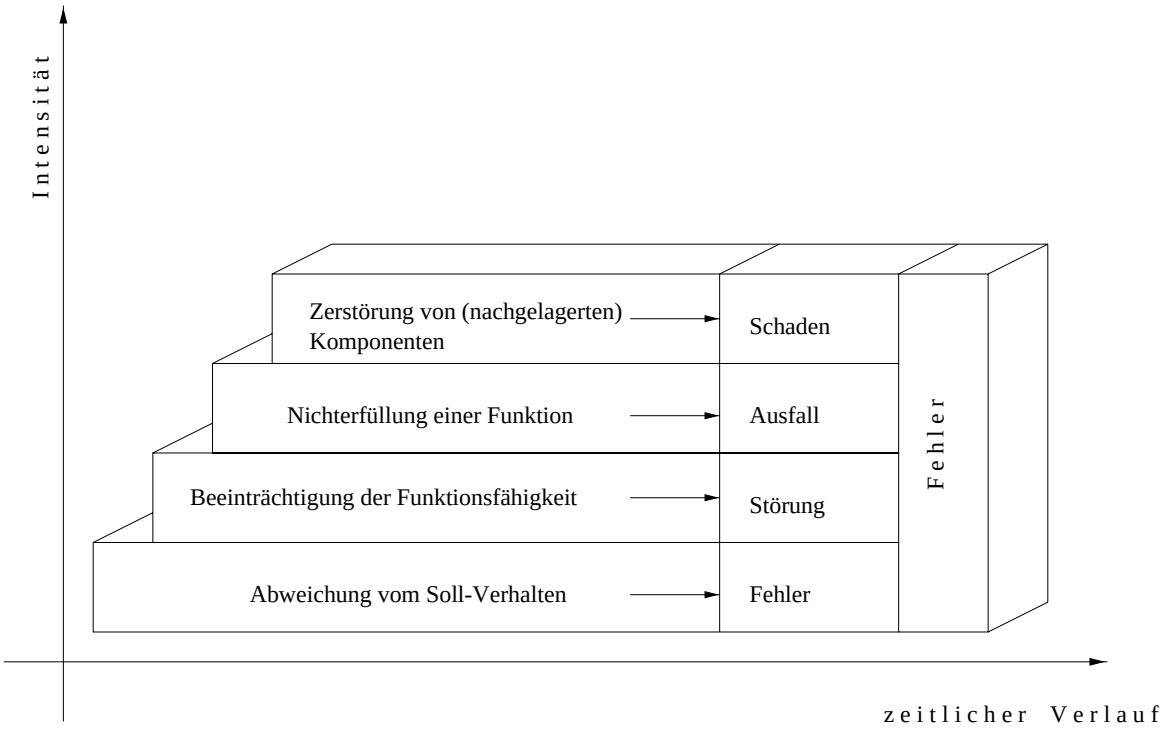


Abbildung 2.8: Auswirkungen eines Fehlers in Abhängigkeit von Zeit und Intensität. Generell gilt: je länger ein Fehler verschleppt wird, desto schwerwiegender sind die daraus resultierenden Folgen (aus [Qua94]).

Deshalb ist es besonders wichtig – um Kosten und Zeit für eine eventuelle Schadensbehebung zu vermeiden – bereits bei der Erstellung des Diagnosesystems Kontrollmechanismen einzubauen, die Fehler und Inkonsistenzen des Diagnosesystems erkennen und auch Vorschläge zu deren Behebung unterbreiten, um somit Schäden an der Applikation frühzeitig vorzubeugen.

Zieht man dazu die Information hinzu (siehe auch Abb. 2.7), daß ca. die Hälfte aller Fehler aus einer inkorrekten Wissensakquisition bzw. Expertenfehler resultiert, ist es dringend geboten, Kontrollmöglichkeiten miteinzubeziehen, die eine inkorrekte Wissensakquisition verhindern oder wenigstens auf Inkonsistenzen in den Expertenaussagen hinweisen.

2.3 Beispiele eingesetzter Diagnosesysteme

In der Industrie werden seit längerem verschiedene Diagnosesysteme erfolgreich eingesetzt (siehe [KS97, Seg96, Lut93, SLS93]), vornehmlich in der Qualitätssicherung und Authentisierung. Die im folgenden näher betrachteten Systeme (vgl. dazu auch die Übersicht in Tabelle 2.1) sind für spezielle Applikationen entstanden, die sich nur partiell oder gänzlich der analytischen Beschreibung entziehen. Es werden die Lösungsansätze dieser Systeme untersucht, um Schlußfolgerungen für die Architektur des ILD-Systems zu erhalten.

2.3.1 Motor-Diagnose

Um den steigenden Qualtitätsanforderungen gerecht zu werden, besteht das Bestreben, Fehler bei der Fertigung von Motoren bereits frühzeitig zu erkennen (siehe [Lut93]). Dazu wird der Montageprozeß überwacht, um so eine schnelle Beurteilung des geprüften Motors zu erhalten. Es soll dazu die Fehlerursache möglichst genau festgestellt werden. Außerdem ist eine Reparaturanleitung automatisch zu generieren.

Um diesen Anforderungen zu genügen, werden für dieses Diagnosesystem vollständig verbundene Backpropagation-Netze eingesetzt. Um diese einzulernen, werden aus den Meßdaten (5 Parameter mit je 750 Meßwerten) fehlertypische Merkmale in Dateien abgelegt, die die kontinuierlichen Meßsignale und ihre Bewertung enthalten. Es treten dabei verrauschte Daten auf, und es sind darüber hinaus noch Fertigungstoleranzen zu beachten. Die Daten werden für die Klassifikation mit Backpropagation-Netzen auf das Intervall $[0 \ldots 1]$ normalisiert. Als erfolgreich hat sich eine $50 - 20 - 20 - 11$ Topologie herausgestellt, bei der jedes der 11 Ausgabeneuronen einer Fehlerklasse zugeordnet wurde.

Das beschriebene Diagnosesystem beantwortet die Fragen, welche Fehler die ordnungsgemäße Funktion des Motors beeinträchtigen, und wo diese Fehler aufgetreten sind. Die Diagnose wird in Realzeit durchgeführt und gibt die gefundenen Fehlerquellen aus.

2.3.2 Getriebe-Diagnose

Die Getriebe-Diagnose dient der Ermittlung und Aufzeichnung des Vollastschaltverhaltens (siehe [SLL93]). Es sollen bereits geringfügigste Abweichungen von den Toleranzen festgestellt werden.

Man unterscheidet dabei drei verschiedene Arten von Schaltvorgängen: Vollasthoch-, Vollastrückschaltvorgänge und Modulationsdruck. Eine Messung von 3 Sekunden ergibt ca. 18000 verrauschte Meßwerte, die durch äquidistante Abtastung auf 150 Merkmale für die drei Backpropagation-Netze reduziert werden. Für jeder Art des Schaltvorgangs ist ein anderes Netz zuständig. Diese sind in der Lage, sechs verschiedene Fehlertypen und auch daraus kombinierte Fehlerursachen zu erkennen. Zum Einlernen der Netze mit einer $150 - 15 - 15 - 30 - 6$ Topologie werden manuell zusammengestellte Musterdateien verwendet, die einen repräsentativen Querschnitt der gesammelten Beispiele enthalten.

Mittels einer graphischen Ausgabe werden sowohl die Eingabedaten als Kurvenverlauf der Drehzahl als auch der Fehlertyp dargestellt.

2.3.3 ZN-Face

Bei ZN-Face handelt es sich um ein Zutrittskontrollsystem (siehe [Gmb]). Mittels automatischer Gesichtserkennung, die auf Neuronalen Netzen basiert, wird der Zugang zu einem Gebäude überprüft.

ZN-Face ist ein Komplettsystem aus Verifikationskonsole mit integrierter Kamera und dazugehöriger Rechnereinheit. Am Kontrollpunkt wird automatisch ein Bild der Person aufgenommen, die das Gebäude oder den Raum betreten will. Dazu wird das Gesicht in ein Gitter zerlegt und diese durch neuronale Netze mit sog. Referenzbildern verglichen.

Es besteht die Möglichkeit die Übereinstimmungsgüte einzustellen, so daß die Wahrscheinlichkeit für falsche positive Erkennungen beeinflußt werden kann. Dies geht natürlich auf Kosten der Anzahl der Abweisungen berechtigter Personen, denen der Zugang verweigert wird. Diese Werte lassen sich durch die Verwendung mehrerer Referenzbilder verbessern.

2.3.4 ZN-LISA

ZN-LISA ist ein System, mit dem unter nur geringem Aufwand, akustische Daten erfaßt, visualisiert, gespeichert und analysiert werden können (siehe [HG97]). Dazu werden offline neuronale Klassifikatoren entwickelt, die Fehler in Bauteilen, Materialien oder Fertigprodukten diagnostizieren. Das Einlernen der Klassifikatoren basiert auf Beispielsdateien. Über ein Benutzerinterface werden die Klassifikationsergebnisse angezeigt.

Der komplette Aufbau von ZN-LISA ist in Abbildung 2.9 genauer dargestellt.

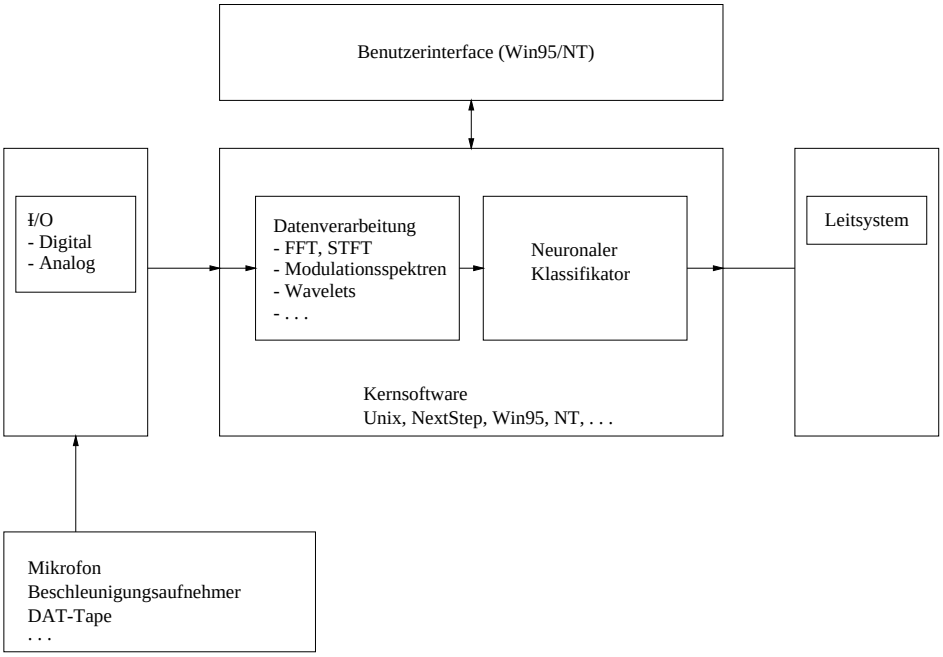


Abbildung 2.9: Architektur des LISA-Systems

ZN-LISA wird bereits in verschiedenen Produktionen eingesetzt, beispielsweise bei dem Sound-Quality-Engineering, der Vermessung der Blaslanzenposition am Elektro-Lichtbogen oder der Kontrolle vielbenutzter mechanischer Komponenten wie Bankautomaten.

2.3.5 Qualitätskontrolle von Kassettenlaufwerken

Dieses Diagnosesystem dient der Erkennung von Montage-, Fertigungs- und Funktionsfehler von Kassettenlaufwerken (oder auch anderen rotierenden Geräten), die durch unterschiedliche Geräusche festgestellt werden können (siehe [FWH97b, FWH97a]).

Das vorliegende Expertenwissen läßt sich nicht in Regeln fassen, deshalb wird eine Klassifikation durch Kohonen-Netze durchgeführt. Jedes Laufwerk wird vier Sekunden

akustisch überprüft. Dabei entstehen 32768 Abtastwerte, aus denen Merkmale wie die Varianz, die Schiefe oder die Wölbung des gemessenen Signals ermittelt werden. Es wurden 500 Beispielsdatensätze gesammelt, mit denen ein Kohonen-Netz eintrainiert wurde.

Die Klassifikationsergebnisse werden mittels verschiedenfarbiger Symbole am Bildschirm dargestellt. Außerdem werden als fehlerhaft gekennzeichnete Laufwerke automatisch aussortiert.

2.3.6 DIAMOND

Verwendet wird dieses Diagnosesystem zur Diagnose von Dieselmotoren, um dabei die Reparatur fehlerhafter Motoren zu beschleunigen, und um eine kostenmäßige Optimierung zu erreichen.

Bei DIAMOND handelt es sich um ein hybrides Diagnosesystem (siehe [SLS93]), das für die Diagnose Neuronale Netze einsetzt. Das Diagnoseergebnis wird dann anhand sog. Plausibilitätsregeln, die aus Expertenwissen abgeleitet wurden, überprüft.

Alle motorrelevanten Daten und ebenso die Regeln werden in einer relationalen Datenbank gespeichert. Dadurch findet eine schnelle Adaption an aktuelle Gegebenheiten statt.

Die Diagnose wird in Verbindung mit anderen Informationen, wie dem aktuellen Motorstatus, der Gegenüberstellung von Soll- und Istdaten, der Motorhistorie und diversen Motorparametern, ausgegeben. Die Ergebnisse werden am Bildschirm angezeigt.

Tabelle 2.1 zeigt noch einmal eine Übersicht über die oben aufgeführten Diagnosesysteme.

2.4 Schlußfolgerungen für Interaktive Diagnosesysteme

Bei den beispielhaft vorgestellten subsymbolischen Diagnosesystemen wird beschrieben, daß die Wissensakquisition als separater Schritt bei der Erstellung eines Diagnosesystems durchgeführt wird. Zuerst werden durch Experten Beispielsdaten gesammelt, die im nachfolgenden Schritt im System eingelernt werden. Diese Vorgehensweise bewirkt, daß keine Berücksichtigung des geänderten Verhaltens des Diagnosesystems aufgrund der Hinzunahme neuer Beispiele durchgeführt wird. Die Operationalisierungsphase ist also zeitlich von der Wissenserhebung getrennt.

Es wird des Weiteren aufgezeigt, daß eine wesentliche Fehlerquelle bei der Erstellung von Diagnosesystemen bei der Wissensakquisition verursacht wird. Demzufolge ist es für die Fehlervermeidung seitens der Experten beim Aufbau eines Diagnosesystems notwendig, eine umfangreiche Unterstützung im Zuge der Wissensakquisition zu bieten. Dies bedeutet, daß eine weitgehend automatische Wissensakquisition eingesetzt werden sollte, um diese Fehlerquelle zu minimieren. Darüber hinaus ist die Wissensakquisition auf eine möglichst einfache gemeinsame Sprache zu reduzieren, damit beide Beteiligten über die gleichen Informationen verfügen und nicht "aneinander vorbeireden" und auf diese Weise inkorrektes Wissen in das Diagnosesystem einfließt.

Das System muß eine Vorfilterung der Beispiele durchführen, und den Benutzer nur in Zweifelsfällen zu Rate ziehen, um schon auf diese Weise einer möglichen Falschein-

Diagnosesystem	Problematik	Wissensakquisition	Lernkomponente	Fehlerbehebung	Dialog-unterstützung
Motor-Diagnose	Überwachung im Montageprozeß	Beispielsdatei mit z.T. verrauschten u. kontinuierlichen Daten	vollständig vernetzte BP-Netze	automatische Generierung einer Reparaturanleitung	Dialogausgabe der Diagnose
Getriebe-Diagnose	Ermittlung geringfügigster Abweichungen von Fehlertoleranzen beim Schaltverhalten	Musterdatei mit repräsentativem Querschnitt	fünfschichtiges BP-Netz	mehrere Fehlerarten in Kombination diagnostizierbar	graphische Anzeige mit Aufbereitung der Meßdaten und Angabe des Fehlertyps
ZN-Face	automatische Gesichtserkennung zur Zugangskontrolle	Gesichtsabbildungen als Grauwertbilder i.V.m. Referenzbilder	Neuronale Netze	Beschränkung des Zugangs	Verifikationskonsole mit integrierter Kamera
Kontrolle von Kassettenlaufwerken	akustische Erkennung von Montage-, Fertigungs- u. Funktionsfehlern	Beispielsdatei	Kohonen-Netze	automatisches Aussortieren fehlerhafter Laufwerke	symbolische Darstellung des Klassifikationsergebnisses
ZN-LISA	akustische Erkennung von Fehlern in Bauteilen, Materialien und Fertigprodukten	Beispielsdatei mit Phonemen	Neuronale Netze	Kennzeichnung des fehlerhaften Gerätes	Anzeigen des Klassifikationsergebnisses am Benutzerinterface
DIAMOND	Diagnose von Dieselmotoren zur Beschleunigung der Reparatur	Speicherung aller motorrelevanten Daten	Diagnose mit FF-Netzen	Diagnosevorschlag u. Plausibilitätsprüfung d. Diagnose mit Hilfe von Regeln (Expertensystem)	Darstellung der Ergebnisse am Bildschirm

Tabelle 2.1: Diagnosesysteme aus der Industrie.

gabe entgegenzuwirken. Anderseits muß die Eingabe des Experten durch eine Plausibilitätsprüfung validiert werden, um mögliche Widersprüche sofort aufzudecken.

Durch die Architektur der getrennten Wissenserhebung und der Operationalisierung des Wissens ist die Erkennung von Fehlern im Diagnosesystem nur sehr verzögert möglich. Die verwendeten Lernmethoden in den aufgeführten Diagnosesystemen lassen eine andere Vorgehensweise allerdings auch nicht zu.

Demzufolge muß es das Klassifikationssystem ermöglichen, neue Informationen umgehend in die Wissensbasis zu adaptieren. Dies würde die Handhabung des Systems enorm erleichtern. Darüber hinaus wächst dadurch die Akzeptanz von Seiten des Benutzers in Bezug auf das System, da die Korrekturen und Änderungen sofort umgesetzt und nachvollzogen werden können.

Bei den vorgestellten Diagnosesystemen existiert keine geschlossene Vorgehensweise, um die Vorverarbeitung der Daten durchzuführen oder den Diagnosesystemkern auf andere Problemstellungen zu übertragen. Die Beispielsdaten werden meistens in unterschiedlichen Dateien vorgehalten, ohne daß dabei eine zentrale Datenhaltung oder Wissensbasis vorliegen würde. Darüber hinaus ist nicht geklärt, wie bereits vorhandenes Vorwissen wie Regeln, physikalische Modelle, usw. in das Diagnosesystemen integriert werden können.

Aus den oben getroffenen Aussagen lassen sich folgende Schlußfolgerungen ableiten. Ein interaktives Diagnosesystem sollte in der Lage sein, online mittels Interaktion mit dem Benutzer zusammen zu lernen. Der Benutzer hat dabei die Rolle eines Lehrers, der dem Diagnosesystem vorhandene Regeln vermittelt und die Aussagen des Systems anhand von Beispielen verifiziert. Als Voraussetzung dazu ist eine Kommunikationsschnittstelle zu sehen, die dem Benutzer sowohl die Eingabe neuen Wissens als auch die Überwachung des Diagnosesystems selbst erlaubt. Das Wissen sollte in einer Wissensbasis verwaltet werden, und das System sollte dabei weitgehend automatisiert arbeiten, um Benutzerfehler zu minimieren.

Diese Forderungen lassen sich durch das Konzept des Interaktiv Lernenden Diagnosesystems für verrauschte Inspektionsdaten umsetzen. Dabei spielt sowohl die uniforme Datenrepräsentation für das Diagnosesystem und den Benutzer als auch die unmittelbare Wissensadaption eine zentrale Rolle. Die Architektur für ein derartiges Diagnosesystem wird im folgendem Kapitel vorgestellt.

Kapitel 3

Architekturansatz für ein lernendes Diagnosesystem

Diese Kapitel führt die Architektur eines Interaktiv Lernenden Diagnosesystems (ILD-Systems) ein. Darin werden die Merkmale einer Applikation spezifiziert, bei der die Interaktiv Lernende Diagnose (ILD) eingesetzt wird. Die Architektur und deren Komponenten werden vorgestellt und erläutert. Ebenso werden die Aufgabenstellungen der Komponenten als auch die Zusammenhänge zwischen den einzelnen Komponenten spezifiziert.

3.1 Motivation eines Interaktiv Lernenden Diagnosesystems

In Kapitel 2 wurde bereits erläutert, daß die Wissensakquisition sowie Fehler der Experten die größten Fehlerquellen bei der Erstellung von Diagnosesystemen darstellen. Um dieser Problematik zu begegnen, wird in der vorliegenden Arbeit eine Lösung vorgeschlagen, mit der sich derartige Fehlerquellen weitgehend vermeiden lassen.

Basierend auf dem Phasenmodell zur Ausprägung eines Diagnosesystems wird hier eine Methode beschrieben, die einen inkrementellen Aufbau des Diagnosesystems ermöglicht, ohne dabei wesentliche Kenntnisse über die Lernalgorithmen oder die interne Darstellung des Diagnosewissens zu besitzen.

Wie bereits erwähnt, durchläuft ein herkömmliches Diagnosesystem während seiner Entstehung und Anwendung verschiedene Phasen. Dieser Prozeß wird zum Einen von den Diagnoseexperten, die das Diagnosesystem aufbauen (vgl. Abb. 2.1), zum Anderen von den Anwendern, die das Diagnosesystem benutzen, unterstützt. Diese Arbeitsphasen lassen sich dabei wie folgt beschreiben:

1. Wissensakquisition durch die Diagnoseexperten,

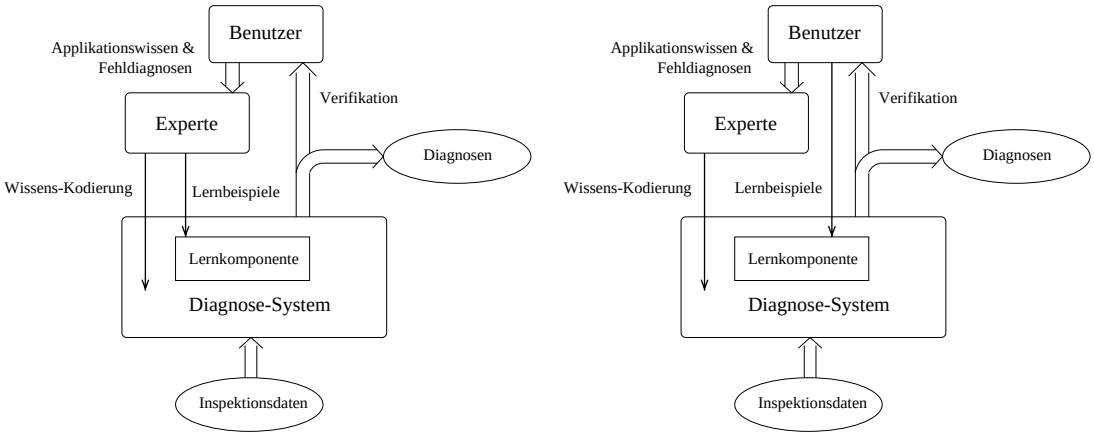
2. Kodierung des Wissens durch die Diagnoseexperten,

3. Anwendung des Diagnosesystems durch die Anwender.

Diese Art des Vorgehens ist jedoch recht starr. Entstehen durch die Anwendung des Systems neue Erkenntnisse, so muß der oben beschriebene Prozeß der Kodierung

erneut durchgeführt werden. Dabei treten oftmals Konsistenzprobleme auf. Außerdem wird die Verfügbarkeit der neuen Erkenntnisse wesentlich verzögert (siehe [SBGB93]).

Diese Konsistenzprobleme liegen dabei in der Arbeitsweise des Systems begründet. Diese ist unbeweglich und adaptiert das neu gewonnene Wissen über die Applikation nicht sofort in ihr Modell. Der Weg führt in der Regel über einen Diagnoseexperten, der das neue Wissen bewertet und es anschließend wieder dem System zuführt (vgl. Abb. 3.1(a)).



(a) Standard-Diagnosesystem

(b) Interaktiv Lernendes Diagnosesystem

Abbildung 3.1: Vorgehen bei der Erstellung eines Diagnose-Systems.

Bei einem ILD-System ist dagegen der Transformationsschritt über den Diagnoseexperten für das Einlernen des Systems nicht notwendig. Ferner kommuniziert und interagiert es direkt mit dem Anwender selbst und adaptiert seine eigene Diagnose an die Vorgaben des Benutzers.

Das System lernt dabei aus den Beispielen. Da kein Prozeßmodell existiert, ist es auch nicht möglich, Daten künstlich zu generieren, um damit das Diagnosesystem einzulernen. Der lernprozeß ergibt sich also im wesentlichen aus der Interaktion zwischen dem Diagnosesystem und dem Benutzer (interaktives lernen).

Der Begriff des *interactive learning* wurde von Morik (siehe [Mor94]) wie folgt definiert:

> "The interactive learning where the user prepares examples and background knowledge and then interact with a learning system."

Diese Definition wurde in bezug auf symbolische Systeme getroffen. In dieser Arbeit wird sie auf subsymbolische Systeme erweitert. Zusätzlich wird von einem interaktiv lernenden System verlangt, daß es inkrementell lernt (siehe auch [Mor94]), d.h. durch die Hinzunahme neuer Beispiele "vergißt" das System keine früher präsentierten Eingaben, und es ist online lernfähig, was durch eine sofortige Adaption der Beispiele vorausgesetzt wird.

Der Anwender übernimmt somit die Rolle eines Lehrers für das Diagnosesystem, das so alle notwendigen Informationen über den zu diagnostizierenden Prozeß erlernen

kann. Auf diese Weise wird anhand der Beispielsdaten, unterstützt durch die Interaktion mit dem Benutzer, konstruktiv eine Wissensbasis aufgebaut. Die eintreffenden Daten werden dabei diagnostiziert, und das Ergebnis wird dem Anwender umgehend präsentiert, damit er das System gegebenenfalls korrigieren kann. Die gewonnenen Informationen werden dann durch das Diagnosesystem auf Plausibilität gegenüber früheren Erkenntnissen verifiziert. Außerdem fließt das neu gewonnene Wissen sofort in die nachfolgenden Entscheidungen des Systems mit ein.

Auf diese Art und Weise wird sukzessive eine Wissensbasis aufgebaut, bei der das gelernte Wissen nach dessen Eingabe in der nachfolgenden Diagnose zur Verfügung steht (vgl. Abb. 3.1(b)).

Basierend auf den in Kapitel 2 eingeführten Komponenten für Diagnosesysteme, werden im folgenden zunächst die geeigneten Eingabedaten und daran anschließend die Anforderungen und Komponenten eines ILD-Systems spezifiziert.

3.2 Profil der zur ILD–Diagnostik geeigneten Eingabedaten

Die zu diagnostizierenden Rohdaten sind numerischer Natur. Dabei handelt es sich beispielsweise um diskrete Sensorwerte. Auf jeden Fall enthalten die Rohdaten keinerlei Informationen in Form von Symbolen.

Ist das Wissen nicht explizit in Form von Regeln oder Prozeduren bekannt (siehe Abschnitt 2.1.3), besteht nur die Möglichkeit, das heuristische Wissen in Form von Beispielen dem Diagnosesystem implizit zur Verfügung zu stellen, d.h. das Wissen, das noch in den Daten verborgen ist, muß direkt aus den Daten extrahiert werden. Es stellt sich dabei aber das Problem, daß die Daten a priori nicht klassifiziert sind und in großer Menge zur Verfügung stehen, so daß es unter Berücksichtigung des Kostenfaktors in keinem Verhältnis steht, die Gesamtdatenmenge manuell zu überprüfen (siehe Abschnitt 1.1).

Ein weiteres Problem bei der Verarbeitung der Daten besteht in deren Qualität. In vielen Fällen sind die Daten verrauscht oder nicht vollständig verfügbar. Aus diesem Grund werden fehlertolerante Verfahren für ihre Auswertung benötigt. Liegen dagegen eindeutige, also unverrauschte, Daten vor, so handelt es sich dabei lediglich um einen Spezialfall verrauschter Daten. Deshalb konzentrieren sich die nachfolgenden Untersuchungen auf verrauschte subsymbolische Daten.

3.3 Anforderungen an die Architektur eines Interaktiv Lernenden Diagnosesystems

Eine wesentliche Eigenschaft Interaktiv Lernender Diagnosesysteme ist ihre Adaptivität. Systeme sollen, basierend auf den bereits erworbenen Kenntnissen, die Diagnose durchführen und diese dem Benutzer präsentieren. Dieser überprüft die Ergebnisse und kann dann Korrekturen an ihnen vornehmen, falls er mit den Ergebnissen nicht einverstanden ist. Die Diagnose der nachfolgenden Ereignisse berücksichtigt bereits das neu spezifizierte Wissen (interaktives Lernen). Von dem System wird darüber hinaus auch eine gewisse Generalisierungsfähigkeit erwartet, damit nicht jede neue Kombination

von Eingangszuständen eine neue Abfrage an den Benutzer nach sich zieht, d.h. das Diagnosesystem muß in der Lage sein, von Einzelfällen ausgehend zu abstrahieren.

Damit der Benutzer und das System sinnvoll miteinander kommunizieren können, müssen die zu diagnostizierenden Daten in einer einheitlichen Repräsentation vorliegen, d.h. die Daten werden sowohl für den Benutzer als auch für das System auf die gleiche Weise präsentiert. Die gesamte interaktive Wissensakquisition beschränkt sich dann auf diese Daten.

Die gesamte Zeit protokolliert das System seine Diagnosen und schätzt deren statistische Güte ab. Diese Ergebnisse werden dann dem Benutzer mitgeteilt, um ihm Aufschluß über die aktuelle Leistungsfähigkeit der Klassifikation zu geben.

Die oben beschriebenen Systemanforderungen implizieren eine Wissensakquisition mittels interaktiver Überwachung und Eingabe. Um das Diagnosesystem möglichst leistungsfähig zu gestalten, wird das erworbene Wissen verifiziert und optimiert in die Wissensrepräsentation integriert. Damit wechseln sich also zwei Phasen ständig ab, nämlich die Interaktionsphase, in der neues Wissen erworben wird, und die Optimierungsphase, in der die Wissensrepräsentation wiederholt verifiziert und optimiert wird. Dadurch entsteht eine iterative Annäherung an das gewünschte Diagnoseergebnis.

Dieses Verhalten wird auch durch das kognitive Modell von Anderson untermauert (siehe [And77]), in dem die Phasen 2 und 3 abwechselnd durchlaufen werden. Die Phase 1 des Modells vom Anderson stellt dabei die Integration vom a priori Wissen in das Diagnosesystem dar.

Kognitives Lernmodell von Anderson

Die neueren Lerntheorien aus der kognitiven Psychologie betrachten den Lernprozeß als Wissenserwerb (vgl. [Gör95]). Den bekanntesten Ansatz hat dabei Anderson (siehe [And77]) formuliert. Dieser Ansatz beschreibt drei Phasen:

Phase 1: Das Wissen des Systems wird auf der Basis des deklarativen Wissens, also aufgrund von Regel und Fakten erworben. Aus dieser Wissensmodellierung können noch vollständig die ursprünglichen Regeln und Fakten wiedergegeben werden. Der Zugriff auf das Wissen verläuft jedoch recht ineffizient.

Phase 2: Durch Anwendung des Wissens (Training), wird das Wissen umstrukturiert, und es werden mehrere Teilschritte zu einem Schritt vereinigt. Dadurch werden beispielsweise häufig verwendete Abfolgen oder Wissensfragmente zusammengefaßt. Anderson nennt dies "knowledge compilation".

Phase 3: Die letzte Phase dient dem Feinschliff des Wissens (Anderson bezeichnet dies als "honing"). Es werden die letzten Ungereimtheiten in der Wissensrepräsentation beseitigt und eine weitergehende Optimierung der Repräsentation durchgeführt. In diese Phase wird nicht mehr der gleiche Geschwindigkeitszuwachs wie bei dem Übergang von Phase 1 zu Phase 2 erreicht, da kein neues Wissen einfließt.

Diese drei Phasen werden bis zum Abschluß des Lernprozesses immer wieder wiederholt. Der Lebenszyklus eines Diagnosesystems ist somit durch ein Spiralmodell beschreibbar (siehe Abb. 3.2).

Daraus ergibt sich für die ILD folgende Phasenunterteilung des iterativen Prozesses:

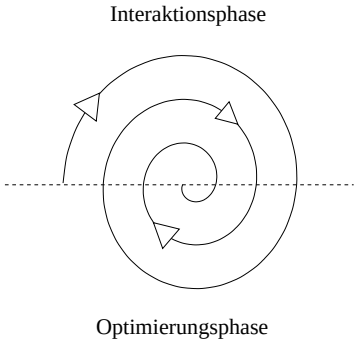


Abbildung 3.2: Spiralmodell des Lebenszyklus eines Interaktiven Lernenden Diagnosesystems.

Interaktionsphase: Interaktives Lernen des Diagnosesystems durch Korrekturen des Lehrers (Benutzers), der interaktiv die Diagnoseergebnisse überwacht.

Optimierungsphase: Reorganisationsphase zur Optimierung und Verifikation der gelernten Modelle ohne Beeinflussung durch den Benutzer.

Diese beiden Phasen wechseln sich dabei jeweils ab, so daß sich das ILD-System immer entweder in der Interaktionsphase oder der Optimierungsphase befindet. Die manuellen Korrekturen des Benutzers sollen für spätere Optimierungsschritte bzw. für die Nachvollziehbarkeit des Systemverhaltens protokolliert werden. Die Optimierungsphase wird ohne Benutzerüberwachung als Stapelprozeß ausgeführt. Auch die Optimierungen in dieser Phase, sollen protokolliert werden, um die Überprüfbarkeit des Systemverhaltens zu ermöglichen. Außerdem soll damit gleichzeitig gewährleistet werden, daß bereits erworbenes Wissen nicht verloren geht.

3.4 Komponenten eines Interaktiv Lernenden Diagnosesystems

Die Zielsetzung eines Diagonsesystems besteht darin, aus einer komplexen Datenmenge die relevanten Informationen über mögliche Probleme herauszufiltern und basierend auf diesen Informationen Entscheidungen zu deren Behebung zu präsentieren. Dies trifft ebenfalls in vollem Umfang für das ILD-System zu. Darüber hinaus basiert die ILD auf einer beispielsbasierten Methodik, wobei die Beispiele durch die Interaktion zwischen dem Benutzer und dem System akquiriert werden. Dies bietet die Möglichkeit, den Suchraum sukzessive einzuschränken.

Die ILD-System Architektur besteht aus den in Abbildung 3.3 aufgeführten und nachfolgend aufgelisteten Komponenten. Die einzelnen Komponenten des ILD-Systems realisieren die in Abschnitt 3.3 beschriebene Vorgehensweise und sind derart konzipiert, daß sie die dort spezifizierten Aufgabenstellungen erfüllen:

Datenvorverarbeitung: Umwandlung der Rohdaten in Merkmale.

Lernkomponente: Einlernen des Modells anhand von Beispielen.

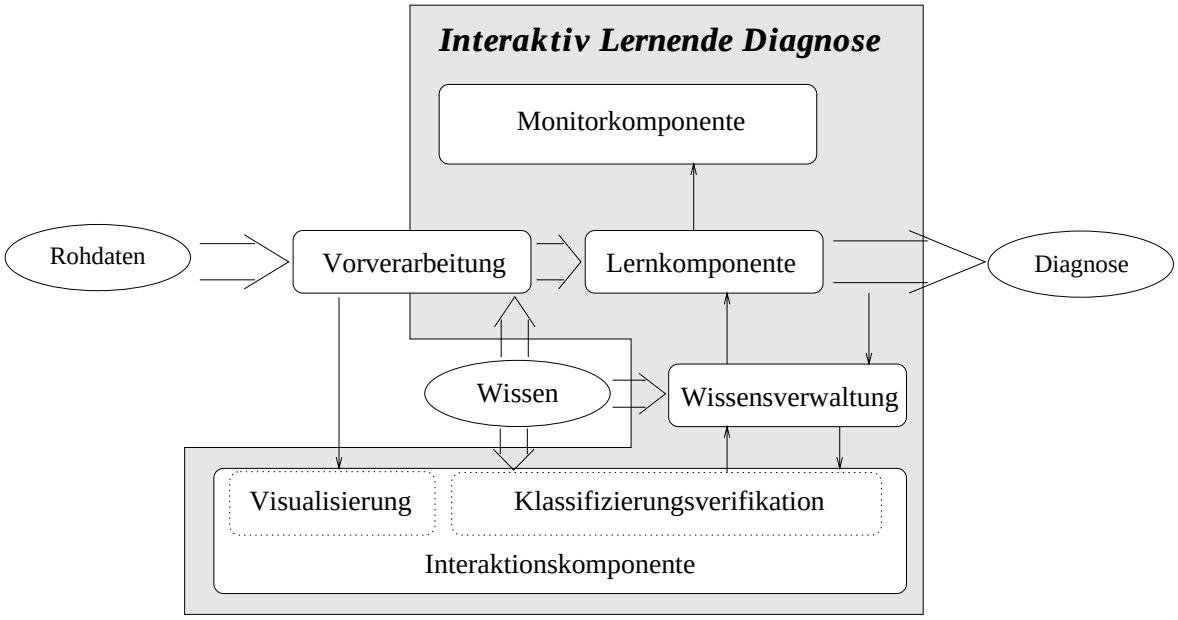


Abbildung 3.3: Architektur des Interaktiv Lernenden Diagnosesystems mit den unterschiedlichen Komponenten. Die grau unterlegten Bereiche sind Teile des Diagnosesystemkerns.

Wissensverwaltungskomponente: Speicherung der vom Anwender klassifizierten Daten sowie der von ihm vorgegebenen Regeln.

Monitorkomponente: Auskunft über den Zustand des Diagnosesystems, Güteabschätzung sowie Detektion von Widersprüchen.

Interaktionskomponente: Datenvisualisierung, Verifikation und Anpassung der Klassifikation.

Im nachfolgenden wird nun auf die Anforderungen an die einzelnen Komponenten näher eingegangen.

3.4.1 Vorverarbeitung der Rohdaten

Die Rohdaten $\mathcal{D}$ müssen in Merkmale $\mathcal{M}$ (siehe Kapitel 4) konvertiert werden, damit sie von der nachfolgenden Lernkomponente verarbeitet werden können. Dies ist notwendig, da die Rohdaten oft in einem Format vorliegen, das von der Lernkomponente nicht verarbeitet werden kann.

Dabei kann die Gewinnung der subsymbolischen Merkmale ($\mathcal{M}$) aus den unverarbeiteten Rohdaten ($\mathcal{D}$) als Verarbeitungsfunktion (V) betrachtet werden:

$$V : \mathcal{D} \longrightarrow \mathcal{M} \tag{3.1}$$

Die Funktion V der Rohdaten auf die subsymbolischen Merkmale wird im wesentlichen durch die Umsetzung des vorhandenen Wissens über die Applikation in Algorithmen durchgeführt. Diese Umsetzung enthält eine geeignete Auswahl von "Standard"-Signalverarbeitungsalgorithmen und deren Kombination mit applikationsspezifischen Operatoren. Dabei sollen folgende Fragestellungen untersucht werden, die den Aufbau der Vorverarbeitung wesentlich vereinfachen können:

- Lassen sich Aussagen über die Rohdaten $\mathcal{D}$ gewinnen, die den Aufbau der Funktion V vereinfachen können?
- Ist es möglich, die Qualität der Merkmale $\mathcal{M}$ zu erfassen ohne Kenntnisse über die Lernkomponente?
- Existiert eine Möglichkeit zur Analyse der Komplexität der Merkmale $\mathcal{M}$?

Darüber hinaus wird exemplarisch untersucht, wie Merkmale mit Hilfe der Signalverarbeitung, statistischer Ansätze, der Informationstheorie, empirischer und heuristischer Funktionen extrahiert werden können. In Kapitel 4 wird aufgezeigt, wie die gewonnenen Kenngrößen für die Monitorkomponente verwendet werden können.

3.4.2 Lernkomponente

Die Lernkomponente ist verantwortlich für die korrekte Klassenzuordnung zu den Merkmalen (siehe Abschnitt 2), also für die Klassifikation. Die Klassifikation bildet das Fundament, auf dem sich die Diagnose aufbaut. Sie beschäftigt sich mit der Zuordnung der Merkmale aus dem Merkmalsraum $\mathcal{M}$ zu bestimmten vorgegebenen Klassen $\mathcal{C}$. Diese Klassifikationsfunktion K kann dann folgendermaßen beschrieben werden:

$$K : \mathcal{M} \longrightarrow \mathcal{C} \tag{3.2}$$

Diese Funktion ist im allgemeinen nicht surjektiv, da durch die Klassifikation mehrere Elemente aus dem Merkmalsraum $\mathcal{M}$ auf den Vektor[1] $\vec{c} \in \mathcal{C}$ abgebildet werden. Dadurch entsteht ein Informationsverlust bzw. eine Informationsreduktion auf genau die resultierenden Klassen $\vec{c}$.

Bei der Lernkomponente besteht das Ziel in der Ermittlung der Abbildung K durch einen Lernalgorithmus. Lernen kann prinzipiell auf drei Arten durchgeführt werden: überwacht, unüberwacht oder als sog. Reinforcement-Lernen (siehe [HKP91, Bra97]). Jede dieser Lernarten basiert auf Beispielen. Der Lernprozeß kann dabei als eine Suche im Parameterraum des Lernalgorithmus aufgefaßt werden.

Die Diagnose besteht in ihrem Kern aus der Zuordnung von Merkmalen $\vec{x} \in \mathcal{X}$ zur Klassen $\vec{c} \in \mathcal{C}$. Genau diese Korrelation kann durch einen überwacht arbeitenden Lernalgorithmus gelernt werden. Im weiteren werden die bekannten Zuordnungen als Beispiele bezeichnet.

Definition 3.1 (Beispiele)

$$B := \{(\vec{x}, \vec{c}) | \vec{x} \in \mathcal{X}, \vec{c} \in \mathcal{C}\}$$
$$mit \qquad \mathcal{X} \subseteq \mathcal{M}$$

Die Aufgabe eines überwachten Lernalgorithmus lautet also wie folgt:

> *Wie findet man bei bekannten Beispielen $\vec{b} \in \mathcal{B}$ eine Abbildung K, die für alle oder möglichst viele Beispiele $\vec{b}$ gilt?*

[1] Die Klassifikationen werden als Vektoren von Klassen betrachtet, damit auch Mehrfachklassifikationen oder Nichtklassifikationen auf einfache Art und Weise ausgedrückt werden können.

Das Hauptziel bei der Erstellung einer Lernkomponente besteht in der Automatisierung der Wissensakquisition und des Lernprozesses. Die Lernkomponente ist derart strukturiert, daß sie die durch den Benutzer spezifizierte Beispiele direkt integriert und dieses Wissen für die nachfolgende Bearbeitung verfügbar macht.

Außerdem sollte der Entwickler während der Trainingsphase zu jeder Zeit die Gütestatistik abfragen können, damit er interaktiv in den Lernprozeß eingreifen kann, um so ein zielgerichteteres Lernen zu ermöglichen. Dadurch kann der Lernvorgang gerade bei praxisnahen Anwendungen beschleunigt werden, und die Gewährleistung der Funktionalität des Erlernten wird damit erhöht.

Bereits vorhandenes Wissen in Form von Modellwissen, sogenanntes a priori Wissen[2], muß darüber hinaus in die Lernkomponente über eine interaktive Schnittstelle integriert werden können.

Ausgehend von diesen Rahmenbedingungen ergeben sich folgende Anforderungen an die Lernkomponente:

Interaktives Lernen: Es muß möglich sein, die Eingaben des Experten zu "beobachten", um aus ihnen online und inkrementell zu lernen.

Keine Steuerungsparameter: Durch die Vermeidung von Methoden die benutzerdefinierte Steuerungsparameter benötigen, bzw. das Ausnutzen einer Strategie für das Auffinden der richtigen Steuerungsparameter muß es möglich sein, den Lernvorgang weitgehend zu automatisieren. Damit braucht der Benutzer kein Wissen über die verwendeten Algorithmen zu besitzen.

Korrekturfähigkeit: Getätigte Eingaben / Hypothesen können auch zum späteren Zeitpunkt wieder korrigiert werden.

Anwendungsunabhängigkeit: Es sollen Methoden und Strategien gewählt werden, die applikationsunabhängig arbeiten, damit das ILD-System auf eine große Klasse von Applikationen angewendet werden kann.

Generalisierungsfähigkeit: Die Lernkomponente stellt automatisch Hypothesen über unbekannte Eingaben auf und bewertet die Qualität dieser Hypothesen.

Automatische Hypothesenüberdeckung: Nicht alle Beispiele sollen vom Experten verifiziert werden müssen, sondern es sind nur so viele Beispiele zu kontrollieren, daß der gesamte Eingaberaum hinreichend abgedeckt ist. Für diese Aussage müssen entsprechende Bewertungskriterien erarbeitet werden.

Monitorschnittstelle: Der Benutzer kann sich immer über den internen Zustand der Lernkomponente informieren. Ein qualitatives Maß für die Güte des Klassifikators wird bereitgestellt.

Integration von Applikationswissen: In vielen Fällen existiert ein gewisses Maß an Bereichs- und Applikationswissen in Form von Regeln über die Anwendung, das auf einfache Weise in die Lernkomponente integrierbar sein muß.

[2]Dabei handelt es sich beispielsweise um Ausnahmesituationen, wie das Auslösen eines Endabschalters einer Maschine. Es handelt sich dabei um Wissen, das es sich nicht zu erlernen lohnt bzw. zum Erlernen zu kostspielig ist.

Mit diesen Eigenschaften kann die Lernkomponente eine umgehende Adaption des Wissens gewährleisten. Dazu muß jedoch auch eine geeignete Wissensbasis mit einer entsprechenden Verwaltunskomponente existieren, die das erlernte Wissen vorhält und ggf. Korrekturen ermöglicht.

3.4.3 Wissensverwaltung

Die Wissensverwaltung dient dem Erhalt des Applikationswissens über die gesamte Lebensdauer des Diagnosesystems. Da das Wissen aus verschiedenen Quellen stammen kann (vgl. Abschnitt 2.1.3), muß dies auch durch die Wissensverwaltung aufgenommen werden können. Dies umfaßt sowohl das Modell-/Regelwissen, die a priori bekannten Beispiele als auch die während der Benutzung des Systems gesammelten Beispielsdaten und Regeln. Darüber hinaus werden die Konfigurationen der Applikation gesichert, um die Nachvollziehbarkeit der Beispielsdaten in Abhängigkeit der jeweiligen Applikationskonfiguration sicher zu stellen.

Die Wissensverwaltung muß außerdem in der Lage sein, die Konsistenz des Wissens sicher zu stellen und etwaige Widersprüche in den Beispielen aufzuzeigen.

Näheres zur Wissensverwaltung ist in Kapitel 6 beschrieben.

3.4.4 Monitorkomponente

In dem ILD-System ist es sinnvoll, daß der Benutzer jederzeit den Status der einzelnen Komponenten abfragen kann. Sowohl die Resultate der Vorverarbeitung, die der Lernkomponente als auch der Zugriff auf die Wissensbasis müssen dem Benutzer zur Verfügung stehen, damit er die Entscheidungen des Systems nachvollziehen kann. Des weiteren werden dem Benutzer Informationen zugänglich gemacht, mit deren Hilfe es ihm ermöglicht wird, eine qualitative Aussage über den Zustand der Komponenten des Diagnosesystems zu treffen.

Entscheidungen des Systems können durch die verschiedenen Entscheidungs- und Vorverarbeitungsinstanzen verfolgt werden. Neben der Möglichkeit den Datenfluß nachzuvollziehen, ist auch die Sicht auf die Modelle und Parameter der Komponenten transparent gestaltet. Damit wird dem Benutzer Einblick in die interne Sicht des Diagnosesystems gewährt, sowie dessen Arbeitsweise und dessen Entscheidungsfindungsprozeß zugänglich gemacht.

In Kapitel 6 wird darauf näher eingegangen.

3.4.5 Interaktionskomponente

Das in dieser Arbeit angewandte interaktive Lernen zielt darauf ab, die Expertenfehler zu minimieren. Aus diesem Grund ist die Wissensakquisition zwischen der automatisierten Methodik und der direkten Methodik einzuordnen (siehe Abb. 2.6), da ein wesentlicher Teil automatisiert ist und ein geringer durch strukturierte Benutzerinteraktion abgedeckt wird.

Die Möglichkeiten der Interaktion in der Interaktionskomponente werden im folgenden auf die Überwachung und Korrektur der Diagnose beschränkt. Demzufolge übernimmt der Benutzer mittels der Interaktionskomponente die Rolle eines Lehrers.

Die Interaktionskomponente ist zuständig für die Kommunikation zwischen dem Diagnosesystem und dem Benutzer. Einerseits werden die Ergebnisse und die Daten,

auf denen die einzelnen Diagnoseschritte basieren, dem Benutzer präsentiert. Andererseits wird das vom Benutzer einzugebende Wissen an das Diagnosesystem übermittelt. Wesentlich dabei ist die einheitliche Sprache beider Parteien. Das bedeutet, daß das Diagnosesystem und der Benutzer die gleiche Terminologie benutzen, um gleiche Sachverhalte auszudrücken.

Dazu gehört insbesondere eine geeignete einheitliche Präsentation der Daten sowie eine einheitliche standardisierte Eingabe von Wissen bezüglich dieser Daten. Diese Punkte werden im Kapitel 6 ausführlich diskutiert.

3.5 Resümee der ILD – Architektur

Die ILD weist aufgrund der gesetzten Schwerpunkte wie Verbesserung der Wissensakquisition und Reduktion von Expertenfehlern eine fünfkomponentige Struktur auf:

1. Vorverarbeitung der Daten zur Eingabe für die Lernkomponente.
2. Lernkomponente, mit deren Hilfe das Modell anhand von Beispielen eingelernt werden wird.
3. Wissensverwaltung zur Vorhaltung und Erweiterung der Wissensbasis (siehe auch Kapitel 6).
4. Monitorkomponente, die die Entscheidungen des Systems nach außen hin für den Benutzer verständlich darstellt (vgl. dazu Kapitel 6).
5. Interaktionskomponente mit standardisierter Eingabe / Präsentation, die die Kommunikation zwischen Benutzer und dem Diagnosesystem in einer einheitlichen Form unterstützt.

Ein wesentliches Merkmal des ILD-Systems ist dabei die Interaktionskomponente, die durch ihre einheitliche Kommunikation zwischen dem Benutzer und dem System, den Lernvorgang dahingehend unterstützt, neues Wissen, aber auch Wissenskorrekturen über die Wissensverwaltung umgehend dem System zuzuführen. Auf diese Weise wird gewährleistet, daß das veränderte Wissen den nachfolgenden Entscheidungen des Diagnosesystems bereits zur Verfügung steht. Durch die Wissensverwaltung wird das Wissen für die Lernkomponente und den Benutzer persistent vorgehalten (Näheres siehe dazu Kapitel 6).

Darüber hinaus kann der Zustand der Systemkomponenten mittels einer Monitorkomponente überwacht werden. Sie liefert Aussagen über die Qualität des Systems, stellt also eine Art Selbstbewertung des Systems für den Benutzer dar. Diese Komponente wird im Kapitel 6 näher betrachtet.

Die Lernfähigkeit aus Beispielen und die automatische Optimierung des internen Diagnosemodells sind ebenfalls wesentliche Merkmale der ILD. Daraus ergibt sich die Zweiphasigkeit der ILD mit der Interaktionsphase, in der die Wissensbasis aufgebaut wird einerseits, und der Optimierungsphase zur Reorganisation des Modells andererseits.

Dies führt dazu, daß die andere zentrale Komponente des ILD-Systems – die Lernkomponente – folgende Eigenschaften aufweisen muß (siehe auch Kapitel 5):

- Interaktives Lernen durch direkte Umsetzung der Beispiele,

- Abwesenheit von Steuerungsparametern für den interaktiven Lernvorgang,
- Applikationsunabhängigkeit der verwendeten Lernalgorithmen,
- Generalisierungsfähigkeit der Lernalgorithmen,
- Automatische Generierung von Hypothesenregionen,
- Monitorschnittstelle für die Erläuterungen der getroffenen Klassifikationen und
- die Integration von Applikationswissen in den Klassifikator anhand bekannter Regeln .

Durch diese einzelnen Komponenten des Interaktiven Lernenden Diagnosesystems wird der Benutzer in die Lage versetzt, bei verrauschten subsymbolischen Inspektionsdaten, das ILD-System online und interaktiv einzulernen. Dabei wird sowohl der Entwicklungsprozeß beschleunigt, als auch die Fehler bei der Identifikation der Diagnosefehler durch die interaktive Vorgehensweise stark reduziert. Zur Verifikation steht die konkrete Implementierung dieser Architektur in Form des ***CMS***–Baukastens (siehe Anhang A) zur Verfügung.

Als erstes wird bei dem Datenfluß durch das ILD-System die Vorverarbeitung durchgeführt, auf die im folgenden Kapitel näher eingegangen wird.

Kapitel 4

Prinzipien der Vorverarbeitung verrauschter mehrdimensionaler Inspektionsdaten

Als Vorstufe zu einem interaktiven beispielsbasierten Einlernen durch ein Diagnosesystem müssen die Rohdaten durch die Vorverarbeitung aufbereitet und in eine Form konvertiert werden, die der Lernkomponente eine Weiterverarbeitung ermöglicht. Dabei läßt sich diese Vorverarbeitung in sechs Phasen unterteilen, die auch in dieser Reihenfolge durchlaufen werden (siehe Abbildung 4.1).

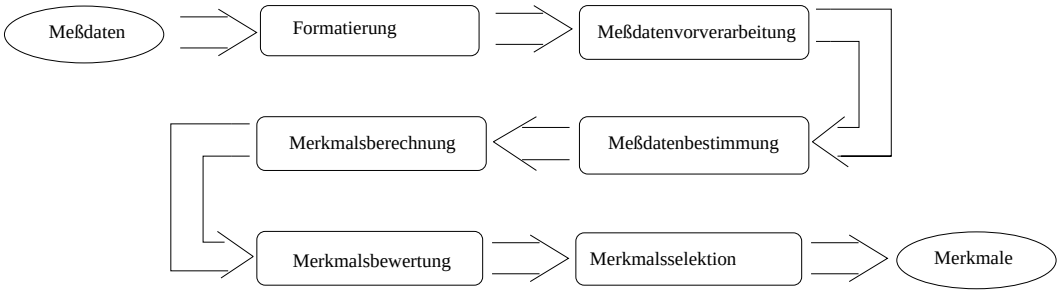


Abbildung 4.1: Phasen der Vorverarbeitung von Inspektionsdaten.

Die Phasen der Merkmalsbewertung und der Merkmalsbestimmung werden nur in der Aufbauphase des Diagnosesystems durchgeführt, da sie ausschließlich im Zusammenhang mit dem Einlernprozeß der Lernkomponente benutzt werden.

Bei der Vorverarbeitung sind insbesondere folgende Fragestellungen zu klären:

- Welche Daten stehen zur Verfügung, und in welcher Reihenfolge können sie verarbeitet werden? (Charakterisierung der Ein-/Ausgaben)
- Kann Modellwissen in der Vorverarbeitung der Daten verwendet werden?
- Welche Merkmale sollen extrahiert werden?
- Welchen Informationsgehalt bieten die Merkmale bzgl. der Klassifikation, und wie kann ggf. eine Bestimmung durchgeführt werden?

Darüber hinaus muß geklärt werden, ob eine geeignete modellunabhängige Aussage über den Informationsgehalt zur Untersuchung bzw. Bewertung der Daten existiert. Hierauf aufbauend können die richtigen Methoden zur Ermittlung der Merkmale ausgewählt werden. Diese Auswahl kann zum einen aufgrund vorhandenen Vorwissens über die Anwendung durchgeführt werden, zum anderen können neue Anforderungen zur Erstellung der Merkmale auch von Seiten des Diagnosesystems erfolgen. Durch die Kombination dieser beiden Methoden zur Merkmalsbestimmung kann dann in einem iterativen Prozeß die Merkmalsauswahl verbessert werden.

Die Methoden der Vorverarbeitung werden im folgenden mit Hilfe eines Applikationsbeispiels untermauert.

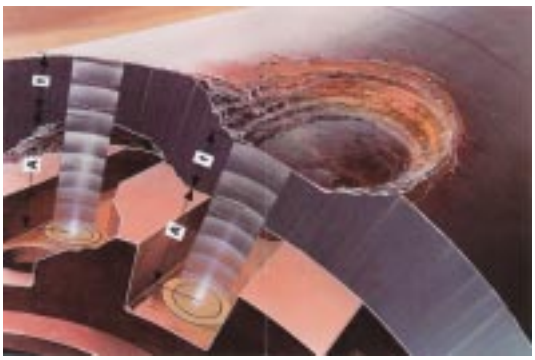
4.1 Applikationsbeispiel: Ultraschallbasierte Inspektion von Pipelines

Der im Abschnitt 1.1 beschriebene UltraScan-Molch für die Pipelineinspektion basiert auf Ultraschallsensoren. Das Prinzip der Ultraschallmessung kann der Abbildung 4.2 entnommen werden. Bei dieser Messung wird die Laufzeit des ersten und des zweiten Echos gemessen, wodurch die Entfernung des Sensors zur Innenwand und die Wanddicke ermittelt werden können. Der Molch ist mit je einem Sensor pro 8mm in Umfangsrichtung ausgestattet, und durch die Flüssigkeit in der Pipeline wird er auf einer Geschwindigkeit von ca. 1m/s gehalten. Dabei werden alle Sensoren etwa 300 mal gefeuert.

Dadurch wird eine Messung pro 24mm^2 der Rohrinnenwand durchgeführt. Es werden mit bis zu 512-Ultraschallsensoren bestückte Molche dieses Typs verwendet, die mit einer Frequenz von 400kHz feuern. Durch die Kodierung ergibt sich eine Auflösung von ca. 0.225mm für den Wanddicken- und den Sensorabstandswert. Die Messungen werden in konstanten Zeitabständen durchgeführt, was bei einer optimalen Geschwindigkeit von einem Meter pro Sekunde in einer Messung eines Sensors pro 3mm Wegstrecke resultiert. Bei dieser optimalen Geschwindigkeit des Molchs ergibt sich für eine 100km lange Pipeline mit einem Durchmesser von 28 Zoll = 71.12 cm ein Datenvolumen von ca. 20 GByte. Da der Molch aber auch still stehen oder durch Berg- und Talfahrten verlangsamt bzw. beschleunigt werden kann, muß mit einem höheren Datenaufkommen gerechnet werden. Es kann also nicht von der Einhaltung einer idealen Geschwindigkeit ausgegangen werden. Dies wirkt sich auf die Verzerrung der Meßdaten aus, die durch das Diagnosesystem kompensiert werden müssen.

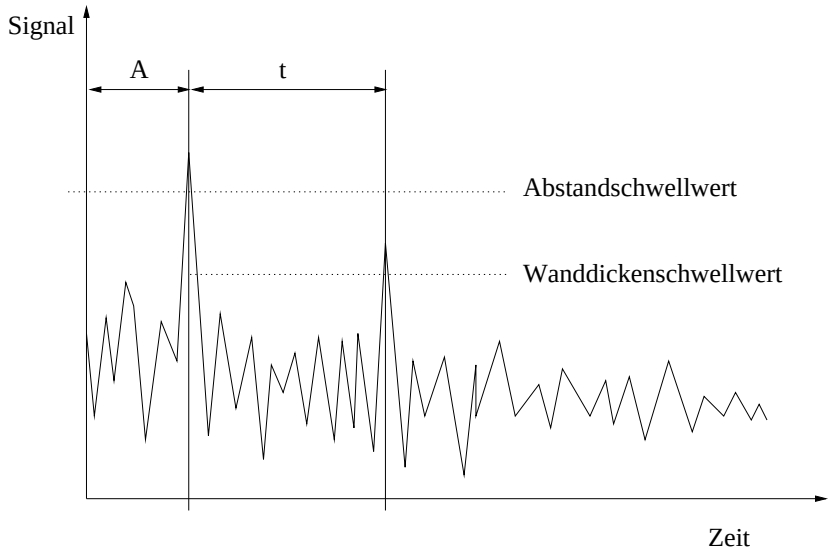
Die Meßwerte für das erste und zweite Echo jeder Messung werden komprimiert auf Datenträgern abgelegt. Nach Beendigung des Laufs werden die Messungen in einem speziellen Format auf viele Dateien aufgeteilt. Dabei erfolgt die Feuerung der Ultraschallsensoren in einer derartigen Reihenfolge, daß kein Übersprechen zu stande kommt. Durch die diagonale Anordnung der Sensoren auf dem Sensorträger (siehe Abbildung 4.2(b)) entsteht eine verzerrte Aufnahme. Um die Meßwerte ihrer korrekten Meßposition zuzuordnen, wird ein Odometriesignal aufgezeichnet, das in äquidistanten Abständen durch Laufräder erzeugt wird.

Je nach Molchtyp werden die Meßwerte bereits durch die Aufnahmeelektronik zu Gruppen von 64 Sensoren zusammengefaßt und jeweils in Matrizen mit der Länge von 512 Werten zu Blöcken komprimiert. Abhängig von Anzahl der vorhandenen Sensoren wird eine entsprechende Anzahl von Blöcken für die Aufnahme eines kompletten



(a) Ultraschallmeßverfahren mit Laufzeitdifferenzen A (erstes) und t (zweites Echo).

(b) Sensorkopf eines 28Zoll UltraScan–Molchs.



(c) Signalverlauf einer Ultraschallmessung. Mittels voreingestellter Schwellwerte werden die Peaks des ersten Echos von der Rohrinnenwand und des zweiten Echos von der Rohraußenwand detektiert. Aus den Laufzeiten werden die Laufzeitdifferenzen A für den Abstand und t für die Wanddicke ermittelt.

Abbildung 4.2: Sensoren eines UltraScan–Molchs.

Sensorumfangs benötigt. In Laufrichtung werden 64 Blöcke in eine Datei geschrieben. Für einen 28 Zoll Molch, der mit 256 Sensoren ausgestattet ist, sind es demzufolge 256 Blöcke pro Datei.

Diese Rohdaten werden durch das ***NeuroPipe I***–System (vgl. Abschnitt 7.1) aufbereitet und können als zwei normierte C-Scans dargestellt werden (siehe Abbildung 4.3).

Bei den Messungen in der Pipeline können durch die Zerstreuung des Ultraschallsignals die Signalpegel derart abgeflacht werden, daß eine Zuordnung zum ersten bzw. zweiten Echo nicht möglich ist. Solche Fälle werden als Signalverluste bezeichnet. Neben anderen Quellen treten diese Ausfälle vermehrt an den Schweißnähten auf. Dies ist

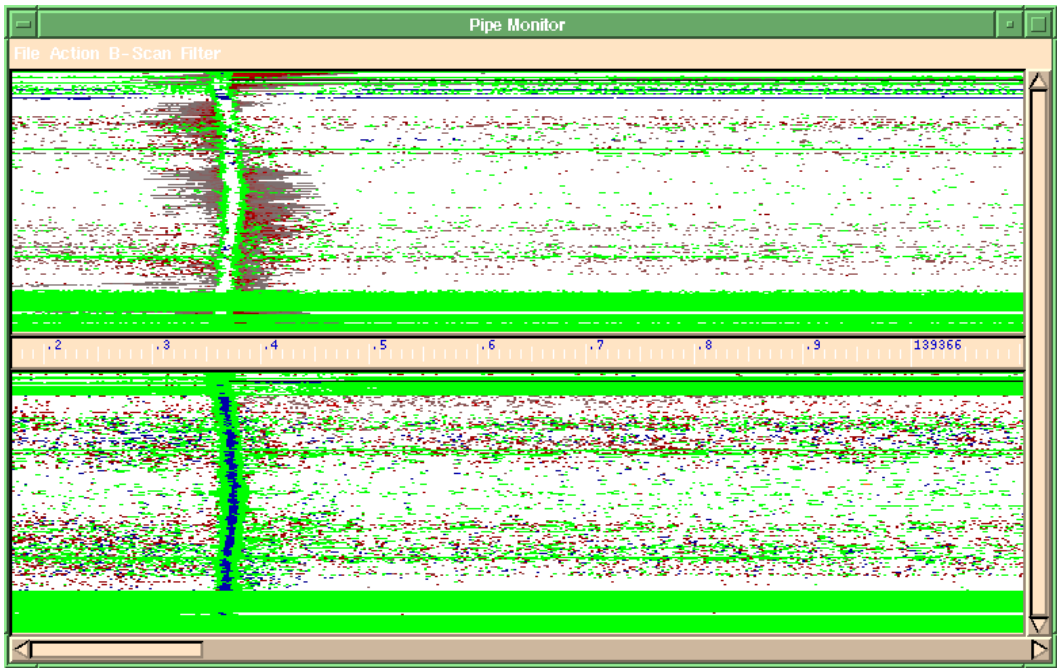


Abbildung 4.3: Ultraschallscan-Falschfarbendarstellung (C-Scan) eines Rohrabschnittes einer Erdölpipeline aus dem ***NeuroPipe I***–System. Die obere Hälfte stellt den Abstand des Sensors zur Wand und die untere Hälfte die Wanddicke dar. Dabei wird das Rohr in Längsrichtung an der 12 Uhr Position für die Darstellung aufgeklappt.

in Abbildung 4.3 zu sehen: auf der linken Seite verläuft eine leicht gekrümmte vertikale Linie (Rundnaht) mit grauen Rändern (Signalverluste) durch beide C-Scans. Signalverluste befinden sich auch im unteren Bereich als dicke horizontale Balken und mitten im Bild als vereinzelte und kleine Punktgruppierungen.

Darüber hinaus bestehen Verfälschungen der Messung mit den sog. Meßsperrzeitfehlern, die durch das Messen des eigenen Ultraschallschusses entstehen. Das Meßfenster muß daher sehr schmal eingestellt werden, damit auch sehr geringe Entfernungen erfaßt werden können. Durch die variierenden Bedingungen (Druck, Temperatur, Abrieb) in der Pipeline verändern sich allerdings die Laufzeiten des Signals, womit ein Übersprechen zu stande kommen kann.

4.2 Charakterisierung der Meßdaten und der Applikation

Bei der Vorverarbeitung der Meßdaten wird in einem ersten Schritt eine Analyse dieser Daten durchgeführt, um sie so in eine geeignete interne Repräsentation überführen zu können. Zu diesem Zweck wird im folgenden ein System zur Klassifikation der Daten eingeführt, das helfen soll, anhand der Klassifizierung die richtigen Meßdatenvorverarbeitung- und Merkmalsextraktionsverfahren auszusuchen.

Die Meßdaten auf denen das ILD-System operiert, können unterschiedliche Typen und Strukturen (Arrays, Listen) besitzen. Alle im ILD-System beteiligten Algorithmen kennen den Typ und die Struktur der Daten, die sie verarbeiten und erzeugen (vgl. Anhang A). Dadurch wird ein Konfigurationsbaum der anwendbaren Algorith-

men aufgespannt, in dem die Verarbeitungsreihenfolge durch die Typen und verarbeiteten Strukturen festgelegt ist. Diese Typenhierarchie wurde durch den sogenannten Connectionst Modell Simulator (***CMS***) umgesetzt (siehe Abschnitt A.1.2.3), um die Algorithmen miteinander zu koppeln.

Die Wurzel dieses Konfigurationsbaums bilden die Eingabedaten des Diagnosesystems, die in der Regel von einer Vielzahl von Sensoren geliefert werden. Damit verbunden ist eine große Datenvielfalt, bedingt durch die unterschiedlichen Meßprinzipien. Deshalb muß unterschieden werden, ob zeitdiskrete oder kontinuierliche Meßdaten vorliegen, oder ob statische bzw. dynamische Messungen erfolgt sind, und um welche Wertebereiche es sich bei diesen Daten handelt.

Ein Beispiel dafür ist die Zeitreihenanalyse, bei der die Anzahl der Eingaben nicht von vornherein bekannt ist. Andererseits existiert bei einer Vielzahl von Applikationen eine fixe Anzahl von Eingaben. Beispielsweise sind bei der Ultraschalldiagnose von Pipelines bis zu 512 Sensoren mit jeweils zwei Meßwerten über den Ort beteiligt.

Damit sie durch das ILD-System ausgewertet werden können, müssen sie zunächst einer Formatierung unterzogen werden.

4.3 Formatierung der Meßdaten

Zunächst werden die Meßdaten aus ihrer Rohdarstellung (siehe Abbildung 4.4) in das interne Format konvertiert.

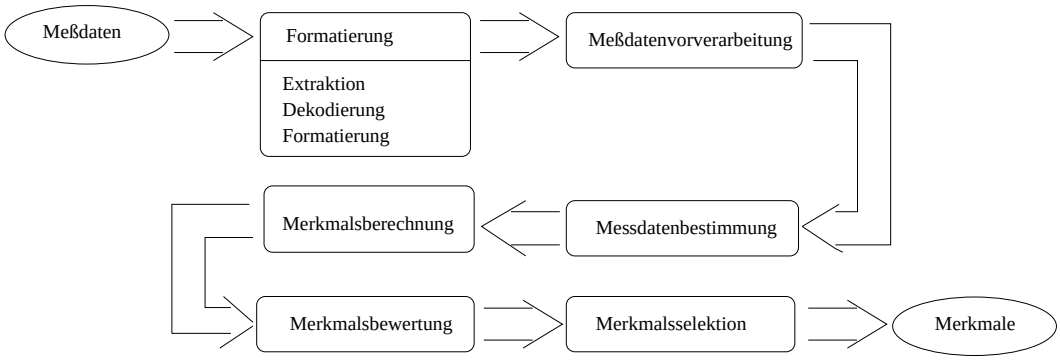


Abbildung 4.4: Formatierung der Meßdaten in eine standardisierte interne Darstellung für die weitere Vorverarbeitung.

Dazu ist die Modellinformation über die Position der Rundnähte[1] notwendig. Diese Information wird durch eine vorherige Analysephase des ***NeuroPipe I***–Systems berechnet (siehe auch Abschnitt 4.6.1). Mit Hilfe dieser Distanzangaben und des angefragten bzw. des zu diagnostizierenden Abschnittes werden diejenigen Dateien ermittelt, die diesen Rohrabschnitt beinhalten (siehe Abbildung 4.5). Die Dateien werden dekomprimiert und in einer Matrix aus zweidimensionalen Feldern gespeichert. Die Felder enthalten die dekomprimierte Information der Blöcke aus den Dateien. Damit ist die Größe der Matrix von der Anzahl der Sensoren pro Block, und der Anzahl der Ultraschallaufnahmen auf der zu analysierenden Wegstrecke abhängig.

[1] Als Rundnähte werden diejenigen Schweißnähte bezeichnet, durch die zwei Rohrsegmente zusammengefügt werden.

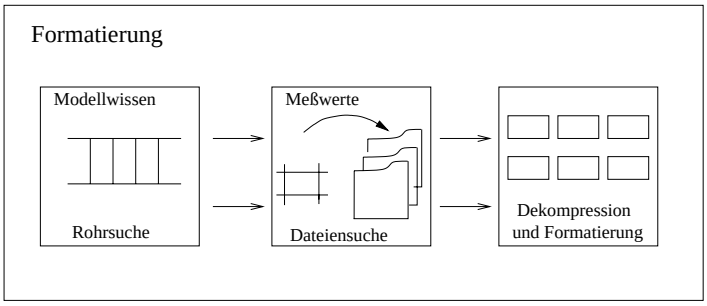


Abbildung 4.5: Formatierung der Ultraschall-Meßdaten in eine standardisierte interne Darstellung für die weitere Vorverarbeitung.

Dabei ist die diagonale Anordnung der Sensoren zu berücksichtigen, da die in einer Spalte eines Blocks gespeicherte Information dem Feuerungszeitpunkt der Ultraschallsensoren entspricht. Durch die Geometriebeschreibung des Sensorkopfes ist es erst im nachfolgenden Schritt möglich eine distanznormierte Darstellung zu ermitteln. Diese Geometrie muß aber bereits in der Formatierung berücksichtigt werden, um alle hierfür notwendigen Daten zur Verfügung zu stellen.

Wie bereits erwähnt, werden für die Distanzmessung Odometerräder verwendet, die pro Umdrehung n Signale liefern, die den jeweiligen Messung zu diesem Zeitpunkt zugeordnet werden. Anhand diese Signale kann die Distanzlokalisation durchgeführt werden.

Die Felder in der Matrix repräsentieren die Information der Blöcke aus den Dateien. Die Blöcke aus den Dateien werden dazu dekomprimiert, wobei vier übereinander liegende zweidimensionale Felder mit jeweils 7 Bit Auflösung entstehen (siehe Abbildung 4.2(a),(c)):

Sensorabstand: Kodiert die Entfernung des Sensors von der Innenwand des Rohres. Die Sensoren besitzen durch ihre Anbringung immer einen Mindestabstand. Damit wird die Hälfte der Laufzeit des ersten Echos, das über einer vordefinierten Schwelle liegt, als Sensorabstand gemessen. Da der Molch immer in einer Flüssigkeit schwimmt, liegt die Auflösung bei ca. 0.225mm.

Wanddicke: Stellt die Dicke der Metallrohrwand mit einer Auflösung von ca. 0.225mm dar. Dazu wird die Hälfte der Laufzeitdifferenz zwischen dem ersten und zweitem Echo kodiert.

Kompressionswert des Sensorabstandes: Der Kompressionswert wird für jedem Sensor separat als gleitender Mittelwert bereits auf dem Molch ermittelt. Er dient dazu, alle Werte, die sich in einem vorgegebenem Abstand zu dem Kompressionswert befinden, auf diesen abzubilden, wodurch "Lücke" in dem Wertebereich entsteht. Durch das Herausfiltern dieser kleinen Abweichungen werden sehr hohe Kompressionsraten erzielt.

Kompressionswert der Wanddicke: Für die Kompression der Wandwerte wird ein gleitender Mittelwert über alle Sensoren eines Blocks und in einer vordefinierten Länge durchgeführt. Dieser Kompressionswert wird dazu verwendet, um alle Meßwerte in einer vordefinierten Umgebung auf diesen Kompressionswert abzubilden. Durch diese Schwellwertfilterung wird die Kompression erheblich verstärkt.

Durch die Kompression entsteht zwar eine wesentlich kleinere Datenmenge, was für die Speicherung auf den Datenträgern im Molch von großem Interesse ist, gleichzeitig entsteht eine Lücke im Wertebereich, die sich in bestimmten Situationen nachteilig auswirkt. Ein Beispiel hierfür sind die gewalzten Rohre, bei denen die Wanddicke herstellungsbedingt schwankt. Diese Schwankung ist derart groß, daß sie aus dem Kompressionsfenster herausfällt und dadurch in den späteren Phasen u.U. unnötige Anomalieregionen erzeugt (siehe Abschnitt 4.5).

Bei einen 28 Zoll Molch, der mit einer durchschnittlichen Geschwindigkeit von 1m/s und mit 256 Sensoren ein 12m langes Rohr aufnimmt, werden die Daten in ca. 36 Feldern mit jeweils $4*7$Bit gespeichert, die in einer vierzeiligen Matrix angeordnet sind. Demzufolge beträgt die Datenmenge für ein 12m langes Rohr $(64*4)*(512*9)*(4*7)$Bit = 33030144Bit = 4128768Byte. Diese Datenmenge wird durch die anschließende Vorverarbeitung derart manipuliert, daß Merkmale extrahiert werden können.

4.4 Vorverarbeitung der Meßdaten

Unter der Vorverarbeitung der Meßdaten ist eine Aufbereitung und Transformation der Daten zu verstehen, bei der die Dimensionalität der Daten erhalten bleibt. Durch Transformationen (Bildung von Ableitungen, FFT, etc.) und Datenfusion können neue Repräsentationen bzw. Sichten bzgl. der Daten generiert werden (siehe auch Abbildung 4.6).

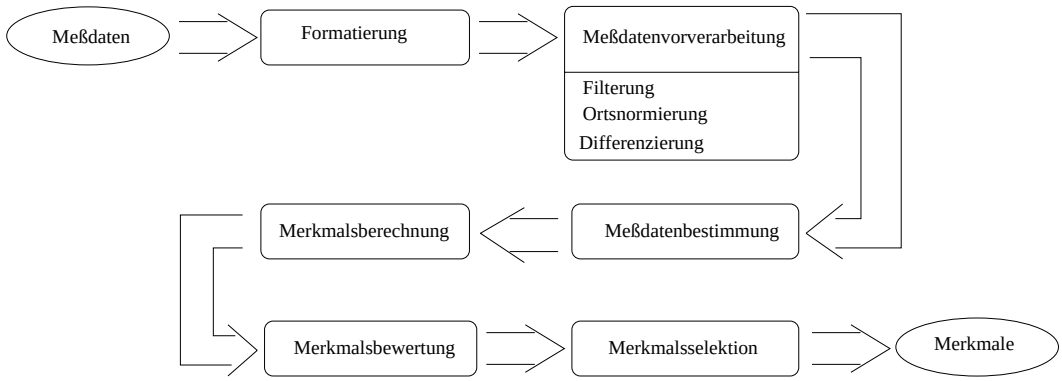


Abbildung 4.6: Phase der Meßdatenverarbeitung während der Vorverarbeitung.

Zur Vorverarbeitung der Meßdaten können die Algorithmen der Signal- und Bildverarbeitung eingesetzt werden. Dazu zählen beispielsweise folgende Verfahren aus der Pipelinediagnose (siehe auch [Jäh97, DH73, Spi90, Hei94, SE94]):

- Algorithmen zur Normierung der Daten:
  Die formatierten Daten werden mit Hilfe der vorhandenen Orts- und Geometrieinformationen in eine distanznormierte Darstellung umgewandelt (siehe Abbildung 4.7).

- Filterungsverfahren:
  In den Versuchsphasen des ***NeuroPipe I***–Diagnosesystems wurden Tiefpaßfilter verwendet, um Meßsperrzeitfehler, die durch große Amplituden gekennzeichnet

sind, zu eliminieren. Es hat sich aber herausgestellt, daß unter bestimmten Konstellationen auch die "normalen" Meßwerte gleiche Werte annehmen können, womit auch diese eliminiert wurden. Außerdem dürfen die Meßwerte nicht verändert werden, da sonst ihre Aussagekraft verloren geht. Aus diesen Gründen wurde insgesamt auf die Filterung verzichtet.

- Transformationsverfahren wie z.B. FFT, Differenzierung:
  Dadurch, daß die Anomalien keine spezifische Ausdehnung, Amplitude oder gar ein sich wiederholendes Muster enthalten, wird die FFT nicht für die Vorverarbeitung von Anomalien eingesetzt. Hingegen wurde versucht die Spiralnähte, die durch eine periodische diagonale Linie durch die C-Scan Darstellung gekennzeichnet sind, mittels der FFT zu detektieren. Es hat sich aber herausgestellt, daß eine Detektion mittels geneigter Histogramme wesentlich effizienter ist.

  Bei der Magnetstreufluß-Pipeline-Inspektion (siehe Abschnitt 7.3) wird mittels der Differenzierung der Signale in Längsrichtung der Pipeline eine neue Repräsentation der Daten erzeugt.

- Oversampling - Algorithmen:
  Um eine äquidistante Auflösung der Daten zu erhalten, muß die schwankende Geschwindigkeit des Molches durch Oversampling ausgeglichen werden.

- Verfahren zur Korrektur von Daten anhand von Modellen:
  Die Ortsinformation der Messungen steht leider nicht immer korrekt zur Verfügung oder fehlt sogar ganz. Aus diesem Grund muß eine Odometerkorrektur stattfinden, die durch Löschen, Hinzufügen und wahlweise Lineare- oder Bisplineinterpolation behoben werden. Dabei ist die Masse des Molchs ausschlaggebend, der nur beschränkte Beschleunigungen durchführen kann.

Bei der Ultraschall–Pipelinediagnose schwankt die Geschwindigkeit des Molches in der Pipeline. Die Messungen werden in konstanten Zeitabständen durchgeführt, wodurch eine unregelmäßige Verteilung der Meßpunkte bzgl. des Orts entsteht. Des weiteren sind die Ultraschallsensoren auf schräg angeordneten Kufen (siehe Abb. 4.2(b)) montiert. Dadurch muß ein horizontaler Versatz für jeden Sensor korrespondierend zu seiner Kufenposition bei der Ermittlung der zu einer Distanz zugehörigen Meßwerte interpoliert werden. Um keine Fehlinterpretationen durch falsche Modellierungen zu erhalten, wird immer die nächste Messung verwendet, mit der keine Interpolation der Meßwerte durchgeführt wird.

Die formatierten Meßdaten werden durch diese Interpolation und Oversampling derart transformiert, daß die Anzahl der Messungen pro zurückgelegtem Meter der Pipeline konstant wird (siehe Abb. 4.7). Dazu wird eine Auflösung vom 800 Messungen pro Meter gewählt, die an der Schwelle des Meßbereichs der Ultraschallsensoren liegt.

Dabei wird keine Rekonstruktion von nicht vorhandenen Daten durchgeführt, da diese Information in den späteren Schritten sehr wichtig und die Modellierung nicht eindeutig ist.

4.5 Bestimmung von Meßdatenregionen

Die Bestimmung der Meßdatenregionen umfaßt das Auffinden abweichender bzw. für die Aufgabenstellung interessanter Regionen in den Daten, die der nachfolgenden Klas-

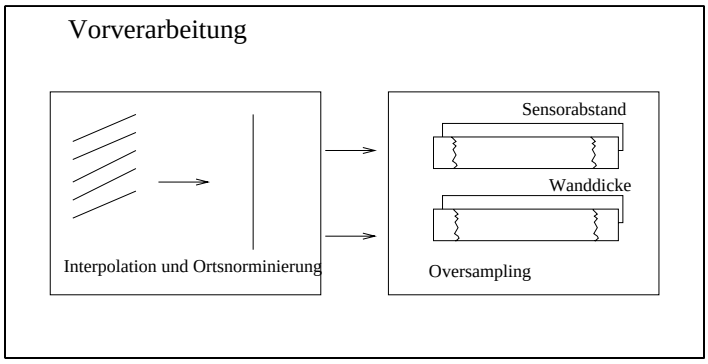


Abbildung 4.7: Vorverarbeitung der Ultraschall-Meßdaten durch Ortsnormierung, wobei die diagonale Sensoranordnung und Geschwindigkeitsschwankungen kompensiert werden, und das Oversampling der Meßdaten, damit eine konstante Repräsentation entsteht. Dabei entsteht die Sensorabstandsrepräsentation mit zugehörigen Kompressionswerten als auch die Wanddickenrepräsentation mit ihren Kompressionswerten.

sifikation unterzogen werden (siehe Abb. 4.8). Diese Regionen können entweder durch Intervalle, Polygonzüge oder Marken eingegrenzt werden.

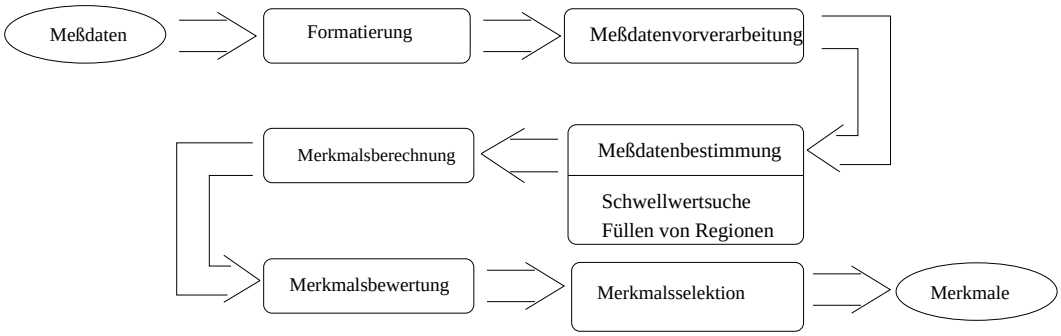


Abbildung 4.8: Aus den Meßdaten werden die für die nachfolgende Klassifikation "interessanten" Teilgebiete herausgeschnitten.

Eine stark verbreitete Methode aus der Bildverarbeitung ist die Schwellwertsuche bei einem optimal ausgeleuchteten Objekt. Dieses Verfahren besitzt den Nachteil, daß zuerst oder im nachfolgenden Schritt auf die Interessenregion fokusiert werden muß, falls sich mehrere unabhängige Regionen gleichzeitig im Bild befinden.

Die hier angewandte Methode der Bestimmung geht davon aus, daß zusammenhängende Regionen in den n-dimensionalen Feldern von Interesse sind. Deshalb wird zunächst mittels eines ersten Kriteriums nach einem Startpunkt gesucht, von dem aus die transitive Hülle aller mit ihm in Verbindung stehenden Meßwerte nach einem zweiten Kriterium ermittelt werden.

Bei der Pipelineinspektion handelt es sich um Regionen, die durch parallele Schwellwertfilterung in beiden Repräsentationen (Wanddicke und Sensorabstand) einen Anfangspunkt für die Region suchen (siehe Abb. 4.9). Dieser Anfangspunkt wird anhand einer weiteren Schwellwertsuche nach Nachbarpunkten wiederum in beiden Repräsentationen erweitert. Die derart ermittelte Region wird als Anomalie selektiert. Wegen der starken Abhängigkeit der Schwellwertparameter von der Qualität der Daten und

des angestrebten Auswertungszieles wird die Parametrisierung durch den Auswerter vorgenommen.

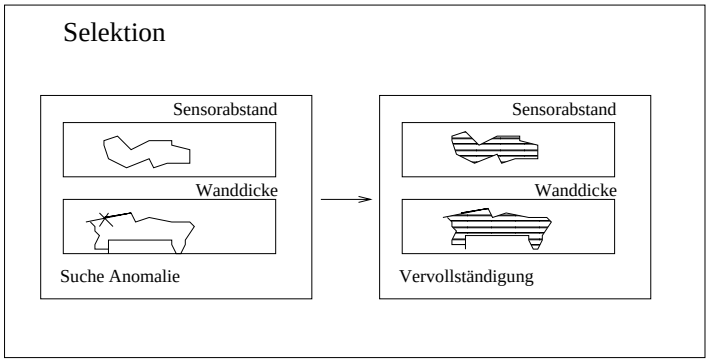


Abbildung 4.9: Bestimmung der Regionen der Ultraschall-Meßdaten. Im ersten Schritt wird ein Anfangspunkt der Anomalieregion gesucht (mit einem Kreuz markiert) der im zweiten Schritt Betrachtung der Nachbarspunkte zur Region ausgedehnt wird. Die Suche findet gleichzeitig in der Sensorabstand- und Wanddickenrepräsentation statt.

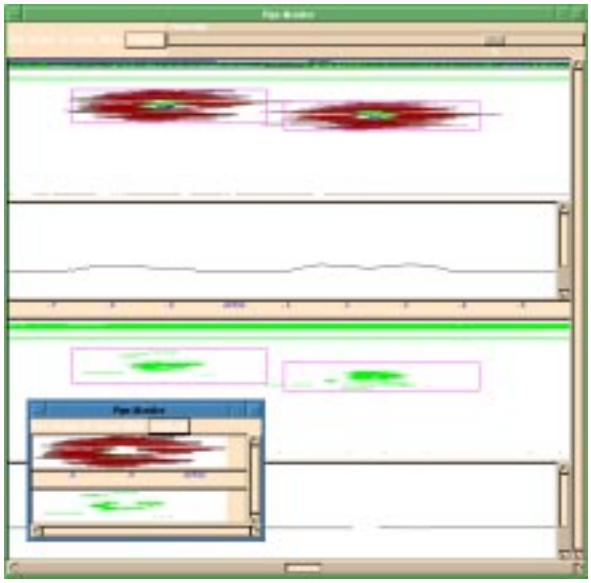


Abbildung 4.10: Falschfarbenvisualisierung zweier Regionen aus den Ultraschall-Inspektionsdaten. Die obere Hälfte enthält die Abstandswerte und die untere die Wanddickenwerte. Die selektierten Regionen werden gleichzeitig in beiden Ebenen durch Rechtecke markiert. Dabei werden die zur linken Region gehörenden Meßwerte in einem separaten Fenster dargestellt.

Aus den selektierten Regionen werden im nachfolgenden Schritt Merkmale extrahiert.

4.6 Extraktion von Merkmalen

Die Aufgabe der Merkmalsextraktion besteht in der systematischen Aufarbeitung der vorverarbeiteten Sensordaten sowie der zielgerichteten Suche innerhalb dieser Daten nach plausiblen Merkmalen. In der Regel ist dies mit einer Reduktion der Datenmenge verbunden. Die Merkmale sollen dabei die für den nachfolgenden Klassifikationsschritt notwendige Information enthalten (siehe dazu Abbildung 4.11). Dies bedeutet, daß durch die Berechnung eines Merkmals die für den Klassifikationsprozeß notwendige Information komprimiert wird. Im allgemeinen gilt: je mehr Vorwissen durch eine Merkmalsfunktion genutzt wird, um so größer ist die Datenreduktion und um so höher ist der Informationsgewinn.

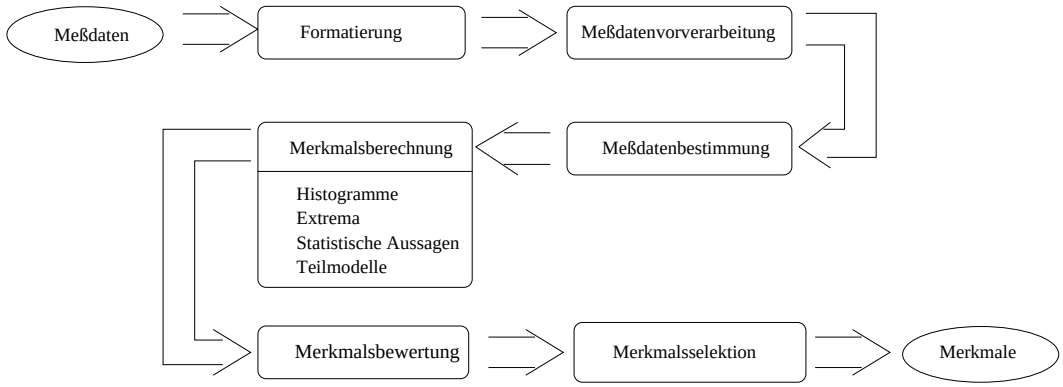


Abbildung 4.11: Extraktion der Merkmale ist wesentlich durch die auf die Vorverarbeitung folgende Klassifikation geprägt

Die Nutzung von Modellwissen in der Pipelinediagnose ist in Abbildung 4.12 schematisch dargestellt. Dabei wird die reale Welt durch die kontinuierliche Originalfunktion $F(X, Y)$ repräsentiert, die durch den Molch mit Hilfe der Ultraschallsensoren und der Vorverarbeitung auf eine diskrete Funktion $F'(x, y)$ abgebildet wird. Um zielorientiert aussagekräftige Merkmale zu erhalten, ist es notwendig, möglichst viel des vorhandenen Modellwissens auszunutzen. Daraus ergibt sich eine Hierarchie von unterschiedlich dimensionalen Räumen (siehe Abb. 4.12), aus denen die Merkmale gewonnen werden können.

Die Merkmalsextraktion läßt sich also auch als Funktion beschreiben:

$$f : R^m \rightarrow M^n \qquad \text{mit } m \geq n \tag{4.1}$$

Dabei stellt R^m einen m-dimensionalen Unterraum der Rohdaten und M^n den n-dimensionalen Merkmalsraum des durch f extrahierten Merkmals dar.

Die Berechnung von Merkmalen dient hauptsächlich zur Reduktion des Datenvolumens bei gleichzeitiger Erhaltung der für die nachfolgende Klassifikation notwendigen Informationen.

Die in dieser Arbeit vorgeschlagene Lernkomponente zum Generieren von Klassifikatoren trifft ihre Entscheidungen auf der Basis von Merkmalen. Diese liegen in numerischer Form vor. Da die Lernkomponente auch Backpropagation–Netze nutzt (siehe Abschnitt 5.1.2), ist es sinnvoll, die Merkmale auf einen Zahlenwert im Intervall $[0 \ldots 1]$ zu normieren. Damit wird bei den Backpropagation–Netzen der Bereich

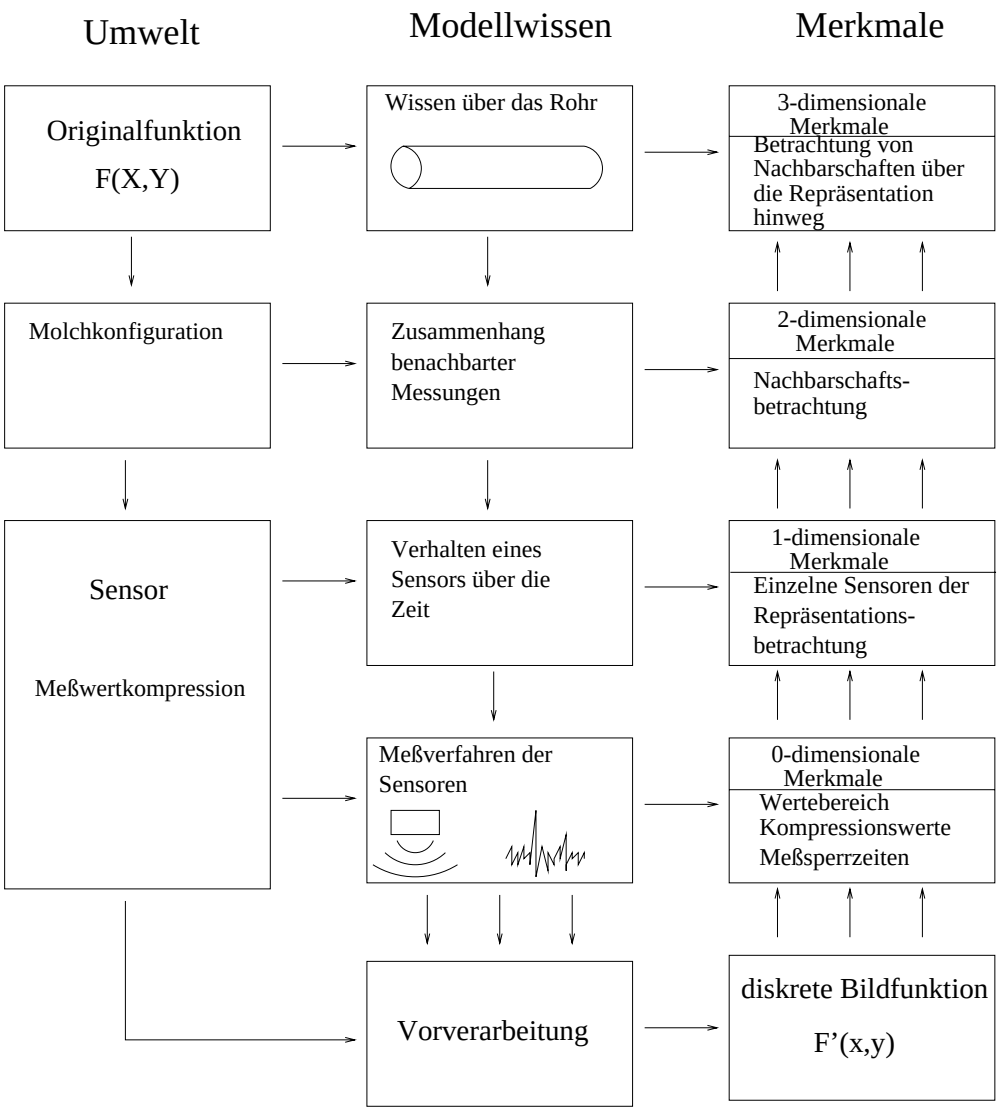


Abbildung 4.12: Einbeziehung des aus der Umwelt abgeleiteten Modellwissens in die Vorverarbeitung sowie auch in die Herleitung und Erstellung der Merkmale. Die Merkmale werden aus der Bildfunktion $F'(x, y)$ berechnet.

der Aktivierungsfunktion mit großer Steigung bei dem Lernprozess ausgenutzt, was zu einem schnellerem Lernerfolg (Konvergenz) führt.

Die Merkmale werden mit Hilfe von Filter- und statistischen Verfahren extrahiert und verdichtet. Viele der in der Bildverarbeitung üblichen Merkmalsextraktoren, wie z.B. Richtungsfilter oder Kantendetektion stellen nicht die Art der Information zur Verfügung, die Anomalien in den betrachteten C-Scans hinreichend beschreiben. Mit dieser Information kann daher keine Klassifikation erfolgreich durchgeführt werden. Der Grund hierzu liegt in der variierenden Kardinalität der Mengen dieser Merkmale. Deshalb werden weitere spezielle *subsymbolische* Merkmale benötigt, die diese variablen Mengen beschreiben. Eine Forderung an solche *subsymbolischen* Merkmale besteht darin, daß sie weitere signifikante Charakteristika der Meßdaten enthalten wie (siehe auch [DH73, SBH95, Gar98]):

- Statistischen Methoden (Histogramme, Mittelwerte, Abweichungen),

- Mathematischer Analyse (Minima, Maxima, Steigung, Ableitung, Integral, Summe, etc.),
- Umsetzung des Modellwissens in Algorithmen.

Typische Merkmale umfassen dabei sowohl Minima, Maxima als auch Ableitungen der diskreten Bildfunktion. Zur weiteren Verarbeitung durch eine folgende Klassifikation der zu bestimmenden Anomalie ist es notwendig, einen Merkmalsvektor fester Länge als Eingabe zu bilden. Dabei ist die richtige Auswahl der Merkmale aus der Menge der möglichen Merkmale für das verwendete Lernverfahren von entscheidender Bedeutung.

Im allgemeinen werden die Merkmale mit Hilfe von Modellwissen im Sinne der Inspektionsaufgabe ausgenutzt. Bei der Auswertung der Merkmale wird in der Regel Erfahrungswissen von Diagnoseexperten mit herangezogen, um deren Vorwissen bezüglich der Aussagekraft spezieller Merkmale nutzen zu können. Zusätzlich sind gezielte Versuchsreihen sinnvoll, um mehr Sicherheit bei der Merkmalsanomalie erzielen zu können.

Betrachtet man dazu beispielsweise die Ultraschall–Pipelinediagnose, dann wurden für die Anomalieklassifikation insgesamt 85 Merkmale und für die Längsnahtdetektion 82 Merkmale bestimmt (siehe Abschnitt 7.2).

Diese Merkmale sollen durch eine automatische Merkmalsselektion (siehe Abschnitt 4.8) oder das Lernverfahren auf ihre "Nützlichkeit" hin validiert werden. Ist dann die Klassifikationsgüte nicht ausreichend, müssen neue Merkmale in einem iterativen Prozeß gesucht werden. Im Abschnitt 4.7 werden darüber hinaus Verfahren untersucht, die einen Anhaltspunkt über die Aussagekraft der Merkmale geben. Durch diese zusätzliche Information über die Merkmale kann die Anzahl der Iterationen verringert werden, da die Zusatzinformation Anhaltspunkte für die Merkmalserstellung liefert.

Im nachfolgenden sollen zunächst einige Merkmalsbeispiele aus der Pipelinediagnose näher betrachtet werden.

4.6.1 Beispiele für Merkmale

Im folgenden Unterabschnitt werden einige ausgewählte Merkmale aus den 167 Merkmalen der Ultraschall-Pipelinediagnose vorgestellt, um einen näheren Einblick in die Merkmalsarten zu erhalten. Dabei werden diese nach ihrer Quelle, nämlich ob es sich um 1D, 2D oder 3D Daten handelt, unterschieden. Merkmale, die sich aus Daten höherer Dimensionen ergeben, kamen im Rahmen dieser Arbeit nicht vor und werden deshalb nicht weiter betrachtet.

Merkmale eindimensionaler Daten

Merkmale, die aus statistischen und geometrischen Zusammenhängen von Nachbarschaften in einer Dimension ermittelt werden, werden in der Längsnahtdetektion bei der Pipelinediagnose verwendet (siehe Abschnitt 7.2). Bei dieser Detektion sind insbesondere die Signalverluste von Bedeutung, die den Verlauf der Schweißnaht charakterisieren. Dabei besteht die Schwierigkeit darin, eine Naht auch in einem stark verrauschten Rohr, also in einer Repräsentation, in der viele Sensorausfälle vorkommen, zu erkennen. Dazu sind folgende Merkmale entscheidend:

- Merkmal für prozentuale Anzahl der Signalverluste pro Sensor im Rohr. Dies ist relevant, um die Verrauschtheit der Daten zu quantifizieren.

- Maß für die Separation der Meßwerte durch Signalverluste.

- Die Diskontinuität der Signalverläufe in der Sensorabstand- und der Wanddickenrepräsentation. Dieses Merkmal beschreibt die Häufigkeit von großen Wertesprüngen innerhalb eines Sensors.

Merkmale zweidimensionaler Daten

Diese Merkmale werten die Zusammenhänge von Nachbarschaften in zwei Dimensionen aus. Dazu zählen nicht nur Merkmale, die Zusammenhänge innerhalb einer Repräsentation ausdrücken, sondern auch die Zusammenhänge zwischen zwei eindimensionalen Repräsentationen. Alle nachfolgenden Merkmale werden bei der Klassifikation der Regionen bei der Ultraschall-Pipelinediagnose verwendet:

- Häufigkeitsverteilung über die dem Kompressionsfenster benachbarten Wertebereiche in der Wanddicken- und Sensorabstandsrepräsentation.

- Relative Anzahl von Signalverlusten innerhalb der Anomalieregion.

- Maximale und minimale Ableitung innerhalb der Anomalieregion sowie deren Standardabweichung.

Ein weiteres Beispiel für die Bestimmung zweidimensionaler Merkmale ergibt sich aus dem Problem der Suche nach Rundnähten in Pipelines. Dafür wird aus beiden C-Scan Darstellungen (siehe Abbildung 4.3) in einem kleinen Intervall um jede X-Koordinate oder Distanz in der Pipeline jeweils ein Histogramm über die Anzahl der Signalverluste und die Anzahl der Kanten bestimmt (siehe Abbildung 4.13).

Durch die Betrachtung des so entstandenen zweidimensionalen Unterraums, der sich als mehrere Histogramme unterschiedlicher Merkmale manifestiert, können durch weiteres Aufsummieren dieser Merkmale sowie durch eine Schwellwertbetrachtung die Rundnahtpositionen ermittelt werden.

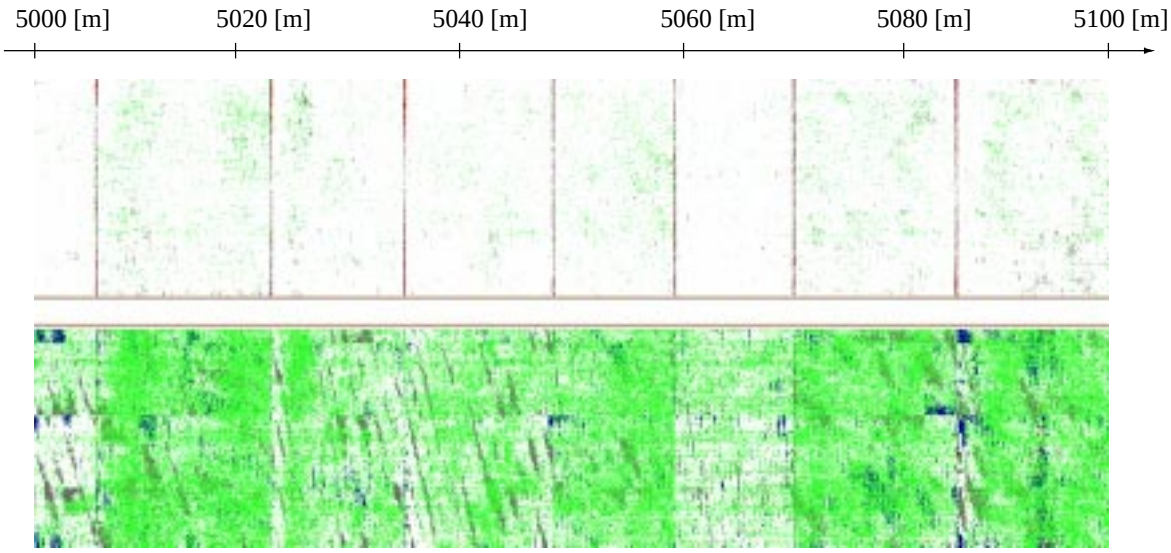
Merkmale dreidimensionaler Daten

Bei der Pipelinediagnose werden Merkmale aus den geometrischen Zusammenhängen zwischen den unterschiedlichen Repräsentationen (Wanddicke und Sensorabstand) ermittelt:

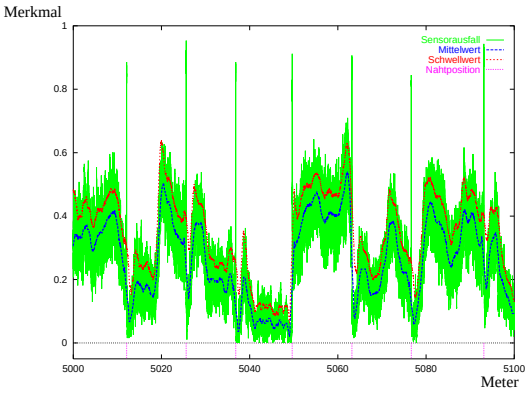
- Korrelation der Signalverluste bei der Messung der Wanddicke und des Sensorabstands in der Anomalieregion.

- Verhältnis der Ausdehnung der Anomalieregion in der Wanddickenrepräsentation und der Ausdehnung in der Repräsentation des Sensorabstands.

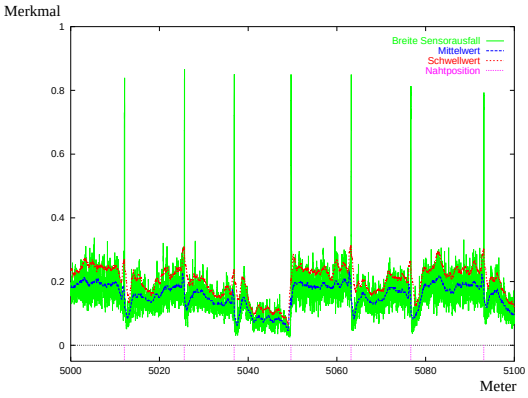
Durch die Verknüpfung der Repräsentationen, die entweder von verschiedenen Sensoren herrühren oder durch unterschiedliche Vorverarbeitungsschritte entstanden sind, können Informationsverschiebungen durch die Merkmale erfaßt werden.



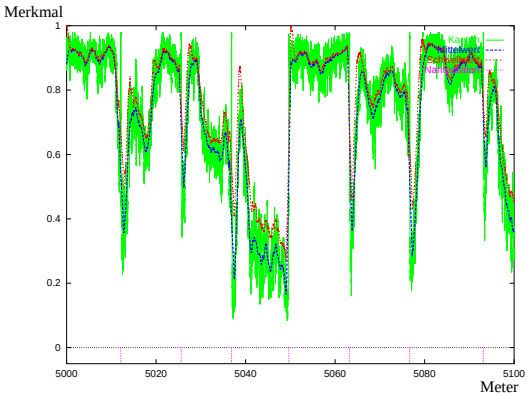
(a) Originaldaten eines 100 Meter Pipeline-Abschnittes. Die obere Hälfte stellt den Sensorabstand, die untere die Wanddicke dar.



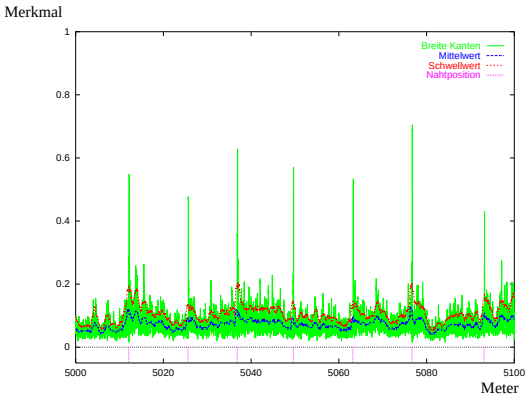
(b) Histogramm der Signalverluste



(c) Differenz des Histogramms der Signalverluste



(d) Histogramm der Sensorwertänderungen



(e) Differenz des Histogramms der Sensorwertänderung

Abbildung 4.13: Merkmale für die Suche von Schweißnähten in Stahlrohren mittels Ultraschall. Die aus Abb. (a) gewonnenen Merkmale sind Histogramme von Signalverlusten (Abb. (b),(c)) und Sensorwertänderungen (Abb. (d),(e)) in Umfangsrichtung des Rohres aufsummiert über der Position im Rohr.

4.6.2 Einlernen von Merkmalen

Neben der Kodierung des Vorwissens in Form von Merkmalen besteht aber auch die Möglichkeit, die Merkmale einzulernen. Dies kann sowohl automatisch als auch interaktiv mit Unterstützung des Experten erfolgen. Dieser Ansatz wird häufig verwendet, um sog. Signalfilter aufzubauen (siehe [SBH95, Hei94, EC96]). Dabei kann das Einlernen von Merkmalen als Aufteilung eines komplexeren Problems in kleinere Teilaufgaben aufgefaßt werden. Das Problem besteht dabei in der Definition eines aussagekräftigen Merkmals an sich, die zu lösenden Teilaufgaben in der Bewertung des jeweiligen Beispiels durch den Benutzer. Dazu sollte das Merkmal für den Benutzer einfach und eindeutig interpretierbar sein, damit er dem System die richtigen Klassifikationen aufzeigen kann.

Aus diesem Grund lassen sich nicht alle vorstellbaren und erfaßbaren Merkmale durch interaktives Lernen ausdrücken, da für die Bewertung die Benutzerinteraktion (siehe auch Kapitel 6) notwendig ist, die eine interpretierbare Vorstellung von dem Merkmal voraussetzt.

Durch das Sammeln dieser Benutzerbewertungen eines Merkmals, ist es prinzipiell möglich, eine Merkmalsfunktion zu erstellen. Hierzu bieten sich zum Beispiel interaktive Lernverfahren an. Auf diese Weise können komplexe Merkmalshierarchien entstehen.

Die oben besprochenen Methoden zur Merkmalsextraktion beruhen in starkem Maße auf Wissen (explizit oder implizit) über die zugrundeliegenden Merkmale in Verbindung mit dem verwendeten Klassifikationsalgorithmus. Aus diesem Grund soll im folgenden Abschnitt die Frage geklärt werden, ob generelle Aussagen über Merkmale und ihre implizierte Informationsgehalt getroffen werden können, ohne a priori Modellwissen anzuwenden.

4.7 Ermittlung der Charakteristik der Merkmale

In diesem Abschnitt werden Verfahren vorgestellt, die Aussagen über die Merkmale anhand repräsentativer Beispiele erlauben, ohne Kenntnis über den später verwendeten Klassifikationsalgorithmus (siehe Abb. 4.14) zu besitzen. Diese Aussagen bieten dem Benutzer objektive Maßstäbe für die Interpretation der Merkmale.

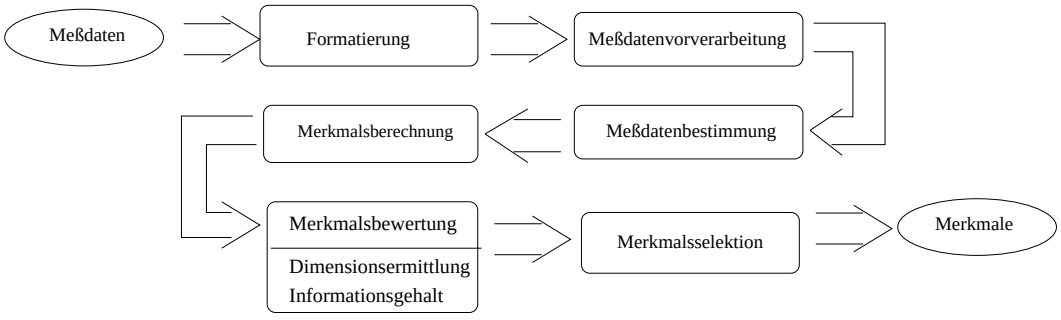


Abbildung 4.14: Bewertung der Merkmale ohne die Einbeziehung spezifischer Charakteristiken des nachfolgenden Klassifikators.

Die Untersuchungsmethoden aus der Diskriminanzanalyse wie zum Beispiel der Hauptkomponentenanalyse (siehe auch Abschnitt 4.8.1), Test auf Gleichheit der Mittelwertvektoren und Kovarianzmatrizen (siehe [Eis88]) bzw. der Test auf multivariate

Normalverteilung (siehe [Mar70]) besitzen den Nachteil, daß sie auf konkreten Annahmen bzw. auf a priori Wissen über die Merkmale beruhen. Diese sind geeignet für Merkmale, bei denen diese Zusammenhänge bekannt sind, wovon in dieser Arbeit allerdings nicht ausgegangen wird.

Aus diesem Grund wird die Charakterisierung der Merkmale bezüglich ihrer Aussagekraft ausschließlich mittels informationstheoretischer Aussagen und des Verfahrens der Fraktalen-Dimension durchgeführt, auf die in folgenden Abschnitten eingegangen wird.

4.7.1 Fraktale-Dimension

Eine wichtige Aussage über die Merkmale ist ihre Unabhängigkeit, bzw. die Aussage, wie viele Merkmale wären nötig, um die gleiche Information zu Repräsentieren, gesetzt den Fall, es gelingt, unabhängige Merkmale zu finden. Durch diese Aussage kann die Wahl der Merkmale überprüft werden. Ein zu diesem Zweck geeignetes Verfahren kommt aus der Chaosforschung.

In der Chaostheorie wird die fraktale Dimension verwendet, um die "Zerklüftetheit" eines chaotischen Systems zu messen. Mit dieser Dimension wird ein abstrakter Dimensionsbegriff definiert, der sich algorithmisch aus den Zuständen des chaotischen Systems bzw. in dem hier vorliegenden Fall den Merkmalsdaten berechnen läßt. die Um diese fraktale Dimension zu bestimmen, wird eine Methode aus der nichtlinearen Dynamik verwendet, die sog. Bestimmung der verallgemeinerten Fraktalen-Dimension (siehe [Gra83]).

Mit der verallgemeinerten Fraktalen-Dimension besteht die Möglichkeit, die "Zerklüftetheitseigenschaften" eines Datensatzes zu charakterisieren. Das Verfahren betrachtet nur die Merkmalsvektoren, ohne die Kenntnis der jeweiligen Klassenzugehörigkeit.

Auf diese Weise können Informationen über die internen Zusammenhänge der Merkmale zur Verfügung gestellt werden. Außerdem kann die Struktur des Merkmalsraums damit näher auf den Grad der räumlichen Ausbreitung dieses Raumes erfaßt werden.

Die Bestimmung der verallgemeinerten Fraktalen-Dimension geht von einem Datensatz aus, der aus N Datenpunkten besteht. Dieser wird mit Volumenelementen $V_i(l)$ überdeckt, wobei l eine charakteristische Größe darstellt. Eine charakteristische Größe ist beispielsweise eine Kugel mit Radius l oder ein Hyperwürfel der Kantenlänge l. Nach Überdeckung des Datensatzes mittels der Volumenelemente $V_i(l)$ wird das Skalierungsverhalten einer bestimmten Eigenschaft in Bezug zur abnehmenden Größe l der Volumenelemente gemessen. Als Resultat ergibt sich dann eine abstrakte Definition der verallgemeinerten Fraktalen-Dimension $D(q)$ mit der Partitionierung $q \in \mathbb{R}$.

In der Regel wird mit dieser Methode die räumliche Ausbreitung von Attraktoren[2], die aus unendlich vielen Punkten bestehen, quantifiziert.

Die verallgemeinerte Fraktale-Dimension lautet (vgl.[Gra83, Spe94]):

$$D(q) = \lim_{l \to 0} \frac{I^q}{\log(\frac{1}{l})} \tag{4.2}$$

[2]Attraktoren beschreiben diejenige Menge der Zustände, Koordinaten gegen die ein chaotisches System für $n \to \inf$ strebt.

Dabei stellt I^q den fraktalen Informationsbegriff dar, der wie folgt definiert ist:

$$I^q(l) = \frac{1}{1-q} \log \sum_{i=1}^{A(l)} p_i^q(l) \tag{4.3}$$

mit

p_i	die Wahrscheinlichkeit, einen Punkt in einem Volumenelement i zu lokalisieren,
l	die Größe des Volumenelementes,
$A(l)$	die minimale Anzahl der Volumenelemente, die für die Überdeckung des Datensatzes benötigt wird.

Für $q = 1$ stimmt die Gleichung 4.3 mit der Shannonschen Information (Entropie) überein (siehe Gl. 4.4) Der Informationsbegriff errechnet sich also im wesentlichen aus der logarithmierten Aufsummierung der Wahrscheinlichkeiten p_i, die sich aus den überdeckenden Volumenelementen ergeben.

Diese Betrachtung erfolgt für verschiedene Partitionen q. Bei dem Spezialfall $q = 1$ kann der ermittelte Wert wie in Abbildung 4.15 interpretiert werden. Dadurch stellt der ermittelte fraktale Dimensionswert $D(1)$ die minimale Merkmalsraumdimension, bzw. die minimale Anzahl von Merkmalen, dar, um die Aussage des Merkmalsraums zu erhalten.

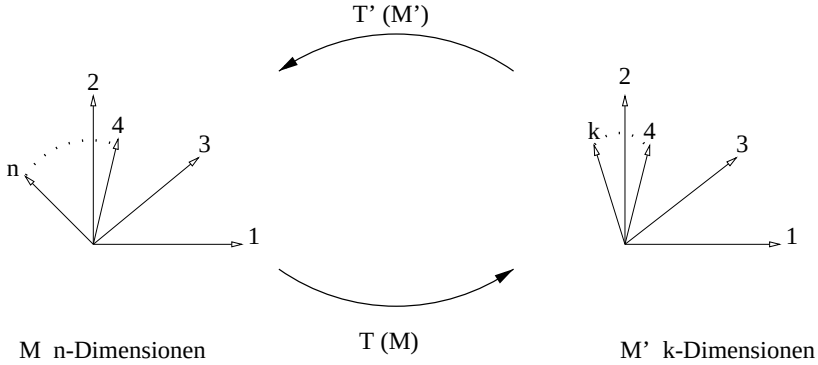


Abbildung 4.15: Für dem Merkmalsraum M mit der Dimension n (Anzahl der Merkmale) wird dabei die minimale Merkmalsraumdimension k eines unbekannten Raums M' ermittelt, der durch eine unbekannte invertierbare Abbildung T entstehen kann.

Für größere q's wird die fraktale Dimension für wahrscheinlichere Merkmalskonstellationen und für kleinere q's werden die "Zerklüftung" von weniger wahrscheinlicheren Merkmalskonstellationen betrachtet.

Die Berechnung der Fraktalen-Dimension des durch die Merkmalsvektoren $\mathcal{X}$ aufgespannten Raums ermöglicht es, die notwendige Anzahl von Dimensionen (Merkmalen) zu ermitteln. Für Kohonennetze wurde die Kenntnis der fraktalen Dimension des Merkmalsraums erfolgreich umgesetzt, in dem die Dimension des Gitters auf diese Art bestimmt wurde (siehe [Spe94]).

Als Beispiel für die Anwendung der Fraktalen-Dimension sind hier die *defects1* und *lweld* Datensätze aufgezeichnet (siehe auch Abschnitt E.1, E.2 sowie Abbildung 4.16).

Anhand der Ergebnisse in Abbildung 4.16 kann daraus abgelesen werden, daß der Merkmalsraum für *defects1* 3.37 fraktale Dimensionen bei $q = 1$ besitzt. Das bedeutet,

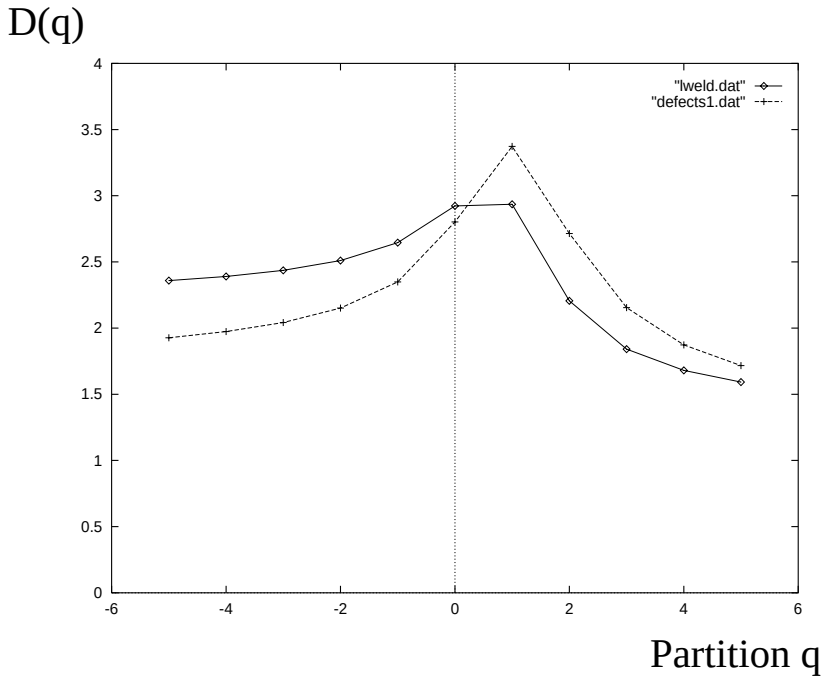


Abbildung 4.16: Fraktale-Dimension $D(q)$ der *defects1* (aus 1211 Datensätzen) und *lweld* (aus 9880 Datensätzen) in Abhängigkeit von der Partitionierung q.

daß bei richtiger Wahl bzw. Kombination der Merkmale mindestens vier Merkmale notwendig sind, um diesen Raum zu repräsentieren. Bei dem *lweld* Datensatz ergibt sich eine Wert von 2.93 für die fraktale Dimensionen bei $q = 1$. Auch hier könnte demzufolge bei einer geschickten Konstruktion der Merkmale eine sehr geringe Anzahl von Merkmalen ausreichend sein.

Anders ausgedrückt, es existiert für die *defects1* Daten also kein Verfahren, um diesen Raum in zwei Dimensionen ohne Verzerrung abzubilden. Nichtsdestotrotz ist das Ergebnis erstaunlich, da es zeigt, daß von den 41 Dimensionen des Merkmalsraums nur vier essentiell wären, falls die *richtigen* Merkmale konstruiert werden können. Analog gilt diese Betrachtung auch für den *lweld* Datensatz, der theoretisch mit drei Merkmalen auskommen könnte.

Darüber hinaus erhält man ein intuitives Gefühl für die Komplexität des Merkmalsraums durch die Betrachtung von $D(q)$ für $q \neq 1$.

4.7.2 Informationstheoretische Betrachtung

Eine weitere Möglichkeit einer modellunabhängigen Charakterisierung der abgeleiteten Merkmale besteht in der Ermittlung ihrer informationstheoretischen Aussagen über die Merkmale (siehe [Wos88]). Im folgenden wird mit der Zufallsvariable X ein Merkmal und mit der Zufallsvariable Y entweder ein anderes Merkmal oder eine Klasse repräsentiert. $p(x)$ ist die Wahrscheinlichkeit von x und $p(x, y) = p(x)p(y|x)$ (siehe [Lip89]).

Entropie Ein zentrales Maß zur Gewinnung von Information stellt die Entropie dar. (siehe [Sha48, DO96]):

$$H(X) = -\sum_{x \in \mathfrak{X}} p(x) \log(p(x)) \tag{4.4}$$

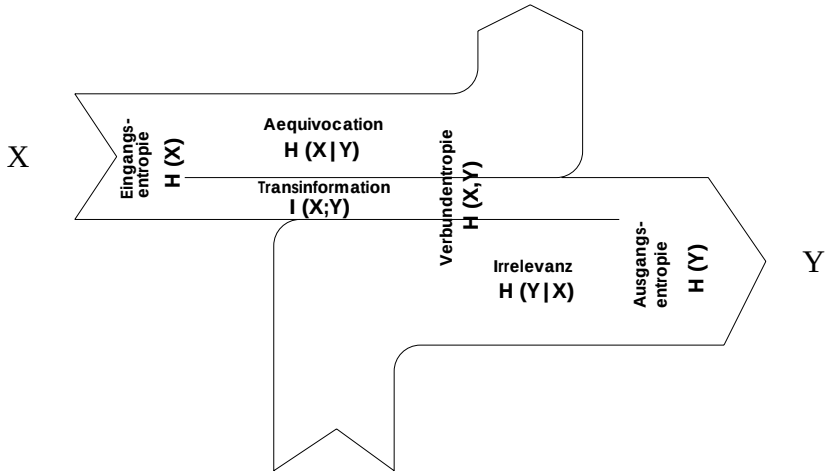


Abbildung 4.17: Informationstheoretischer Zusammenhang zwischen zwei Zufallsvariablen X und Y.

Verbundentropie

$$H(X,Y) = -\sum_{x\in\mathfrak{X}}\sum_{y\in\mathfrak{Y}} p(x,y)\log(p(x,y)) \tag{4.5}$$

Bedingte Entropie: Äquivocation und Irrelevanz

$$H(X|Y) = -\sum_{x\in\mathfrak{X}}\sum_{y\in\mathfrak{Y}} p(x,y)\log(p(x|y)) \tag{4.6}$$

Transinformation zwischen den Merkmalen X,Y und auch zwischen Merkmalen und Klassen (vgl. Abbildung 4.17 und siehe [DO96]):

$$I(X;Y) = H(X) + H(Y) - H(X,Y) \tag{4.7}$$

Aus diesen Betrachtungen kann die "Aussagekraft" eines Merkmals bezüglich aller betrachteten Beispiele abgeleitet werden. Ein Problem besteht allerdings auch in der Frage, wie die n *richtigen* Merkmale, die den Merkmalsraum eindeutig und vollständig aufspannen, zu finden sind. Diese Frage wird im folgenden Abschnitt untersucht.

4.8 Merkmalsselektion

In diesem Abschnitt wird die Frage diskutiert, inwieweit Merkmalsselektion zur Bestimmung der aussagekräftigsten Merkmale für die Klassifikation von Defekten ohne a priori Wissen über die gesuchten Klassifikatoren möglich ist.

Das Ziel der Merkmalsselektion ist es, ohne Einbeziehung des Klassifikators Aussagen über die Merkmale zu treffen (siehe Abbildung 4.18). Die Anzahl der Merkmale soll dabei derart reduziert werden, damit der Berechnungsaufwand und die Komplexität des Klassifikators möglichst gering gehalten wird. Nachfolgend werden die bekanntesten Verfahren der Merkmalsselektion vorgestellt. Dazu gehören die Hauptkomponentenanalyse und diverse Verfahren die auf den informationstheoretischen Betrachtungen basieren (MIFS, MMIP, SMIFE).

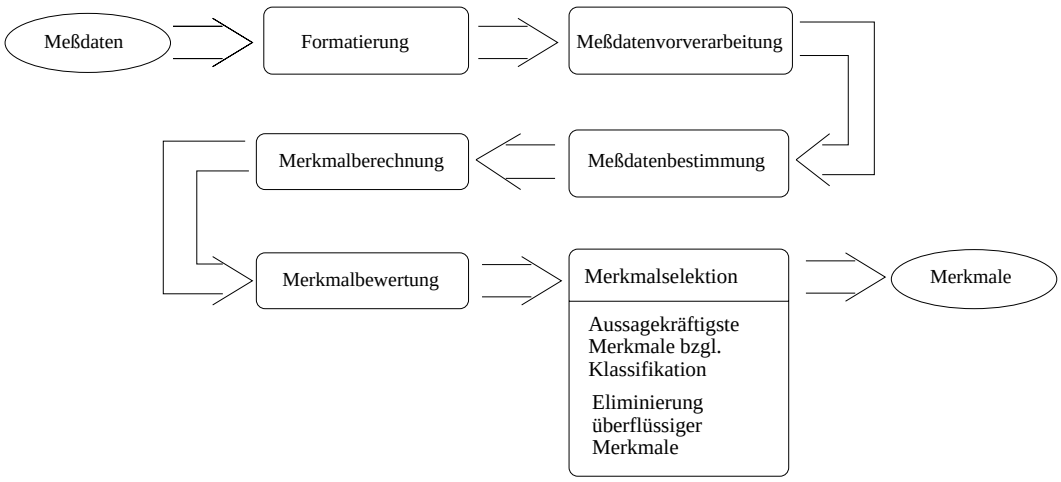


Abbildung 4.18: Durch die Selektion der Merkmale wird eine Menge der Merkmale ausgewählt, die für die nachfolgende Klassifikation gute Ergebnisse versprechen.

4.8.1 Hauptkomponentenanalyse

Bei der Hauptkomponentenanalyse, auch als Karhunen–Loeve Transformation bekannt, werden die *linearen* Zusammenhänge der Merkmale ohne Berücksichtigung der Klassifikationen betrachtet (siehe [Bis95]). Dieses Verfahren arbeitet unüberwacht und eignet sich speziell dafür, lineare Abhängigkeiten zwischen den Merkmalen zu entdecken.

In der Realität sind die Merkmale aber nur in seltenen Fällen streng linear abhängig. Des weiteren geht keinerlei Information über die Klassenzugehörigkeit in die Selektion der Merkmale ein. Deshalb ist die Hauptkomponentenanalyse für die Merkmalsselektion für Klassifikationssysteme in der Pipelinediagnose ungeeignet.

4.8.2 Mutual Information Feature Selection (MIFS)

Dieses Verfahren interpretiert die Klassen und Merkmale im n-dimensionalen Merkmalsraum als diskrete Zufallsvariablen (siehe [Bat94]). Die Merkmale $\vec{f} \in \mathcal{X}$ werden nach ihrem Beitrag zur Klassifikation der Klassen $\vec{t} \in \mathcal{C}$ geordnet. Es werden die wichtigsten k Merkmale für die Klassifikation ausgewählt. Dazu wird folgender Algorithmus verwendet:

Initialisierung: $F := \{f_i : 1 \leq i \leq n\}$ und $S := \{\}$.

Berechnung der Transinformation: für jedes Merkmal $f_i \in F$:

$$I(\vec{t}; f_i) := \sum_{j=1}^{dim(\vec{t})} I(t_j; f_i) \tag{4.8}$$

$$I(t_j; f_i) := \sum_{\vec{c} \in \mathcal{C}} \sum_{\vec{x} \in \mathcal{X}} p(c_j, x_i) \log \frac{p(c_j, x_i)}{p(c_j)p(x_i)} \tag{4.9}$$

Die Wahrscheinlichkeiten $p(c_j, x_i), p(c_j), p(x_i)$ werden dabei durch die relativen Häufigkeiten bzgl. des Datensatzes $\mathcal{B}$ angenähert.

Auswahl des ersten Merkmals: Finde das erste Merkmal f_i, das $I(\vec{t}, f_i)$ maximiert. Setze $F := F \backslash \{f_i\}$ und $S := S \cup \{f_i\}$.

Greedy-Auswahl: Wiederhole solange bis $|S| = k$.

1. Berechnen der Transinformation zwischen Merkmalen:
   Für alle Paarungen der Variablen (f_i, s_j) mit $f_i \in F, s_j \in S$ berechne $I(f_i, s_j)$.
2. Auswahl des nächsten Merkmals:
   Wähle dasjenige f_i, das $I(\vec{t}; f_i) - \beta \sum_{s_j \in S} I(f_i, s_j)$ maximiert.
   Setze $F := F \backslash \{f_i\}$; $S := S \cup \{f_i\}$.
   (β ist ein Parameter, um der Informationsüberlappung der Merkmale empirisch zu begegnen; normalerweise gilt: $0 \leq \beta \leq 1$.)

Ausgabe der Menge S: S enthält dabei die ausgewählten Merkmale, in der Reihenfolge ihrer Wichtigkeit.

MIFS ordnet mittels des Greedy-Verfahrens die Merkmale nach ihrem Beitrag zur Klassifikation an, wobei allerdings nicht berücksichtigt wird, inwieweit die Information verschiedener Merkmale gleich ist.

4.8.3 Maximum Mutual Information Projection Feature Extractor (MMIP)

Dieses Verfahren versucht durch das Auffinden orthogonal normierter Projektionen des Merkmalsvektors nach dem Algorithmus aus (siehe [GMW90]). Damit die Transinformation zum Klassifikationsvektor maximiert wird, werden die Merkmale mit dem höchsten Informationsgehalt in der orthogonal normierten Projektion des Merkmalsvektors ermittelt (siehe [BG96]). Diese Merkmale lassen sich beschreiben als

$$\vec{a}^* = \max_{\vec{a}} I(\vec{a}^T \vec{X}; Y), \tag{4.10}$$

wobei $\vec{a}^T \vec{X}$ das Merkmal mit der höchsten Transinformation ist. Das Merkmal wird mittels des MIFS-Algorithmus (siehe Abschnitt 4.8.2) ermittelt. Die Projektion mit der maximalen Transinformation wird bestimmt und daran anschließend aus den Merkmalsvektoren eliminiert. Dieser Vorgang wiederholt sich, bis der Merkmalsvektor komplett abgearbeitet ist.

Das Problem dieser Methode besteht in der hohen Überlappung der Transinformation zwischen den Merkmalen. Dieser Problematik wird bei der oben beschriebenen Vorgehensweise keinerlei Rechnung getragen.

4.8.4 Separated Mutual Information Feature Extractor (SMIFE)

Das Verfahren SMIFE1 berechnet die Transinformation zwischen drei Zufallsvariablen, wobei zwei davon Merkmale (X_i, X_j) und die dritte die Klassenvariable Y darstellen. Die Transinformation dreier Variablen wird gemäß der Gleichung 4.11 berechnet. Aus

Merkmalsselektionsverfahren	Komplexität
SMIFE1	$O(b \times n^2)$
SMIFE2	$O(b \times n^2)$
MMIP	$O(b \times n \times$ (Mittelwert der Iterationen))
Hauptkomponentenanalyse	$O(b \times n^2)$
MIFS	$O(b \times n^2)$

Tabelle 4.1: Laufzeitkomplexität der Algorithmen. Dabei bedeutet $|\mathcal{B}| = b$ die Anzahl der Beispiele, $dim(\vec{b} \in \mathcal{B}) = n$ die Dimension der Merkmale (siehe [BG96])

. .

diesen Werten wird eine Matrix aufgebaut, aus der mittels der Hauptkomponentenanalyse (siehe Abschnitt 4.8.1) die informativsten Merkmale ermittelt werden (siehe [BG96]).

$$I(X_i; X_j; Y) = H(X_i, X_j, Y) - H(X_i) - H(X_j) - H(Y) \tag{4.11}$$
$$+ I(X_i; Y) + I(X_j; Y) + I(X_i; X_j)$$
$$= I(X_i; Y) + I(X_j; Y) - I(X_i, X_j; Y) \tag{4.12}$$

Die Verbesserung dieses Verfahrens stellt SMIFE2 dar (siehe [BG96]). Anstatt die Merkmale durch die absteigende Reihenfolge der durch die Hauptkomponentenanalyse der Matrix über $I(X_i; X_j; Y)$ gewonnenen Eigenwerte zu betrachten, wird bei SMIFE2 die Gleichung 4.12 ausgenutzt. Dabei wird die Reihenfolge der Merkmale durch die ansteigenden Eigenwerte der Matrix über $I(X_i, X_j; Y)$ bestimmt.

4.8.5 Vergleich der Merkmalsselektionsverfahren

Die Hauptkomponentenanalyse erfaßt nur den linearen Zusammenhang der Merkmale und läßt dabei die Klassen außer acht. Hingegen werden bei den informationsbasierenden Verfahren wie MIFS, MMIP, SMIFE1 und SMIFE2 auch die Klassen in den Auswahlprozeß miteinbezogen. Es werden also bei den Merkmalen nicht nur lineare Zusammenhänge berücksichtigt. Durch die Betrachtung der Information können beispielsweise Permutationen der Wertebereiche zwischen den Merkmalen entdeckt werden, was bei der Hauptkomponentenanalyse nicht möglich ist.

Diese Vermutung wurde durch empirische Tests nachgewiesen (siehe [Bat94, BG96]). Die Betrachtung der Laufzeitkomplexität der Algorithmen liefert keine signifikanten Unterschiede (siehe Tabelle 4.1).

Einen Nachteil besitzen allerdings alle oben aufgeführten Verfahren. Sie sind nicht in der Lage, eindeutig festzustellen, ob die Information, die in den k selektierten Merkmalen enthalten ist, ausreicht, um einen Klassifikator aufzubauen, der die besten möglichen Ergebnisse bezüglich der Beispiele erzielt.

4.9 Resümee

In diesem Kapitel wurde die Verarbeitungskette zur Merkmalsextraktion und Merkmalsselektion aus Anomalien in großen Meßdatenmengen der Pipelinediagnose disku-

tiert. Unter dem Aspekt der bestmöglichen Auswahl eines aussagekräftigen Merkmalssatzes wurden unterschiedliche Verfahren untersucht.

Zu diesen Zweck wurde eine Charakterisierung der Meßdaten der Applikation aufgestellt, die eine intuitive Vorstellung über die Daten widerspiegelt. Basierend auf diesem Modell über die Rohdaten sind dann die entsprechenden Verarbeitungsalgorithmen erstellt worden. Es wird eine klare Trennung zwischen der Meßdatenvorverarbeitung und der Merkmalsextraktion definiert, die es vereinfacht, vorhandenes Applikationswissen in die Vorverarbeitung zu integrieren. In diesem Zusammenhang wurden Beispiele für unterschiedlich dimensionale Merkmalsextraktoren präsentiert. Darüber hinaus sind auch die Aspekte des direkten Einlernens eines Merkmalsextraktors untersucht worden. Es hat sich herausgestellt, daß eine erschöpfende Integration des a priori Wissens in die Merkmale sinnvoll ist. Ferner konnte gezeigt werden, daß Merkmale, bei denen der Experte über eine intuitive Vorstellung verfügt, mit Hilfe von Lernverfahren gut modelliert werden können.

Weiterhin wurden Methoden untersucht, die sich mit den klassifikatorunabhängigen Aussagen über die Merkmale beschäftigen.

Der Vorteil der Fraktalen-Dimension (Abschnitt 4.7.1) gegenüber dem Test auf Dimensionen besteht in der Tatsache, daß die Berechnung der Fraktalen-Dimension ausschließlich von den Merkmalen $\mathcal{X}$ ausgeht, wohingegen der Dimensionstest auch die Klassen $\mathcal{C}$ für die Berechnung der Eingabedimension heranzieht.

Des weiteren basiert die Berechnung des Dimensionstests auf linearen Zusammenhängen zwischen den Merkmalen, wie sie in der Realität im allgemeinen nicht auftreten. Die Klassenverteilungen müssen korrekt modelliert werden. Insbesondere bei multimodalen Daten müssen Klassen u.U. in mehrere Unterklassen aufgeteilt werden, falls sie aus mehreren Ballungsgebieten bestehen.

Aus den dargelegten Gründen wurde für die Pipelinediagnose ausschließlich die Fraktale-Dimension und die informationstheoretischen Aussagen als Merkmalscharakteristik verwendet.

Bei der Erstellung eines Diagnosesystems auf der Basis Neuronaler Netze nimmt die Kodierung des a priori Wissens in der Vorverarbeitung viel Zeit in Anspruch. Deshalb wurden hier verstärkt Methoden untersucht, die eine Merkmalsbewertung erlauben, ohne einen Klassifikator dafür konzipieren zu müssen.

Normalerweise wird eine Merkmalsauswahl in der Art und Weise getroffen, daß iterativ wiederholt Merkmale selektiert werden, dann ein Klassifikator aufgebaut und anschließend dessen Güte überprüft wird. Reicht diese Güte nicht aus, muß eine erneute Selektion von Merkmalen erfolgen bzw. die Erzeugung neuer Merkmale durchgeführt werden.

Dieser iterative Prozeß kann zwar nicht gänzlich aufgehoben, aber mit Hilfe der Merkmalsselektionsverfahren und der Ermittlung der Charakteristik der Merkmale erheblich verkürzt werden.

Die Kombination aus Merkmalscharakteristik und Merkmalsselektion führt dazu, daß zumindest die *aussagekräftigsten* Merkmale in die nachfolgend behandelte Lernkomponente übergeben werden.

Kapitel 5

Lernkomponente

In diesem Kapitel wird eine Lernkomponente vorgestellt, die auf den Anwendungsbereich der Klassifikation von Pipelineanomalien zugeschnitten ist. Charakteristisch dabei ist das Sammeln repräsentativer Daten und das damit verbundene Einlernen Neuronaler Netze (siehe [SB96b, Hei94, SBG95]). Ziel ist es, einen Klassifikator zu kreieren, der Änderungen aus neu hinzukommenden Beispielen sofort erlernen kann, und eine gute Generalisierungsfähigkeit aufweist. Es wird angestrebt, ein Lernverfahren zu realisieren, bei dem der Benutzer keine Lernparameter für den Klassifikator spezifizieren bzw. einstellen muß. Darüber hinaus werden statistische Methoden untersucht, die eine Bewertung bzw. Vorhersage über die Güte des Klassifikators erlauben. Die Einbindung der Lernkomponente in das Diagnosesystem ist in Abbildung 5.1 schematisch dargestellt.

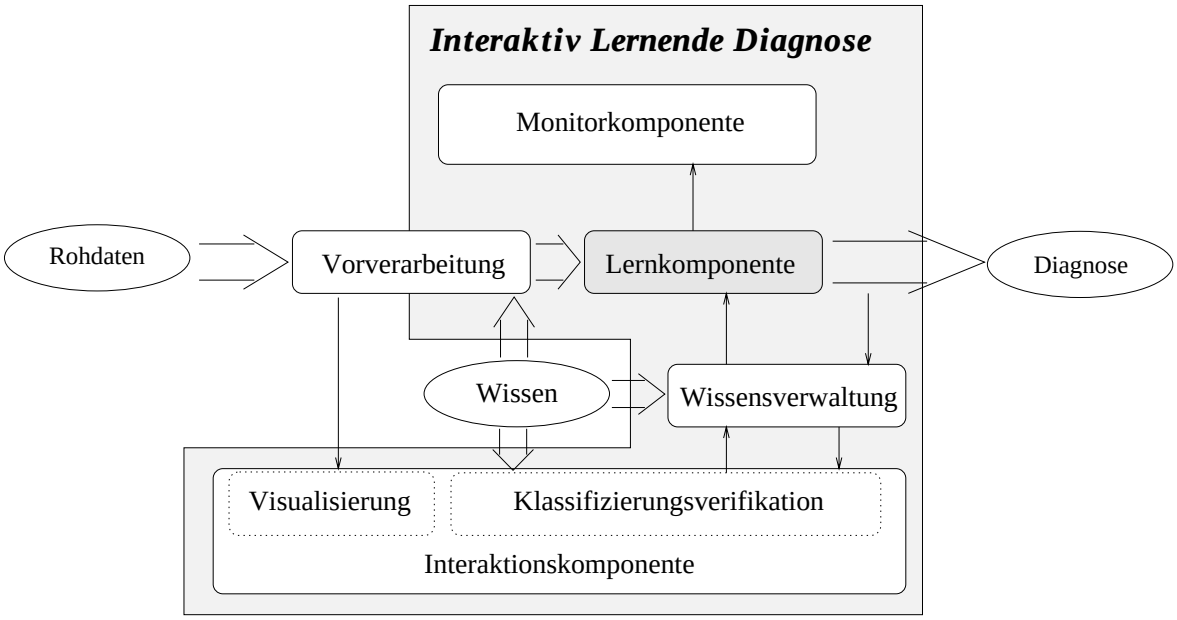


Abbildung 5.1: Einbettung der Lernkomponente in die Architektur des Interaktiv Lernenden Diagnosesystemkerns.

In diesem Abschnitt wird ein hierarchischer Ansatz für den gesuchten Klassifikator entwickelt, der einen Weg aufzeigt, wie die aufwendige Konfiguration von Neuronalen Netzen und die Einstellung ihrer Lernparameter weitgehend automatisiert werden

kann. Dabei wird Lernparameter wie folgt definiert:

Definition 5.1 (Lernparameter)
Unter Lernparameter werden all diejenigen Parameter eines Lernverfahrens verstanden, die durch den Benutzer von Außen vorgegeben werden müssen.

Für diesen Zweck wird ein inkrementeller *Dynamische Regionen* Lernalgorithmus (*Dynamic Bounds*) entwickelt, dessen Eigenschaften und seine Verbindung mit Backpropagation–Netzen zu *Hybriden Knoten* im Folgenden beschrieben wird. Es wird eine Strategie vorgestellt, nach der diese Knoten gebildet werden. Mit dieser Strategie kann die Komplexität des Klassifikators reduziert werden. Sowohl die hybride Architektur als auch die des *Dynamic Bounds*-Algorithmus wird anhand mehrerer Versuchsreihen evaluiert.

Die für die Experimente verwendeten Datensätze sind in [Pre94, MM96] sowie im Anhang E detailliert beschrieben. Es handelt sich dabei um Datensätze aus konkreten Applikationen.

5.1 Auswahl der Lernverfahren

Für die Klassifikation großer Mengen verrauschter numerischer Daten eignet sich ganz besonders die Gruppe der subsymbolischen Lernverfahren (siehe Tabelle 5.1). Unter lokalen Lernverfahren versteht man dabei diejenigen Verfahren, bei denen durch die Adaption ihres Modells an ein Beispiel ausschließlich die Ausgaben des Netzes in eine unmittelbaren Umgebung des gegebenen Beispiels beeinflussen. Hingegen wirkt sich bei globalen Lernverfahren eine Adaption über den gesamten Merkmalsraum aus. Genetische Verfahren besitzen dagegen auch eine gute Generalisierungsfähigkeit. Sie eignen sich aber nicht für die in dieser Arbeit betrachteten Anwendungen, da sie lange Lernzeiten benötigen, und die Optimierungsphase bei ILD jedoch zeitlich beschränkt ist, damit der Benutzer rasch in die Interaktionsphase treten kann.

Um die Leistungsfähigkeit des vorgeschlagenen Lernverfahrens zu erhöhen, werden hierarchische Anordnungen untersucht, die die Schwächen der einzelnen Verfahren ausgleichen.

Verfahren	Generalisierungsfähigkeit	Lerngeschwindigkeit
lokale Lernverfahren	schlecht	sehr hoch
globale Lernverfahren	gut	niedrig
hierarchische Ansätze	gut	mittel
genetische Algorithmen	gut	sehr niedrig

Tabelle 5.1: Eigenschaften verschiedener subsymbolischer Lernmethoden (vgl. [Ber94, Bis95]).

Im folgenden wird auf die wichtigsten Ansätze näher eingegangen.

5.1.1 Lokale Lernverfahren

Als der älteste und wohl auch bekannteste Vertreter der Klasse der lokalen Lernverfahren ist das sogenannte k-nächster Nachbar Verfahren zu nennen (siehe [DS79]). Der Nachteil dieses Verfahrens besteht in der großen Anzahl benötigter Neuronen (Stützstellen) im Zusammenhang mit einer großen Anzahl von Lerndaten und der ungenügenden Generalisierungsfähigkeit.

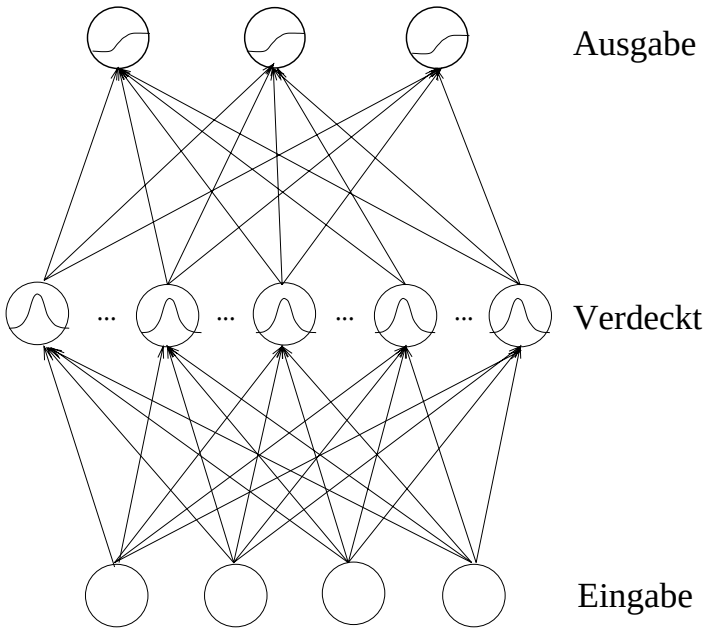


Abbildung 5.2: Netztopologie eines RBF–Netzes mit gaußschen Aktivierungsfunktionen in der verdeckten Schicht und sigmoiden Aktivierungsfunktionen in der Ausgabeschicht.

Zu den neueren Vertretern der überwacht lernenden lokalen Verfahren zählen die RBF–Netze (siehe [MD88]). Wie in Abbildung 5.2 skizziert, besteht ein RBF–Netz aus drei Schichten, wobei die verdeckten Neuronen (mittlere Schicht) gaußsche Aktivierungsfunktionen besitzen. Im wesentlichen unterscheiden sich RBF–Netze von den k-nächster Nachbar-Netzen durch die Gaußglocke als Aktivierungsfunktion der Neuronen, die Festlegung der Neuronenanzahl und der Gewichtung der einzelnen Gewichte zu der Netzausgabe. Damit ist der Nachteil der wachsenden Größe der Repräsentation gelöst, aber die Schwierigkeit besteht dabei nach wie vor in der a priori Bestimmung der richtigen Anzahl der zu wählenden Neuronen, um eine gute Generalisierungsfähigkeit zu erzielen.

Dieser Nachteil wird durch den sogenannten *Dynamic Decay*[1] Algorithmus (siehe [BF94, BF94, Ber97]) ausgeräumt. Dieses Verfahren fügt bei Bedarf Neuronen hinzu, ist aber andererseits nicht in der Lage, überflüssige Neuronen, die für das Klassifikationsergebnis nicht mehr benötigt werden, wieder zu entfernen. Der Algorithmus arbeitet ebenfalls mit Gaußglocken als Aktivierungsfunktionen. Für die Entscheidung, ob ein Neuron eingefügt werden soll, sind zwei Lernparameter (θ^+, θ^-) verantwortlich, die

[1] Der Name Dynamischer Verfall kommt von der Tatsache, daß die Ausdehnungsbereiche der Neuronen dynamisch verkleinert werden.

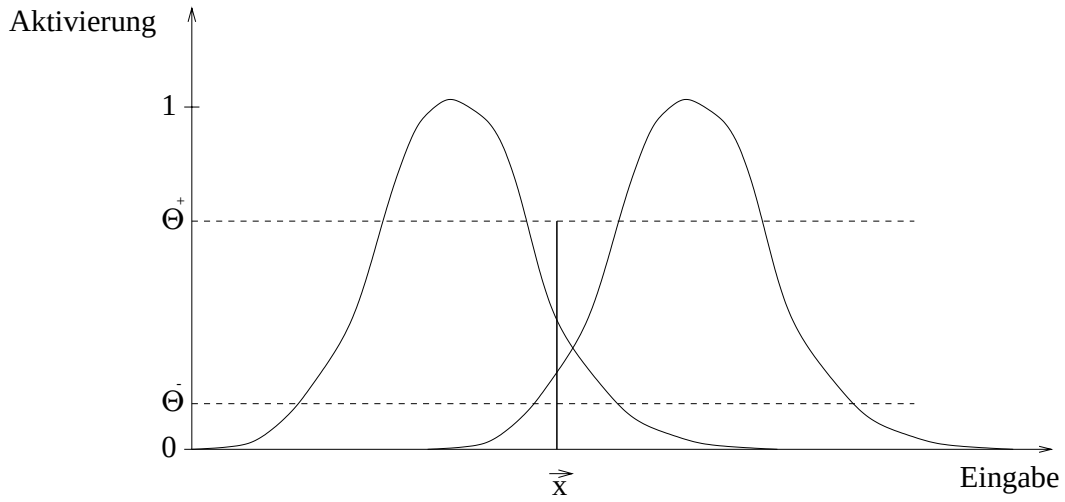


Abbildung 5.3: Einfügen eines neuen Neurons bei einer unbekannten Eingabe $\vec{x}$ bei dem sogenannten *Dynamic Decay* Verfahren.

den Einflußbereich eines Neurons bestimmen (siehe Abbildung 5.3). Über diese Lernparameter läßt sich indirekt die Anzahl der eingefügten Neuronen und die Generalisierungsfähigkeit steuern. Dabei ist anzumerken, daß dieser Algorithmus den Einflußbereich eines Neurons nur verkleinern kann, und das Verschmelzen oder gar Löschen von Neuronen nicht durchgeführt wird. Eine eingehende Untersuchung des Dynamic Decay Algorithmus wird in [Giz96] durchgeführt.

Eine Erweiterung des Dynamic Decay stellt das sogenannte Rectangular Basis Function Network (*RecBFN*) Klassifikationsverfahren dar (siehe [Ber97]). Es unterscheidet sich vom Dynamic Decay durch die trapezartigen Aktivierungsfunktion. Es existieren zwei ineinander geschachtelte Hyperquader, die den Kern und den Einflußbereich darstellen. Jede Dimension besitzt dabei ihren eigenen Ausdehnungsradius, der durch den Algorithmus angepaßt wird. Aber auch beim RecBFN werden Neuronen nur eingefügt und deren Volumen verkleinert.

5.1.2 Globale Lernverfahren

Zu den bekanntesten global lernenden Verfahren zählt das Backpropagation–Verfahren (siehe [RHW86]). Bei den Backpropagation–Netzen werden vorwiegend sigmoide Funktionen als Aktivierungsfunktionen verwendet, was die globale Auswirkung einer Änderung nach sich zieht. Neben der richtigen Wahl des Lernparameters ε ist auch die Topologie des Neuronalen Netzes zu bestimmen.

Da das Backpropagation–Verfahren sehr lange Lernzeiten aufweist, wurden verschiedene Beschleunigungsmethoden entwickelt. Eine der bekannteren ist das sogenannte *QuickProp* Verfahren (siehe [Fah88]). Dieses Verfahren nutzt die Information der vorhergehenden Adaptation der Gewichte, um den nächsten Adaptationsschritt durchzuführen. Es verwendet eine Annäherung an die zweite Ableitung, wodurch die Konvergenz erheblich beschleunigt wird. Das Verfahren benötigt allerdings drei Lernparameter.

Ein weiteres bekanntes Beschleunigungsverfahren ist das sogenannte Resilient Pro-

pagation Verfahren – *RPROP*[2] (siehe [Rie92, RB93, Rie96]). RPROP benutzt ebenfalls die Information der vorhergehenden Adaptation der Gewichte, um den nächsten Adaptionsschritt durchzuführen. Für das Verfahren werden jedoch nur zwei Lernparameter benötigt, die im Gegensatz zu QuickProp in ihrer Wahl weitgehendst unkritisch sind. Darüber hinaus konvergiert es schneller als QuickProp (siehe [Rie96]).

Das Problem der Bestimmung der korrekten Topologie eines Backpropagation–Netzes wird im allgemeinen heuristisch gelöst. In der Literatur existieren unterschiedliche Ansätze, um dieses Problem durch Lernalgorithmen zu lösen. Das bekannteste von ihnen ist das sogenannte Cascade Correlation Verfahren – *CasCorr*, siehe [FL90, Fah90]). Es fügt sukzessiv Neuronen in die Topologie ein und versucht so konstruktiv eine kaskadierte, schichtweise Netztopologie zu bilden. Leider haben sich derartige Verfahren als instabil bzw. die Wahl der Lernparameter als kritisch herausgestellt. Eine eingehende Untersuchung des Cascade Correlation Verfahrens wird in [Win94] durchgeführt.

Außerdem existiert eine Vielzahl heuristischer Regeln, um die Anzahl der Neuronen in den verborgenen Schichten a priori zu bestimmen. Dadurch reduziert sich der Suchraum nach einer geeigneten Netztopologie. Eine eingehende Untersuchung derartiger Regeln wurde in [Kop99] durchgeführt.

5.1.3 Hierarchische Ansätze

In letzter Zeit wird immer mehr versucht, die Fähigkeiten unterschiedlicher Lernverfahren miteinander zu kombinieren. Eine gute Übersicht über die bekanntesten Ansätze bietet [AK98]. Das Hauptproblem der hierarchischen Ansätze besteht in der wachsenden Anzahl von Lernparametern bei steigender Komplexität des Modells und das Fehlen jeglicher Vorgehensweise bei der Ermittlung der *richtigen* Konfiguration des Ansatzes. Im folgenden werden die wichtigsten Vertreter der hierarchischen Ansätze vorgestellt.

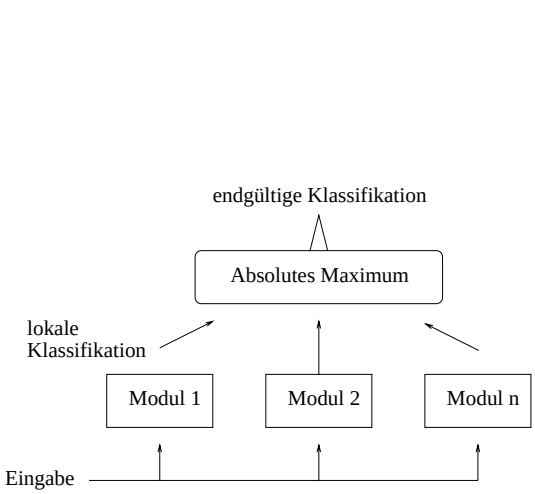
Die Entkoppelten Module (*Decoupled modules*) Variante benutzt ein ART–Netz[3] (siehe [Bis95]), um jede Klasse in Gruppen aufzuteilen. Für jede Gruppe ist dann ein Backpropagation–Netz (BP–Netz) zuständig (siehe Abbildung 5.4(a)). Diese Netze können dann parallel eingelernt werden. Als Klassifikation wird die maximale Aktivierung über alle Netze genommen. Das Problem besteht darin, daß die Beispielsmenge in disjunkte Gruppen aufgeteilt wird. Jedes der BP-Netze wird nur mit den Beispielen seiner Gruppe trainiert, und es besteht keine Verbindung zu den anderen Netzen. Aus diesem Grund ist die Generalisierungsfähigkeit nicht besonders hoch.

Das *ART-BP* Modell benutzt auch ein unüberwacht eingelerntes ART–Netz, um ein Beispiel zu einem der nachgeschalteten Backpropagation–Netze zu dirigieren (siehe Abbildung 5.4(b)). Eine ungenügende Separierung der Daten führt dazu, daß die BP–Netze sehr komplex werden (vgl. [AK98]). Damit kann diese Vorgehensweise das Netz beinahe zu einem nicht modularen BP-Netz degenerieren.

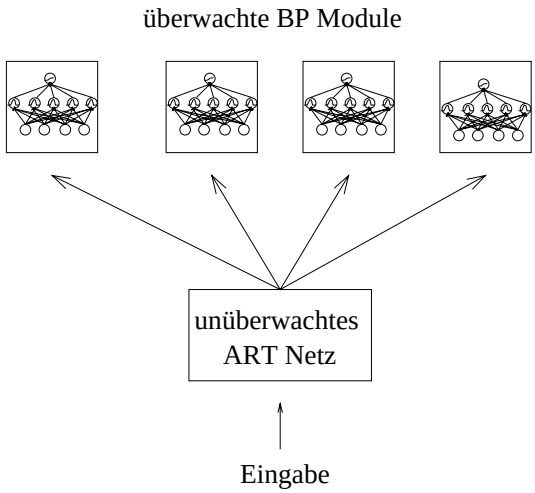
Ein weit verbreitetes Modell ist das *Ensemble*. Es besteht aus einer zu wählenden Anzahl von Netzen (meistens BP–Netze), die parallel und unabhängig voneinander mit unterschiedlichen Startinitialisierungen eingelernt werden. Bei der Klassifikation wird

[2]Der Name "Elastisches Propagieren" spielt auf die Fähigkeit des Verfahrens an, für jedes Gewicht eine separate Lernrate adaptiv zu bestimmen.

[3]ART–Netze werden unüberwacht eingelernt und dienen zur Clusterung der Merkmalsraums.

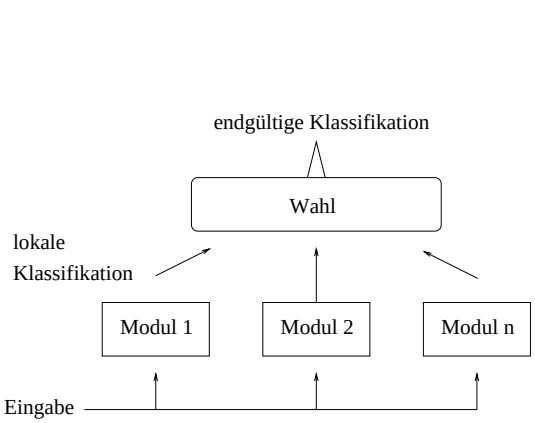


(a) Entkoppelte Module (*Decoupled modules*)

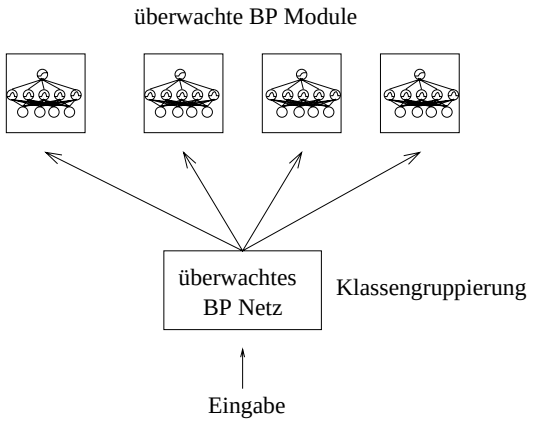


(b) ART-BP Model

Abbildung 5.4: Hierarchische Ansätze mit unüberwachter Verteilungskomponente.



(a) Ensemble Mittelwert/Mehrheit



(b) Hierarchische Module

Abbildung 5.5: Überwachte hierarchische Ansätze.

dann entweder die häufigste Klassifikation, bei der Ensemble Mehrheit (*Ensemble majority*), oder der Mittelwert der Klassifikationen, bei der Ensemble Mittelwert (*Ensemble average*) Methode gewählt. Da jedes Modul die komplette Information verarbeiten muß, entstehen komplizierte Netze und lange Trainingszeiten (siehe [Bis95, AK98]).

Zum Abschluß der hierarchischen Ansätze sei noch der Hierarchische–Module (*Hierarchical Modules*) Ansatz erwähnt. Der Hierarchische–Module Ansatz besteht aus einem überwacht eingelernten Gruppierungsmodul auf Backpropagationbasis und nachgeschalteten Blättermodulen, die sich ebenfalls aus BP–Netzen zusammensetzen. Die Gruppenaufteilung wird mit Hilfe unüberwachter Lernverfahren durchgeführt, und diese Gruppierung wird durch das untere BP–Netz realisiert. In den Blättern wird dann die Feinklassifikation innerhalb der Gruppen durchgeführt. Neben dieser zweischichtigen Anordnung können die Netze auch in mehreren Schichten angeordnet werden.

Problematisch sind hier die vielen BP–Netze wegen der langen Lernzeiten. Die Leistungsfähigkeit ist etwa mit dem ART–BP Modell vergleichbar (siehe [AK98]).

5.1.4 Beurteilung der Lernarchitekturen

Die oben abgesprochenen Verfahren wurden mit Hilfe der in Tabelle 5.4 verwendeten realen[4] Datensätzen ausgetestet. Die Ergebnisse sind in Tabelle 5.2 zusammengefaßt. Aus dieser Zusammenstellung wird ersichtlich, daß die lokalen Lernverfahren schneller lernen können, aber bei der Anwendung (Propagierung) langsamer sind als die globalen Verfahren. Hingegen sind die global basierten Verfahren dominant bei der Generalisierungsfähigkeit. Ein weiterer signifikanter Unterschied ist das Vorhandensein eines konstruktiven Verfahrens für die Bestimmung der notwendigen Neuronen (Dynamic Decay) bei den lokal lernenden Verfahren, was sich bei den global lernenden (CasCorr) als nicht praktikabel herausgestellt hat.

Verfahren	Lerngeschwindigkeit	Propagierungsgeschwindigkeit	Anpassung der Neuronenzahl	Generalisierung	Anzahl der Lernparameter
k-nächster Nachbar	++	−−	+	+−	1
RBF-Netze	+	−	−−	+	1
Dynamic Decay	++	−	+	+	2
RecBFN	++	−	+	+	2
Backpropagation	−	++	−−	++	1
QuickProp	+−	++	−−	++	2
RPROP	+−	++	−−	++	2
CasCorr	−−	++	++	+−	3
Entkoppelte Mod.	+−	+	−−	+−	$n * M$
ART-BP mod.	+−	+	−−	++	$n * 2$
Hierarchische Mod.	−	+	−−	++	$n * M$
Ensemble	−	+	−−	++	$n * 2 + 2$

Tabelle 5.2: Vergleich der ausgewählten Lernverfahren. n ist die Anzahl der Module und M die Anzahl der Parameter eines Moduls.

Der Hauptproblempunkt bei den hierarchischen Ansätzen besteht in der Segmentierung des Merkmalsraums (bis auf den Ensemble–Ansatz) in einfachere Unterräume. Die Segmentierung ist allerdings recht aufwendig. Da meistens BP–Netze eingesetzt werden, multipliziert sich der Aufwand für das Einlernen (Topologien, Initialisierungen). Das Einlernen kann jedoch parallel durchgeführt werden. Durch die Verkleinerung des Problems ist bei manchen Architekturen (Hierarchical moduls) der Lernprozess schneller als bei vergleichbaren BP–Netzen für das Gesamtproblem.

[4] Datensätze die aus realen Problemstellungen und Applikationen stammen.

5.2 Architektur der Lernkomponente

Ausgehend von dem Applikationsprofil der ILD steht eine sehr große Menge an Beispielen $\mathcal{X}$ zur Verfügung. Der Benutzer überwacht die Klassifikation und klassifiziert online neue Beispiele (vgl. Def. 3.1). Diese Beispiele bilden dann die Grundlage, um den Klassifikator K einzulernen.

Eine erfolgreiche Suche nach dem Klassifikator K impliziert eine für die ILD geeignete Lernarchitektur. Diese muß in der Lage sein, interaktiv zu lernen und gut zu generalisieren (vgl. Abschnitt 3.3, 3.4.2). Die Lernarchitektur ist eng mit der Wahl der Merkmale verknüpft. Die Frage, die sich in diesem Zusammenhang stellt, ist:

> *Welche Netze sind in welcher Reihenfolge auszuführen, um die gestellte Aufgabe zu lösen?*

Dabei ist zu beachten, daß die Anzahl der Freiheitsgrade des Gesamtsystems möglichst gering ist, damit das System mit einer akzeptablen Anzahl von Lernbeispielen eingelernt werden kann.

Das Vorgehen für das Erlernen einer solchen Klassifikation kann folgendermaßen gegliedert werden (vgl. Anderson's Lernmodell Abschnitt 3.3):

Grobklassifikation: Diese Phase ist zuständig für die Klassifikation des Offensichtlichen, der sog. Trivialbeispiele.

Feinklassifikation: Durch weitere Optimierung der internen Repräsentation wird die Klassifikationsgüte verbessert.

Um dem Benutzer die Manipulation großer nichtklassifizierter Datenmengen zu ermöglichen, wird das in Abschnitt 3.3 vorgestellte Spiralmodell mit abwechselnder Interaktions- und Optimierungsphase durchgeführt, bis die angestrebte Klassifikationsgüte erreicht worden ist.

Um eine möglichst schnelle Adaption der Beispiele während der Interaktionsphase zu erreichen, wird ein lokales, überwachtes Lernverfahren benötigt. Dies hat allerdings zur Folge, daß die Generalisierungsfähigkeit des lokalen Lernverfahrens nicht für eine eindeutige Klassifikation ausreichend ist. Um hingegen eine bessere Generalisierung zu erzielen, benötigt man ein global agierendes Verfahren. Um beiden Anforderungen gerecht zu werden, wird der Klassifikator in zwei Teile aufgespaltet. Auf diese Weise entsteht ein hybrider Klassifikator (siehe Abbildung 5.6), da zwei unterschiedlich arbeitende Verfahren gekoppelt werden (vgl. [SG97]).

Der mit dem lokalen Lernverfahren arbeitende interaktive *Dynamic Bounds*-Klassifikator hat die Aufgabe, oft auftretende Beispiele, die sich innerhalb eines Klassenclusters befinden, zu klassifizieren. Durch das lokale Lernen lassen sich neue Neuronen einfügen oder modifizieren, ohne daß sich diese Adaption auf die Klassifikationen des restlichen Merkmalsraums auswirkt. Damit ist eine schnelle online Adaption an neue Beispiele möglich. Dies gewährleistet dann, daß die nachfolgenden Beispiele bereits mit dem neu erworbenen Wissen verarbeitet werden. Um die Anwendung dieses Verfahrens zu vereinfachen, darf es keine Lernparameter besitzen.

Für die Entscheidungsgrenzen an den Rändern der Klassencluster wird der *Hybride Knoten*, ein hierarchisches Lernverfahren auf der Basis mehrerer global arbeitender

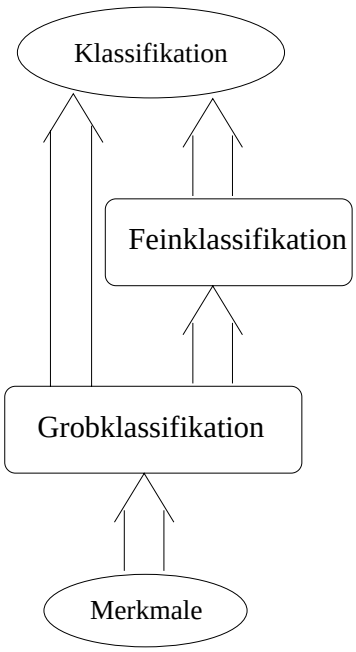


Abbildung 5.6: Architektur eines hybriden Klassifikators einer Lernkomponente. Die Grobklassifikation übernimmt die Bewertung der "Trivialbeispiele" und ist in der Lage schnell adaptiert zu werden, wohingegen die Feinklassifikation die Auswertung der "schwierigen" Kandidaten übernimmt und nur langsam adaptiert wird.

RPROP–Netze eingesetzt. Auf diese Weise werden die schwierigen Regionen (Klassenclusterränder) durch einen generalisierungsstarken Klassifikator abgedeckt (siehe Abschnitt 5.4).

Alle bisher bekannten Lernverfahren besitzen gewisse Nachteile beim Einsatz für Klassifikationsaufgaben. In [Giz96] wurde deshalb versucht, bestehende Lernverfahren so abzuwandeln, daß diese Nachteile zumindest teilweise kompensiert werden können.

Die Abwandlungen bestehender Lernverfahren können das Klassifikationsverhalten verbessern. Dennoch bleiben viele der genannten Probleme wie z. Bsp. die ungenügende Generalisierungsfähigkeit immer noch erhalten. Aus diesem Grund wurde das hier vorgestellte *Dynamic Bounds* Lernverfahren und das Verfahren des *Hybriden Knotens* entwickelt, das einige entscheidende Vorteile bezüglich der Lerngeschwindigkeit und der dynamischen Konfiguration gegenüber den vorgestellten Verfahren aufweist.

5.3 Lernen mit dem *Dynamische–Regionen*-Algorithmus

In diesem Abschnitt wird der sogenannte *Dynamische–Regionen* (*Dynamic Bounds*) Verfahren vorgestellt und entwickelt. Es ist ein inkrementeller Lernalgorithmus, der nach der Methode des lokalen überwachten Lernens arbeitet. Dieses Verfahren wurde im Rahmen dieser Arbeit für die schnelle, grobe Klassifikation entwickelt.

Bei der Entwicklung des *Dynamic Bounds*-Algorithmus wurden die Gesichtspunkte besonders stark berücksichtigt und darauf geachtet, daß *keine* Lernparameter (vgl. Definition 5.1) für dieses Verfahren notwendig sind, der Algorithmus eine Abbruchbe-

dingung besitzt und möglichst wenige Neuronen benutzt werden.

Im Folgenden wird nun der *Dynamic Bounds*-Algorithmus näher erläutert.

5.3.1 Die Idee des *Dynamische–Regionen*-Algorithmus

Die Grundidee des neuen Algorithmus ist es, den Eingaberaum in *sichere Bereiche*, für die Beispielsdaten existieren, und sog. *Hypothesenbereiche*, die die Annahmen über die Klassenzugehörigkeit des jeweiligen Unterraumes repräsentieren, aufzuteilen. Dadurch ist es möglich, auch unbekannte Eingaben zu erkennen, und trotzdem eine Hypothese über die jeweilige Klassenzugehörigkeit aufzustellen.

Das zugrundeliegende Neuronale Netz besteht aus zwei Schichten: aus der Eingabeschicht $\mathcal{I}$ und der Ausgabeschicht $\mathcal{O}$, die vollständig verbunden sind. Da dieses Verfahren für Klassifikationsprobleme eingesetzt wird, besteht die Aktivierungsfunktion des Neurons n aus zwei ineinander geschachtelten Hyperquadern, dem Hyperquader $\mathcal{S}_n$ für den *sicheren Bereich* und den Hyperquader $\mathcal{H}_n$ für den *Hypothesenbereich*. Diese Quader werden durch die Gewichte w_{nj} des Netzes aufgespannt:

$$w_{nj} = (z_j, h_j^{(l)}, h_j^{(r)}, s_j^{(l)}, s_j^{(r)}) \tag{5.1}$$

Die Elemente des Tupels w_{nj} beschreiben gerade die Stützstelle z_j und die Intervalle in der j-ten Dimension des Merkmalsraums $\mathcal{M}$. Das durch $(h_j^{(l)}, h_j^{(r)})$ aufgespannte *offene* Intervall repräsentiert den *Hypothesenbereich*, und das durch $[s_j^{(l)}, s_j^{(r)}]$ aufgespannte *geschlossene* Intervall repräsentiert den *sicheren Bereich* in der j-ten Dimension. Sie beschreiben die Unterräume des Merkmalsraums $\mathcal{M}_j$ in dieser Dimension, für die gilt:

$$\mathcal{S} \subseteq \mathcal{H} \subseteq \mathcal{M} \tag{5.2}$$

$$\left[s_j^{(l)}, s_j^{(r)}\right] \subseteq (h_j^{(l)}, h_j^{(r)}) \subseteq \mathcal{M}_j \tag{5.3}$$

Diese Kodierung der Gewichte erlaubt eine asymmetrische Ausdehnung der Einflußbereiche in jeder Dimension.

Um die Abwesenheit von Lernparametern zu gewährleisten, wird die benötigte Anzahl der Neuronen der Ausgabeschicht $\mathcal{O}$ während des Lernprozesses ermittelt. Aus diesem Grund wird das Lernen mit einer *leeren* Ausgabeschicht begonnen. Jedes Neuron der Ausgabeschicht wird im Verlauf des Einlernens eindeutig einer Klasse zugeordnet.

5.3.2 Propagierung

Die Klassenzugehörigkeiten der einzelnen Neuronen n werden als Vektoren $\vec{c}_n$ der Dimension von C repräsentiert. Alle Elemente von $\vec{c}_n$ sind gleich 0, außer dem Element, das die zu repräsentierende Klasse darstellt. Dieses wird auf 1 gesetzt.

Für das Propagierungsergebnis einer Eingabe $\vec{x} \in \mathcal{X}$ existieren die drei Fälle aus Algorithmus 5.1.

Die Parameter α und β dienen zur Unterscheidung zwischen einer sicheren Entscheidung und einer Hypothese über die Klassenzugehörigkeit. Ohne Beschränkung

$$DBprop(\vec{x}) = \begin{cases} \alpha * \{\vec{c}_n \mid \forall n : \vec{x} \in \mathcal{S}_n\} & \{\vec{c}_n \mid \forall n : \vec{x} \in \mathcal{S}_n\} \neq \emptyset \\ \beta * \{\vec{c}_n \mid \forall n : \vec{x} \notin \mathcal{S}_n \wedge \vec{x} \in \mathcal{H}_n\} & \{\vec{c}_n \mid \forall n : \vec{x} \notin \mathcal{S}_n \wedge \vec{x} \in \mathcal{H}_n\} \neq \emptyset \\ \vec{0} & \text{sonst} \end{cases}$$

Algorithmus 5.1: Propagierung bei *Dynamic Bounds*.

der Allgemeinheit wird $\alpha = 1$ und $\beta = 0.8$ gesetzt[5]. Durch diese Vorschrift ist es einfach zu entscheiden, ob ein neues Beispiel aufgetreten ist, für das keine Hypothese zu Verfügung steht (erkennbar an der Ausgabe $\vec{0}$), oder ob die Eingabe in der Hypothesen- oder sicheren Region eines oder mehrerer Neuronen liegt.

Auf diese Weise kann auch eine mehrdeutige Zugehörigkeit zu verschiedenen Klassen dargestellt werden. Dies unterstützt die Abschätzung der Qualität der Klassifikation.

5.3.3 Lernen

Bei diesem Lernverfahren handelt es sich um eine Kombination aus muster- und epochenweisem Lernen (siehe Algorithmus 5.2 und [Bis95]). Der Algorithmus verkleinert $\mathcal{H}$, vergrößert die Einflußbereiche $\mathcal{S}$ und fügt benötigte Neuronen während des Musterlernens ein. Während des Epochenlernens werden die Einflußbereiche $\mathcal{H}$ vergrößert, die Einflußbereiche $\mathcal{S}$ verkleinert und ungenutzte Neuronen gelöscht.

```
1 function DBlearn(B)
2   do
3     foreach n              // Initialisierung für Löschen
4       Dhit_n := 0; Dact_n := 0
5     foreach (x, t) ∈ B
6       e = DBprop(x)
7       DBpatternlearn(x, t, e)
8   until DBepochlearn()
```

Algorithmus 5.2: Lernalgorithus für *Dynamic Bounds*.

Für die Lernphase ist es notwendig, das Gewichtstupel folgendermaßen zu erweitern (siehe auch Abbildung 5.7):

$$w_{nj} = (z_j, h_j^{(l)}, h_j^{(r)}, s_j^{(l)}, s_j^{(r)}, h_{old,j}^{(l)}, h_{old,j}^{(r)}, s_{old,j}^{(l)}, s_{old,j}^{(r)}, s_{min,j}^{(l)}, s_{min,j}^{(r)}) \tag{5.4}$$

Dabei sind h_{old}, s_{old} Duplikate der h, s Elemente, die nach der Beendigung jeder Epoche kopiert werden. Die dazugehörigen Regionen werden mit $\mathcal{H}_{old}$ und $\mathcal{S}_{old}$ bezeichnet. Auf diese Weise läßt sich feststellen, ob ein Neuron im Verlauf einer Epoche

[5] α und β dienen lediglich der Unterscheidung im Klassifikationsergebnis und müssen deshalb $\alpha \neq \beta$ erfüllen.

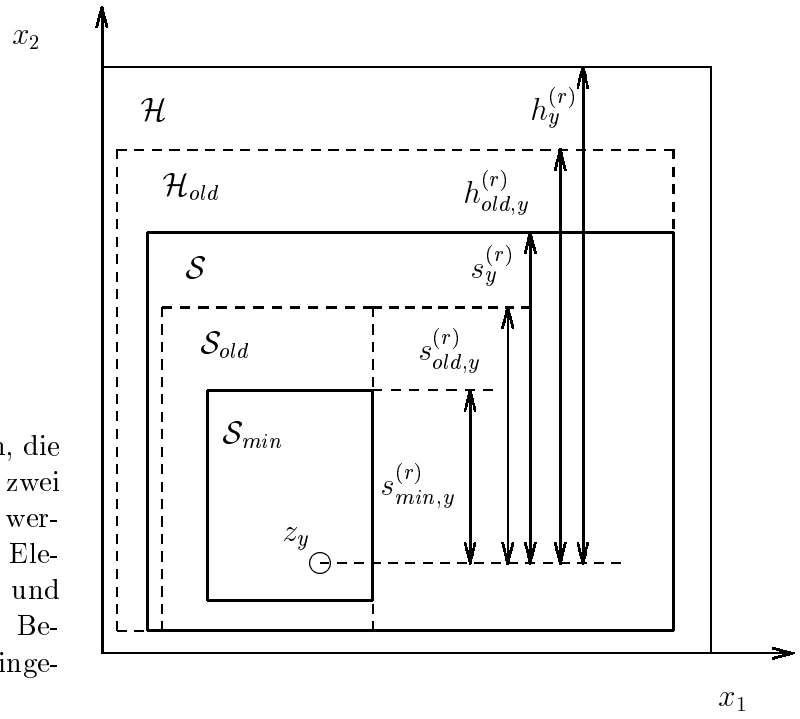


Abbildung 5.7: Regionen, die durch die Gewichte in zwei Dimensionen aufgespannt werden. Die entsprechenden Elemente des Gewichtstupels und die daraus resultierenden Bereiche sind entsprechend eingezeichnet.

modifiziert wurde. Die s_{min} Elemente beschreiben epochenübergreifend die maximale Ausdehnung der sicheren Bereiche und spannen einen Unterraum auf, der mit $\mathcal{S}_{min}$ bezeichnet wird.

Das Lernverfahren startet, wie oben bereits erwähnt, mit einer leeren Ausgabeschicht. Bei jeder Präsentation eines Beispiels werden, falls nötig, die Einflußbereiche $\mathcal{H}$ und $\mathcal{S}$ aller beteiligter Neuronen angepaßt, indem ihre Regionengrenzen *verkleinert* oder *expandiert* werden. Wird durch den Propagierungsschritt kein geeignetes Neuron gefunden, wird ein neues Neuron eingefügt.

5.3.3.1 Musterlernen

Das Musterlernen wird auf jedes Beispiel angewendet und sorgt für die unmittelbare Beseitigung von Kollisionen. Der Musterlern-Algorithmus kann aus den Algorithmen 5.4 und 5.3 entnommen werden. Die fünf möglichen Fälle werden im folgenden erläutert (siehe auch Abbildung 5.8).

Sei $\vec{x}$ ein Lernbeispiel mit der Klassifikation $\vec{t}$ und durch den Propagierungsschritt wird jedem Neuron n die Ausgabeklasse $\vec{c}_n$ zugeordnet:

Fall 1: Korrekte Klassifikation:

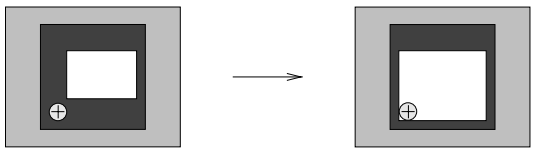
$$\vec{t} = \vec{c}_n \wedge \vec{x} \in \mathcal{S}_n \tag{5.5}$$

Das Lernbeispiel der richtigen Ausgabeklasse liegt innerhalb des sicheren Bereichs. Das Neuron n klassifiziert das Lernbeispiel $\vec{x}$ bereits richtig.

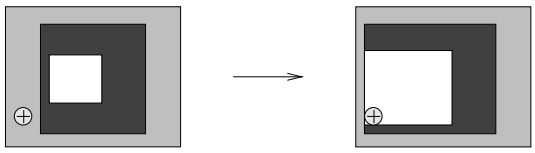
Die Region $\mathcal{S}_{min,n}$ wird derart erweitert, daß sie $\vec{x}$ enthält.

Fall 2: Expansion der *sicheren Region*:

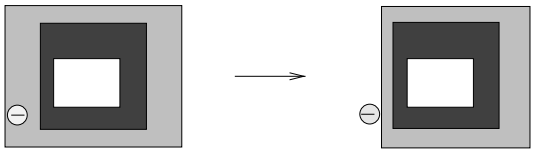
$$\vec{t} = \vec{c}_n \ \wedge \ \vec{x} \notin \mathcal{S}_n \ \wedge \ \vec{x} \in \mathcal{H}_n \ \wedge \ \vec{x} \in \mathcal{H}_{old,n} \tag{5.6}$$



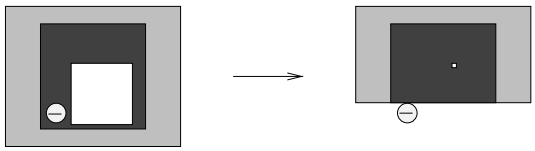
(a) Fall 1: Korrekte Klassifikation



(b) Fall 2: Expansion der *sicheren Region*



(c) Fall 3: Verkleinern der *Hypothesenregion*



(d) Fall 4: Verkleinern der Regionen

Abbildung 5.8: Die vier unterschiedlichen Fälle beim Musterlernen von *Dynamic Bounds* im zweidimensionalen Merkmalsraum für **DBprop**$(\vec{x}) \neq \vec{0}$. $\mathcal{S}_{min}$ ist weiß, $\mathcal{S}$ schwarz und $\mathcal{H}$ grau dargestellt. Positives Beispiel ist als +, negatives als − abgebildet. Der fünfte, nicht dargestellte Fall ist die Nichtklassifikation der Eingabe, die zur Neuroneinfügung führt.

Hier liegt ein Lernbeispiel der richtigen Ausgabeklasse innerhalb des alten Hypothesenbereiches. Das Neuron n wäre in der Lage, das Lernbeispiel richtig zu klassifizieren. In diesem Fall kann der sichere Bereich *expandiert* werden. Dies muß in jeder Dimension geschehen, da das Neuron n das Lernbeispiel $\vec{x}$ ab sofort als sicher klassifizieren soll.

Die Region $\mathcal{S}_{min,n}$ wird derart erweitert, daß sie $\vec{x}$ enthält.

Fall 3: Verkleinern der *Hypothesen–Region*:

$$\vec{t} \neq \vec{c}_n \wedge \vec{x} \notin \mathcal{S}_n \wedge \vec{x} \in \mathcal{H}_n \tag{5.7}$$

In diesem Fall liegt eine Fehlklassifikation des Lernbeispiels $\vec{x}$ vor. Hier liegt ein Lernbeispiel der falschen Klasse im Hypothesenbereich von Neuron n vor. Um diese Fehlklassifikation zu beseitigen, muß der Hypothesenbereich $\mathcal{H}_n$ in mindestens einer Dimension *geschrumpft* werden (siehe Abschnitt 5.3.3.2).

Fall 4: Verkleinern der Regionen:

$$\vec{t} \neq \vec{c}_n \wedge \vec{x} \in \mathcal{S}_n \tag{5.8}$$

In diesem Fall liegt ebenfalls eine Fehlklassifikation des Lernbeispiels $\vec{x}$ vor. Da das Lernbeispiel aber sogar vollständig im sicheren Bereich liegt, müssen sowohl $\mathcal{H}_n$ als auch die $\mathcal{S}_n$ des Neurons in mindestens einer Dimension *geschrumpft* werden (siehe Abschnitt 5.3.3.2).

Liegt $\vec{x}$ innerhalb der Region $\mathcal{S}_{min,n}$, dann werden alle $s^{(l)}_{min,i}, s^{(r)}_{min,i}$ auf 0 gesetzt.

Fall 5: Falsche Ausgabe: Wird durch den Propagierungsschritt kein Neuron der gleichen Klasse gefunden (dritter Fall im Algorithmus 5.1), so wird ein neues Neuron n eingefügt, dessen Gewichte wie folgt initialisiert werden:

$$w_{nj} = (z_j, -\infty, \infty, 0, 0, -\infty, \infty, 0, 0, 0, 0) \tag{5.9}$$

Der *Hypothesenbereich* wird gleich unendlich gesetzt, und der *sichere Bereich* wird so gesetzt, daß nur die Stützstelle enthalten ist. $\vec{c}_n$ wird auf $\vec{t}$ gesetzt.

```
1  function DBexpand(x⃗, z⃗, l⃗, r⃗)
2    ∀ j : l_j := min(z_j − l_j, x_j)
3    ∀ j : r_j := max(z_j + r_j, x_j)
```

Algorithmus 5.3: Vergrößern von Regionen.

```
1  function DBpatternlearn(x⃗, t⃗, e⃗)
2    foreach n ∈ Neuronen
3      case t⃗ = c⃗_n ∧ x⃗ ∈ S_n
4        // Fall 1: korrekte Klassifikation
5        Dact_n+ = 1
6      case t⃗ = c⃗_n ∧ x⃗ ∉ S_n ∧ x⃗ ∈ H_n ∧ x⃗ ∈ H_old,n
7        // Fall 2: Expansion der sicheren Region
8        DBexpand(x⃗, z⃗_n, s_n^(l)→, s_n^(r)→)
9      case t⃗ ≠ c⃗_n ∧ x⃗ ∉ S_n ∧ x⃗ ∈ H_n
10       // Fall 3: Verkleinern der Hypothesen-Region
11       DBshrink(x⃗, z⃗_n, h_n^(l)→, h_n^(r)→)
12     case t⃗ ≠ c⃗_n ∧ x⃗ ∈ S_n
13       // Fall 4: Verkleinern der Regionen
14       DBshrink(x⃗, z⃗_n, s_n^(l)→, s_n^(r)→); DBshrink(x⃗, z⃗_n, h_n^(l)→, h_n^(r)→)
15       s_min,n^(r)→ := 0⃗; s_min,n^(l)→ := 0⃗
16   end
17   if x⃗ ∉ S_n      // Fall 5: neues Neuron n Einfügen
18     c⃗_n := t⃗; ∀j : w_nj := (x_j, −∞, ∞, 0, 0, −∞, ∞, 0, 0, 0, 0)
19   if x⃗ ∈ S_n ∧ ∃k ≠ n : x⃗ ∈ S_k
20     Dhit_n+ = 1
21   if x⃗ ∈ S_n ∧ ∀k ≠ n : x⃗ ∉ S_k
22     S_min,n := S_n
```

Algorithmus 5.4: Musterlernalgorithmus

Weiterhin werden für die Dauer einer Epoche in $Dact_n$ alle sicheren Aktivierungen des Neurons ($\vec{x} \in \mathcal{S}_n$) gezählt und in $Dhit_n$ werden diejenigen sicheren Aktivierungen, bei denen gleichzeitig auch ein anderes Neuron der gleichen Klasse aktiv war, gezählt.

Falls das Neuron n als einziges als sicher aktiviert wird, dann wird dessen Minimalbereich erweitert

$$\mathcal{S}_{min,n} := \mathcal{S}_n$$

um die Minimalregion zu ermitteln, innerhalb derer das Neuron als einziges zuständig ist. Wird hingegen eine Aktivierung des sicheren Bereichs durch eine falsche Klasse aktiviert, dann wird die Minimalregion zurückgesetzt:

$$\vec{x} \in \mathcal{S}_n \quad \wedge \quad \vec{t} \neq \vec{c}_n \qquad \longrightarrow \qquad \mathcal{S}_{min,n} := \vec{0}$$

Diese Berechnungen werden später für die Löschstrategie benötigt (siehe Abschnitt 5.3.3.3).

5.3.3.2 Anpassungsstrategie der Regionengrenzen

Für das *Verkleinern* der Regionengrenzen gibt es keine eindeutige Strategie. Prinzipiell werden aber beim *Verkleinern* folgende Ziele verfolgt:

- Ein fehlerhaft klassifiziertes Lernbeispiel muß nach dem *Verkleinern* außerhalb des Einflußbereichs des *verkleinerten* Neurons liegen.
- Das Volumen, das das Neuron überdeckt, sollte möglichst groß bleiben, damit nicht zu viele Neuronen mit kleinem Einflußbereich entstehen. Dies setzt ansonsten die Leistungsfähigkeit des Algorithmus entscheidend herab.

Um diese beiden Bedingungen zu erfüllen, gibt es genau drei Möglichkeiten, die in Abbildung 5.9 dargestellt sind.

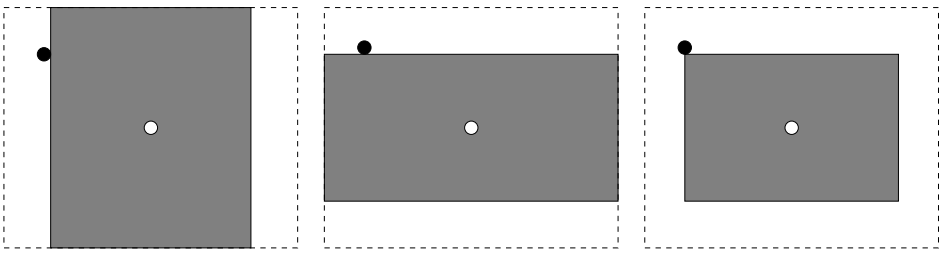


Abbildung 5.9: Drei Möglichkeiten, wie die Regionengrenzen bei *Dynamic Bounds* im zweidimensionalen Merkmalsraum aufgrund von Falschklassifikation durch Verkleinerung angepaßt werden können.

Wie aus Abbildung 5.9 deutlich wird, ist es nicht notwendig, die Regionengrenzen in allen Dimensionen zu verkleinern. Es genügt, die Einschränkung des Einflußbereichs in nur einer Dimension vorzunehmen, denn jede weitere würde das Volumen $\mathcal{H}$ nur unnötig verringern. Für ein Lernbeispiel $\vec{x} = (x_1, ..., x_n)$ und Neuron n mit den Intervallgrenzen $h_{nj}^{(l)}$ und $h_{nj}^{(r)}$ für alle j Dimensionen liegt $\vec{x}$ genau dann innerhalb des Einflußbereichs eines Neurons, wenn gilt:

$$\vec{x} \in \mathcal{H}_n \;\Leftrightarrow\; \forall j : x_j \in (z_{nj} - h_{nj}^{(l)}, z_{nj} + h_{nj}^{(r)}) \tag{5.10}$$

Der Verlust an Volumen durch die Einschränkung des Einflußbereichs in nur einer Dimension kann durch die Gleichung 5.11 ausgedrückt werden. Es sei hier ohne Einschränkung der Allgemeinheit das Verkleinern der rechten Hälfte des Volumens in der Dimension j von rechts ($h_{nj}^{(r)}$) angenommen:

$$\begin{aligned}(h_{nj}^{(r)} - \mid z_{nj} - x_j \mid) \times \prod_{\substack{1 \leq k \leq j \\ k \neq n}} h_{nk}^{(r)} \\ = (h_{nj}^{(r)} - \mid z_{nj} - x_j \mid) \times \frac{\prod_{1 \leq k \leq j} h_{nk}^{(r)}}{h_{nj}^{(r)}}\end{aligned} \tag{5.11}$$

Der konstante Term $\prod_{1 \leq k \leq j} h_{nk}^{(r)}$ kann wegfallen, und somit erhält man die optimale Dimension zum *Verkleinern* durch die Formel 5.12. Deshalb genügt es, die Regionengrenzen in einer Dimension zu verkleinern.

$$\min_{1 \leq k \leq j} \left\{ \frac{(h_{nk}^{(r)} - \mid z_{nk} - x_k \mid)}{h_{nk}^{(r)}} \right\} \tag{5.12}$$

Ein weiteres Problem ergibt sich aus der Tatsache, daß die *Hypothesenbereiche* $\mathcal{H}$ der Neuronen zu Beginn auf unendlich initialisiert werden. Da immer die Dimension, die den geringsten Volumenverlust verursacht, verkleinert wird, wird eine unendliche Dimension niemals wieder reduziert, sobald eine endliche existiert. Dies kann dazu führen, daß ein j-dimensionales Volumen zu einem $j-1$-dimensionalen *Schlauch* degeneriert. Um die Bildung solcher *Schläuche* zu verhindern, sind unterschiedliche Strategien denkbar:

1. Für das Verkleinern einer Dimension wird ein Minimum eingeführt. Erreichen alle endlichen Dimensionen diesen Schwellwert, muß beim nächsten Schrumpfvorgang eine unendliche Dimension aufgegeben werden.

2. Bisher wurde nur die Annahme getroffen, daß das Volumen der Regionen von Interesse ist. Es kann aber auch die Varianz der Ausdehnung, also die räumliche Lokalität interessieren. Um diese räumliche Lokalität zu erreichen, werden die unendlichen Regionengrenzen zuerst aufgegeben (siehe Algorithmus 5.5).

Durch empirische Beobachtungen hat sich gezeigt, daß die zweite Strategie zur Anpassung der Regionengrenzen als die bessere anzusehen ist. Die Anzahl der Neuronen steigt während des Musterlernens, weil die unendlichen Regionengrenzen relativ schnell aufgegeben werden, und dadurch das Volumen geringer ist als bei der ersten Strategie. Die Anzahl der Neuronen geht aber während des Epochenlernschrittes zurück und zwar so weit, daß die Anzahl geringer wird als bei der ersten Strategie. Die Generalisierungsfähigkeit wird ebenfalls wesentlich verbessert.

Ein schwerwiegendes Problem der ersten Strategie stellen Bereiche dar, die quer durch den Eingaberaum führen und damit Fehlklassifikationen erzeugen. Aus diesem Grund wird die zweite Strategie für die Anpassung von $\mathcal{H}$ verwendet.

```
1  function DBshrink(x⃗, z⃗, l⃗, r⃗)
2    case ∃j : x_j < z_j ∧ l_j = ∞
3      l_j := z_j − x_j
4    case ∃j : x_j > z_j ∧ r_j = ∞
5      r_j := x_j − z_j
6    else
7      min { (l_j − |z_j − x_j|) / l_j , (r_j − |z_j − x_j|) / r_j }
8      if l_j ist Minimum
9        l_j := z_j − x_j
10     if r_j ist Minimum
11       r_j := x_j − z_j
```

Algorithmus 5.5: Das Verkleinern der Regionengrenzen.

5.3.3.3 Epochenlernen

Nach der Beendigung einer Epoche werden die nicht benötigten, durch andere überdeckten Neuronen gelöscht, und es wird überprüft, ob keine Modifikation der Einflußbereiche stattgefunden hat, um gegebenenfalls den Lernprozess zu beenden. Falls Änderungen auftreten, wird versucht, den Hypothesenbereich weiter auszudehnen (siehe auch Algorithmus 5.6).

Um ein Neuron löschen zu können, ohne daß dabei Wissen verloren geht, muß es während der gesamten Epoche durch andere Neuronen überdeckt worden sein. Dies bedeutet, daß diese Neuronen die gleichen Aktivierungen zum gleichen Zeitpunkt wie das zu löschende Neuron aufweisen müssen. Zu Anfang jeder Epoche werden zu diesem Zweck die *Dhit*'s und *Dact*'s initialisiert (siehe Algorithmus 5.2), um die Aktivierungen durch den **DBpatternlearn** Algorithmus zu protokollieren.

Die Überdeckung aller Neuronen wird am Ende jeder Epoche untersucht, in dem geprüft wird, ob $Dhit_n = Dact_n$ gilt. Ist dies der Fall, kann das Neuron n gelöscht werden, da es bei jeder sicheren Aktivierung durch ein oder mehrere Neuronen überdeckt wird. Es wurde nämlich im Verlauf der letzten Epoche genau dann aktiv, wenn auch sein Überdeckungsnachbar aktiv war.

Um ein späteres Löschen, und damit die Vereinfachung des Modells zu erzielen, müssen die Regionen erst expandiert werden, um auf diese Weise eine Überlappung zu erzielen und dadurch das Löschen von Neuronen zu ermöglichen. Dazu müssen die Hypothesenregionen $\mathcal{H}$ der Neuronen expandiert werden, damit die sicheren Regionen $\mathcal{S}$ die Möglichkeit bekommen zu expandieren. Zuerst wird jedoch die alte Regionsausdehnung der Neuronen gesichert, damit später erkannt werden kann, ob diese Expansion tatsächlich zur Veränderung der Einflußbereiche geführt hat:

$$\mathcal{H}_{old,n} := \mathcal{H}_n \quad \text{und} \quad \mathcal{S}_{old,n} := \mathcal{S}_n$$

Die Expansion der Hypothesenregionen wird durch folgende Formel für alle Neuronen n und Gewichtskomponenten j durchgeführt:

$$\begin{aligned} h_j^{(l)} &:= h_j^{(l)} + \max(\epsilon * (h_j^{(l)} - s_j^{(l)}), \mu) \\ h_j^{(r)} &:= h_j^{(r)} + \max(\epsilon * (h_j^{(r)} - s_j^{(r)}), \mu) \end{aligned} \tag{5.13}$$

Dabei bildet ϵ den Vergrößerungsmultiplikator, der nur der Bedingung $\epsilon > 0$ genügen muß. Im Rahmen dieser Arbeit wurde er auf einen Wert von 1.5 festgelegt. μ stellt den kleinsten Vergrößerungsschritt dar und muß ebenfalls größer als 0 sein (hier bei 0.005). Diese Parameter beeinflussen die Anzahl der benötigten Epochen bis zur Terminierung des Verfahrens.

Die Überprüfung, ob Änderungen während der letzten Epoche aufgetreten sind, wird anhand von Gleichung 5.14 durchgeführt:

$$\forall n : \mathcal{H}_n = \mathcal{H}_{old,n} \quad \wedge \quad \mathcal{S}_n = \mathcal{S}_{old,n} \quad \wedge \quad \mathcal{S}_n = \mathcal{S}_{min,n} \tag{5.14}$$

Werden weder Veränderungen festgestellt, noch sind Neuronen hinzugefügt oder gelöscht worden, ist der Lernvorgang abgeschlossen.

Der vollständige Epochenlern-Algorithmus ist in Algorithmus 5.6 dargestellt.

```
1  function DBepochlearn()
2    foreach n ∈ Neuronen
3      if Dhit_n = Dact_n ∧ ¬∃(Nachbarneuron verkleinert)
4        Lösche Neuron k :
5          min{k = n ∨ k ∈ {Nachbarn von n}| Volumen von S_k}
6        ∀j ∈ {Nachbarn von k} : Dhit_j := 0
7    if ¬(Neuron gelöscht) ∧ ¬(Neuron eingefügt)
8        ∧ (∀n : H_n = H_old,n ∧ S_n = S_old,n ∧ S_min,n = S_old,n)
9      return true                    // Terminieren
10   foreach n ∈ Neuronen
11     if H_n = H_old,n ∧ S_n = S_old,n)
12       S_n := S_min,n ; H_old,n := H_n   // Verkleinern auf S_min,n
13     else
14       H_old,n := H_n
15       foreach Dimension j           // Expansion von H_n
16         h_nj^(l) := h_nj^(l) + ε * max(h_nj^(l) − s_nj^(l), μ)
17         if z_nj − h_nj^(l) ≤ min_{b∈B} b_j
18           h_nj^(l) := ∞
19         h_nj^(r) := h_nj^(r) + ε * max(h_nj^(r) − s_nj^(r), μ)
20         if z_nj + h_nj^(l) ≥ max_{b∈B} b_j
21           h_nj^(r) := ∞
22     S_old,n := S_n
23   return false
```

Algorithmus 5.6: Epochenlern-Algorithmus.

5.3.4 Integration von Regelwissen

Ist a priori Wissen in Form von Regeln vorhanden, dann kann es ohne großen Aufwand in die Repräsentation von *Dynamic Bounds* überführt werden. Abgesehen von der Eingabeform der Regeln, die im Abschnitt 6.4.1 näher betrachtet wird, ist es ausschlaggebend, ob es sich um eine *scharfe* oder *fuzzy* Regel handelt, die in die interne Darstellung des Algorithmus überführt wird, ohne daß es die Funktionalität des Algorithmus beeinflußt.

Eine Regel stellt dabei die Abbildung eines Bereichs des Merkmalsraums zu einer Klasse dar. Diese Regeln lassen sich in den *Dynamic Bounds*-Algorithmus integrieren, indem sie gerade die Ausdehnungen der $\mathcal{S}$-Regionen bestimmen. Die $\mathcal{H}$-Regionen werden für die *scharfen* Regeln den $\mathcal{S}$-Regionen gleichgesetzt. Bei *fuzzy* Regeln werden die $\mathcal{H}$-Regionen derart gesetzt, daß eine trapezoide Aktivierungsfunktion entsteht (siehe auch [Har98, Ber97]).

Jede Regel wird durch ein Neuron repräsentiert. Diese Neuronen werden explizit markiert und während des Lernprozesses nicht beachtet. Sie erzeugen bei der Propagierung Aktivierungen, die höher sind als die der nicht markierten Neuronen. Auf diese Art läßt sich auch unterscheiden, aufgrund welcher Entscheidungsgrundlage klassifiziert worden ist. Dies kann auch als Hinweis für den Benutzer während der Interaktion dienen.

5.3.5 Aufwandsabschätzung

Der Aufwand des *Dynamic Bounds*-Algorithmus für eine Epoche beträgt $O(\min\{d * n^2, n * d * p\})$. Dabei bezeichnet d die Dimension des Eingaberaumes, n die Dimension der Merkmale und $p = |\mathcal{B}|$ die Anzahl der Lernbeispiele. Der quadratische Aufwand des Verfahrens wird durch den Algorithmus *DBepochlearn* verursacht (siehe Tabelle 5.3), der für die Löschung der Neuronen verantwortlich ist.

Algorithmus	Aufwand
DBprop	$O(d * n)$
DBpatternlearn	$O(d * n)$
DBexpand	$O(d)$
DBshrink	$O(d)$
DBepochlearn	$O(d * n^2)$
Eine Epoche mit **DBlearn**	$O(\min\{d * n^2, n * d * p\})$

Tabelle 5.3: Aufwandsabschätzung der Teilalgorithmen bei *Dynamic Bounds*. Dabei bezeichnet d die Dimension des Eingaberaumes, n die Dimension der Merkmale und $p = |\mathcal{B}|$ die Anzahl der Lernbeispiele.

Durch die Löschstrategie von *Dynamic Bounds* wird die Anzahl der benötigten Neuronen verringert. Je nach Beschaffenheit des Problems schwankt die Anzahl der Neuronen während des Lernvorgangs. Die Anzahl der Neuronen kann aber die Anzahl der Lernbeispiele niemals überschreiten und liegt im Allgemeinen bei einem Bruchteil. Durch die wiederholte Eliminierung der Neuronen wird auch der Aufwand niedrig gehalten, der im wesentlichen durch die Anzahl der Neuronen bestimmt wird.

5.3.6 Bewertung

Der wesentliche Vorteil des *Dynamic Bounds*-Verfahrens besteht in der lernparameterlosen und adaptiven Anpassung der Anzahl der Neuronen an die gegebene Aufgabe. Dabei werden Neuronen sowohl dynamisch hinzugefügt als auch gegebenenfalls entfernt. Darüber hinaus kann der *Dynamic Bounds*-Algorithmus für das online Lernen verwendet werden, indem nur der **DBpatternlearn** Algorithmus für das Adaptieren des Modells durch die neu hinzukommenden Beispiele verwendet wird.

Die Löschstrategie sei am Beispiel des bekannten Spiralbenchmarks verdeutlicht (siehe Abbildung 5.10).

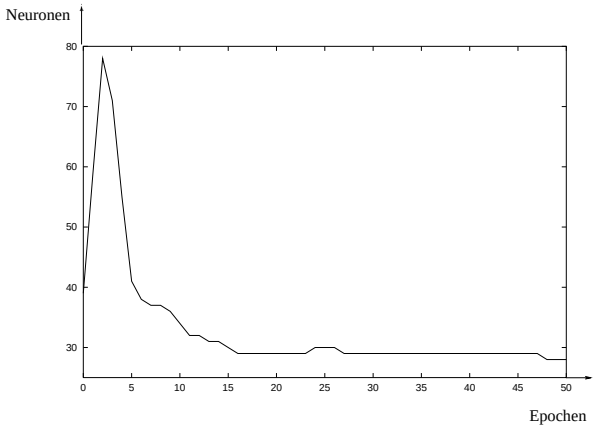


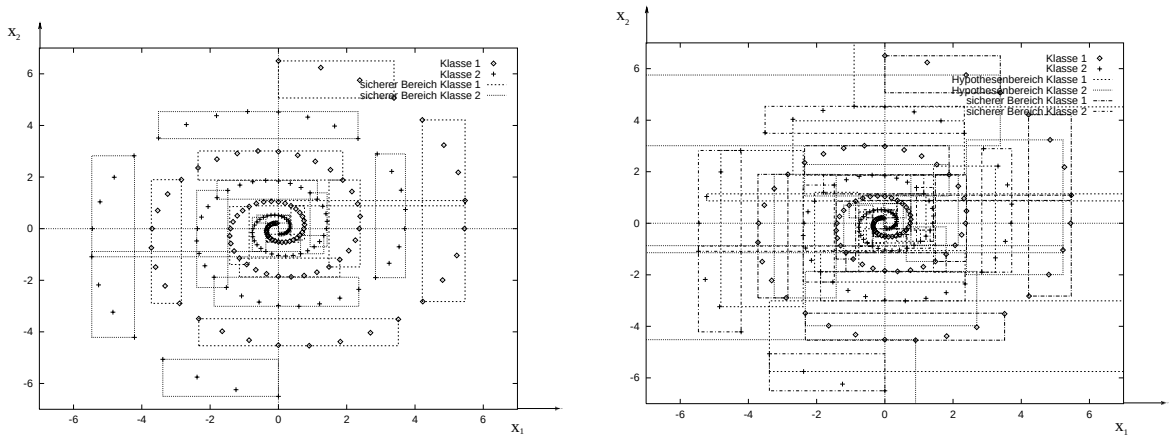
Abbildung 5.10: Veränderung der Anzahl der Neuronen während des Lernvorgangs bei dem Spiralbenchmark. Anfangs steigt die Anzahl durch die Hinzunahme neuer Neuronen, da die Neuronausdehnung zu diesem Zeitpunkt nicht optimal war. Später werden die Neuronen dynamisch entfernt, da durch die Ausdehnungen der Neuronen sich ihrem Optimum annähern und dadurch Überlappungen der Regionen entstehen.

Durch den oben beschriebenen Lernvorgang entsteht eine suboptimale Segmentierung des Merkmalsraums, denn der Einflußbereich der Neuronen wird im Verlauf des Lernvorgangs vergrößert, wodurch Neuronen vollkommen durch andere überdeckt werden. Das Endergebnis des Lernvorgangs am Beispiel des Spiralbenchmarks ist in Abbildung 5.11 zu sehen. Der *Dynamic Bounds* benötigt 28 Neuronen für dieses Klassifikationsproblem.

Als korrekte Klassifikationen der Eingabe werden nur diejenigen betrachtet, die eindeutig innerhalb von $\mathcal{S}$ liegen (vgl. Abschnitt 5.3.2).

An Hand von Abbildung 5.11(a) kann gut die Aufteilung des Merkmalsraums nachvollzogen werden. Die $\mathcal{S}$-Regionen umschließen gerade alle Beispiele, die sie repräsentieren. Zwischen Bereichen gleicher Klassen entstehen dabei Überlappungen. Leider besteht auch eine Überlappung zwischen Bereichen unterschiedlicher Klassen, wie in Abbildung 5.11(a) deutlich an den zwei waagerechten vom Zentrum ausgehenden Schläuchen zu sehen ist. Sie durchqueren die Bereiche anderer Klassen. Dadurch, daß diese Bereiche zu Beginn des Lernvorgangs große Einflußbereiche besitzen, degenerieren sie durch ungünstige Reihenfolge der Präsentation der Beispiele - aufgrund der durchgeführten Verkleinerung - zu Schläuchen.

Dieses Verhalten läßt sich unterbinden, indem ein Lernparameter λ für den maxi-



(a) $\mathcal{S}$ Bereiche beider Klassen **(b)** $\mathcal{H}$ und $\mathcal{S}$ Bereiche beider Klassen

Abbildung 5.11: Einflußbereiche der Neuronen bei dem Spiral-Benchmark.

malen Unterschied der Dimensionsausdehnumg eines Neurons eingeführt wird:

$$\min_i\{h_i^{(r)} + h_i^{(l)}\} * \lambda \geq \max_i\{h_i^{(r)} + h_i^{(l)}\} \tag{5.15}$$

Das Problem des Lernparameters λ besteht in der Abhängigkeit von der Struktur des Problems. Er ist also stark applikationsabhängig und muß für jede Problemstellung gefunden werden. Dies steht aber im Widerspruch zu den gesetzten Zielen für die Lernkomponente. Andererseits wird durch die Propagierung (siehe Algorithmus 5.1) eine Mehrfachklassifikation zu unterschiedlichen Klassen erkannt. Dieses Beispiel kann also als uneindeutig erkannt und abgelehnt werden. Der Benutzer wird auf diese Weise sofort auf solche Inkonsistenzen aufmerksam gemacht. Aus diesen Grund kann auf den Lernparameter λ verzichtet werden.

Durch die großen Restriktionen, unter denen die sicheren Regionen gebildet werden, und dem Ablehnen aller mehrdeutigen oder außerhalb der sicheren Regionen liegenden Klassifikationen, ist *Dynamic Bounds* ein sehr zuverlässiger lernparameterloser Klassifikator für diese Bereiche. Diese Behauptung kann durch die in Tabelle 5.4 zusammengefaßten Ergebnisse des Vergleichs zwischen *Dynamic Bounds*, k-nächster Nachbar Klassifikationsverfahren und dem Entscheidungsbaum-Klassifikationsverfahren C4.5 (siehe [Qui93]) bestätigt werden.

Die Tests wurden mit Hilfe des 10-fach Cross-Validation Verfahrens ermittelt (siehe Abschnitt D.4.1). Die hier verwendeten Beispielsdatensätze entstammen [Pre94, MM96] und werden im Anhang E detailliert beschrieben. Es handelt sich dabei größtenteils um *real world datasets*, also um Datensätze aus konkreten Applikationen.

Um das *Dynamic Bounds*-Verfahren zu vergleichen wurden Experimente mit 40 unterschiedlichen *real world datasets* durchgeführt. Die Ergebnisse sind in der Tabelle 5.4 zusammengefaßt. Aufgetragen sind die unterschiedlichen Größen der Lern- und der Testdatensätze und die Falschklassifikationen der einzelnen Verfahren bei den 40 unterschiedlichen Datensätzen.

Bei *Dynamic Bounds* stellt die Spalte $\notin \mathcal{S}$ die durchschnittliche prozentuelle Anzahl von Falschklassifikationen dar und $\emptyset$ die der Zurückweisungen. Bei den anderen Verfahren ist nur die durchschnittliche prozentuelle Anzahl von Falschklassifikationen

Projekt			*Dynamic Bounds*			C4.5	k-NN
	$\lvert\mathcal{B}_l\rvert$	$\lvert\mathcal{B}_t\rvert$	$\notin \mathcal{S}$	$\emptyset$	$\sum$		
ann-thyroid	6480	720.00	0.5	1.8	(2.3)	2.0	7.6
australian	621	69.00	11.2	18.3	(29.5)	31.4	36.7
balance-scale	562.5	62.50	13.3	8.2	(21.5)	21.3	29.8
bupa	310.5	34.50	18.8	31.9	(50.7)	40.9	35.1
car	1555.2	172.80	5.0	6.2	(11.2)	5.7	53.9
clean1	428.4	47.60	2.7	73.9	(76.6)	26.7	18.7
clean2	5940	660.00	2.2	21.0	(23.2)	4.8	3.7
defects1	1091.00	120.00	4.3	28.2	(32.5)	13.2	14.1
ecoli	302.4	33.60	8.9	23.2	(32.1)	19.0	73.2
flag	174.6	19.40	2.6	91.8	(94.4)	49.5	68.6
german	900	100.00	16.0	32.4	(48.4)	31.4	39.8
glass	192.6	21.40	0.0	29.9	(29.9)	2.8	0.9
hayes-roth	118.8	13.20	15.9	34.1	(50)	23.5	56.1
heart	243	27.00	14.4	24.8	(39.2)	29.3	42.6
ionosphere	315.9	35.10	1.4	21.1	(22.5)	19.7	11.4
iris	135	15.00	4.0	10.7	(14.7)	6.0	3.3
kr-vs-kp	2876.4	319.60	1.2	1.7	(2.9)	0.6	50.6
krkopt	25250.4	2805.60	9.9	15.0	(24.9)	32.0	76.9
lenses	21.6	2.40	12.5	20.8	(33.3)	20.8	91.7
letter-recognition	18000	2000.00	2.5	14.1	(16.6)	14.0	11.9
lweld	8892	988.00	2.6	4.6	(7.2)	3.5	5.5
monks-1	500.4	55.60	1.3	0.2	(1.5)	4.1	20.0
monks-2	540.9	60.10	2.5	4.0	(6.5)	1.8	24.6
monks-3	498.6	55.40	2.7	1.1	(3.8)	1.1	20.2
new-thyroid	193.5	21.50	2.8	12.6	(15.4)	7.4	7.9
nursery	11664	1296.00	2.6	3.1	(5.7)	1.2	55.7
page-blocks	4925.7	547.30	2.1	3.4	(5.5)	3.3	4.3
sat	5791.5	643.50	7.3	11.7	(19)	14.9	11.6
segment	2079	231.00	1.9	8.0	(9.9)	6.2	5.6
segmentation	2079	231.00	1.4	8.7	(10.1)	6.7	5.5
shuttle	52200	5800.00	0.0	0.1	(0.1)	0.2	0.2
sonar	187.2	20.80	7.7	68.8	(76.5)	29.8	23.6
soybean-small	42.3	4.70	0.0	23.4	(23.4)	4.3	23.4
tic-tac-toe	862.2	95.80	0.3	3.5	(3.8)	15.3	30.5
vehicle	761.4	84.60	12.2	38.7	(50.9)	28.4	40.5
vowel-context	891	99.00	4.6	45.6	(50.2)	50.6	1.5
waveform	4500	500.00	11.1	16.3	(27.4)	25.7	23.0
waveform-+noise	4500	500.00	12.2	22.0	(34.2)	25.8	24.8
wine	160.2	17.80	1.7	35.4	(37.1)	7.3	28.1
zoo	90.9	10.10	0.0	28.7	(28.7)	38.6	70.3

Tabelle 5.4: Vergleich der Falschklassifikationen zwischen *Dynamic Bounds*, C4.5 und k-NN (k-nächster Nachbar) Klassifikationsverfahren mittels 10-CV. $|\mathcal{B}_l|$ ist die durchschnittliche Anzahl der Lern-, $|\mathcal{B}_t|$ die der Testbeispiele. $\notin \mathcal{S}$ stellt die durchschnittliche prozentuelle Anzahl von Falschklassifikationen dar und $\emptyset$ die der Zurückweisungen. Beide Werte sind in $\sum$ aufaddiert.

aufgezeichnet. Diese Zahlen stellen eine Bewertung der Verfahren bezüglich der Falschklassifikationen ohne Zurückweisungen.

Aus diesen Werten läßt sich der Schluß ziehen, daß *Dynamic Bounds* gut für die sichere Klassifikation geeignet ist. Durch seine Möglichkeit der Unterscheidung der Klassifikation in sichere Ergebnisse, Hypothesen, Mehrfachklassifikationen und Nichtklassifikationen erzielt er gerade bei den sicheren Klassifikationen sehr gute Ergebnisse. Zu diesem Zweck werden die Ausgaben $\vec{c}$ der Funktion *prop* (siehe Algorithmus 5.1) ausgenutzt. Ist genau ein Element von $\vec{c} = \alpha$ (Aktivität $\alpha = 1.0$), dann wird die Klassifikation als sicher gewertet, bei $\vec{c} = \beta$ (Aktivität $\beta = 0.8$) liegt eine Hypothese vor und sind mehrere Elemente von $\vec{c} \neq 0$ wird das Ergebnis als Mehrfachklassifikation bezeichnet. Im Falle von $\vec{c} = \emptyset$ ist eine Nichtklassifikation aufgetreten. Zur Vereinfachung werden die Hypothesen-, Mehrfach- und Nichtklassifikationen als Ablehnungen ($\neg\mathcal{S}$) bezeichnet.

Aus dieser detaillierten Fallunterscheidung lassen sich Schlüsse über die Lernmenge $|\mathcal{B}_l|$ ziehen:

1. Ist die Anzahl der abgelehnten Beispiele ($\neg\mathcal{S}$) zu hoch, dann existieren noch zu viele für den *Dynamic Bounds* unbekannte Beispiele. Dieser Fall ist gut anhand des *flag* Datensatzes (siehe Tabelle 5.4) zu beobachten. Der Datensatz besitzt 28 Merkmale mit 8 Klassen und wird mit 174 Beispielen trainiert, was die verhältnismäßig hohe Ablehnungsanzahl erklärt.

2. Ist die Anzahl der falschklassifizierten Beispiele ($\notin \mathcal{S}$) zu hoch, dann existieren unbekannte Überlappungen zwischen benachbarten Regionen. Dies kann generell zwei Gründe haben: Erstens kann die Anzahl der Beispiele zu gering sein, zweitens kann die Klassentrennung in Cluster nicht möglich sein, weil sich die Wahrscheinlichkeitsverteilungen der Klassen überlappen.

Neben der guten Klassifikationseigenschaft ist hervorzuheben, daß *Dynamic Bounds* keine Lernparameter benötigt. Bei der Anwendung im Rahmen der ILD wird er während der Interaktionsphase im Musterlern-Modus verwendet. Dies erlaubt eine schnelle lernparameterlose Adaption an die Beispiele. Darüber hinaus können die Klassifikationsergebnisse dem Benutzer eine Hilfestellung geben, und zwar dahingehend, ob sich die Beispiele innerhalb einer Region oder in mehreren Hypothesen- bzw. sicheren Regionen befinden, bzw. ob eine Mehrfachklassifikation aufgetreten ist und zwischen welchen Bereichen dies der Fall ist.

Um dem Benutzer eine Hilfestellung bei der Klassifikation der vom *Dynamic Bounds* abgelehnten oder nah der Klassengrenzen liegenden Beispiele zu bieten, werden im nachfolgenden Abschnitt entsprechende Verfahren entwickelt und eingehend untersucht.

5.4 Synergese durch *Hybride Knoten*

Das in dieser Arbeit vorgestellte Konzept für hybride Lernarchitekturen in Diagnosesystemen basiert auf einem hierarchisch gegliederten hybriden Ansatz. Die Anwendung dieser Vorgehensweise wird in [SBG95] beschrieben. Das Training und die Architektur des sogenannten *Hybriden Knotens* ist in zwei Stufen eingeteilt (siehe Abbildung 5.12):

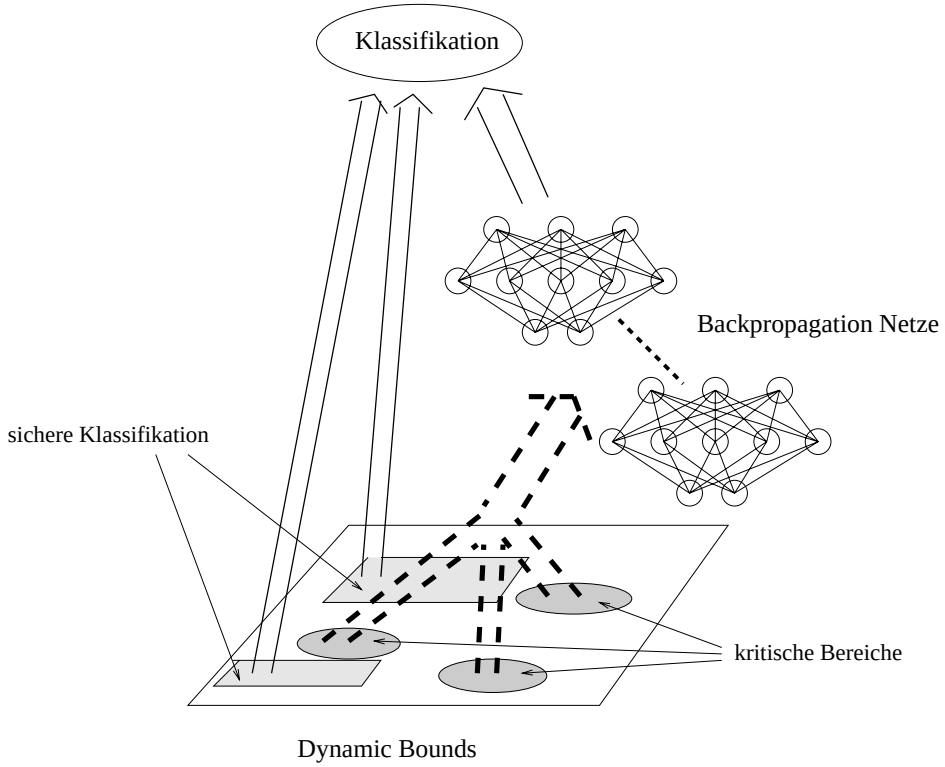


Abbildung 5.12: Der prinzipielle Aufbau des *Hybriden Knotens*. Zur Vorklassifikation wird *Dynamic Bounds* verwendet. Die Klassifikation der kritischen Bereiche (Randbereiche) wird durch eines oder mehrere Backpropagation- Netze durchgeführt.

1. Vorklassifikation des Merkmalsraums durch das lokale Lernverfahren *Dynamic Bounds*.

2. Feinklassifikation der kritischen Bereiche durch ein oder mehrere Backpropagation-Netze.

Das Ziel des *Hybriden Knotens* besteht in der Unterscheidung zwischen *Innenbereichen*, die innerhalb einer homogenen Klassenregion liegen, und den kritischen *Randbereichen*, die die Klassengrenzen bilden. Dies kann am Beispiel der *defects1*-Daten[6] aus Abbildung 5.13, die in ein zweidimensionales Kohonennetz eingelernt wurden, nachvollzogen werden (siehe auch [SBH95, Giz94]).

Die Randbereiche der Klassenregionen sind deshalb kritisch, da sie bei realen Anwendungen oft sehr zerklüftet sind. Das hat zur Folge, daß unabhängig von der Wahl der Aktivierungsfunktion des lokalen Lernverfahrens, der Grenzverlauf nicht genau nachgebildet werden kann. Aus diesem Grund begnügt sich *Dynamic Bounds* mit einer parametrisierbaren asymmetrischen Rechteckaktivierungsfunktion. Da *Dynamic Bounds* alle Lernbeispiele auch 100% korrekt klassifiziert, d.h. diese Beispiele sind alle innerhalb den sicheren Regionen ($\mathcal{S}$) enthalten, müssen die potentiell gefährlichen Randregionen, im Folgenden mit $\mathcal{R}$ bezeichnet, detektiert werden (siehe Abschnitt 5.4.1).

Die Randregionen können ohne Probleme in die Klassifikation von *Dynamic Bounds* integriert werden, indem der Algorithmus **DBprop** (siehe Algorithmus 5.1) zur **DBhybprop** (siehe Algorithmus 5.7) erweitert wird.

[6]Dieser Datensatz wurde zum einlernen der Netze bei der Pipelinediagnose verwendet.

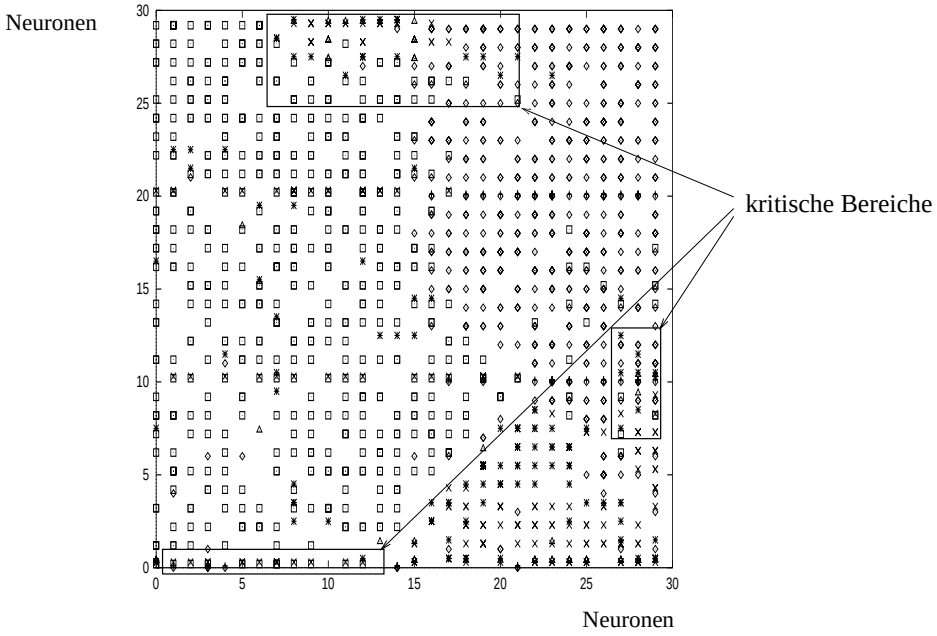


Abbildung 5.13: Der 41-dimensionale Eingaberaum des *defects1*-Datensatzes (siehe auch [SBH95]), der mit Hilfe eines Visualisierungsverfahrens für zweidimensionale Kohonennetze (siehe [Giz94]) in einem Neuronengitter dargestellt wird. An den Neuronen des Kohonennetzes sind die repräsentierten Klassen aufgeführt. Die beispielhaft eingerahmten Bereiche, stellen die kritischen Bereiche dar.

$$\textbf{DBhyb-prop}(\vec{x}) = \begin{cases} \alpha * \{\vec{c}_n \mid \forall n : \vec{x} \in \mathcal{S}_n\} \\ \qquad\qquad \{\vec{c}_n \mid \forall n : \vec{x} \in \mathcal{S}_n\} \neq \emptyset \\ \gamma * \{\vec{c}_n \mid \forall n : \vec{x} \notin \mathcal{R}_n \backslash \mathcal{S}_n\} \\ \qquad\qquad \{\vec{c}_n \mid \forall n : \vec{x} \notin \mathcal{S}_n \wedge \vec{x} \in \mathcal{R}_n\} \neq \emptyset \\ \beta * \{\vec{c}_n \mid \forall n : \vec{x} \in \mathcal{H}_n \backslash (\mathcal{S}_n \cup \mathcal{R}_n)\} \\ \qquad\qquad \{\vec{c}_n \mid \forall n : \vec{x} \notin (\mathcal{S}_n \cup \mathcal{R}_n) \wedge \vec{x} \in \mathcal{H}_n\} \neq \emptyset \\ \vec{0} \qquad\qquad \text{sonst} \end{cases}$$

Algorithmus 5.7: Erweiterung der Propagierung bei *Dynamic Bounds* um die Ausgabe der Randregionen.

Ähnlich wie die Parameter α und β dient γ zur Unterscheidung über die Klassenzugehörigkeit. Ohne Beschränkung der Allgemeinheit wird $\gamma = 0.9$ gesetzt. Dadurch ergibt sich auch die Rangfolge der Klassifikation mit $\alpha > \gamma > \beta$.

Zur Klassifikation der Regionsränder $\mathcal{R}$ werden Backpropagation–Netze verwendet, da mit diesen Netzen hinsichtlich der Generalisierungsfähigkeit bei nichtlinearen Eingaberäumen sehr gute Ergebnisse erzielt werden.

Die Arbeitsweise des *Hybriden Knotens* stellt sich also wie in Algorithmus 5.8 dar.

Im Abschnitt 5.4.2 wird die Problematik des Aufbaus des **BPhyb-prop** Algorithmus besprochen, und es wird eine neue Strategie für die Wahl der Konfiguration der Backpropagation–Netze entwickelt. Im Folgenden wird auf die Problematik bei der

```
1 function HKprop(x⃗)
2   e⃗ := DBhyb-prop(x⃗)
3   case es ex. genau ein e_i = α
4     return e⃗
5   case ∃ e_i = λ
6     return BPhyb-prop(x⃗)
7   else
8     return e⃗
```

Algorithmus 5.8: Propagierungsalgorithmus für *Hybriden Knoten*.

Suche nach den *Randregionen* $\mathcal{R}$ im *Dynamic Bounds*-Netz eingegangen.

5.4.1 Heuristik für Randregionen

Der *Dynamic Bounds*-Algorithmus integriert alle ihm präsentierten Beispiele in die sicheren Regionen $\mathcal{S}$. Wie bereits oben angesprochen, besteht das Ziel in der Suche nach den Klassengrenzen im Merkmalsraum. Diese werden durch spezielle Randneuronen n repräsentiert, die diese Randregionen $\mathcal{R}_n = \mathcal{S}_n$ aufspannen. Dazu ist es nötig, einzelne Neuronen zu vereinigen und zuzulassen, daß ein einzelnes Neuron mehr als nur eine Ausgabeklasse vertritt.

Grundlage der Überlegung ist die Tatsache, daß in den homogenen Bereichen wenig Neuronen, die weiter auseinander liegen, viele Lernbeispiele klassifizieren, während in den Randbereichen viele Neuronen nah beieinander liegen, die wenige Lernbeispiele klassifizieren. Die angestrebte Heuristik soll die Einflußbereiche der Neuronen als Wahrscheinlichkeitsverteilungen der Klassen im Merkmalsraum anhand der vorhanden Segmentierung durch die Einflußbereiche der Neuronen approximieren und daraus die Randregionen (Überlappungsbereiche der Wahrscheinlichkeitsverteilungen) identifizieren. Demzufolge existieren zwei Parameter, die diesen Zusammenhang beschreiben:

- ein Parameter für den Abstand der Neuronen,
- ein Parameter für die Aussagekraft des einzelnen Neurons.

Aus diesem Grund werden für die Heuristik zwei Lernparameter benötigt. Zum Einen muß eine obere Grenze für den Abstand der Neuronen angegeben werden. Dies geschieht mit dem Lernparameter *MinDist*. Neuronen, die weiter auseinander liegen als dieser Schwellwert, kommen für eine Vereinigung erst gar nicht in Frage, da davon ausgegangen werden muß, daß diese nicht in einer Randregion liegen.

Ein zweiter Lernparameter, im Folgenden als *MinNumber* bezeichnet, gibt an, wieviele Lernbeispiele ein Neuron maximal klassifizieren darf, damit eine Vereinigung in Frage kommt. Neuronen, die mehr Lernbeispiele klassifizieren, liegen gemäß der Annahme der Heuristik in einer homogenen Region und kommen deshalb für eine Vereinigung ebenfalls nicht in Frage.

Die Vorgehensweise des Algorithmus **DBprune**$(MinNumber, MinDist)$ besteht darin, anhand der eingeführten Heuristik, Mischregionen Schritt für Schritt zu erschlie-

ßen. Ausgehend von einem Startneuron wird eine Überlappungsregion durch kontinuierliches Vereinigen von Neuronen erschlossen. Hierbei werden immer zwei Neuronen, ein Randneuron und ein Nicht-Randneuron vereinigt. So werden in einer Art *Greedy*-Verfahren die Randregionen erschlossen.

Wird kein weiteres Neuron mehr gefunden, das gemäß dem Lernparameter *MinDist* nah genug bei dem Randneuron liegt, aber noch Kandidaten übrig sind, wird ein neues Startneuron aus diesen gewählt. Auf diese Weise können bei geeigneter Wahl der Schwellwerte alle Randregionen detektiert werden.

Auswahlkriterien für die Lernparameter der Heuristik

Um ein möglichst optimales Ergebnis bei der Randregionssuche zu erzielen, müssen die Lernparameter *MinDist* für den Abstandsschwellwert, sowie der Schwellwert *MinNumber* für die maximal erlaubte Anzahl von klassifizierten Lernbeispielen sehr sorgfältig gewählt werden. Da es sich bei dem Verfahren nur um eine Heuristik handelt, können diese Werte nicht für den allgemeinen Fall bestimmt werden. Es gibt aber durchaus Kriterien für eine passende Auswahl. Dieser Abschnitt soll dazu einige Anregungen geben.

Eine vielversprechende Größe für den Abstandsschwellwert *MinDist* ist die mittlere Distanz der Lernbeispiele zueinander. Der Nachteil dabei besteht darin, daß dieser mittlere Abstand vorher bekannt sein muß. Zur Berechnung ist ein Algorithmus mit einem Aufwand von $O(n^2)$ nötig, wobei n die Anzahl der Lernbeispiele angibt.

Eine weitere Möglichkeit zur Ermittlung vom *MinDist* wäre eine Art Iterationsverfahren. Zuerst wird mit einem relativ großen Abstand begonnen. Nach der Randregionssuche mit diesem Abstandsmaß werden die Neuronen überprüft. Gibt es fast nur noch Randneuronen, wurde der Abstandsschwellwert zu groß gewählt. Liefert dieses Vorgehen dagegen einen zu geringe Anzahl, so ist der Wert des Abstandsschwellwert zu klein angesetzt. Auf diese Weise kann sukzessive auf einen guten Wert hingearbeitet werden. Diese Wert ist allerdings stark applikationsabhängig.

Der Schwellwert für *MinNumber* kann nur ungefähr ermittelt werden. Eine mögliche Methode stellt hierbei die Analyse eines eingelernten Netzes dar. Hierfür wird die Anzahl der Neuronen mit der Anzahl der Lernbeispiele in Beziehung gesetzt. Durch Gleichung 5.16 wird die durchschnittliche Anzahl der Beispiele (num), die durch ein Neuron repräsentiert wird, ausgedrückt.

$$num = \frac{|\mathcal{B}|}{\text{Neuronenanzahl}} \tag{5.16}$$

Als Abwandlung der hier vorgestellten Heuristik kann anstatt eines Schwellwertes für den Abstand bzw. für die Anzahl klassifizierter Lernbeispiele, auch die sogenannte Dichte ρ_n eines Neurons n genommen werden. Die Dichte berechnet sich gemäß Gleichung 5.17 aus dem Volumen, daß das Neuron abdeckt, und der Anzahl klassifizierter Lernbeispiele die sich im Bereich des Neurons befinden.

$$\rho_n = \frac{|\vec{x} \in \mathcal{S}_n|}{\text{Volumen von } \mathcal{S}_n} \tag{5.17}$$

Grundlage dieser Heuristik ist wieder die Annahme, daß ein Neuron in einer Randregion eine kleinere Dichte besitzt, also wenige Lernbeispiele klassifiziert, während ein Neuron in einer homogenen Region viele Lernbeispiele repräsentiert.

Allen Verfahren ist gemeinsam, daß sie auf einen oder zwei Lernparameter hin optimieren, wobei die Auswahl der Parameter vom Problem selbst abhängt. Hier können nur Testreihen und Experimente Abhilfe schaffen. Es müssen mehrere verschiedene Lernparametereinstellungen ausprobiert werden, um eine Vorstellung über passende Werte zu der jeweiligen Problemstellung zu erhalten.

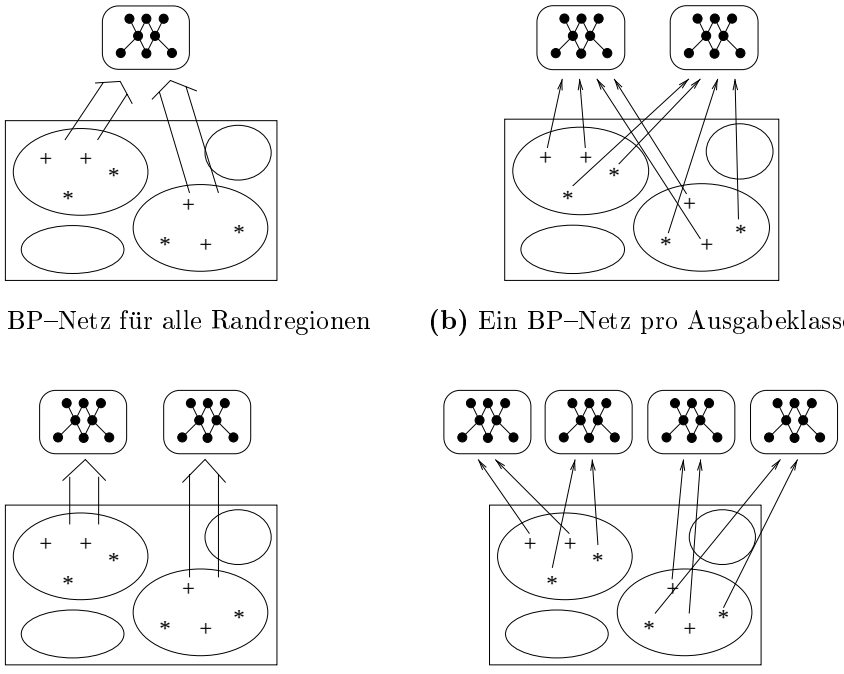
5.4.2 Hybride Lernarchitekturen

Die Randregionen $\mathcal{R}$ werden wegen der guten Generalisierungseigenschaften durch Backpropagation–Netze klassifiziert. Dabei stellt sich die Frage, wieviele Netze und mit welchen Zuständigkeiten die Backpropagation–Netze konfiguriert werden sollen. Die möglichen Konfigurationen des *Hybriden Knotens* sind in Abbildung 5.14 aufgezeigt. Die Konfigurationen sind nach der Fähigkeit, komplexe Eingaberäume zu klassifizieren, geordnet.

Die Auswahl der Lernbeispiele für die Backpropagation–Netze läßt sich unabhängig von der Konfiguration wie folgt ausdrücken:

$$\mathcal{B}sub_{S,\delta} := \{(\vec{x}, \vec{c}) \in \mathcal{B} | \exists n \in S : (\vec{x} - \mathcal{R}_n) < \delta\} \tag{5.18}$$

Es werden all diejenigen Beispiele in die Beispielsmenge für das Einlernen der Backpropagation–Netze aufgenommen, die sich nicht weiter als δ außerhalb von $\mathcal{R}$ liegen. Die Wahl von δ hängt im Wesentlichen von der Anzahl der Beispiele in $\mathcal{B}sub_{S,\delta}$ und der Komplexität der Randregion ab:



(a) Ein BP–Netz für alle Randregionen

(b) Ein BP–Netz pro Ausgabeklasse

(c) Ein BP–Netz pro Randregion

(d) Ein BP–Netz pro Ausgabeklasse und Randregion

Abbildung 5.14: Mögliche Konfigurationen der Backpropagation–Netze (BP–Netze) des *Hybriden Knotens* geordnet nach aufsteigender Komplexität.

(a) *Ein BP–Netz für alle Randregionen:* In diesem Fall wird genau ein Backpropagation–Netz zum Nachlernen sämtlicher Randregionen verwendet (siehe Abbildung 5.14(a)). Die Lernbeispiele für diese Konfiguration sind durch $\mathcal{B}sub_{\forall\mathcal{R},\delta}$ gegeben.

Ein Nachteil dieser Konfiguration kommt dann zum Tragen, wenn die einzelnen Randregionen weit auseinander liegen bzw. zu komplizierte Entscheidungsgrenzen zwischen den Klassen bestehen. Es werden dem BP–Netz nur Lerndaten aus weit entfernt liegenden Unterregionen (je nach Wahl von δ) des Merkmalsraums präsentiert. Aufgrund dieser nur lokal vorhandenen Information ohne globalen Zusammenhang ist es dem Netz unter Umständen nicht möglich, alle Mischregionen zufriedenstellend zu erlernen.

(b) *Ein BP–Netz pro Ausgabeklasse:* In diesem Fall werden genau so viele Backpropagation–Netze zum Nachlernen sämtlicher erschlossener Randregionen verwendet wie die Anzahl der Klassen (siehe Abbildung 5.14(b)). Die Lernbeispiele ergeben sich durch $\mathcal{B}sub_{\forall\mathcal{R},\delta}$.

Mit dieser Konfiguration wird eine höhere Komplexität der Klassifikation möglich als mit Methode (a). Es ist allgemein bekannt, daß durch die Aufteilung der Klassifikationsaufgabe nach Klassen auf mehrere Netze eine bessere Klassifikation erzielt werden kann.

(c) *Ein BP–Netz pro Mischregion:* Hier wird für jede Randregion $\mathcal{R}_r$ ein separates Netz BP_r eingelernt (siehe Abbildung 5.14(c)). Die Lernbeispiele für jedes Backpropagation–Netz sind $\mathcal{B}sub_{\mathcal{R}_r,\delta}$.

Diese lokale Betrachtung des Merkmalsraums vereinfacht die Komplexität der Entscheidungsfunktion. Ein Nachteil dieser Architektur ist die unter Umständen zu kleine Anzahl an Lernbeispielen in den einzelnen Randregionen. Um nämlich eine zufriedenstellende Generalisierungsfähigkeit bei den Backpropagationnetzen zu erhalten, ist je nach Komplexität eine bestimmte Mindestanzahl von Lernbeispielen nötig. Deshalb ist es sinnvoll, benachbarte Beispiele für den Lernprozeß mit zu nutzen ($\delta > 0$).

(d) *Ein BP–Netz pro Mischregion und Ausgabenklasse:* Analog zum Vorgehen von Konfiguration (a) zur Konfiguration (b) werden für alle Randregionen $\mathcal{R}_r$ genau so viele Backpropagation–Netze eingesetzt wie Ausgabeklassen existieren (siehe Abbildung 5.14(d)). Für jedes Backpropagation–Netz $BP_{r,c}$ zur Region r und Klasse c werden Lernbeispiele durch $\mathcal{B}sub_{\mathcal{R}_r,\delta}$ bestimmt.

Die obigen Konfigurationen des *Hybriden Knotens* sind in der Lage, immer komplexere Eingaberäume zu bearbeiten. Gleichzeitig steigt aber auch die Anzahl der notwendigen Lernbeispiele. Die aufgeführten Architekturen lassen sich natürlich auch kombinieren, so daß ein kontinuierlicher Übergang zwischen den Grundkonfigurationen entsteht.

Der Propagierungsalgorithmus **BPhyb-prop** besteht darin, daß für jede Klasse c einer jeden Randregion $\mathcal{R}_r$ ein Backpropagation–Netz BP_k zugeordnet wird, wobei ein Netz k durch mehrere Randregionen r und Klassen c referenziert werden kann.

Durch die steigende Komplexität und Anforderung an die Beispielsanzahl, können die vorgestellten Architekturen nacheinander überprüft werden. Sinkt die Klassifikationsgüte, brauchen die restlichen Konfigurationen nicht weiter überprüft werden. Auf

diese Weise erhält man den Algorithmus **BPhyb-learn** bei der Suche nach einem guten Klassifikator.

Bei der Wahl des Parameters δ, für die Auswahl der Beispiele, sollte berücksichtigt werden, daß sich die Einflußbereiche im *Dynamic Bounds* während der Interaktionsphase verschieben können, insbesonders können sie kleiner werden. Damit der Benutzer trotzdem einen Hinweis auf die korrekte Klassifikation von Seiten des Diagnosesystem erhalten kann, sollte $\delta > 0$ gewählt werden,so daß eine Überlappung der Klassifikationsbereiche zwischen den BP–Netzen und dem *Dynamic Bounds* entsteht.

Durch die Verwendung des RPROP-Lernalgorithmus zum Einlernen der Backpropagation–Netze besteht die Parametrisierung der BP–Netze im wesentlichen aus der Wahl der richtigen Topologie, des Abbruchs des Lernvorgangs, um ein Übertrainieren zu verhindern und der Initialisierung der Gewichte. Diese Probleme lassen sich nur durch mehrfaches Ausprobieren und unter Zuhilfenahme von Heuristiken bewältigen (siehe auch [Kop99]).

Des Weiteren können statt eines BP-Netzes mehrere eingesetzt werden, um so ein Ensemble zu kreieren, bei dem die am stärksten aktivierte Klasse gewinnt (majority vote) oder der Durchschnitt gebildet wird (siehe [AK98]). Eine weitere Möglichkeit wären die Ausgaben der Netze zu gewichten (Multiple-experts). Auch die Auswahl der Lernbeispiele könnte weiter verfeinert werden, in dem der Lernparameter δ automatisch und für jedes Netz individuell ermittelt wird. Diese Ansätze werden in dieser Arbeit aus Zeitgründen nicht weiter verfolgt.

5.4.3 Phasenlernen mit *Hybriden Knoten*

Wie aus den Anforderungen an eine ILD hervorgeht (vgl. Abschnitt 3.3), muß ein permanenter Wechsel zwischen der Interaktionsphase und der Optimierungsphase durch die Lernkomponente realisiert werden (siehe Algorithmus 5.9), um so die Diagnose effektiv zu gestalten.

Das schnelle online Lernen des *Dynamic Bounds*, im Speziellen mittels des **DB-patternlearn**-Algorithmus (siehe auch Algorithmus 5.4), wird dabei während der Interaktionsphase verwendet, um so dem Benutzer eine schnelle Wissensakquisition zu ermöglichen. Die in diesem Zusammenhang vom Benutzer klassifizierten Daten werden gesammelt $\mathcal{B}_{new}$ und für das spätere offline Lernen in der Wissensverwaltung (siehe Kapitel 6.5) aufbewahrt.

In der offline Phase wird dann die Reorganisation der durch die neuronalen Netze aufgestellten Modelle vorgenommen. Die offline Phase wird in dem Algorithmus 5.9 definiert und durchläuft folgende Schritte:

1. Einlernen des *Dynamic Bounds* Netzes bis zu seiner Terminierung über mehrere Epochen hinweg.

2. Suche der Randregionen mit Hilfe der Heuristik.

3. Suche der Konfiguration der hybriden Lernarchitektur und deren Einlernen.

4. Bestimmung der repräsentativen Beispiele.

Schritt 2 und 3 dieser Vorgehensweise lassen unterschiedliche Parametrisierungen zu. Zu diesem Zweck werden unterschiedliche Parametrisierungen, ausgehend von den

```
1  function Phase-learn(X, B, MinNumber, MinDist)
2    X_sel = ∅
3    do
4      do          // Interaktionsphase
5        x⃗ = X_sel∪ Auswahl durch den Benutzer
6        e⃗ = HKprop(x⃗)     // (s. Alg. 5.8)
7        if Falsifizieren von e⃗ durch den Benutzer
8          t⃗ = Eingabe vom Benutzer
9          B_new = B_new ∪ (x⃗, t⃗)
10         DBpatternlearn(x⃗, t⃗, e⃗)
11     until Abbruch durch Benutzer

12     if B_new ≠ ∅          // offline Phase
13       B = B ∧ B_new
14       DBlearn(B)
15       DBprune(MinNumber, MinDist)
16       BPhyb-learn(B)
17       X_sel = get-examples(X)
18   until Klassifikatorperformance ausreichend
```

Algorithmus 5.9: Phasenlernen mit *Hybriden Knoten*.

in der letzten Iteration ermittelten Parametern, ausgetestet. Dabei ist zu erwähnen, daß nicht unbedingt die beste Konfiguration bzw. Parametrisierung ermittelt werden muß, sondern auch eine suboptimale Lösung während der Wissensakquisitionsphase ausreicht, da letztendlich der Benutzer in Zweifelsfällen kontaktiert werden kann.

Schritt 4 ist optional und kann nur durchgeführt werden, falls die Sensordaten offline vorhanden sind. Besteht die Möglichkeit, diese Daten offline zu diagnostizieren, dann kann der eingelernte *Dynamic Bounds*-Algorithmus diese Daten klassifizieren und all diejenigen Beispiele protokollieren, die in die Hypothesen oder Randbereiche fallen. Dieses Vorgehen wird durch den **get-examples**-Algorithmus realisiert. Dem Benutzer kann dann eine zufällige Auswahl präsentiert werden.

Mit Hilfe der Risikomatrix $\mathcal{R}$ (siehe Abschnitt D.2) und der Schätzung des wahren Fehlers (siehe Abschnitt D.4) läßt sich die Klassifikatorperformanz ermitteln. Ist diese ausreichend, kann das Diagnosesystem in seinem Lebenszyklus in die Anwendungsphase übergehen.

5.4.4 Evaluierung

In diesem Versuch wurden zwei Konfigurationen des *Hybriden Knotens* mit einem Backpropagation–Netz verglichen. Für das Einlernen der BP–Netze in der hybriden Architektur wurden nur Daten aus den Hypothesen- und sicheren Bereichen der Randneuronen verwendet. Bei näherer Betrachtung der Tabellen 5.5 und 5.6 kann festgestellt werden, daß die Klassifikationsgüte geringfügig gegenüber dem einfachen BP–Netz zugenommen hat.

Datensatz	BP-Netz (%)		Hybride Architektur (%)	
	gut	schlecht	gut	schlecht
thyroid	1764 (98.00)	36 (2.00)	1772 (98.44)	28 (1.55)
pipe	1477 (95.47)	70 (4.53)	1499 (96.9)	48 (3.1)
diabetes	132 (68.75)	60 (31.25)	133 (69.27)	59 (30.73)

Tabelle 5.5: *Hybrider Knoten* mit einem Backpropagation–Netz für alle Mischregionen.

Datensatz	BP-Netz (%)		Hybride Architektur (%)	
	gut	schlecht	gut	schlecht
thyroid	1764 (98.00)	36 (2.00)	1770 (98.33)	30 (1.67)
pipe	1477 (95.47)	70 (4.53)	1480 (95.67)	67 (4.33)
diabetes	132 (68.75)	60 (31.25)	133 (69.27)	59 (30.73)

Tabelle 5.6: *Hybrider Knoten* mit einem Backpropagation–Netz pro Ausgabeklasse und für alle Mischregionen.

Die hier durchgeführten Experimente umfassen nur die (a) und (b) Konfigurationen aus Abschnitt 5.4.2, da die Klassifikationsgüte offensichtlich abnimmt. Es ist festzuhalten, daß sogar die einfachste Form des *Hybriden Knotens* eine bessere Klassifikation produziert als herkömmliche Backpropagation–Netze.

Um die übrigen Konfigurationen des *Hybriden Knotens* anzuwenden, bedarf es einer höheren Anzahl von Trainingsbeispielen bzw. komplexeren Klassengrenzen.

5.5 Resümee

Es wird ein neues Lernverfahren, das sog. *Dynamic Bounds*-Verfahren, vorgestellt, dessen wesentliche Eigenschaften in den folgenden bestehen:

- Keine vom Benutzer extern festzulegende Lernparameter.
- Sehr gute online Lernfähigkeit. Der Algorithmus kann im Musterlern-Modus zum inkrementellen Lernen genutzt werden.
- Dynamische, selbstorganisierende Anpassung der Einflußbereiche der Neuronen und auch das Einfügen und Löschen von Neuronen.
- Es werden wenige Neuronen für eine gute Repräsentation des Merkmalsraums benötigt.

Experimente mit den *Dynamic Bounds*, C4.5 und k–nächster Nachbar Algorithmen anhand der Standardklassifikationsdatensätze aus [Pre94, MM96] und der im Anhang E beschriebenen, zeigen deutlich die Vorteile des neuen Verfahrens im Bereich der Identifikation von sicheren Klassifikationen (siehe Abschnitt 5.3.6).

Für die Feinklassifikation wurde der *Hybride Knoten* vorgestellt, ein hierarchisch arbeitender Ansatz auf der Basis von *Dynamic Bounds*, der den Merkmalsraum in homogene und schwierige Klassifikationsbereiche (Randregionen) unterteilt. Es wurde

eine Heuristik für das *Dynamic Bounds*-Verfahren vorgestellt, die die Lokalisation der Randregionen ermöglicht. Durch die Aufspaltung in zwei Mengen, wird das Klassifikationsproblem wesentlich vereinfacht, wodurch die Anzahl der benötigten Epochen für das Einlernen aller Netze bis auf die Hälfte reduziert werden kann, und die Klassifikationsleistung sogar etwas erhöht wird (siehe Abschnitt 5.3.6). Schwierigkeiten bereitet es weiterhin, für die Backpropagation–Netze geeignete Topologien zu finden, sowie ein Übertrainieren der Netze zu verhindern.

Eine wesentliche Verbesserung des Lernens besteht in der eindeutigen Strategie zur Bestimmung der Konfiguration des *Hybriden Knotens*. Mit Hilfe dieser Strategie ist es möglich festzustellen, ob eine Konfiguration ausreicht, oder ob eine mächtigere Architektur benötigt wird.

Der *Hybride Knoten* bietet dabei auch die Möglichkeit, Klassifikationen interaktiv in einem online Prozeß einzulernen, wobei die Eingabe sofort erlernt wird. Im Verlauf der offline Optimierung durch die Backpropagation–Netze, wird eine optimale Generalisierung erreicht. Dieser Prozeß kann iterativ fortgesetzt werden, ohne eine der beiden Fähigkeiten einzubüßen. Der iterative Prozeß kann, in Abhängigkeit von der Beschaffenheit der Applikation, durch die automatische Auswahl von repräsentativen Beispielen effizienter gestaltet werden.

Die in diesem Kapitel vorgestellten Verfahren wurden durch das ***CMS***-System realisiert (siehe Anhang A).

Kapitel 6

Interaktion zum Überwachen und Einlernen der Lernkomponente

Dieses Kapitel behandelt die Einbeziehung des Benutzers in den Lernprozeß der Lernkomponente (vgl. Abbildung 6.1) mit Hilfe der Visualisierung und Interaktion.

In diesem Zusammenhang wird eine flexible interaktive graphische Visualisierungsmöglichkeit für die unterschiedlichen Phasen der vorverarbeiteten und klassifizierten Meßdaten vorgestellt. Darüber hinaus widmet sich das Kapitel der Problematik der strukturierten Wissensverwaltung, die speziell auf die Bedürfnisse der inkrementellen Wissensakquisition und der subsymbolischen Lernverfahren zugeschnitten ist, als auch einer Monitorkomponente, mit deren Hilfe es möglich ist, den Zustand der Lernkomponente zu überwachen.

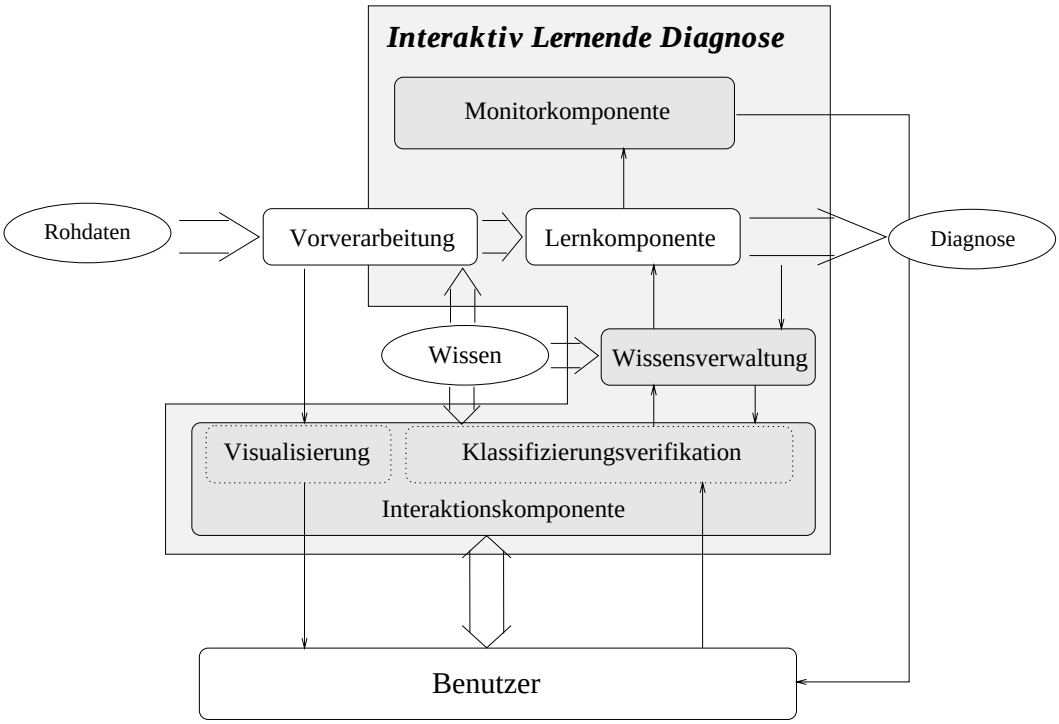


Abbildung 6.1: Die Monitor-, Wissenserwerbs- und Wissensverwaltungskomponente des ILD-Systems bilden die eigentliche Interaktion.

6.1 Anforderung an die Interaktionskomponenten

Der vorgestellte Ansatz der ILD geht davon aus, daß keine oder nur sehr geringe Kenntnisse existieren, die sich in Regeln formalisieren lassen. Die Idee besteht darin, den Experten und das einzulernende Diagnosesystem die gleichen Daten betrachten zu lassen (siehe Abbildung 6.2). Da die zu treffenden Entscheidungen auf subsymbolischen Daten beruhen, ist es notwendig, die Betrachtung sowohl der Rohdaten $\mathcal{D}$, der unterschiedlichen Signalverarbeitungsschritte V, der diversen Merkmale $\mathcal{M}$ als auch der Diagnoseergebnisse $\mathcal{C}$, die dazugehörigen Erklärungen der Entscheidungen und die Qualitätsabschätzungen der Lernkomponente dem Benutzer zur Verfügung zu stellen (vgl. Abschnitt 3.4).

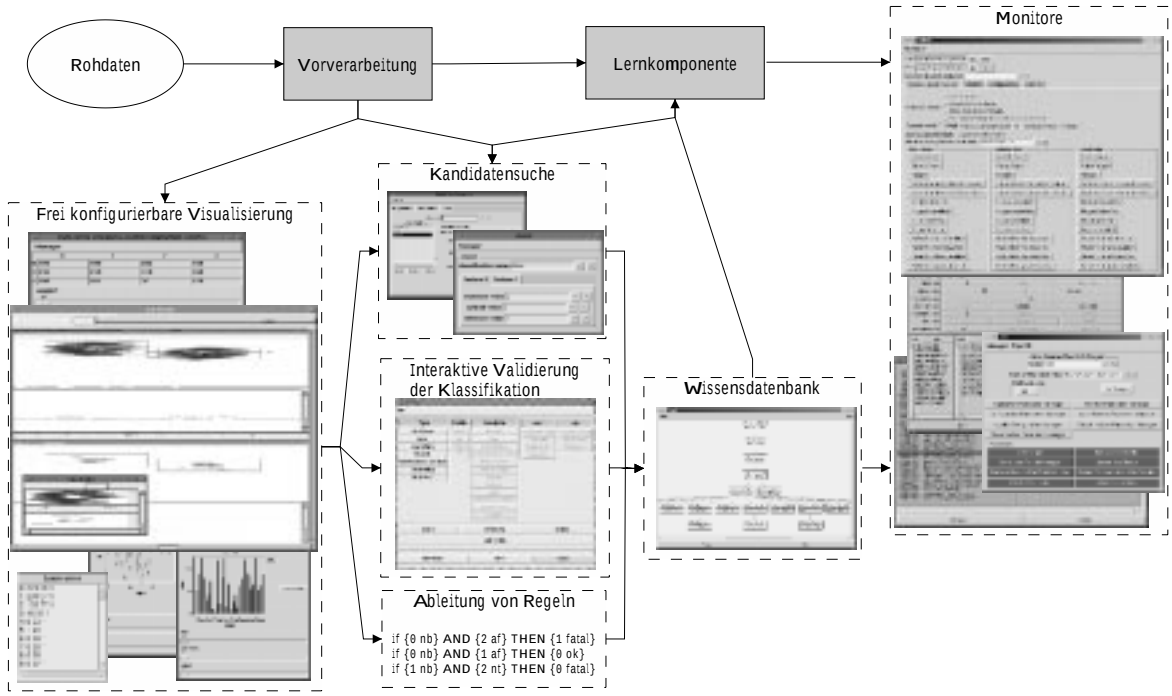


Abbildung 6.2: Ablauf der Interaktion in der ILD anhand der einzelnen Komponenten.

Die ILD kann aber auch ohne Verwendung von Vorwissen gestartet werden. Zu jedem Datum meldet sie dann ihre Diagnose, oder daß aufgrund von fehlendem Wissen keine Diagnose möglich ist. Bei keiner Klassifikation bzw. einer Fehlklassifikation kann das System sofort durch die Interaktion mit dem Experten korrigiert werden.

Die Fehlerkorrektur des Systems geschieht auf Basis derselben Daten, wie sie auch dem Benutzer zur Verfügung stehen. Wenn möglich, werden die Diagnosen für komplette Rohdatenbereiche auf einmal angezeigt, damit der Benutzer auf einen Blick möglichst viele Daten überprüfen kann. Hierzu ist es notwendig, die Daten für den Menschen derart aufzubereiten, daß sie die Applikation widerspiegeln. Zu diesem Zweck eignet sich die graphische Interaktion im besonderen. Der Benutzer kann anhand von Markierungen bei Kurven und anderen Darstellungen, verschiedene Bereiche selektieren, um so das Diagnoseergebnis zu modifizieren. Durch Definition neuer Ausgabeklassen besteht sogar die Möglichkeit, dem Diagnosesystem neue Zusammenhänge zu vermitteln.

Der Grundgedanke der ILD besteht darin, daß neu erworbenes Wissen unmittelbar adaptiert wird. Dies geschieht in den sog. *Hybriden Knoten* (siehe auch Abschnitt

5.4) auf der Basis des *Dynamic Bounds*-Algorithmus. Durch die sofortige Adaption befindet sich das Diagnosemodell immer auf dem aktuellsten Stand. Der Experte wird dadurch entlastet, da er bereits aufgetretene Problemstellungen nicht wiederholt bearbeiten muß. Er konzentriert sich also nur auf die noch nicht erschlossenen Teile des Bereichswissens bzw. der Verifikation des Diagnosesystems.

Neben der direkten Beeinflussung auf der Beispielsebene hat der Experte die Möglichkeit, a priori Wissen in Form von Regeln direkt in die Wissensverwaltung einzugeben, die wiederum durch die Lernkomponente umgehend umgesetzt werden.

Der im Rahmen dieser Arbeit entwickelte Simulator für Neuronale Netze *Connectionist Model Simulator (**CMS**)* ist ein allgemeines Rahmenwerk, um sowohl beliebige Neuronale Netze als auch andere Algorithmen (Vorverarbeitung) in Form eines beliebigen Datenflußgraphen miteinander zu verbinden. Es können diverse Datenquellen und Visualisierungsverfahren kombiniert werden. Die Knoten des Graphen realisieren dabei die Ein- und Ausgabe sowie generell die Verarbeitung der Daten. Die Kanten sind für den Transport, die Aufsplittung oder Zusammenfügung der Daten verantwortlich (siehe auch Anhang A). Für dieses Rahmenwerk ist eine graphische Interaktionsmöglichkeit zu schaffen, die Zugriff auf die internen Parameter des ILD-Systems als auch zu seiner Überwachung dienen kann.

Für die Verwirklichung des Interaktionskonzepts, im Rahmen des *Interaktiven Lernens*, und die einfache Anbindung des ILD-Systems an die Applikation wurde im Rahmen dieser Arbeit das generische Benutzerinterface System ***Iface*** entwickelt (siehe auch Anhang C).

6.2 Konzept der Interaktion

Um eine Kommunikation zwischen dem Benutzer und dem Lernsystem aufzubauen, müssen sinnvolle Visualisierungsmöglichkeiten der Daten vorliegen, denn das System und der Benutzer müssen die gleiche Sprache benutzen, um sich gegenseitig zu verstehen.

Die bei der Pipelinediagnose auftretenden Daten sind meist höherdimensional. Ohne eine graphische Visualisierung solcher Daten ist es in der Regel sehr schwierig, bestehende Zusammenhänge in diesen Daten zu erkennen. Da solche Daten nicht direkt dargestellt werden können, wurden zu diesem Zweck viele Verfahren entwickelt. Ein Überblick unterschiedlicher Darstellungsarten findet sich in [Alb96].

Wie bereits erwähnt, stehen im wesentlichen höherdimensionale Daten zur Darstellung bereit. Die bei der ILD auftretenden Daten, im folgenden auch als *multivariate Daten* bezeichnet, lassen sich als eine Menge von Tupeln darstellen:

Definition 6.1 (Multivariate Daten)

$$\{(\vec{x}, w) | \vec{x} \in \mathbb{N}_0^n \text{ und } w \in \mathbb{R}\}$$

In dieser Menge nimmt $\vec{x}$ alle Werte innerhalb eines bei $\vec{0}$ beginnenden Hyperquaders an. Dabei gilt im allgemeinen $n > 1$. w sind die darzustellenden Daten.

Die multivariaten Daten werden im wesentlichen durch ihre Dimension n bestimmt (siehe Definition 6.1). Für den Benutzer ist es aber wichtig, daß er eine ihm intuitiv verständliche Sicht (Projektion) der Daten präsentiert bekommt.

Darüber hinaus kann generell festgehalten werden, daß Methoden zur Darstellung von k-Dimensionen existieren. Diese lassen sich zur Visualisierung aller Daten höherer Dimensionen verwenden, indem eine Selektion der darzustellenden Dimensionen stattfindet, oder bei Dimensionen geringeren Umfangs die Darstellung vervielfältigt wird.

Dimension	Beispiel der Visualisierung
0-D	Ein Wert, eine Tortengrafik
1-D	Histogramm, Kurve
2-D	Mehrere Kurven verschoben, Balkenfeld, Falschfarben 2-D-Bild
3-D	Falschfarben 3-D-Darstellung projiziert auf ein 2-D-Bild
n-D	Tabellen, Projektion in 2-D-Bild

Tabelle 6.1: Darstellungsmöglichkeiten von Daten in Abhängigkeit von ihrer Dimension.

In Abschnitt 4.2 wird die Charakteristik der Rohdaten der Applikation erfaßt. Diese wird hier fortgeführt und zur Bestimmung der Visualisierungsmethode verwendet (siehe Tabelle 6.1). Nicht alle Darstellungen sind jedoch informationserhaltend. Projektionen verbergen immer Teile der in den Daten enthaltenen Informationen.

Die zufriedenstellende Visualisierung multivariater Daten stellt im allgemeinen ein nicht lösbares Problem dar. Diese Unlösbarkeit basiert unter anderem auf folgenden Tatsachen:

- Das menschliche Vorstellungsvermögen ist in der Regel nicht in der Lage, Daten zu visualisieren, die von mehr als drei Eingabevariablen (Dimensionen) abhängen.
- Da die allgemein verwendeten Ausgabemedien (z.B. Monitor) in der Regel höchstens dreiwertige Ausgaben zulassen (X- und Y-Achsen sowie Farbkodierungen), ist bei der Ausgabe von drei- oder höherdimensionalen Daten immer eine Projektion oder Auswahl[1] notwendig. Dies führt zwangsläufig zu einem Verlust an Übersichtlichkeit oder Daten.

Ist schon die Visualisierung mit Problemen behaftet, so ist die Interaktion mit solchen Daten noch weitaus schwieriger. In der Regel wird dabei die Selektion auf Bereiche bereits existierender Daten beschränkt. Außerdem wird die Auswahl der darzustellenden Daten eingeschränkt.

Da bei der Diagnose keine Rohdaten manipuliert werden, beschränken sich die nachfolgenden Betrachtungen auf die Visualisierung und Selektion der Daten. Die Interpretation multidimensionaler Daten erfordert die Möglichkeit, die Daten in jedweder Weise betrachten zu können. Das impliziert, daß ein mehrdimensionaler Datensatz aus der Perspektive unterschiedlicher Schnitte durch n-dimensionale multivariate Daten dargestellt und selektiert werden kann. Aus diesem Grund werden aus den multivariaten Daten "Scheiben" herausgeschnitten, die ihrerseits visualisiert werden können. Auf diese Art ist es dem Benutzer möglich, durch die Daten nach Belieben zu *browsen*.

Durch die konsequente Anwendung der gleichen Art der Datenhaltung ist es möglich, unterschiedliche Visualisierungsmethoden auch gleichzeitig anzuwenden. Die Bedienung der interaktiven Komponenten ist derart gestaltet, daß sie flexibel, intuitiv und für verschiedene Applikationen leicht angewendet werden kann.

[1] In der englischen Literatur meistens als *Slicing* bezeichnet.

Um der Adaptivität an unterschiedliche Applikationen Rechnung zu tragen und eine möglichst hohe Flexibilität zu gewährleisten, wird die Oberfläche von der Applikation getrennt.

Im weiteren wird nun die eigentliche Visualisierung und Selektion näher erläutert.

6.3 Interaktive Datenvisualisierung

In Tabelle 6.1 wurden bereits einige Darstellungsmöglichkeiten in Abhängigkeit von der Dimension der Sicht auf die multivariaten Daten vorgeschlagen. Es gibt eine Vielzahl von Darstellungsmöglichkeiten für eine feste Dimension. Die in dieser Arbeit verwendeten Darstellungen sind möglichst universell. Durch das vorgestellte Konzept ist es jederzeit möglich, durch Hinzufügen neuer Darstellungsarten, in Form von sogenannten *Displaymodi*, neue Visualisierungsvarianten zu integrieren. Dieses Konzept wurde durch das speziell entwickelte Oberflächensystem ***Iface*** umgesetzt (siehe Anhang C).

Wesentlich für diese Displaymodi ist ihre Fähigkeit, Datenbereiche zu selektieren, um auf diese Art mit der Applikation bzw. dem Diagnosesystem zu kommunizieren. Die Visualisierung kann interaktiv gewählt werden (siehe Abbildung 6.3).

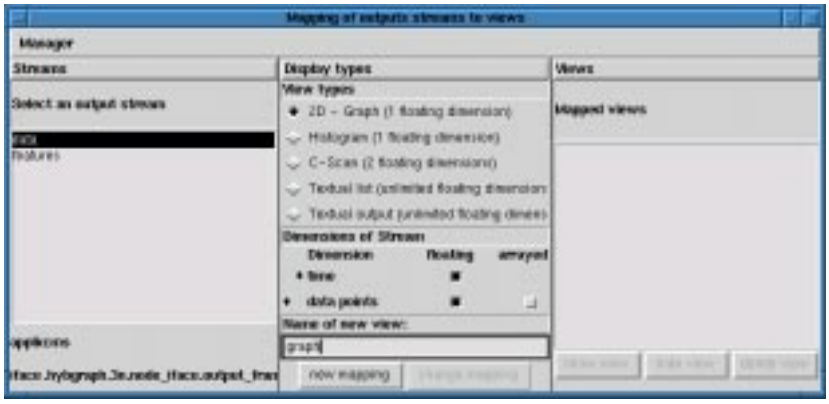


Abbildung 6.3: Konfiguration einer neuen Visualisierung. Zu jeder Datenquelle (stream) können eine oder mehrere unterschiedliche Displaymodi (view types) ausgewählt werden. Dabei können die dargestellten Dimensionen der Daten frei spezifiziert werden.

Im folgenden werden einige Beispiele solcher physikalischer Interaktionselemente aufgeführt. Für weitere Beispiele sei auf Anhang C verwiesen.

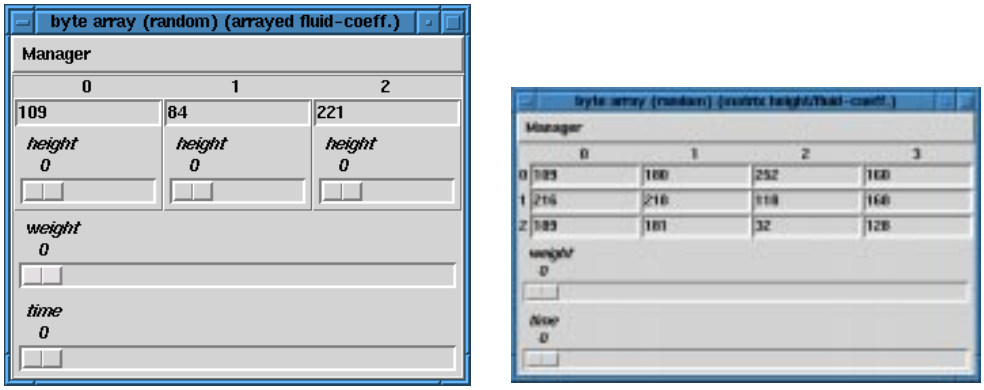
Multidimensionsdarstellung

Das Problem bei der Darstellung multidimensionaler Daten wird durch die Auswahl der zu betrachtenden Dimension gelöst. Durch die Schieberegler kann ein beliebiger Schnitt durch die multivariaten Daten in der jeweiligen Dimension gewählt werden. Die Anzahl der durch die Schieberegler darzustellenden Dimensionen wird automatisch anhand der durch die Grunddarstellung nicht visualisierbaren Dimensionen festgelegt. Beispiele für derartige Dimensionsselektionen werden u.a. in den Abbildungen 6.4(b) und 6.5 dargestellt.

Matrixdarstellung

Mit dieser Form der Darstellung können andere Darstellungsformen entlang einer oder zweier Dimensionsachsen aufgereiht werden. Die Selektion beschränkt sich hier auf die Matrixelemente.

Ein Beispiel für diese Darstellung findet sich in Abbildung 6.4(a). Dabei handelt es sich um einen dreidimensionalen Datensatz. In diesem Fall wird die Dimension `fluid-coeff` in der ersten Matrixachse dargestellt. Für jeden Wert in dieser Dimension[2] wird ein Textfeld erzeugt. Die Werte für die Dimension `height` lassen sich für jeden dieser Werte unabhängig einstellen. Im Gegensatz dazu ändern sich in Abbildung 6.4(b) bei einer Änderung der Dimension alle Werte gleichzeitig. Die Dimensionen `height` und `fluid-coeff` werden in einer zweidimensionalen Matrixform abgebildet.



(a) Dimension `height` wird durch das Textfeld verwaltet.

(b) `height` und `fluid-coeff` in Matrizendarstellung.

Abbildung 6.4: Textuelle Darstellung in Matrixform.

Linien und Säulendiagramme

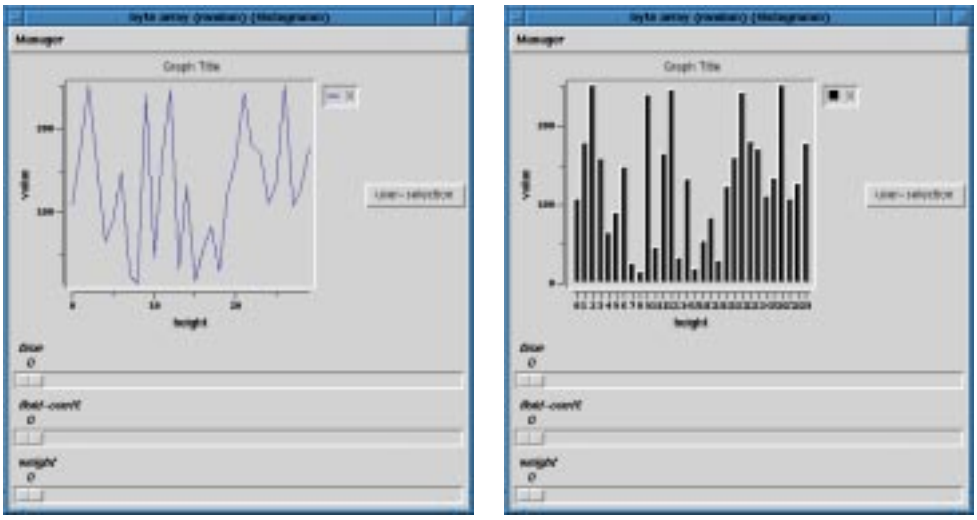
Liniendiagramme, die der Einfachheit halber auch die Säulendiagramme einschließen, stellen einen eindimensionalen Ausschnitt der multivariaten Daten als Linienzug dar (siehe Abbildung 6.5). Der Benutzer kann dabei im Definitionsbereich Selektionen durchführen.

Falschfarbendiagramm

Neben einfachen Kurven existiert eine Vielzahl anderer Visualisierungsmöglichkeiten. Außer einfachen Tabellen seien noch die Falschfarbendiagramme erwähnt, die sich als sehr gut geeignet für große Datenmengen herausgestellt haben. Als Beispiel sei hier ein dreidimensionaler[3] Datensatz aus dem ***NeuroPipe***-Projekt (siehe Abbildung 6.6) dargestellt. Dieser Datensatz besteht aus circa 5.7 MByte Daten.

[2] Dimensionen sind immer 0-basiert.

[3] Die Dimension der einzelnen Bilder ist zweidimensional. Durch die Präsenz mehrerer zueinander gehörender Darstellungen entsteht die dritte Dimension im Intervall $[0, 1]$.



(a) Ausgabe als Linienzug. **(b)** Ausgabe als Säulendiagramm.

Abbildung 6.5: Visualisierung von 1-D-Daten mittels Linien und Säulendiagramm.

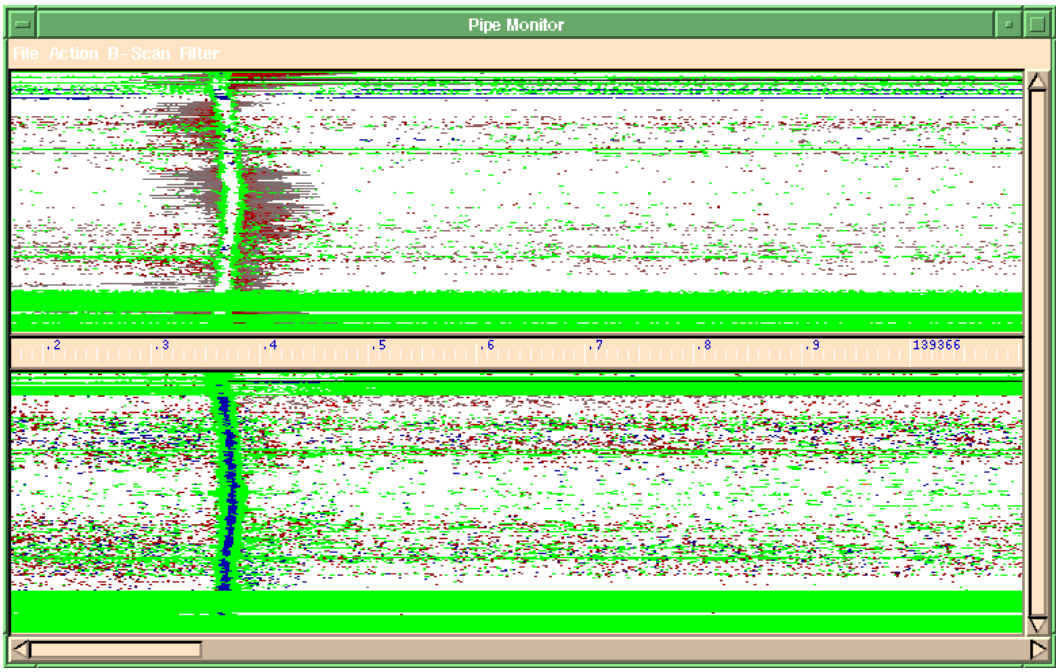


Abbildung 6.6: Falschfarbendarstellung eines Rohrabschnittes aus dem ***NeuroPipe I*** Projekt, angeordnet unter Zuhilfenahme der Matrixdarstellung (für die Anordnung der zwei Bilder übereinander) und der Falschfarbendarstellung.

Die Selektion in dieser Darstellung kann mit Hilfe der eingezeichneter Rechtecke direkt interaktiv durchgeführt werden (vgl. Abbildung 6.8).

Diese Möglichkeiten zur Benutzerinteraktion sind nicht erschöpfend, stellen aber durch das hier verfolgte Konzept eine einfach erweiterbare modulare Architektur dar. Diese Komponenten werden im folgenden zur Akquisition von Anomalieklassen verwendet.

6.4 Interaktive Akquisition von Anomalieklassen

Im Kapitel 2.1.3 wurden die unterschiedlichen Stufen der Wissensakquisition erläutert. Dadurch, daß die ILD im wesentlichen auf subsymbolischen Werten arbeitet, beschränkt sich die Wissenserhebung auf die Möglichkeit, die multivariaten Daten interaktiv zu selektieren bzw. durch das Diagnosesystem Selektionen angezeigt zu erhalten. Zusätzlich wird zu jeder Selektion eine Diagnose des selektierten Datenbereichs präsentiert.

Eine weitere Möglichkeit der direkten Wissenseingabe stellt sich in Form von Regeln. Dies ist wesentlich, falls a priori Regelwissen existiert. Es wäre allerdings u.U. nur sehr mühsam oder vielleicht gar gefährlich, die richtigen Beispiele auszusuchen, um auf diese Art der Lernkomponente genügend Informationen bereitzustellen, damit diese sie einlernen kann. Aus diesem Grund wird nachfolgend eine Methode vorgestellt, die es erlaubt, Regeln direkt zu akquirieren.

6.4.1 Integration von Regelwissen

Wissen, das in Form von Regeln existiert, soll dem Diagnosesystem zur Verfügung gestellt werden. In Abschnitt 2.1.2.1 wurden bereits einige Wissensrepräsentationen erläutert. Die meisten von ihnen bedienen sich Symbole als Eingabesprache. Die ILD arbeitet auf subsymbolischen Merkmalen, was impliziert, daß das eingegebene Wissen sich auch subsymbolisch darstellen lassen muß. Die Vorgehensweise bei der Erstellung von Regeln ist in Abbildung 6.7 dargestellt.

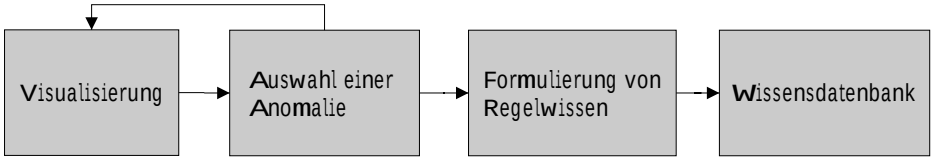


Abbildung 6.7: Ablauf bei der Erstellung von Regeln. Zuerst wird eine Idee für eine Regel durch wiederholte Betrachtung unterschiedlicher Anomalien überprüft. Diese wird als Regel mit Hilfe von linguistischen Variablen formuliert und in die Wissensbasis übertragen.

Zu diesem Zweck wurde das Vorgehen an die Art und Weise angelehnt, wie es bei der Regelspezifikation von Fuzzy–Controllern der Fall ist, d.h. es existiert eine Eingabemöglichkeit für Regeln in subsymbolischer Form (siehe Abschnitt 5.3.4).

Die Idee besteht dabei in der Benennung unterschiedlicher Bereiche der einzelnen Eingabemerkmale und Benennung der Ausgabewerte. Mit Hilfe von einfachen Wenn–Dann-Beziehungen werden diese zu Regeln gruppiert. Damit beschreiben sie eine konkrete Abbildung aus dem Merkmalsraum auf die unterschiedlichen Klassen. Diese Abbildung läßt sich direkt in die Repräsentation eines *Dynamic Bounds* Netzes im speziellen und auf ein RBF–Netz im allgemeinen überführen. In Tabelle 6.2 ist ein Beispiel für eine derartige Regelbeschreibung dargestellt.

Die Regeln werden basierend auf linguistischen Termen definiert, die die Aufteilung der Eingabewerte in Symbole spezifizieren. Diese werden durch die Regeln in den unterschiedlichen Eingabedimensionen verknüpft und einer Ausgabe zugeordnet, die wiederum als ein Symbol definiert ist. Diese Regeln lassen sich mittels des im Abschnitt 5.3.4 beschriebenen Verfahrens in den *Dynamic Bounds*-Algorithmus integrieren.

```
in_dim: 0
{
nb:{-1.000000000 100.000000000 1.000000000}
nz:{0.000000000 0.100000001 0.050000001}
pz:{0.000000000 0.050000001 0.100000001}
pb:{1.000000000 1.000000000 100.000000000}
}

out_dim: 0
{
fatal:{3.0}
total:{5.0}
}

rules:
{
0: IF {0 nb} AND {1 nb} THEN {1 fatal}
1: IF {0 nb} AND {1 nz} THEN {0 total}
2: IF {0 nb} AND {1 pz} THEN {2 fatal}
}
```

Tabelle 6.2: Subsymbolische Regel-Darstellung mit Hilfe von Wenn–Dann–Regeln. *in_dim* und *out_dim* definieren die linguistischen Terme für die Eingabedaten bzw. die zu erzeugende Klassifikation. Die Regeln spezifizieren die Verknüpfung der Terme in den unterschiedlichen Dimensionen.

Den wesentlichen Anteil bei der ILD stellt allerdings die interaktive Diagnose dar.

6.4.2 Interaktive Akquisition von Defektklassen

Durch die visuelle Darstellung der Daten und die dazugehörigen Selektionen zur Markierung der Interessengebiete können auch mehrere Bereiche gleichzeitig dargestellt werden, um so dem Benutzer einen besseren Überblick über die Daten zu gewähren. Dabei ist es besonders wichtig, daß die Daten in einer möglichst komprimierten, aber für den Betrachter verständlichen Sichtweise präsentiert werden. In Abbildung 6.8 wird die im ***NeuroPipe***-Projekt verwendete Darstellung zur Visualisierung von Anomalien gezeigt.

Durch die geschickte Anordnung wird ermöglicht, direkt unterschiedliche Anomalien (Selektionen) mittels des oberen Schiebereglers anzufahren, und es wird auch sofort die zugehörige Klassifikation anhand der hervorgehobenen Beschriftung links neben dem Schieberegler angezeigt.

Des weiteren läßt sich dabei die Applikation der unterschiedlichen Visualisierungsmöglichkeiten für multivariate Daten gut beobachten. In Abbildung 4.10 ist die Visualisierung durch die Matrixdarstellung in zwei Bereiche aufgeteilt, in eine obere und eine untere Hälfte. Diese Matrixelemente bestehen wiederum aus einer Matrixdarstellung mit jeweils einer oberen und unteren Hälfte. Der obere Teil besteht aus einer zweidimensionalen Falschfarbendarstellung und der untere aus einem Liniendiagramm, das die Werte der Daten in x-Richtung für eine wählbare y-Koordinate darstellt. Dadurch wird es möglich, diesen dreidimensionalen Datensatz aus unterschiedlichen Perspektiven zu betrachten, um so die Korrektheit der Diagnose des Systems zu validieren.

Die Selektion der Daten setzt sich durch alle Dimensionen fort. Die Änderung der

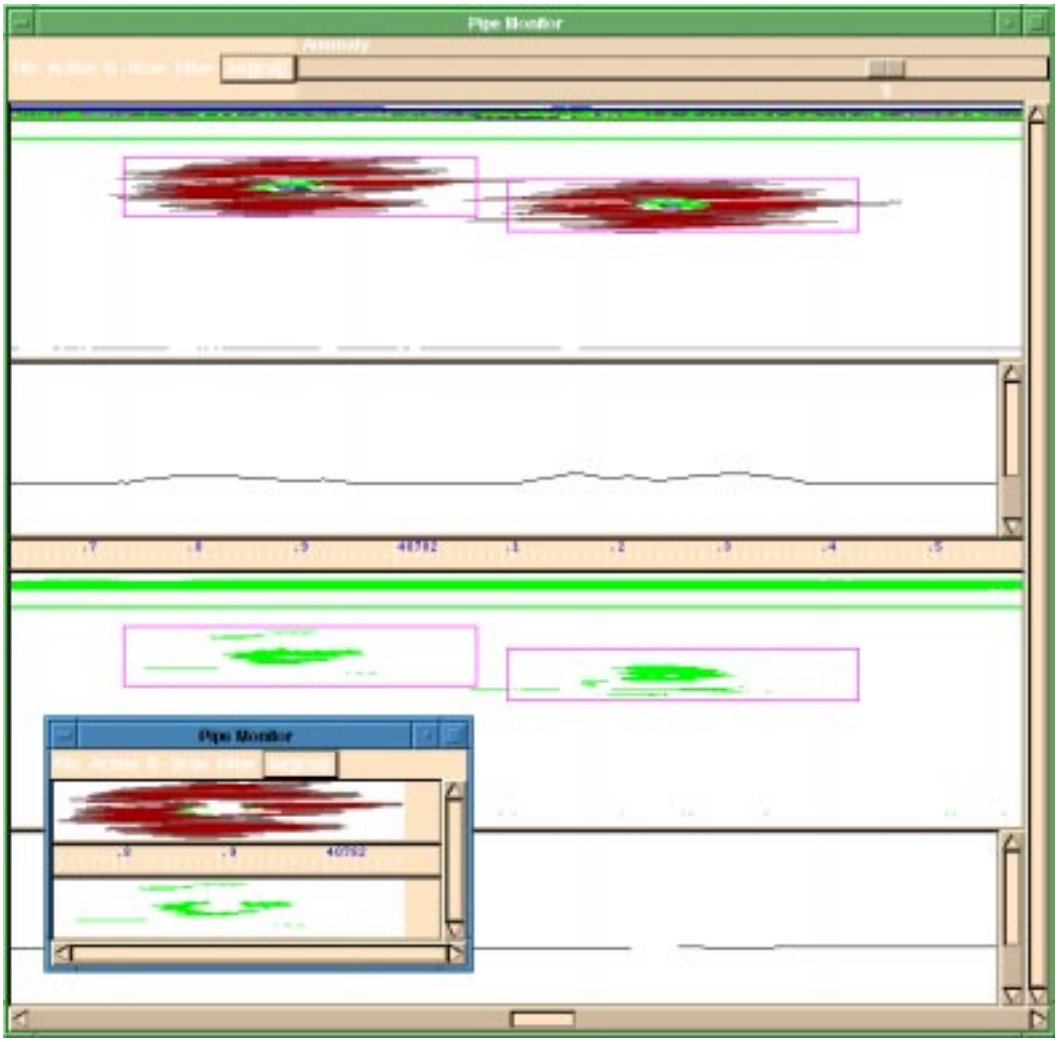


Abbildung 6.8: Falschfarbenvisualisierung eines Ausschnitts der Daten aus dem ***NeuroPipe***-Projekt. Dabei wurde ein Gebiet zum besseren Verständnis vom Benutzer extrahiert. Die mit Rechtecken eingerahmten Selektionen kennzeichnen die zu klassifizierenden Gebiete.

Selektion in einer Sicht überträgt sich automatisch, wegen der Restriktion der Selektion (siehe Abschnitt C.3.1), durch alle Darstellungen.

Falls eine falsche Klassifikation durch den Experten entdeckt wird, muß die Möglichkeit der Korrektur bestehen. Dies geschieht entweder durch eine auf die Applikation zugeschnittene Klassifizierungseingabe, oder wie später erläutert wird, unter applikationsunspezifischen Klassifikationen. Ein Beispiel für die applikationsspezifische Klassifizierung stellt Abbildung 6.9 dar. Neben der reinen Klassifikation werden noch andere für die Diagnose nicht notwendige Informationen gesammelt.

Nicht immer besteht die Notwendigkeit, zusätzliche Klassifizierungshilfen zur Verfügung zu stellen, beispielsweise existieren bei der interaktiven Klassifikation von Längsnähten in Pipelines nur zwei Klassen: Naht und Nicht-Naht. Dies kann bereits durch die reine Selektion erledigt werden. In Abbildung 6.10 ist wieder ein dreidimensionaler Ultraschall-Datensatz zu sehen, in dem eine waagerechte Selektion durchgeführt wurde. Sie markiert die Position der Längsnaht. Dabei ist zu beachten, daß die Selektion eindimensional ist. Es wird lediglich die y-Koordinate in der Darstellung selektiert.

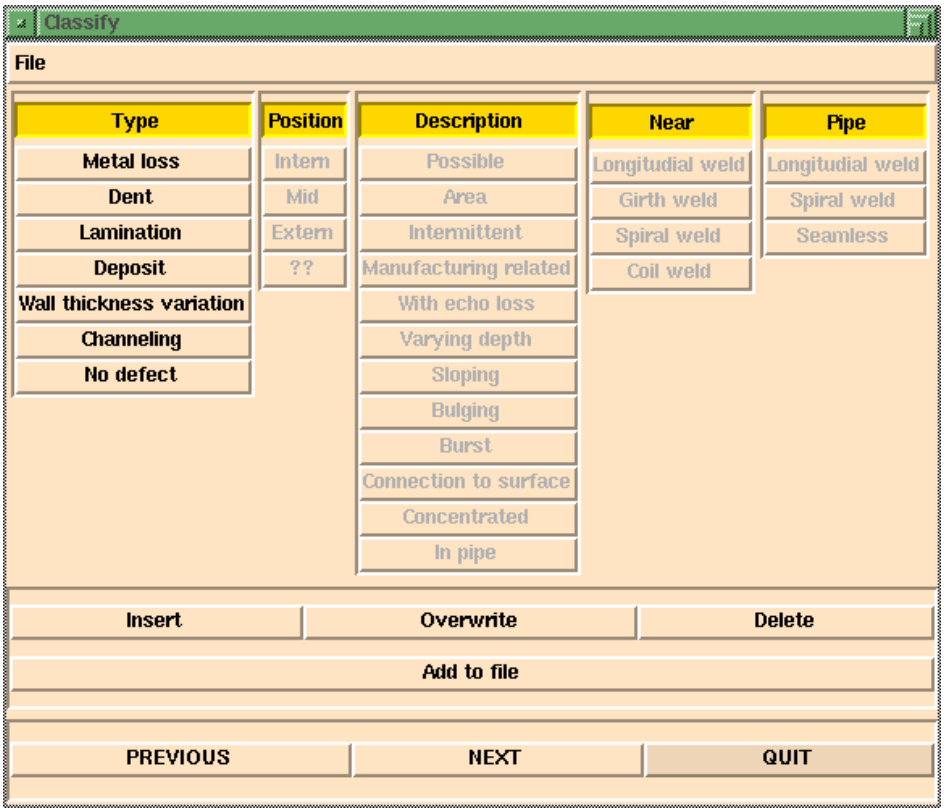


Abbildung 6.9: Klassifizierung am Beispiel der Anomalien im ***NeuroPipe I***-Projekts. Es werden optional zusätzliche Informationen bezüglich der Anomalien gesammelt.

Dadurch entsteht eine Projektion durch alle restlichen Dimensionen, die sich in der Abbildung als ein einzelner Strich darstellt.

6.4.3 Interaktive Suche nach Anomaliesonderfällen

Bevor ein Lernverfahren eingesetzt werden kann, müssen Lernbeispiele zur Verfügung stehen. Diese sind entweder a priori gegeben, oder sie müssen explizit gesammelt werden. Im vorherigen Abschnitt wurde die Interaktive Vorgehensweise bei der Validierung der Diagnosen aufgezeigt.

Das Problem beim Sammeln von Beispielsdaten besteht in der Unkenntnis der internen Repräsentation der Lernkomponente. Der Mensch beurteilt die Signifikanz einzelner Beispiele nach seinen eigenen Maßstäben und nicht aus dem Blickwinkel des Systems. Demzufolge kann es leicht geschehen, daß der Benutzer dem System nur einen Teil der zu diagnostizierenden Daten angibt, und zwar jene, die für ihn schwierig zu interpretieren sind; und die anderen Bereiche bleiben dann unvollständig oder sind dem Diagnosesystem nur partiell bekannt (vgl. Abschnitt 5.4.3).

Für diese Problematik wurden in dieser Arbeit zwei Ansätze entwickelt: das automatische Sammeln von Beispielen, auf das im nächsten Abschnitt (6.4.4) eingegangen wird, und die Möglichkeit, benutzerdefinierte Filter bzgl. der Klassifikatorausgaben zu definieren.

Diese Filter erlauben es, die Ausgabenklassifikationen zu bewerten und zu protokollieren. Der Benutzer ist damit später in der Lage, die gefundenen Daten zu inspizieren und deren Klassifikation zu überprüfen.

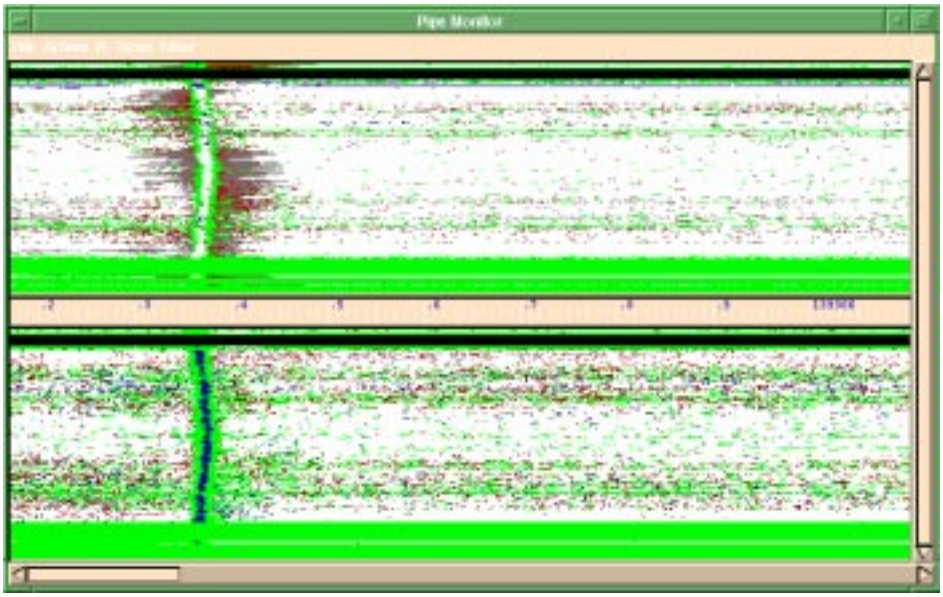
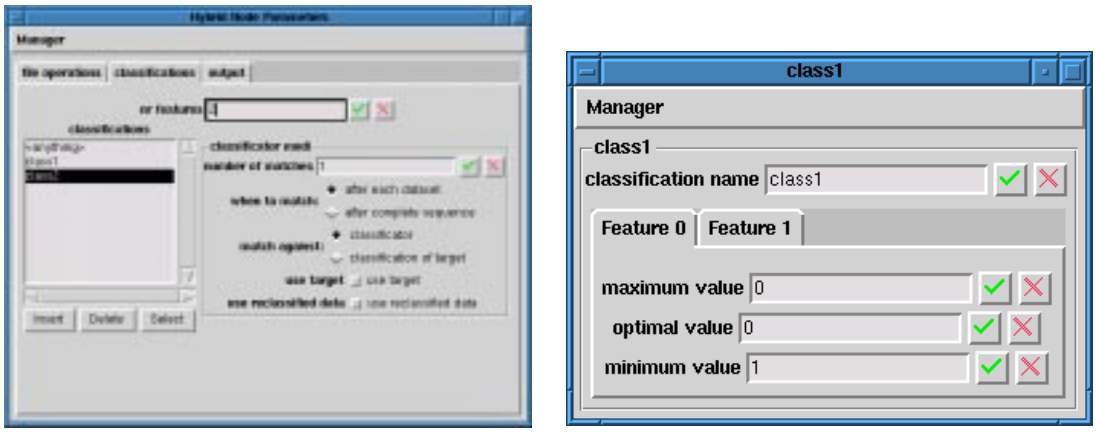


Abbildung 6.10: Klassifizierung der Längsnähte im ***NeuroPipe I***-Projekt mittels direkter Selektion. Der schwarze waagerechte Balken bezeichnet dabei die getätigte Selektion.

Diese Vorgehensweise wurde in Form eines Streamfilters umgesetzt, der es dem Benutzer erlaubt, Kriterien zu erstellen, nach deren Muster im Datenstrom gesucht werden kann (z.B. Fehlklassifikationen). Die Muster, die diese Kriterien erfüllen, können später visualisiert werden. Außerdem kann der Benutzer diese Daten neu klassifizieren und an die Wissensverwaltung übergeben und/oder sofort der Lernkomponente zuführen.



(a) Einstellung der zu detektierenden Klassen bzw. Aktivierungen der Neuronalen Netze.

(b) Einstellung eines Klassifizierungsfilters für jede Ausgabe der Neuronalen Netze.

Abbildung 6.11: Benutzerdefinierte Filterung der Klassifikation.

Der Benutzer kann ferner in der Liste aus Abbildung 6.11(a) beliebige Filter eingeben, gegen die er die Datensätze prüfen möchte. Ein solcher Filter ist in Abbildung 6.11(b) dargestellt. Für jedes Merkmal muß ein Intervall angegeben werden, das angibt, ob der Filter aktiviert wird. Weiterhin kann ein optimaler Wert angegeben

werden, der als Vorschlag zur Umklassifizierung dient.

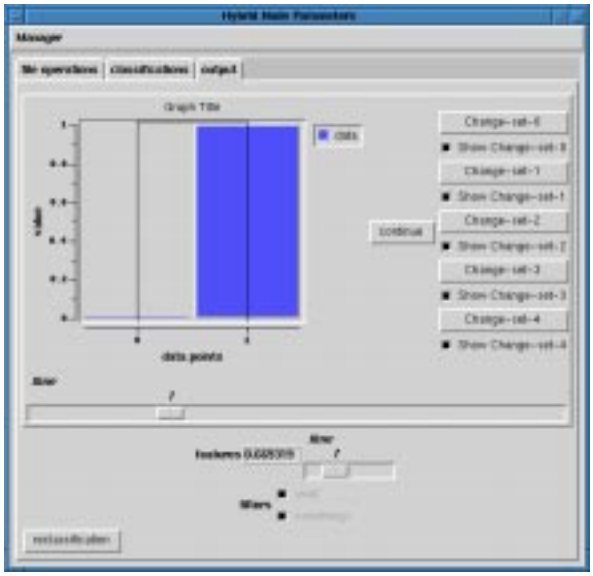


Abbildung 6.12: Visualisierung der Merkmale eines durch den Filter gefundenen Datensatzes am Beispiel eines zweidimensionalen Merkmalsvektors.

In Abbildung 6.12 ist eine Beispielsausgabe zu sehen, nachdem der Klassifikator vier Treffer gefunden hat. Jeder Datensatz besteht aus zwei Punkten, die über der x-Achse des Ausgabefensters aufgetragen sind. Mit der Zeitachse (*time*-Dimensionsschieberegler) läßt sich das entsprechende Pattern wählen. Unter der Ausgabe für die Datenpunkte ist die aktuelle Klassifikation für diesen Datensatz zu sehen (`features`).

Für jede gefundene Klassifikation wird eine Selektion erzeugt. Anhand dieser Selektionen lassen sich die Klassifizierungen ändern. Die Aktivierung der Selektion durch den Benutzer bereitet einen der gefundenen Patterns für die Umklassifizierung vor.

Nachdem eine entsprechende Anzahl von Datensätzen gefunden wurde, bzw. wenn das Ende der Sequenz oder Epoche erreicht wurde, stoppt der Klassifikator die Verarbeitung und erlaubt dem Benutzer, die Daten umzuklassifizieren.

Nachdem der Klassifikator gestoppt hat, besteht für den Benutzer die Möglichkeit, die klassifizierten Daten mit anderen Klassenzugehörigkeiten zu versehen. Dazu kann er sich anhand der Selektionsschaltflächen auf der rechten Seite eine gefundene Klassifikation aussuchen, die er ändern möchte. Anschließend werden im Reklassifikationsfenster alle Filter aufgeführt, die auf diesen Datensatz zutreffen. Mittels der Selektionsknöpfe auf der rechten Seite kann er die Daten auf den vordefinierten Wert des Filters umklassifizieren.

6.4.4 Automatisches Sammeln von Beispielsdaten

Die Idee beim Sammeln von Beispielsdaten besteht darin, möglichst objektiv diejenigen Daten zu detektieren, bei denen die Lernkomponente keine hinreichend gute Klassifikation liefert. Es handelt sich also um die Detektion neuer Beispiele aus den Rohdaten, zu denen noch keine Klassifikation durch den Experten durchgeführt worden ist, d.h. die also noch unbekannt sind. Um vorurteilsfrei eine Beispielsammlung anfertigen zu können, müssen zufällig ausgewählte Rohdaten diagnostiziert werden, und aus der Klassifikation muß abgeleitet werden können, ob das betreffende Beispiel protokolliert

werden soll. Zusätzlich kann eine Abschätzung der Dringlichkeit der Betrachtung dieses Beispiels prognostiziert werden.

Dies kann dadurch bewerkstelligt werden, indem direkt die Ausgaben des *Dynamic Bounds*-Netzes verwendet werden. Dabei existieren die folgenden Möglichkeiten, eine Eingabe $\vec{x}$ zu klassifizieren (vgl. Kapitel 5.3):

1. $prop(\vec{x}) = \vec{0}$.
   Das Beispiel wird abgelehnt. Das bedeutet, daß für diese Regionen des Merkmalsraums noch *keine* Beispielsdaten existieren.

2. $\exists \vec{c_1}, \vec{c_2} \in prop(\vec{x}) : \vec{c_1} \neq \vec{c_2}$
   Es existieren mehrere Klassifikationen von $\vec{x}$ mit unterschiedlicher Klassenzugehörigkeit. Das Beispiel befindet sich also im Einflußbereich verschiedener Klassen, d.h. in der internen Repräsentation von *Dynamic Bounds* überlappen sich die Regionen.

3. $\vec{x} \in \mathcal{H}$ von genau einer Klasse.
   Der *Dynamic Bounds* klassifiziert dieses Beispiel nur als Hypothese. $\vec{x}$ befindet sich also in der Nähe einer sicheren Region der gleichen Klasse.

4. $\vec{x} \in \mathcal{R}$.
   Das Beispiel befindet sich in einem bekannten Randbereich zwischen zwei oder mehreren Klassen.

5. $\vec{x} \in \mathcal{S}$ von genau einer Klasse.
   Das Beispiel ist innerhalb einer sicheren Region.

Die oben angeführten Klassifikationsmöglichkeiten sind nach ihrem Grad der Notwendigkeit zur Überprüfung absteigend sortiert. Dieser Prozeß des Beispielsammelns wird offline in der Optimierungsphase durchgeführt. Zu jeder der aufgeführten Klassen wird eine maximale Anzahl an Beispielen gesammelt bzw. für den gesamten Prozeß eine obere Zeitschranke angesetzt. Dies ist deswegen sinnvoll, da die gesammelten Beispiele anschließend dem Experten direkt im Zusammenhang mit den Rohdaten präsentiert werden, damit dieser sie klassifizieren kann.

Durch das interaktive Lernen verändert sich die Klassifikationsgüte der Lernkomponente, und dadurch könnte eine Vielzahl der gesammelten Problemfälle durch wenige Repräsentanten abgedeckt werden. Aus diesem Grund ist es sinnvoll, zu Beginn auf diese Art des Sammelns von Beispielen zu verzichten und erst interaktiv einen Grundklassifikator aufzustellen. Dann wird mit geringen Suchmengen und häufigeren Intervallen nach zusätzlichen Beispielen gesucht und die Größe der Suchmenge sukzessive erhöht.

Dieser Prozeß kann prinzipiell auch parallel zum Interaktiven Lernen stattfinden, wenn dafür in Kauf genommen wird, daß bereits korrekt klassifizierte Beispiele nochmals als *Problembeispiele* präsentiert werden.

Die Einbeziehung der letzten beiden Gruppen von Klassifikationen (Nummer 4 und 5) dient im wesentlichen zur Validierung des Klassifikators. Diese Beispiele können nach erfolgreicher Überprüfung der Klassifikation in den Testdatensatz aufgenommen werden.

Bei der ILD kommt das Wissen aus den Beispielen, die durch den Experten klassifiziert werden. Deswegen ist es essentiell, alle diese Beispiele zu protokollieren. Denn

das aufwendigste der ILD besteht gerade in der Wissensakquisition, und deshalb darf das einmal akquirierte Wissen nicht wieder verloren gehen.

6.5 Verwaltung von Beispielswissen

Gewöhnlich werden Beispiele in Form einfacher Dateien abgelegt, so daß die später immer wieder auftretende Aufteilung in Lern-, Test- und andere Datensätze sowie die große Anzahl von entstehenden Netzen und unterschiedlichen Vorverarbeitungs-, Merkmalsextraktionsalgorithmen oder Anordnungen zur Verwirrung des Entwicklers führen.

Während der Erstellung eines Diagnosesystems müssen notgedrungen unterschiedliche Konfigurationen evaluiert werden, und es werden auch zu diesen unterschiedlichen Konfigurationen die entsprechenden Klassifikatoren und ihre Qualität benötigt, um sie miteinander zu vergleichen.

Die Wissensverwaltung hat die Aufgabe, alle anfallenden Beispiele, Regeln, Applikations- und Vorverarbeitungskonfigurationen sowie die Netzkonfigurationen und die erzielten Ergebnisse persistent zu sichern. Sie werden in der ***CMS***-Datenbank aufbewahrt (siehe auch Anhang B).

Alle anfallenden Beispiele werden in die Datenbank aufgenommen. Dabei wird geprüft, ob das Beispiel nicht im Konflikt mit einem bereits vorhandenen Beispiel steht. Dies wird anhand einer applikationsunabhängigen Erfassung der Position des Beispiels ermöglicht. Die Position beschreibt dabei auf abstrakte Art und Weise, wie ein Beispiel wieder in den Daten lokalisiert werden kann. Dabei muß berücksichtigt werden, daß sich durch eine andere Art der Vorverarbeitung die Positionsinformation nicht ändern darf, damit immer eine eindeutige Zuordnung der Beispiele erfolgen kann.

Die Datenbank bietet darüber hinaus die Möglichkeit, unterschiedliche statistische Prozeduren durchzuführen. Dazu zählt beispielsweise die Generierung der Cross-Validation- Datensätze und das parallele Einlernen unterschiedlich konfigurierter Hybrider Knoten, was insbesondere für das Auffinden der richtigen Topologien und Initialisierungen für die Backpropagation-Netze notwendig ist. Dies geschieht mittels eines sog. *JobManagers*. Weiterhin ist es möglich, die erzeugten Resultate zu verifizieren.

Im folgenden Abschnitt wird die ***CMS***-Datenbank näher erläutert.

CMS-Datenbank

Die ***CMS***-Datenbank orientiert sich hauptsächlich am Fluß der Beispielsdaten und ihrer Verwendung innerhalb des Diagnoseprozesses (siehe Abbildung 6.13). Jedes Projekt basiert auf einer *Stammdatei*, das die Positionen aller bekannten Beispiele enthält. Von dieser Stammdatei werden sogenannte Stammdatei-Selektions-Dateien abgeleitet. Jede CMS-Datei besitzt eine Versionsnummer, um so unterschiedliche Generationen verwalten zu können. Im Gegensatz zu den CMS-Dateien existiert immer nur eine Stammdatei zu einer Versionsnummer. Eine neue Versionsnummer wird immer dann erzeugt, wenn Beispiele gelöscht bzw. umklassifiziert werden müssen. Das Hinzufügen von Beispielen verändert die Versionsnumerierung nicht.

Die Ableitung der CMS-Dateien von der Stammdatei oder anderen CMS-Dateien ist immer eine Datenselektion, d.h. daß niemals neue Beispiele in die abgeleiteten CMS-Dateien einfließen, sondern daß sie immer eine Untermenge des Vorgängers sind. Von

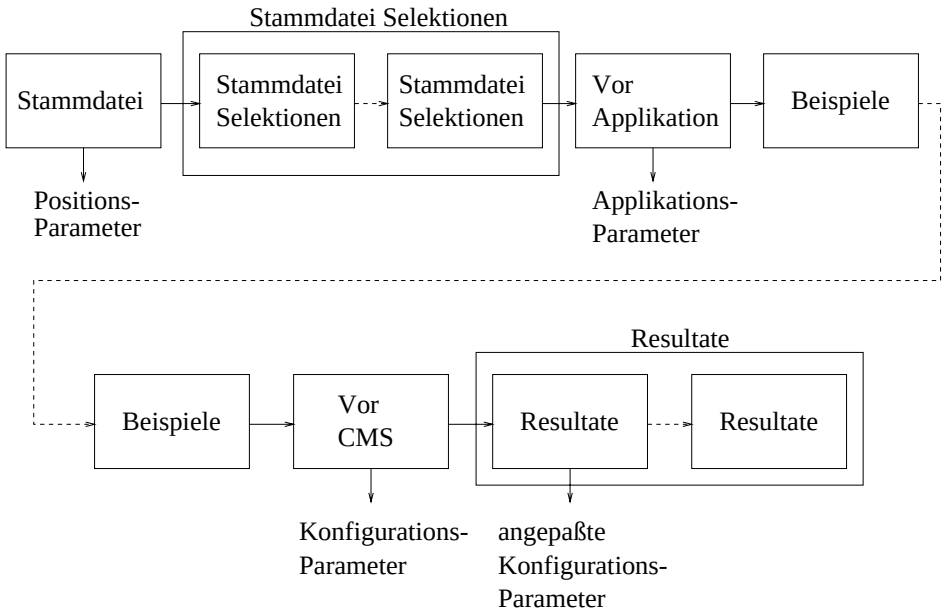


Abbildung 6.13: Die Abhängigkeiten der einzelnen Elemente der ***CMS***-Datenbank. Sie repräsentieren die Beispielsselektion zu den unterschiedlichen Zeitpunkten des ILD-System-Lebenszyklus.

einer CMS-Datei können grundsätzlich mehrere Nachfolger abgeleitet werden (vgl. Abbildung 6.13).

Die gesamte Struktur stellt also einen azyklischen gerichteten Graphen dar. Dieser kann in drei Ebenen eingeteilt werden, die durch zwei Prozeßschritte getrennt sind (siehe Abbildung 6.14).

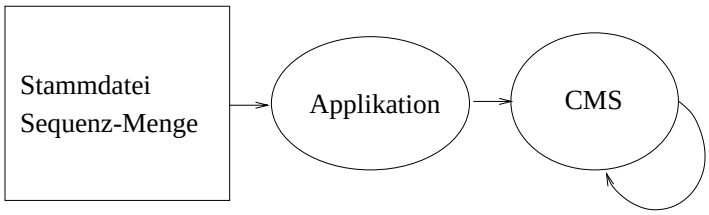


Abbildung 6.14: Der Ablauf der Prozeßschritte in der ***CMS***-Datenbank.

Die Prozeßschritte gliedern sich dabei wie folgt:

- Das Ausführen der Applikation und die Anwendung der Vorverarbeitung und der Merkmalsberechnung auf die Beispiele, die durch die Positionsinformation definiert sind. Dazu werden alle notwendigen Parameter der involvierten Algorithmen von der Datenbank gespeichert, um eine spätere Rekonstruktion durchführen zu können.

- Die Ausführung des ***CMS***, um weitere Vorverarbeitungs- und Merkmalberechnungsschritte durchzuführen und alle involvierten Lernalgorithmen ablaufen zu lassen. Die Parametrisierung diese Prozesses wird wiederum in der Datenbank abgespeichert.

Dadurch ergibt sich eine Dreiteilung des Beispielsflusses:

- Die Selektion der Beispiele, die durch die Applikation ausgewertet werden sollen. Zu diesem Zeitpunkt besteht das Beispiel im wesentlichen aus der Klassifikation und der Positionsangabe.
- Die Beispiele sind um die durch die Applikation berechneten Merkmale erweitert und können weiter selektiert werden. An dieser Stelle wird beispielsweise die Aufsplittung für die Cross-Validation durchgeführt.
- Die durch das ***CMS*** verarbeiteten Beispiele enthalten zusätzlich die Klassifikation und diverse Angaben zur Güte des Ergebnisses.

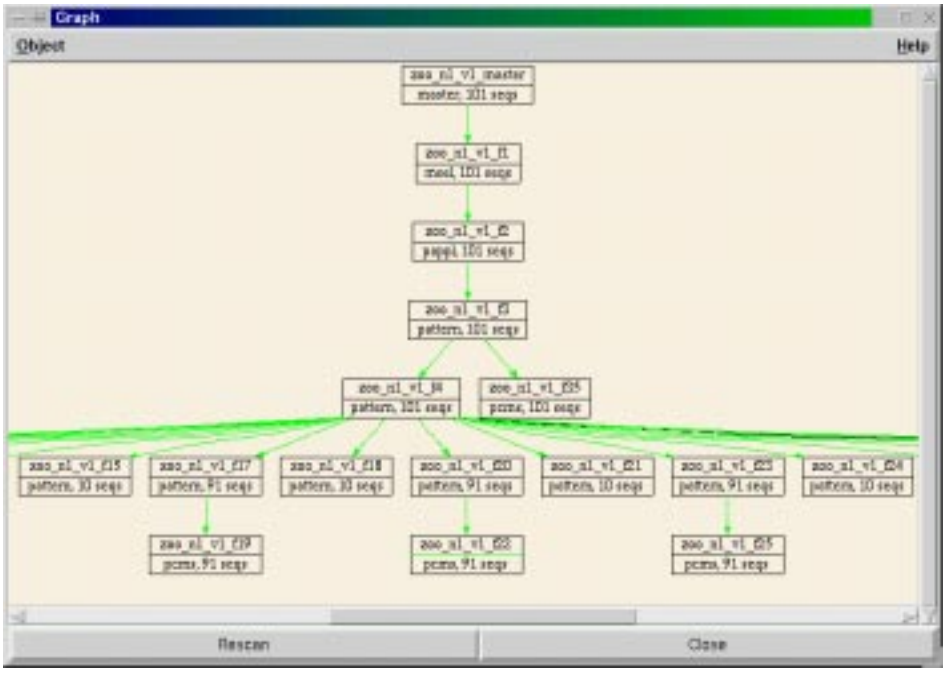


Abbildung 6.15: Graphische Darstellung der Abhängigkeit der CMS-Dateien innerhalb eines Projekts. Deutlich zu sehen sind die unterschiedlichen CMS-Datei-Typen und Anzahlen der selektierten Beispiele.

Dieser Prozeß kann sich an einer beliebigen Stelle aufspalten, indem unterschiedliche Versuche mit geänderter Parametrisierung durchgeführt werden (siehe Abbildung 6.15). Es entsteht eine baumartige Struktur. Durch diese Art der Sicherung der Ergebnisse ist es jederzeit möglich, Ergebnisse abzurufen und zu vergleichen. Durch die Protokollierung der Parameter für die Applikation sowie für die Konfiguration des ***CMS*** können Experimente jederzeit wiederholt bzw. objektiv verglichen werden.

Als Grundlage für die ***CMS***-Datenbank wurde OBST verwendet. Das ist ein am FZI (Forschungszentrum Informatik an der Universität Karlsruhe) im Rahmen eines öffentlich geförderten Projekts entwickeltes objektorientiertes-DBMS. Mit Hilfe dieser objektorientierten Datenbank wurden in dieser Applikation bis zu einer Million Objekte verwaltet.

Um diese Datenbank und die Lernkomponente handhaben zu können, wurden die nachfolgenden Monitore entwickelt.

6.6 Monitorkomponente

Die Monitorkomponente setzt sich aus unterschiedlichen Teilkomponenten zusammen: den ***CMS***-Monitor, für die Visualisierung des Lernfortschritts der Lernkomponente, die Inspektion und Abfrage der ***CMS***-Datenbank als auch der Zustände der laufenden parallelen Diagnoseprozesse (siehe Abschnitt B).

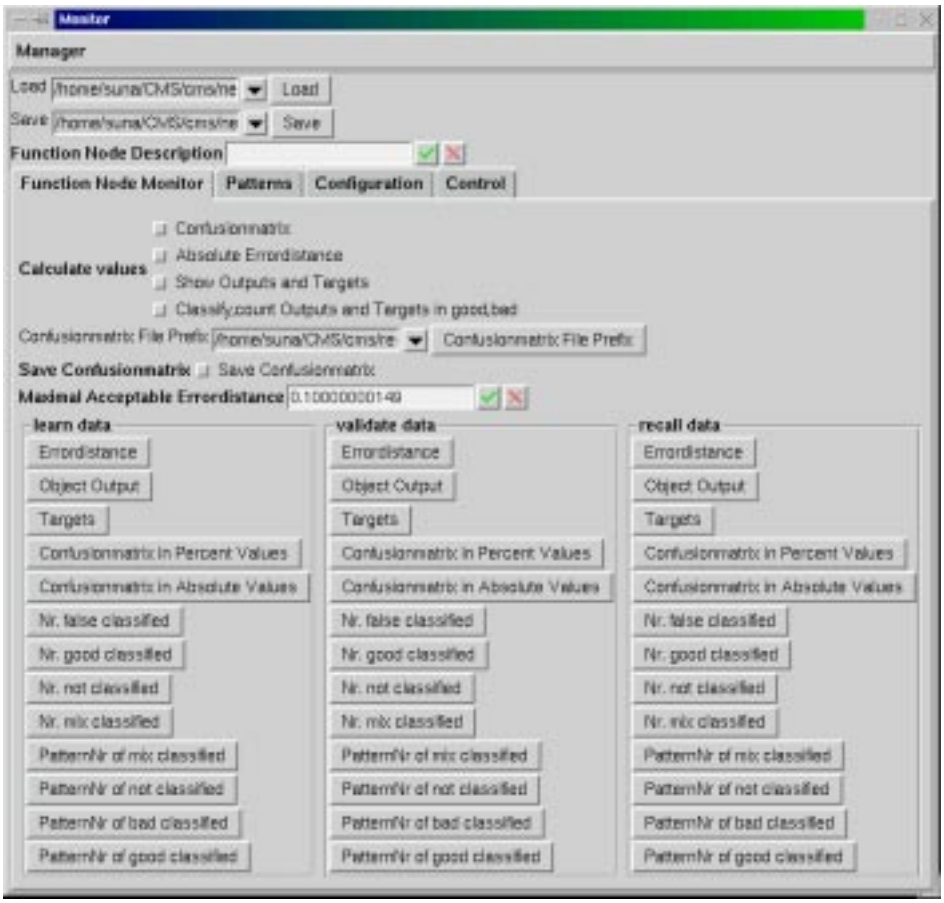


Abbildung 6.16: Ein ***CMS***-Monitor zur Überwachung der Klassifikationsgüte.

Der ***CMS***-Monitor erlaubt es, die Klassifikationsgüte direkt anzuzeigen. Es ist möglich, unterschiedliche Kriterien für die Klassenzuordnung zu spezifizieren, bzw. die Schwellen für die Richtig/Falsch- und Nichtklassifikation zu variieren (siehe Abbildung 6.16). Durch den Monitor kann eine Konfusionsmatrix erstellt werden, die ein Spiegelbild der Treffsicherheit des Klassifikators darstellt, indem sie die Ausgabe des Klassifikators den vorgegebenen Klassifikationen gegenüberstellt. Des weiteren ist auch die Ermittlung der applikationsabhängigen Risikomatrix möglich, indem die klassenspezifischen Risikofaktoren eingetragen werden können (siehe Abschnitt D.2). Dadurch ist eine objektive Bewertung der Klassifikatorgüte möglich.

Basierend auf der ***CMS***-Datenbank sind diverse Vergleiche, Abfragen und Reports verfügbar (siehe Abbildung 6.15). Dazu gehört auch die Berechnung der Konfidenzintervalle in Abhängigkeit von der Anzahl der Testbeispiele und die Ausgabe der darauf basierenden Zusicherungen für die Klassifikatorgüte (siehe D.4).

Des weiteren können die parallel laufenden Diagnoseprozesse einer Applikation, spe-

ziell die auf mehreren Rechnern laufenden Auswertungen der Pipelinediagnose, überwacht werden (siehe Abschnitt B).

6.7 Resümee

Die Interaktion bietet für den Benutzer eine Kommunikationsmöglichkeit mit dem System. Dabei erhält er die Möglichkeit, über diverse Visualisierungstechniken und Selektionsmöglichkeiten direkt in den Lernvorgang einzugreifen, als auch die Betrachtung der unterschiedlichen Monitore des ILD-Prozesses zu überwachen.

Es wird die graphische Interaktion beschrieben, die mittels des speziell dafür entwickelten Oberflächensystems ***Iface*** realisiert wurde. Mit Hilfe dieses Systems können multivariate Daten auf eine für den Benutzer intuitiv verständliche Weise dargestellt und sogar weitere durch den Anwender definierte Darstellungen verarbeitet werden. Ein wesentlicher Bestandteil der Interaktion besteht in den unterschiedlichen Darstellungsmöglichkeiten der Daten. Dazu gehören die Multidimensionsdarstellung, die Matrixdarstellung, die textuelle Ausgabe und die Ausgabe in Diagrammform (Linien- und Falschfarbendiagramme), wobei die Datenmenge mittels spezieller Selektionen eingeschränkt werden kann.

Die Interaktion beschränkt sich jedoch nicht nur auf die Darstellungsmöglichkeiten der Daten, sondern ihre Hauptaufgabe besteht eigentlich darin, eine interaktive Wissensakquisition zu ermöglichen. Dies wird im wesentlichen durch die folgenden Interaktionsmöglichkeiten erreicht:

- Integration von Regelwissen,
- Interaktive Diagnostizierung,
- Interaktive Diagnosefilterung.

Auch wenn bei der vorliegenden Untersuchung davon ausgegangen wurde, daß nur wenige oder keine Regeln bezüglich des zu diagnostizierenden Problems existieren, wurde dennoch die Möglichkeit, Regeln zu integrieren, bereitgestellt. Diese können dann mittels Wenn-Dann-Beziehungen interaktiv als konkrete Abbildung aus dem Merkmalsraum auf unterschiedliche Klassen dargestellt werden.

Eine weitere Interaktionsmöglichkeit besteht in der Interaktiven Diagnostizierung. Neben der visuellen Darstellung und Überprüfung der Klassifikationsergebnisse können Falschklassifizierungen oder Daten, die sich einer Klassifikation entziehen, korrigiert bzw. einer bestimmten Klasse zugeordnet werden, wobei auch vom System diesbezüglich eine Aufforderung an den Benutzer erfolgt.

Als weitere interaktive Unterstützung bietet die Interaktive Diagnosefilterung die Möglichkeit, den zu betrachtenden Datenraum auf bestimmte Ereignisse oder Klassen einzuschränken. Damit lassen sich Teilproblematiken konkret auswerten und die Komplexität reduzieren.

Unterstützt werden diese Möglichkeiten der Interaktion durch das Sammeln von Beispielsdaten. Dazu werden in einem offline-Prozeß alle nicht eindeutig zuordenbaren Beispielsdaten gesammelt und dann komplett dem Benutzer zur Überprüfung präsentiert. Dies dient dazu, dem Benutzer möglichst viele Daten, bei denen eine Interaktion unbedingt erforderlich ist, auf einmal präsentieren zu können, um auf diese Weise eine

zeitliche Verkürzung der interaktiven Phase durch Ausschluß von Wartezeiten zu erreichen. Diese Vorgehensweise ermöglicht es auch, die Vollständigkeit der Beispielsdaten zu erhöhen.

Zur Interaktion gehören außerdem noch Methoden zur Wissensverwaltung, die in der vorliegenden Arbeit von der objektorientierten ***CMS***-Datenbank übernommen werden.

Abgerundet wird die Interaktion durch die Monitorkomponenten, die dem Benutzer die Entscheidungen und Zustände des Systems verständlich darstellen. Basierend auf der ***CMS***-Datenbank können mittels Abfragen und Reports, Entscheidungen des Systems nachvollzogen werden.

Mittels der vorgestellten Interaktion wird also ein hohes Maß an Zusammenarbeit und Transparenz zwischen System und Benutzer erreicht. Durch die flexible Anpassung der Visualisierung und der direkten Kopplung mit der Lernkomponente, werden die gelernten Zusammenhänge sofort umgesetzt. Dadurch wird der gesamte Lernprozeß des Diagnosesystems erheblich beschleunigt. Die experimentelle Validierung ist im folgenden Kapitel enthalten.

Kapitel 7

Anwendung der Interaktiven Diagnose

Im Rahmen der vorliegenden Arbeit wurden vier Projekte mit Hilfe des *Interaktiven Lernenden Diagnose*-Konzepts realisiert. Dieses Kapitel diskutiert die Architektur, die gesammelten Erfahrungen sowie die erzielten Ergebnisse der drei Projekte, die sich mit der Pipelinediagnose beschäftigen. Diese Projekte wurden mit unterschiedlichen Anforderungen durchgeführt, so daß die Komplexität der einzelnen Komponenten der *Interaktiven Lernenden Diagnose* in den Projekten variiert.

Durch diese Applikationen werden insbesondere die in den vorherigen Kapiteln behandelten Komponenten der Vorverarbeitung, Lernkomponente und Interaktionskomponente genauer beispielhaft vorgestellt.

7.1 Ultraschallbasierte Diagnose von Pipelines mit *NeuroPipe I*

Das in Abschnitt 1.1 beschriebene Szenario der Pipelinediagnose spiegelt genau die Situation vor dem Einsatz des im Rahmen dieser Arbeit erstellten Diagnosesystems ***NeuroPipe I*** (siehe auch [SBGB93, Sun94b, SE94, SBG95, SB95, SB96b]) wieder.

Die Ausgangssituation für das ***NeuroPipe I***-Diagnosesystem bestand in der großen Menge an bereits durchgeführten und manuell ausgewerteten Läufen des UltraScan–Tools. Sie stellten neben der Vielfalt an Beispielsdaten auch den großen Erfahrungsschatz seitens der Auswerter sicher.

Die Zielsetzung des Diagnosesystems ist die Klassifikation der wichtigsten in der Pipeline vorkommenden Anomalien und ihrer Bewertung. Die gesammelten Ultraschallmeßdaten werden durch das ***NeuroPipe I***-System offline ausgewertet, und die wichtigsten Anomalien werden anschließend den Auswertern zur Validierung übergeben.

Im folgenden wird der Ablauf und die Architektur des Diagnosesystems ***NeuroPipe I*** näher erläutert.

Architektur des Diagnosesystems *NeuroPipe I*

Die Diagnose einer Pipeline erfolgt in vier Schritten, von denen die Rundnahtsuche und die Defektsuche mit dem ***NeuroPipe I***-System erfolgt:

Datenübersetzung: Die Bänder, Festplatten oder andere Speichermedien werden durch eine Datenkonvertierung in ein spezielles Format übersetzt. Dabei entstehen die sogenannten *.dat Dateien, die als Rohdaten für den weiteren Diagnoseprozess dienen.

Rundnahtsuche: Mit Hilfe von ***NeuroPipe I*** wird das sogenannte Rohrbuch erstellt, das die Positionen der Rundnähte angibt. Dies sind genau diejenigen Distanzen in der Pipeline, an denen jeweils zwei Rohre verschweißt sind. Die Position wird dabei mittels eines Odometerrades vermessen.

Defektsuche: Basierend auf dem Rohrbuch, wird eine rohrweise Diagnose der gesamten Pipeline durchgeführt. Die dabei entdeckten Defekte und Anomalien werden abgespeichert und für die nachfolgende Validierung aufbereitet.

Validierung: Ein Extrakt der gefundenen Defekte wird durch die Auswerter manuell überprüft und für den Pipelilinebetreiber in einem Report zusammengefaßt.

Im folgenden werden die einzelnen Module und Komponenten der ***NeuroPipe I*** Diagnosesystem-Architektur beschrieben. Die komplette modulare Struktur des Diagnosesystems ist in Abbildung 7.1 dargestellt.

Als erster Schritt beim Aufbau des Diagnosesystems mußten die Charakteristika der Meßdaten ermittelt werden (siehe Abschnitt 4.2), damit eine sinnvolle Formatierung (siehe Abschnitt 4.3) durchgeführt werden konnte. Es wurden vier zweidimensionale Felder identifiziert, um die Meßdaten aufzunehmen: der Sensorabstand, die Wanddicke, der Kompressionswert des Sensorabstandes und der Kompressionswert der Wanddicke. Darüber hinaus wurde die Odometerinformation und die Rotation des Molchs in jeweils ein eindimansionales Feld transformiert.

Diese Repräsentation wird durch die Vorverarbeitung der Meßdaten (siehe auch Abschnitt 4.4) derart modifiziert, daß ausschließlich definitiv unplausible Werte herausgefiltert werden, wie beispielsweise Meßsperrzeitfehler der Ultraschallmessung. Darüber hinaus wird kein Modellwissen angewendet, da jede einzelne Messung bei den späteren Betrachtungen ausschlaggebend sein könnte.

Im nächsten Schritt wird die Bestimmung der Meßdatenregionen durchgeführt, die sich ihrerseits in unterschiedliche Untermodule gliedert. Zunächst wird der genaue Verlauf der Nähte ermittelt. Dann werden sie "ausgeblendet". Dies wird deswegen durchgeführt, weil alle Schweißnähte das Ultraschallsignal zerstreuen und "falsche" Anomalien erzeugen. Würde man die Nähte durch die Anomaliesuche detektieren lassen, zerfielen diese in viele Teilbereiche, und es würden wichtige Informationen verloren gehen, ohne die es nicht mehr möglich ist, diese Regionen als Nähte zu detektieren. Aus diesem Grund werden die Nähte vor der Anomaliesuche "ausgeblendet".

Bei der Bestimmung der Meßdatenregionen werden zunächst die genauen Positionen der Rundnähte gesucht und gelöscht. Dies wird mit Hilfe eines adaptiven Verfahrens auf der Basis von Kohonennetzen durchgeführt (siehe [SE94]). Im nächsten Schritt werden alternativ die Positionen und Löschungen von entweder spiralgeschweißten oder längsgeschweißten Rohren durchgeführt. Die spezielle Problematik der Suche nach der Position der Längsnaht in einem Rohr wird im Abschnitt 7.2 behandelt.

Als letzter Schritt bei der Bestimmung der Meßdatenregionen wird die eigentliche Suche nach Anomalien ausgeführt. Die Suche wird nach dem im Abschnitt 4.5 vorgestelltem Verfahren durchgeführt. Dabei sind 12 Parameter für die Arbeitsweise des

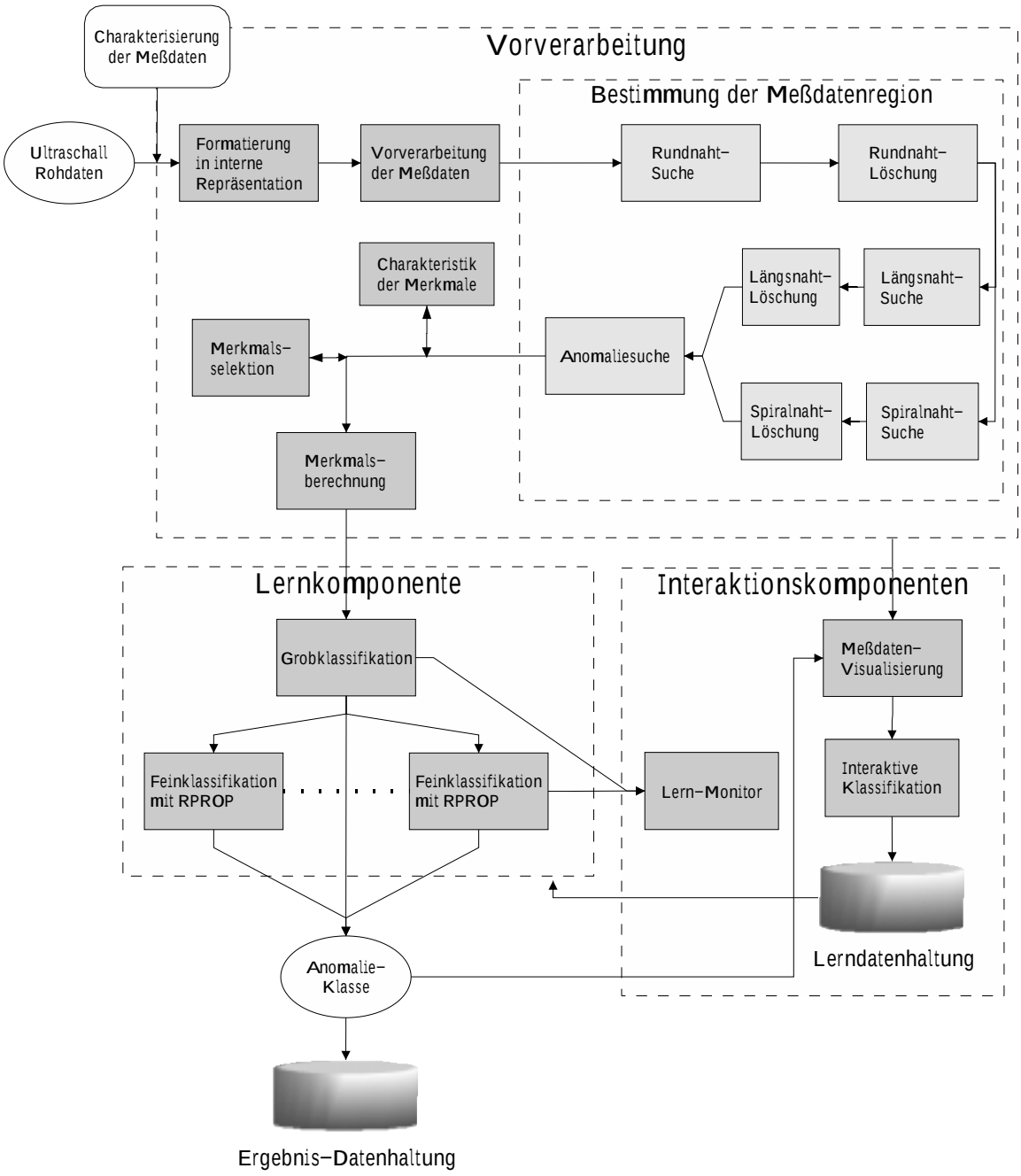


Abbildung 7.1: Module und Datenfluß des ***NeuroPipe I***-Systems.

Algorithmus verantwortlich. Sie sind in Abhängigkeit von der Datenqualität und den zu erzielenden Ergebnissen zu bestimmen. Gerade bei der Einstellung dieser Parameter ist ein tiefes Verständnis für die Arbeitsweise des Algorithmus und die physikalischen Gegebenheiten in der Pipeline notwendig.

Nach dem Auffinden der Anomalien wird aus der jeweiligen Region eine feste Anzahl von Merkmalen extrahiert. Auch hier floß das Wissen über die Applikation in die Auswahl der geeigneten Berechnungsalgorithmen für die Merkmale mit ein. Es wurden anfänglich 82 Merkmale definiert, die nach Überprüfung ihrer Relevanz, mittels des MIFS-Verfahrens, auf 41 reduziert werden konnten. Die Berechnung der Fraktalen

Dimension ergab einen Wert von 3.37 für die Informationsdimension ($q = 1$) (siehe Abschnitt 4.7.1). Daraus läßt sich schließen, daß aussagekräftigere Merkmale existieren, die die gleiche Aussagekraft besitzen. Leider gibt das Verfahren keinen Hinweis darauf, wie diese Merkmale aussehen.

Die Klassifikation der Merkmale der Anomalien wird durch die Lernkomponente vorgenommen. Dazu wurde die Klassifikation in zwei Stufen durchgeführt, die erste umfaßt die Grobklassifikation mittels eines Vorgängers des *Dynamic Bounds*-Netzes, die zweite die Feinklassifikation durch Verwendung der Backpropagation–Netze (RPROP-Netze). Bei diesem Vorgehen werden die durch die Grobklassifikation nicht erkannten Anomalien durch die Feinklassifikation zugeordnet. Für jede Klasse existiert genau ein RPROP-Netz, daß nur für seine Klasse verantwortlich ist. Die so gewonnene Defektklassifikation wird an eine Ergebnis-Datenhaltung weitergeleitet.

Neben dem Problem, die beste Topologie für die RPROP-Netze zu bestimmen, kristallisierte sich das Zusammenstellen der Trainingssätze als weiteres Problem heraus. Aus diesem Grund ist ein interaktives Klassifikationssystem entwickelt worden, daß dem Benutzer die gefundenen Anomalien präsentiert und ihm die Möglichkeit bietet, die Klassifikation des Systems direkt während der Diagnose zu kontrollieren und zu korrigieren.

Die in Abbildung 7.2 dargestellten Anomalien stellen unterschiedliche Arten von Defekten dar, die mit ***NeuroPipe I*** gefunden wurden. Das System präsentiert dem Benutzer interaktiv die Anomalien. Er kann ggf. die Klassifikation korrigieren und in die Lerndatenhaltung übernehmen.

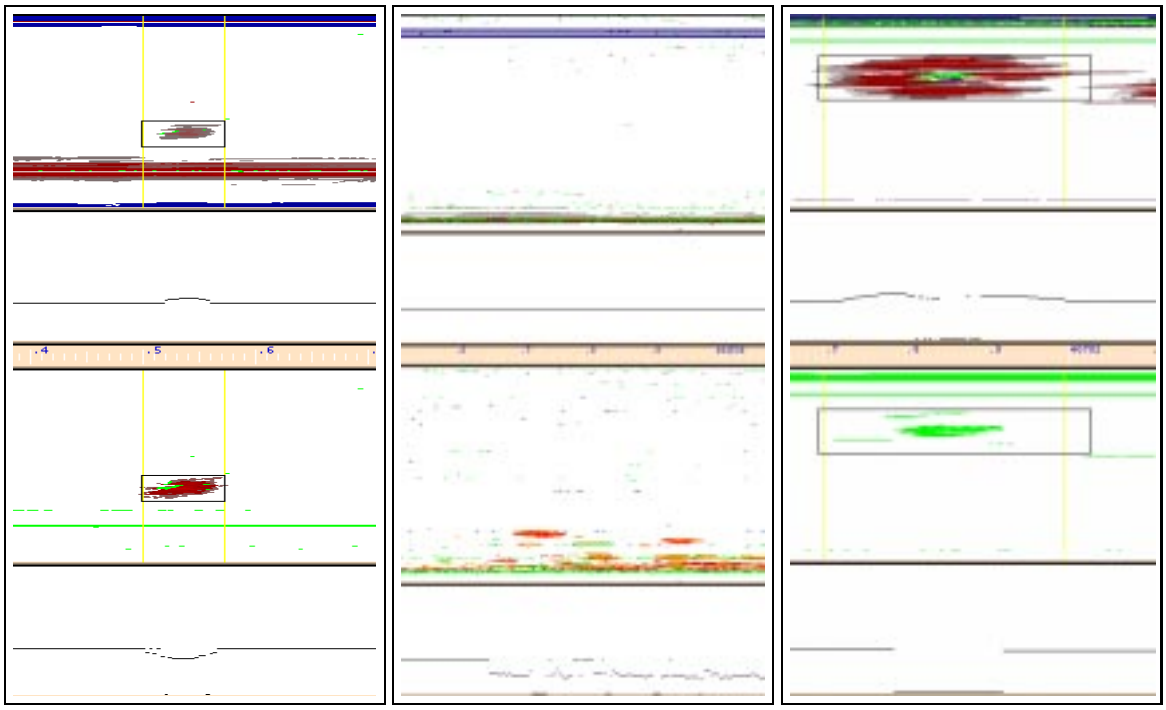


Abbildung 7.2: Verschiedene Anomalien, Defekte in der Pipelinewand in einer C-scan-(Falschfarben) -Darstellung. (a) interner Materialverlust, ist eine Korrosion an der Wandinnenseite; (b) Lamination, ist ein Herstellungsbedingter Einschluß bzw. Inhomogenität des Materials; (c) Beule, ist ein Ausbeulen der Rohrwand z.B. durch einen Stein.

Durch die Prozedur der Interaktion werden neue noch nicht bekannte bzw. falschklassifizierte Beispiele gesucht. Das Problem bei der Suche nach repräsentativen Beispielen besteht in der Tatsache, daß die Beispiele nach subjektiven Kriterien durch den Auswerter ausgesucht und bewertet werden. Das Auffinden neuer aussagekräftiger und geeigneter Beispiele erwies sich als sehr schwierig, da dem Auswerter nichts weiter zur Verfügung stand als die Angabe, welche Anomalien nicht korrekt klassifiziert werden konnten, und sich der Einfluß eines neu hinzugefügten Beispiels auf die Klassifikation des Systems erst nach einer erneuten Lernphase auswirkte. Die daraus resultierende Verzögerung betrug zu diesem Zeitpunkt etwa zwei Tage.

Ergebnisse der Diagnose mit *NeuroPipe I*

Das ***NeuroPipe I*** Diagnosesystem dient der vollautomatischen Erstellung von Defektlisten für komplette Molchläufe. Es wird z.Z. eingesetzt, um die Diagnose durchzuführen. Ein Extrakt der so erstellten Defektanalysen wird durch die Auswerter überprüft.

Der erster Einsatz von ***NeuroPipe I*** erfolgte zur Diagnose der Ultraschalldaten einer 1100km langen Pipeline mit 52Zoll Durchmesser. Die manuelle Auswertung dieser Datenflut (70 CD's mit 50zig-fach komprimierten Daten) hätte das gesamte Auswerteteam (ca. 30 Leute) ein Jahr in Anspruch genommen. Das ***NeuroPipe I***-System konnte die Auswertung im Parallelbetrieb auf 3 SUN Sparc-II Maschinen in drei Monaten abschließen.

Dabei wurden korrekte Ergebnisse von über 95% erzielt, wobei die restlichen 5% zum Teil durch das System selbst als zu überprüfenden Kandidaten ausgewiesen wurden, und der restliche Anteil zum Einem durch die Überprüfung der Ergebnislisten seitens der Auswerter identifiziert werden konnte (siehe [SBG95]), zum Anderen Defektkandidaten darstellten, die auch durch die Experten nicht zugeordnet werden konnten.

Phase	Aufwand in %
Entwicklungsumgebung	30
Anomaliesuche	20
Vorverarbeitung	10
Klassifikation	10
Beispielsuche	30

Tabelle 7.1: Prozentueller Arbeitsaufwand für die einzelnen Teile des Diagnosesystems ***NeuroPipe I***.

In Tabelle 7.1 ist der Aufwand für die einzelnen Phasen bei der Erstellung des Diagnosesystems dargestellt. Die Zahlen belegen, daß der größte Aufwand bei der Erstellung der Entwicklungsumgebung und der Wissensakquisition lag. Es hat sich herausgestellt, daß die zur Verfügung gestellten Defektlisten für das Einlernen des Diagnosesystems nicht benutzt werden konnten, da die Position eines Defektes nicht eindeutig genug ermittelbar war, bzw. die Listen unvollständig waren, da sie nur die tatsächlichen Fehler enthielten, aber die Negativbeispiele (kein Defekt) fehlten. Von den gefundenen Anomalien sind allerdings nur etwa 30% tatsächliche Defekte, die restlichen Anomalien sind auf das Rauschen in den Meßdaten zurückzuführen.

Im Laufe der fast einjährigen Sammlung von Beispielen, wurden etwa 2400 Anomalien klassifiziert. Aus diesen waren wegen unterschiedlicher Gründe nur etwa die Hälfte tatsächlich für das Einlernen des Klassifikators von Nutzen (aus dieser Quelle stammt der *defect1* Datensatz).

Es hat sich herausgestellt, daß sich eine gute Interaktion ohne die sofortige Adaption des Diagnosesystems auf die neu klassifizierten Beispiele den Wissensakquisitionprozeß stark in die Länge zieht. Darüber hinaus wurde die Notwendigkeit einer strukturierten Datenhaltung offensichtlich, da der Überblick über die unterschiedlichen Versuchsreihen und Konfigurationen nur sehr schwierig aufrechtzuerhalten war.

7.2 Interaktive Diagnose von Längsnähten

Eine Teilaufgabe des ***NeuroPipe I***-Projekts befaßte sich mit der Diagnose von Längsnähten. Das Ziel bestand darin, ein Neuronales Netz dahingehend einzulernen, die Längsnähte auch in sehr stark verrauschten Rohdaten wiederzufinden. Dies ist ein sehr wichtiger Schritt bei der Diagnose der Ultraschalldaten, da die Schweißnähte das Ultraschallsignal stark zerstreuen und dadurch anomalien-/defektähnliche Regionen bilden, die durch den Klassifikationsprozess fälschlicherweise als Defekt klassifiziert werden könnten.

Aus diesem Grund wird die Position der Längsnaht gesucht und anschließend aus der Meßwertrepräsentation gelöscht. Eine komplette Darstellung eines Rohrabschnitts mit Längsnaht ist in Abbildung 7.3 zu sehen.

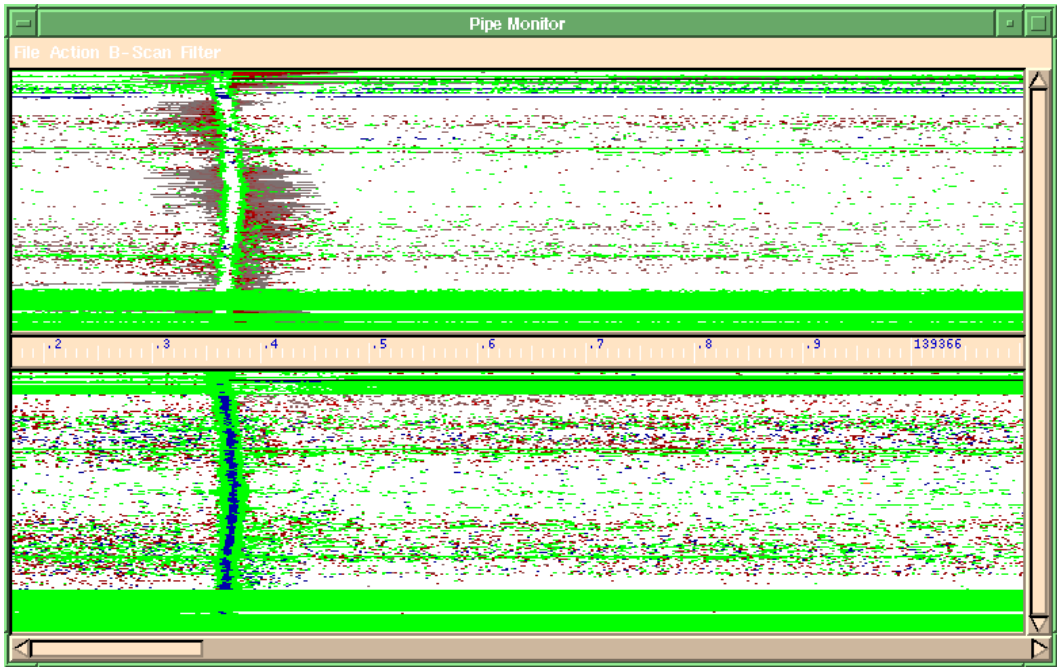


Abbildung 7.3: Falschfarbendarstellung eines Rohrabschnittes aus dem ***NeuroPipe I*** Projekt. Die Längsnaht (Schweißnaht) präsentiert sich im oberen Bereich beider 2D-Bilder als ein mehr oder weniger homogener Waagerechter Balken.

Der spezielle Aufbau der an der Detektion der Längsnaht beteiligten Module ist in Abbildung 7.4 dargestellt.

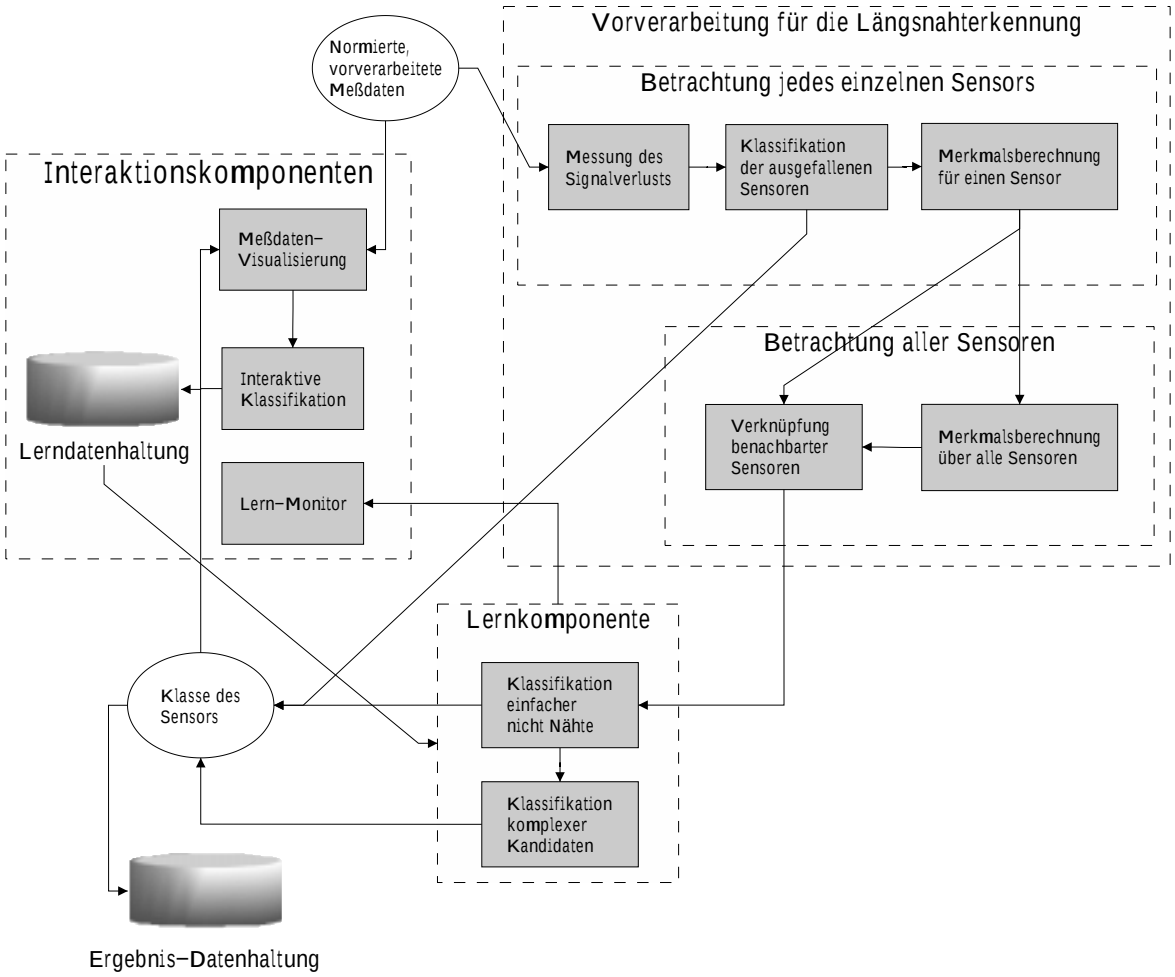


Abbildung 7.4: Module und Datenfluß der Längsnahtsuche im ***NeuroPipe I***-Systems.

Für die Klassifikation wurden insgesamt 75 Merkmale pro Sensor[1] berechnet. Die Lernkomponente erstellt für die aus jeden Sensor stammenden Sensorwerte eine separate Klassifikation, ob diese spezielle Umfangsposition (Sensor) als zur Längsnaht zugehörig betrachtete werden soll, oder zu dem "normalen" Signalmeßwerten gehört.

Um die Akquisition der Längsnahtpositionen zu beschleunigen, wurde eine spezielle Interaktionsvisualisierung entwickelt. Sie stellt die Klassifikationen der Lernkomponente unmittelbar graphisch dar, und bietet dem Benutzer durch Selektion die Möglichkeit, diese zu korrigieren. Die Klassifikation der Position der Längsnaht kann aus Abbildung 7.5 entnommen werden, die bereits die klassifizierte Position der Naht enthält.

Durch diese Art der Darstellung und unter Verwendung der Selektion als direkte Zuweisung der Klassenzugehörigkeit konnte die Geschwindigkeit der Interaktion wesentlich gesteigert werden. Gleichzeitig wurden sehr kurze Zyklen für das Aufstellen neuer Klassifikatoren mit den neu gesammelten Beispielsdaten durchgeführt. Durch diese Maßnahmen konnte die Akquisition der Beispiele auf nur eine Woche reduziert werden, wobei in dieser Zeit 9880 Beispiele gesammelt wurden.

Die aus diesem Projekt stammenden Beispielsdaten sind in dem *lweld*-Datensatz

[1] Die Sensoren entsprechen in der C-Scan Darstellung den waagerechten Linien.

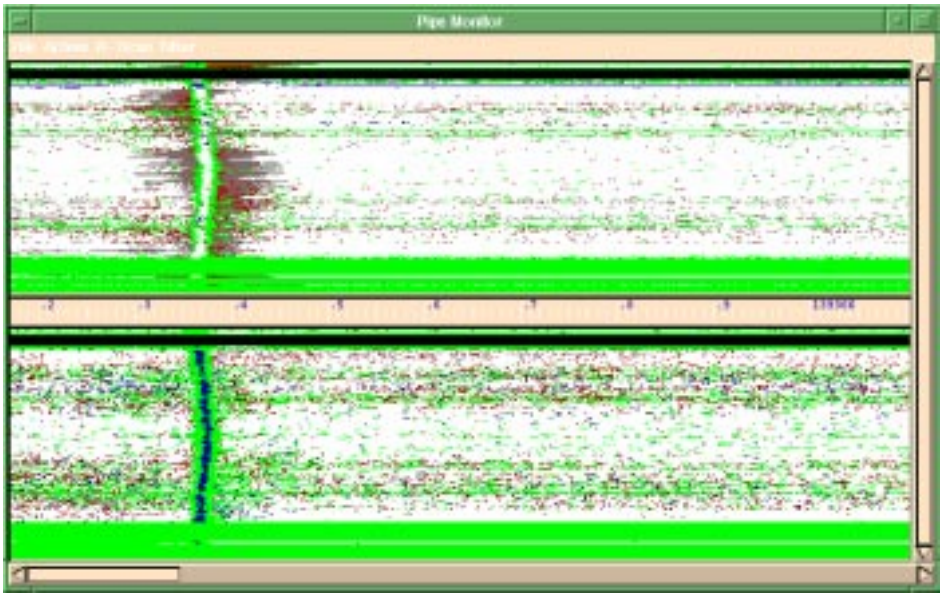


Abbildung 7.5: Klassifizierung der Längsnähte im ***NeuroPipe I***-Projekt. Der schwarze waagerechte Balken bezeichnet dabei die getätigte Klassifikation und/oder die Korrektur durch den Benutzer.

zusammengefaßt (siehe Abschnitt E.2).

7.3 Magnetstreuflußbasierte Diagnose von Pipelines mit *NeuroPipe II*

NeuroPipe II stellt eine Weiterentwicklung von ***NeuroPipe I*** dar. Das Diagnosesystem fokusiert nicht nur auf einen speziellen Molchtyp, sondern ist als eine Toolbox zur Diagnose mit unterschiedlichen Molchtypen und Meßverfahren in der Pipelineinspektionen konzipiert. Als Basis für diese Diagnosesystemtoolbox dient der ***CMS***, der aller verwendeten Module zu einem Datenflußgraph zusammengefügt. Die interaktive Schnittstelle ist mit Hilfe des ***Iface***-Systems aufgebaut.

Das ***NeuroPipe II***-System ist im ersten Schritt für die Diagnose von magnetstreuflußbasierten Meßdaten aufgebaut worden.

Das eingesetzte Magnetstreuflußverfahren bedient sich eines starken Permanentmagneten, der ein Feld aufbaut, dessen Feldlinien in Fahrtrichtung des Molchs orientiert sind. Bei einem Defekt in der Wand (Verminderung der Wanddicke durch Korrosion), werden die Feldlinien aus dem Metall "gedrückt". Die Änderung des Magnetfeldes wird mittels Hall-Sensoren gemessen und abgespeichert. Leider ist dies ein indirektes Meßverfahren, was bedeutet, daß es nicht allgemeingültig möglich ist, anhand der aufgezeichneten Signale auf die Kontur des Defektes zu schließen. Es ist nur eine Schätzung der Ausdehnung möglich.

Die unterschiedliche Charakteristik der Magnetstreuflußdaten gegenüber den Ultraschalldaten fordert auch ein differenziertes Vorgehen bei der Diagnose. Der Datenfluß und die beteiligten Module an der magnetstreuflußbasierten Diagnose können der Abbildung 7.6 entnommen werden.

Die grobe Struktur der Diagnose ist mit der bei den Ultraschalldaten identisch. Die

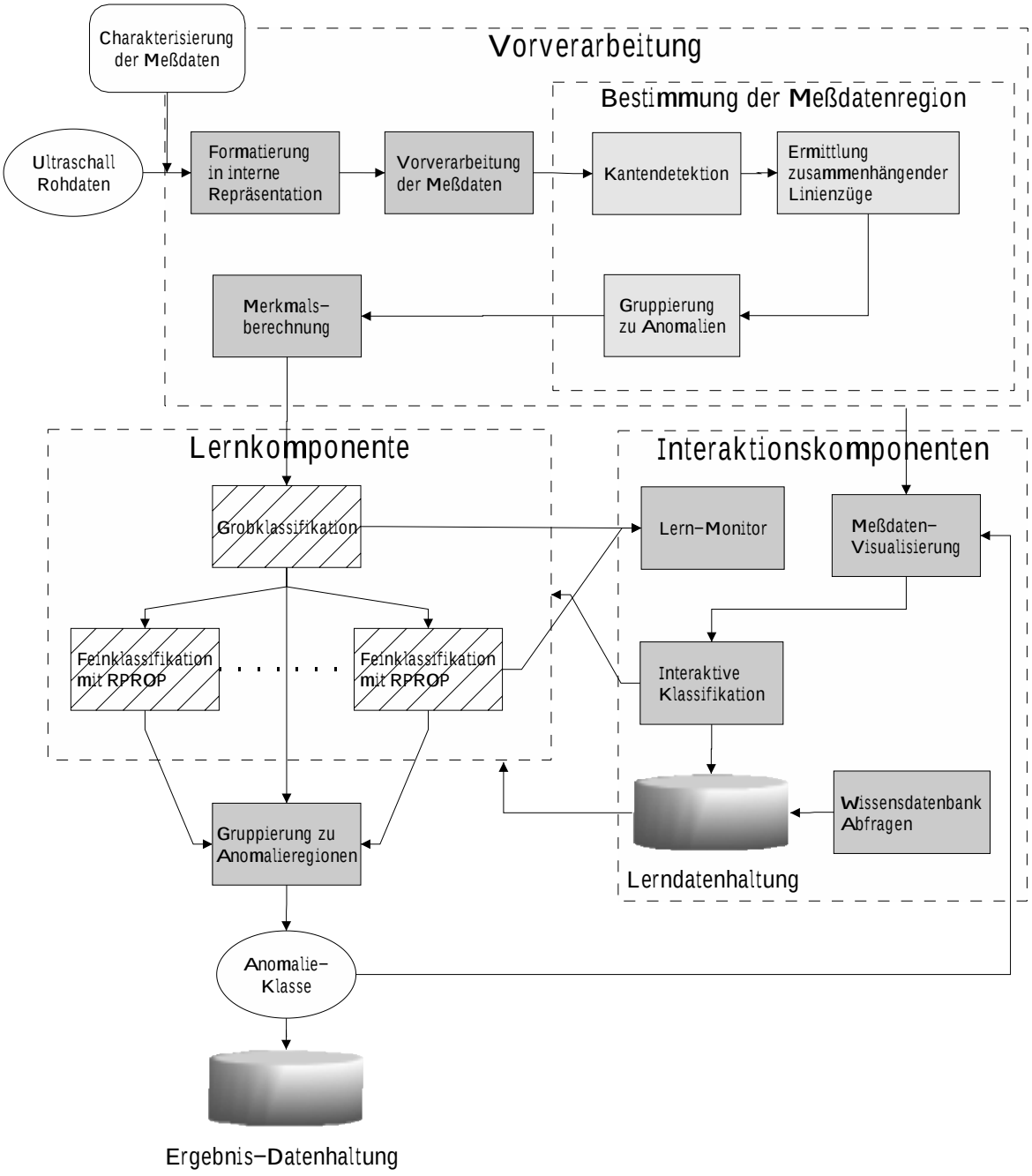


Abbildung 7.6: Module und Datenfluß des ***NeuroPipe II***-Systems für die magnetstreuflußbasierten Meßdaten.

Unterschiede treten bei der Arbeitsweise der einzelnen Module auf.

Die Vorverarbeitung der Meßdaten besteht aus verschiedenen Substitutions-, Differenzierungs- und Duplizierungsoperatoren. Dem schließt sich die Detektion von Kanten mittels Extremabestimmung in der Differenzdarstellung. Diese kanten werden zu Linienzügen gleichen Typs (Maximum, Minimum) zusammengeschlossen. Durch einen endlichen Automaten werden die umschlossenen Flächen identifiziert und zu Anomalien gruppiert. Basierend auf diesen Flächen werden unterschiedliche Merkmale ermittelt. Anschließend wird die Klassifikation mittels der Lernkomponente durchgeführt (ist zum

jetzigen Zeitpunkt noch nicht komplett realisiert). Als letzten Schritt in der Diagnose werden dann die klassifizierten Anomalien anhand der Klassifikation zu Anomalieregion gruppiert, die dann die endgültige Defektregion bildet.

In Abbildung 7.7 sind die Anomalien dargestellt, wie sie dem Benutzer präsentiert werden. Er kann auf einen Blick überprüfen, ob das System korrekt gearbeitet hat. Es ist auch möglich, nur bestimmte Typen von Anomalien darzustellen, so z.B. die unsicheren Kandidaten. Die korrigierte Klassifikation kann sofort in die CMS-Datenbank übernommen werden.

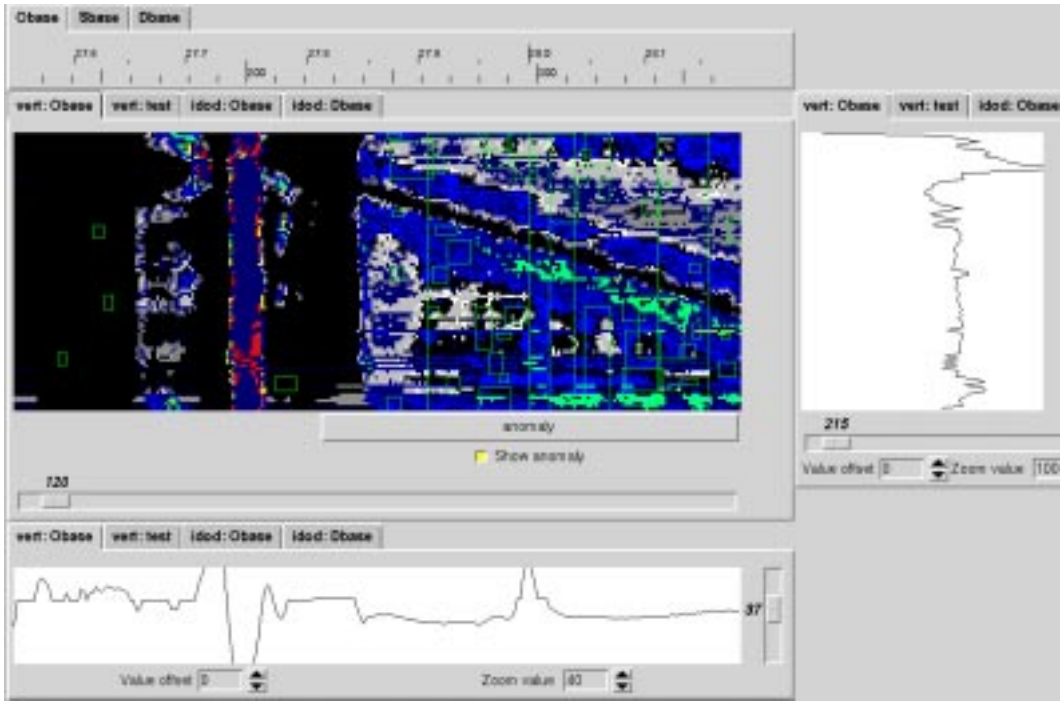


Abbildung 7.7: C-Scan Darstellung der Magnetstreufluß-Daten einer Pipeline Inspektion.

Auch wenn sich die Lernkomponente noch nicht im Echteinsatz befindet, konnte durch das automatische Auffinden der Anomalieregionen, der Berechnung einiger wichtiger Kennzahlen und der dazugehörigen Visualisierung der Regionen die Geschwindigkeit der manuellen Auswertung verdoppelt werden, womit der Vorteil der visuellen Interaktion in einem Diagnosesystem bestätigt wird.

7.4 Resümee

Allen Diagnosesystemen ist es gemeinsam, daß ein großer Arbeitsaufwand mit der Programmierung der Merkmale verbunden ist. Dieser konnte nur im Fall der Längsnaht-Diagnose verringert werden, da viele Merkmale von der Anomaliesuche bei ***NeuroPipe I*** übernommen oder leicht modifiziert werden konnten (vgl. Abschnitt 4.9).

Leider muß festgestellt werden, daß diese Vorgehensweise nicht ohne weiteres verallgemeinbar ist. Aus diesem Grund wurde ***NeuroPipe II*** nach dem Toolbox Prinzip konzipiert, damit schneller neue Algorithmen der Vorverarbeitung integriert werden

können. Es ist davon auszugehen, daß sich die Algorithmen in einem Anwendungsbereich (Pipelinediagnose) unter den verschiedenen Molchtypen zum großen Teil austauschen lassen.

Das nächste zu lösende Problem bestand in der sinnvollen Wahl der Merkmale. Dieses Problem stellt sich insbesondere, wenn nicht genügend Beispiele zum Lernen vorhanden sind. Im ***NeuroPipe I***-Projekt wurde das MIFS-Verfahren mit Erfolg zur Reduktion der Anzahl der Merkmale eingesetzt, indem die Anzahl von 82 auf 41 reduziert werden konnte. In diesem Zusammenhang ist anzumerken, daß die Fraktale-Dimension nach dem Entfernen vom Merkmalen als Indikator für große Änderungen in der Aussagenkraft der Daten dienen kann (vgl. Abschnitt 4.7.1).

Bereits im ***NeuroPipe I***-Projekt wurde der *Hybride Knoten* in einer frühen Form eingesetzt. Es wurde eine Vorklassifikation mit einer früheren Version des *Dynamic Bounds* Netzes durchgeführt. Die weitergehende Klassifikation übernahmen Backpropagation–Netze. Dabei war jedes Netz für eine Klasse zuständig. Die mit dieser Vorgehensweise erzielten Ergebnisse führten zur Weiterentwicklung des *Dynamic Bounds* und des *Hybriden Knotens* zu der im Kapitel 5 vorgestellten endgültigen Form. Beide Konzepte wurden wesentlich weiterentwickelt. Konkrete Testergebnisse zu diesen Ansätzen befinden sich in Abschnitt 5.3.6 und 5.4.4.

Allen hier vorgestellten Diagnosesystemen ist die interaktive Wissensakquisition gemeinsam. Das Problem solcher interaktiver Schnittstellen ist der hohe Erstellungsaufwand. Motiviert durch die Erfahrungen der ***NeuroPipe I*** Applikation wurde im Rahmen dieser Arbeit das ***Iface***-Konzept entwickelt, das in ***NeuroPipe II***, ***CMS*** und der ***CMS***-Datenbank für die Benutzerinteraktion benutzt wird. Für die Wissensakquisition wird dabei vor allem das Konzept der multivariaten Daten und deren Darstellung genutzt. Mit Hilfe von ***Iface*** können Applikationen wesentlich schneller aufgebaut werden und auch flexibel durch Visualisierungs- und Wissenakquisitionskomponenten mit dem Benutzer kommunizieren.

Die Notwendigkeit adäquater Schnittstellen kann insbesondere an der schnellen Akquisition von Beispielen im Falle der Längsnaht-Diagnose nachvollzogen werden, wo innerhalb von nur einer Woche 10.000 Beispiele gesammelt werden konnten. Darüber hinaus hat es sich als notwendig herausgestellt, eine benutzergeführte Filterung der klassifizierten Daten durchzuführen, weil die Anwender oft nur an bestimmten Vorkommnissen interessiert sind.

Als sehr hilfreich hat sich bei den zeitintensiven ***NeuroPipe***-Applikationen das offline-Sammeln von Beispielen herausgestellt. Durch diese Vorgehensweise wird dem Anwender viel Zeit erspart, indem die *Standardfehler* nicht mehr visualisiert werden, sondern nur genau diejenigen, die aus irgendwelchen Gründen der manuellen Überprüfung bedürfen.

Insbesondere bei dem offline-Sammeln von Beispielen werden die Datenmengen unübersichtlich, und die Lernergebnisse können nur mit großer Sorgfalt den korrekten Lerndaten zugeordnet werden. Die diesbezüglichen Erfahrungen aus ***NeuroPipe I*** führten zu der Erstellung der ***CMS***-Datenbank. Sie bietet die Möglichkeit verschiedene verteilte Experimente (mittels des *JobManager-Systems*) durchzuführen und zu vergleichen. Durch die Standardisierung der Wissensverwaltung kann eine Reihe von Auswertungen durch die Monitorkomponente durchgeführt werden. Darüber hinaus hat es sich bei ***NeuroPipe I*** als sehr umständlich herauskristallisiert, die Güte der Klassifikation korrekt zu bewerten. Aus diesem Grund wurden in den ***CMS*** und die ***CMS***-Datenbank Methoden für die Bewertung integriert.

Im folgenden Kapitel werden die Untersuchungsergebnisse abschließend zusammengefaßt, und ein Ausblick auf mögliche weiterführende Arbeiten diskutiert.

Kapitel 8

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde die Diagnose von stark verrauschten subsymbolischen Massendaten aus der Pipelineinspektion untersucht. Da für diese Anwendung so gut wie keine Regeln existieren, wurde ein lernender Ansatz für das Diagnosesystem verfolgt. Das Hauptaugenmerk lag dabei auf der direkten Einbeziehung des Benutzers in den Diagnoseprozess, um eine möglichst schnelle Adaption des Diagnosesystems zu erzielen.

Zu diesem Zweck wurde die Architektur des *Interaktiv Lernenden Diagnosesystems (ILD)* entwickelt, die aus folgenden Komponenten besteht: Vorverarbeitung, Lernkomponente, Wissensverwaltung und Interaktionskomponenten. Dabei wurden unterschiedliche von den Komponenten zu erfüllende Kriterien festgelegt. Insbesondere wurde dabei Wert auf die schnelle Adaption des Diagnosesystems bzw. der Lernkomponente gelegt. Zu diesem Zweck wurde ein zweiphasiges Lernmodell, bestehend aus einer Interaktionsphase und einer Optimierungsphase entwickelt.

Im folgenden werden die in dieser Arbeit erzielten Ergebnisse der einzelnen Komponenten zusammengefaßt:

Vorverarbeitung: Es wurde ein mehrstufiger Prozeß für die Vorverarbeitung der Meßdaten vorgestellt. Er umfaßt die systematische Erfassung der Charakteristik der Meßdaten der Applikation, die daraus resultierende Konvertierung der Daten in die interne Repräsentation, die Vorverarbeitung der Meßdaten durch unterschiedliche Algorithmen, die parametergesteuerte Suche nach Anomalieregionen und der Merkmalsextraktion aus den detektierten Regionen.

Darüber hinaus wurden verschiedene Verfahren zur Merkmalsselektion untersucht. Das MIFS-Verfahren wurde ausgewählt, um damit den Merkmalsraum möglichst niedrigdimensional zu halten. Des weiteren konnte mit Hilfe der Fraktalen Dimensionen die minimale Anzahl notwendiger Merkmale ermittelt werden. Die dabei erzielten Ergebnisse erwiesen sich als hilfreich für die Beurteilung der Merkmalsqualität.

Lernkomponente: Um ein schnelles Einlernen sowie eine gute Generalisierung der Lernkomponente zu erreichen, wurde ein zweistufiger hierarchischer Ansatz entwickelt, der einerseits aus schnell adaptierendem Lernverfahren zur Grobklassifikation und andererseits aus kompakten Netzen zur Feinklassifikation besteht.

Um den Benutzer die Anwendung der *Interaktiven Subsymbolischen Diagnosesystems* zu vereinfachen, wurde ein neuer, speziell für die Interaktionsphase mit dem Benutzer entwickelter Lernalgorithmus (*Dynamic Bounds*) entworfen. Dieser Algorithmus benötigt dabei keine vom Benutzer einzustellenden Parameter

und ist somit ideal für die Benutzung durch fachfremdes Personal. Der *Dynamic Bounds* wird im Einzelschrittlernen in der Interaktionsphase angewendet, um eine optimale Geschwindigkeit bei der Adaption neuer Beispiele zu erzielen, und im Epochenlernen werden nicht genutzte bzw. überflüssige Neuronen automatisch aus dem Modell entfernt. Dabei ist keinerlei Parametereinstellung notwendig. Das Verfahren wurde anhand von vierzig Datensätzen aus unterschiedlichen Applikationen erfolgreich getestet.

Der neu entwickelte *Dynamic Bounds*-Algorithmus erzielte sehr gute Ergebnisse bei der Grobklassifikation und benötigte dafür nur eine geringe Anzahl von Neuronen.

Basierend auf dem *Dynamic Bounds*-Verfahren als Grobklassifikator wurde das Konzept des *Hybriden Knotens* untersucht. Das Konzept sieht es vor für die Feinklassifikation RPROP-Netze einzusetzen. Durch eine strukturierte Vorgehensweise bei der Wahl der Anzahl der beteiligten Netze ist es mit dem Ansatz des Hybriden Knotens möglich, flexibel seine Struktur an die Komplexität der zugrundeliegender Klassifikationsaufgabe anzupassen.

Interaktionskomponente: Mit Hilfe der Interaktion wurde eine Möglichkeit geschaffen, die den Benutzer in die Lage versetzt, den Lernvorgang schnell und bequem zu beeinflussen. Aufgrund der Tatsache, daß für System und Benutzer die gleiche Sicht auf die Daten zur Verfügung steht und der Anwender neue Datenvisualisierungen konfigurieren kann, wird ein schnelleres Verständnis für die Arbeitsweise des Diagnosesystems ermöglicht.

Durch die selektierbaren Anomalieregionen und die Möglichkeit der Änderung der zugehörigen Klassifikation, wurde eine sofortige Adaption des so eingegebenen Wissens ermöglicht.

Darüber hinaus wurden dem Benutzer verschiedene Funktionalitäten des Systems, durch unterschiedliche Monitore zugänglich gemacht. Dadurch wird der Benutzer in die Lage versetzt, unterschiedliche Abläufe des Diagnosesystems zu überwachen. Insbesondere kann die Klassifikationsgüte der Lernkomponente abgefragt werden, was essentiell für die Bewertung der Qualität des Diagnosesystems ist.

Zum Abschluß wurde das Konzept der *Interaktiven Lernenden Diagnose* anhand dreier unterschiedlicher Applikationen aus dem Gebiet der Pipelineinspektion konkret überprüft. Dabei stellte sich heraus, daß die vorgestellten Konzepte durchweg zu einer Beschleunigung des Erstellungsprozesses des Diagnosesystems führen.

Die beschriebenen Komponenten wurden im Rahmen dieser Arbeit durch die ***CMS***-, ***CMS***-Datenbank- und ***Iface***-Toolboxen appliakitonsunabhängig realisiert.

Weiterführende Arbeiten

Das größte Problem stellt zur Zeit die Aufstellung der Vorverarbeitung dar, da sie viele applikationsspezifische Vorgehensweisen in Form von Algorithmen implementiert. Es wäre lohnend zu untersuchen, ob es möglich ist, eine strukturierte, vielleicht sogar halbautomatische Vorgehensweise für den Aufbau der Vorverarbeitung aufzustellen. Da

es dabei einen großen Parameterraum zu untersuchen gilt, wäre es auch an dieser Stelle sinnvoll, das lernbasierte Paradigma anzuwenden.

Um eine noch bessere Bewertung der Merkmale zu gewährleisten, könnten diese mit informationstheoretischen Methoden untersucht werden, die über die in dieser Arbeit vorgestellten hinausgehen. Insbesondere wäre es wünschenswert, eine Bewertung zu erhalten ohne die Einbeziehung von benutzerdefinierten Parametern.

Für die Verbesserung des *Hybriden Knotens* wäre es von Vorteil, wenn eine parameterlose Methode für die Bestimmung der Randregionen im *Dynamic Bounds* gefunden werden könnte. Des Weiteren wären auch andere Lernverfahren für die Realisierung des *Hybriden Knotens* zu untersuchen.

Darüber hinaus wäre es sinnvoll, die im Rahmen dieser Arbeit entwickelten Tools (***CMS***, ***CMS***-Datenbank, ***Iface***) noch zu erweitern, um so die Erstellung beliebiger Diagnosesysteme mit einem möglichst geringen Aufwand zu erreichen.

Literaturverzeichnis

[AK98] Auda, G. und M. Kamel: *Modular Neural Network Classifiers: A Comapative Study.* Journal of Intelligent and Robotic Systems, (21):117–129, 1998.

[Alb96] Albers, Lutz: *Untersuchung generischer interaktiver graphischer Oberflächen.* Diplomarbeit, Forschungszentrum Informatik an der Universität Karlsruhe, 1996.

[And77] Anderson, J.R.: *Acquisistion of cognitive skill.* Psychological Review, 89:369–406, 1977.

[Bat94] Battiti, Roberto: *Using Mutual Information for Selecting Features in Supervised Neural Net Learning.* IEEE Transactions on neural networks, 5(4):537–550, july 1994.

[Ber92] Berlage, Thomas: *Using Taps to Separate the User Interface from the Application Code.* Konferenzband *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, Toolkits, Seiten 191–198, 1992.

[Ber94] Berns, K.: *Steuerungsansätze auf der Basis Neuronaler Netze für sechsbeinige Laufmaschinen.* Infix-Verlag, 1994.

[Ber97] Berthold, M. R.: *Konstruktieves Training von Probabilistischen Neuralen Netzen für die Musterklassifikation.* Doktorarbeit, Universität Karlsruhe, 1997.

[BF94] Berthold, M. und F. Feldbusch: *Ein Trainingsverfahren für Radial Basis Function Netzwerke mit dynamischer Selektion der Zentren und Adaption der Radii.* Konferenzband *4th Dortmund Fuzzy Days.* Springer Verlag, 1994.

[BG96] Bollacker, K.D. und J. Gosh: *Linear feature extractors based on mutual information.* Konferenzband *International Conference on Pattern Recognition*, Nummer 13, Seiten 720–724, August 1996.

[Bis95] Bishop, C. M.: *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[Bra97] Braun, H.: *Neuronale Netze - Optimierung durch Lernen und Evolution.* Springer, 1997.

[CBE+97] Cordes, St., K. Berns, M. Eberl, W. Ilg und R. Suna: *Autonomous sewer Inspection with a wheeled, multiarticulated robot.* Robotics and Autonomous Systems, 21:123–135, 1997.

[Der87] Dersin, P.: *Process Diagnostics beyond Real–Time: The Detecive Expert System.* Konferenzband *Annual Control Engineering Conference*, Band 6, May 19–21 1987.

[DH73] Duda, R.O. und P.E. Hart: *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[DO96] Deco, G. und D. Obradovic: *An Information-Theoretc Approach to Neural Computing.* Perspectives in Neural Computing. Springer–Verlag, Heidelberg, Berlin, New York, 1996.

[Dra95] Drakos, Nikos: *The LaTeX2HTML Translator.* Bericht, Computer Based Learning Unit, University of Leeds, May 2 1995.

[DS79] Dasarathy, B.V. und B.V. Sheela: *A Composite Classifier System Design: Concepts and Methodology.* Konferenzband Dasarathy, Belur V. (Herausgeber): *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, Seiten 108–113. IEEE Computer society Press, 1979.

[EC96] Etemad, K. und R. Chellappa: *A Neural Network Based Edge Detector.* Konferenzband Simpson, P.K. (Herausgeber): *Neural Networks Theory, Technology, and Applications*, IEEE Technology update series, Seiten 603–608. IEEE Technical Activities Board, 1996.

[Ede93] Edelmann, W.: *Lernpsychologie.* Psychologie-Verlags-Union, 1993.

[Eis88] Eisenblätter88, D.: *Ein informationstheoretischer Ansatz der klassischen Diskriminanzanalyse.* Doktorarbeit, Bergisch Gladbach; Köln, 1988.

[ET93] Efron, Bradley und Robert J. Tibshirani: *An Introduction to the Bootstrap.* Nummer 57 im Buch *Monographs on Statistics and Applied Probability.* Chapman & Hall, 1993.

[Fah88] Fahlman, S. E.: *Faster-Learning Variations on Back-Propagation: An Empirical Study.* Morgan Kaufmann, 1988.

[Fah90] Fahlman, Scott E.: *The Cascade–Correlation Learning Architecture.* Konferenzband *Advances in Neural Information Processing Systems 3*, Nummer 3. MK, 1990.

[FL90] Fahlman, S. E. und C. Lebiere: *The Cascade-Correlation Learning Architecture.* Report CMU-CS-90-100, Carnegie Mellon University, 1990.

[FPB96] F. Puppe, U.Gappa, K.Poeck und S. Bamberger: *Wissensbasierte Diagnose- und Informationssysteme.* Springer–Verlag, Heidelberg, Berlin, New York, 1996.

[FWH97a] Fochem, M., P. Wischnewski und R. Hofmeier: *Neuronale Netze zur akustischen Qualitätskontrolle rotierender Teile am Beispiel von Kassettenlaufwerken.* VDI Berichte, Seiten 309–316, 1997.

[FWH97b] Fochem, M., P. Wischnewski und R. Hofmeier: *Online System zur Qualitätskontrolle bei der Produktion von Kassettenlaufwerken.* Industrie Management, 6(9):30–31, 1997.

[Gar98] Garg, Hari Krishna: *Digital signal processing algorithms.* computer engineering series. CRC Press, 1998.

[Giz94] Gizzi, J.: *Verschiedene selbstorganisierende Karten und Visualisierungen.* Studienarbeit, FZI, Juli 1994.

[Giz96] Gizzi, J.: *Untersuchung hybrider Klassifikationskonzepte mit Neuronalen Netzen.* Diplomarbeit, Forschungszentrum Informatik, jan 1996.

[Gmb] GmbH, ZN: *ZN-Face.* URL: http://www.zn.ruhr-uni-bochum.de/DE/PROD/FACE.HTM.

[GMW90] Gill, P. E., W. Murray und M.H. Wright: *Practical Optimization.* Harcourt Brace and Company, London, 1990.

[Gör95] Görz, Günter: *Einführung in die künstliche Intelligenz.* Addison–Wesley Publishing Company, 2 Ausgabe, 1995.

[Gra83] Grassberger, P.: *Generalized dimensions of strange attractors.* Physics Letters, 97A(6):227–230, 1983.

[Har98] Harter, M.: *Intergration von Fuzzy-Control Wissensbasen in eine adaptive Steuerungsarchitektur auf Basis Neuronaler Netze.* Diplomarbeit, Forschungszentrum Informatik an der Universität Karlsruhe, 1998.

[Hei94] Heimannsfeld, Klaus: *Neuronale Netze zur Phasenkorrektur von 1D–NMR–Spektren.* Diplomarbeit, Forschungszentrum Informatik an der Universität Karlsruhe, jun 1994.

[HG97] Hormel, M. und S. Gehlen: *Akustische Qualitätsprüfung mit neuronalen Netzen.* VDI Berichte, 1997.

[HKP91] Hertz, J., A. Krogh und R. G. Palmer: *Introduction to the Theory of Neural Computation.* Addison–Wesley Publishing Company, 1991.

[H.N83] H.Niemann: *Klassifikation von Mustern.* Springer–Verlag, Heidelberg, Berlin, New York, 1983.

[IEC+96] Ilg, W., M. Eberl, St. Cordes, K.Berns und R.Suna: *Ein vielsegmentiger Roboter zur autonomen Inspektion von Abwasserkanälen.* Konferenzband *Fachgespräch Autonome Mobile Systeme (AMS)*, München, 1996.

[Jäh97] Jähne, Bernd: *DIgitale Bildverarbeitung.* Springer–Verlag, Heidelberg, Berlin, New York, 4 Ausgabe, 1997.

[Jös93] Jöst, K.: *Beiträge zum Lebenszyklus objektorientierter Systeme.* Konferenzband Lewerentz, C. (Herausgeber): *Objektorintierte Software-Entwicklung*, FZI-Publikation, Seiten 169–184. Forschungszentrum Informatik an der Universität Karlsruhe, 1993.

[JS96] Jafar-Shaghaghi, M. A. F.: *Maschinelles Lernen, Neuronale Netze und Statistische Lernverfahren zur Klassifikation und Prognose.* Doktorarbeit, Universität Karlsruhe, 1996.

[KL90] Karbach, W. und M. Linster: *Wissensaquisition für Expertensysteme: Techniken, Modelle und Softwarewerkzeuge.* Hanser, 1990.

[Kop99] Kopp, B.C.: *Untersuchung von dynamischen Meta-Lern-Methoden bei Klassifikationsproblemen mit Neuronalen Netzen.* Diplomarbeit, Fernuniversität Haagen, Februar 1999.

[KS97] Köppen-Selinger, B.: *Fehlerdiagnose mit küstlichen neuronalen Netzen.* Doktorarbeit, GH Duisburg, 1997.

[Leo96] Leonhardt, S.: *Modellgestützte Fehlererkennung mit Neuronalen Netzen – Überwachung von Radaufhängugen und Diesel-Einschpritzanlagen.* Doktorarbeit, TH Darmstadt, 1996.

[Lew93] Lewerentz, Claus: *Objektorientierte Software–Entwicklung, Einführung und Vergleich verschiedener Entwurfsverfahren.* FZI–Publikation, Forschungszentrum Informatik an der Universität Karlsruhe, 1993.

[Lip89] Lipschutz, Seymour: *Wahrscheinlichkeit Rechnung.* Schaum. McGraw–Hill, 1989.

[Lut93] Lutz, J.: *Motor-Diagnose mit Neuronalen Netzen.* Konferenzband Schöneburg, E. (Herausgeber): *Industrielle Anwendung Neuronaler Netze*, Kapitel 2. Addison–Wesley Publishing Company, 1993.

[Mar70] Mardia, K.V.: *Measures of Multivariate Skewness and Kurtosis with Applications.* Biometrika, 57:519–530, 1970.

[MD88] Moody, J. und Ch. Darken: *Learning with localized receptive fields.* Konferenzband Touretzky, Hinton und Sejnowski (Herausgeber): *Proceedings of the 1988 Connectionist Models Summer School.* Morgan Kaufmann, 1988.

[MM96] Merz, C.J. und P.M. Murphy: *UCI Repository of machine learning databases.* http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1996.

[Mor94] Morik, K.: *Balanced cooperative modeling.* Konferenzband Michalski, R. und G. Tecuci (Herausgeber): *Machine learning; A Multistrategy Approach*, Band 4, Kapitel 11. Morgan Kaufmann Publishers, San Mateo, California, 1994.

[Ous94] Ousterhout, John K.: *Tcl and the Tk Toolkit.* Addison–Wesley Publishing Company, Reading, MA, USA, 1994.

[PR93] Pfeifer, T. und M. M. Richter: *Diagnose von technischen Systemen. Grundlagen, Methoden und Perspektiven der Fehlerdiagnose.* DUV Informatik. T. Pfeifer and M. M. Richter, Wiesbaden, Deutscher Universitäts Verlag DUV Ausgabe, 1993.

[Pre94] Prechelt, Lutz: *PROBEN1 — A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms.* Bericht 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, September 1994. Anonymous FTP: /pub/papers/techreports/1994/1994-21.ps.Z on ftp.ira.uka.de.

[Pup88] Puppe, F.: *Einführung in Expertensysteme.* Studienreihe Informatik. Springer–Verlag, Heidelberg, Berlin, New York, 1988.

[QRK91] Quante, Ruckhäberle und Kirsch: *Umgebung zur modellgeschtützten Auswertung schnell veränderliche Signale am Beispiel von Verbrennungsmothoren.* FhG–Berichte, 1:49–54, 1991.

[Qua94] Quade, J.: *Automatische Zustandsmodellgenerierung für die Diagnose technischer Prozesse.* Nummer 426 im Buch *Fortschritt-Berichte VDI, Reihe 8 Meß-, Steuerungs- und Regelungstechnik.* VDI-Verlag, VDI Verlag, Postfach 10 10 54, 40001 Düsseldorf, 1994.

[Qui93] Quinlan, J. Ross: *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, San Mateo, California, 1993.

[RB93] Riedmiller, M. und H. Braun: *A direct adaptive method for faster backpropagation learning: The RPROP algorithm.* Konferenzband *International Conference on Neural Networks*, Seiten 586–591, San Francisco, USA, 1993. The Institut of Electrical and Electronics Engineers.

[RHW86] Rumelhart, D. E., G. E. Hinton und R. J. Williams: *Learning Internal Representations by Error Propagation.* Konferenzband *Parallel Distributed Processing*, Band 1, Seiten 318–362. The MIT Press, Cambridge, Massachusetts, 1986.

[Rie92] Riedmiller, M.: *Schnelle adaptive Lernverfahren für mehrschichtige Feedforeward Netzwerke –Vergleich und Weiterentwicklung–.* Diplomarbeit, Universität Karlsruhe, feb 1992.

[Rie96] Riedmiller, M.: *Selbständig lernende neuronale Steuerungen.* Doktorarbeit, Universität Karlsruhe (TH), 1996.

[SB95] Suna, R. und K. Berns: *Neuro Pipe-a neural network based system for pipeline inspection.* Konferenzband M. Kaiser, University of Karlsruhe (Herausgeber): *Fourth European Workshop on Learning Robots.* ESPRIT BR Project 7274 B-Learn II, dec 1995.

[SB96a] Suna, R. und K. Berns: *Interaktives Lernen von Diagnose- und Inspektionsaufgaben.* Konferenzband *CoWAN*, oct 1996.

[SB96b] Suna, R. und K. Berns: *NeuroPipe-a Neural Network Based System for Ultrasonic Inspection.* Studies in Informatics and Control, 5(3):269–278, sep 1996.

[SBG95] Suna, R., K. Berns und K. Germerdonk: *Pipeline-Diagnosis with Hybrid Neural Networks.* Konferenzband *COMADEM*, Band 8, Department of Mechanical Engineering, Queen's University, Kingston, Ontario, Canada K7L3N6, jun 1995.

[SBGB93] Suna, R., K. Berns, K. Germerdonk und A.O. Barbian: *Pipeline Diagnosis Using Backpropagation Networks.* Konferenzband *Neuro-Nimes*, Seiten 351–358, Nimes, oct 1993.

[SBH95] Suna, R., K. Berns und K. Heimannsfeld: *Estimation of Phase Parameters in 1D FT-NMR Spectroscopy.* Konferenzband *Australian Conference on Neural Networks*, Band 6, Seiten 164–167, Department of Electrical Engeneering, University of Sydney NSW 2006, Australia, feb 1995.

[Sch96] Schürmann, Jürgen: *Pattern Classification, A Unified View of Statistical and Neural Approaches.* Wiley, New York, 1996.

[Sch97] Schmajuk, Nestor A.: *Animal Learning and Cognition, A neural network approach.* Cambridge University Press, 1997.

[SE94] Suna, R. und P. Eisenhauer: *Schweißnahtverfolgung mit Kohonennetzen.* Konferenzband Horst Bischof, W.G. Kropatsch und (Herausgeber): *Mustererkennung*, Nummer 16 im Buch *DAGM Symposium*, Seiten 601–608, jul 1994.

[Seg96] Segl, K.: *Integration vo Form- und Spektralmerkmalen durch künstliche neuronele Netze bei der Satelitenbildklassifizierung.* Doktorarbeit, Universität Karlsruhe, 1996.

[SG96] Suna, R. und J. Gizzi: *Dynamic Bounds und Hybride Lernarchitekturen.* Konferenzband *CoWAN*, 1996.

[SG97] Suna, R. und J. Gizzi: *Hybrid Learning with Dynamic Bounds.* Konferenzband Verma, B. und X. Yao (Herausgeber): *ICCIMA*, Seite 439, Gold Coast, Australia, feb. 1997. Griffith University.

[Sha48] Shannon, C.E.: *A Mathematical Theory of Communication.* Bel Sys. Tech. Journal, (27):379–423,623–656, 1948.

[SLL93] Schmid-Lutz, V. und J. Lutz: *Getriebediagnose mit Neuronalen Netzen.* Konferenzband Schöneburg, E. (Herausgeber): *Industrielle Anwendung Neuronaler Netze*, Kapitel 3. Addison–Wesley Publishing Company, 1993.

[SLS93] Schmid-Lutz, V. und E. Schöneburg: *Ein hybrides Diagnosesystem.* Konferenzband Schöneburg, E. (Herausgeber): *Industrielle Anwendung Neuronaler Netze*, Kapitel 4. Addison–Wesley Publishing Company, 1993.

[Spe94] SPECKMANN, H.: *Kohonenkarte und fraktale Dimensionen.* Konferenzband *CoWan94 - Beiträge zum Cottbuser Workshop "Aspekte Neuronalen Lernens"*, Seiten 133–135. TU Cottbus, 1994.

[Spi90] SPIEGEL, MURRAY R.: *Statistik.* Schaum. McGraw–Hill, 2 Ausgabe, 1990.

[Sun91] SUNA, R.: *ParSim – Paralleler Simulator für Neuronale Netze.* Diplomarbeit, Forschungszentrum Informatik an der Universität Karlsruhe, nov 1991.

[Sun94a] SUNA, R.: *Connectionist Model Simulator.* Forschungszentrum Informatik an der Universität Karlsruhe, Haid-und-Neu Str. 10-14, 76131 Karlsruhe, 0.1 Ausgabe, may 1994.

[Sun94b] SUNA, R.: *Neuronale Netze für die Klassifikation von Defekten in Pipelines.* Konferenzband *CoWAN*, Seiten 84–94, oct 1994.

[Sun95a] SUNA, R.: *Connectionist Model Simulator.* Forschungszentrum Informatik an der Universität Karlsruhe, 1995.

[Sun95b] SUNA, ROBERT: *Iface Version 1.0.0.* Forschungszentrum Informatik an der Universität Karlsruhe, 1995.

[Win94] WINTERFELDT, G.: *Untersuchung des Cascade Correlation Lernverfahrens.* Studienarbeit, Forschungszentrum Informatik an der Universität Karlsruhe, apr 1994.

[WK91] WEISS, SHOLOM M. und CASIMIR A. KULIKOWSKI: *Computer Systems That Learn.* Morgan Kaufmann Publishers, San Mateo, California, 1991.

[Wos88] WOSCHNI, EUGEN-GEORG: *Informationstechnik.* Hüthig, 3 Ausgabe, 1988.

[Zöl95] ZÖLLNER, B.: *Adaptive Diagnose in der Elektronikproduktion.* Doktorarbeit, Erlangen, 1995.

Anhang A

Entwicklungssystem *CMS*

Das Entwicklungssystem ***CMS*** wurde im Rahmen dieser Arbeit am Forschungszentrum Informatik entwickelt und dient der Integration und Untersuchung der unterschiedlichsten Aspekte des Lernens. Im ***CMS*** wurden die in dieser Arbeit beschriebenen Ansätze betrachtet. Darüber hinaus wird es ständig erweitert und ergänzt. Ein Anwendungsbeispiel ist in Abbildung A.1 dargestellt.

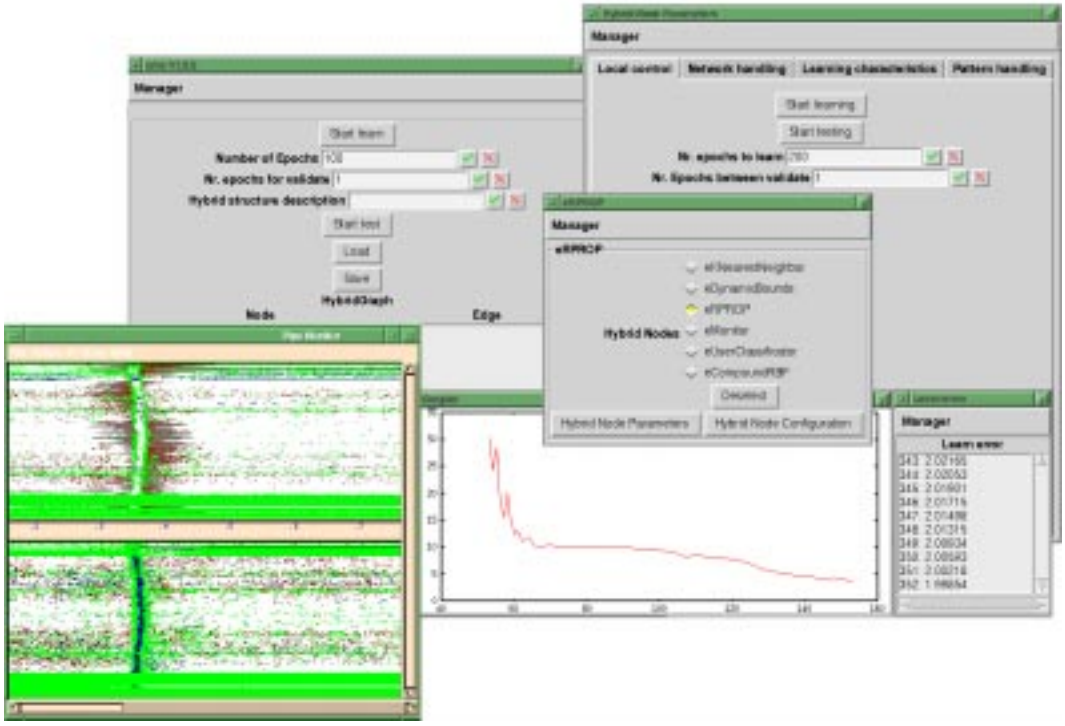


Abbildung A.1: Beispiel für die Benutzung des ***CMS*** mit verschiedenen integrierten Monitoringkomponenten und unterschiedlichen Lernverfahren.

CMS wurde mit den Methoden des objektorientierten Entwurfs konzipiert. Das wesentliche Designkonzept besteht im *White Tool–Box* Prinzip (siehe [Lew93]).

A.1 Das Hybridkonzept

A.1.1 Die hybride Lernarchitektur

Es werden Konzepte benötigt, die es einem Benutzer ermöglichen, verschiedene hybride Lernarchitekturen übersichtlich aufzubauen und einzutrainieren. Diese Konzepte müssen sowohl für die Konsistenz der einzelnen Architekturen sorgen und dem Benutzer Werkzeuge und Hilfsmittel zur Verfügung stellen, um diese Konsistenz zu sichern, als auch durch ein umfangreiches Steuerungskonzept die Synchronisation der beteiligten Komponenten ermöglichen. Schließlich soll das Konzept unabhängig von spezifischen Lernverfahren einsetzbar und erweiterbar sein.

Hierfür wurde ein sogenanntes *Hybridobjekt* konzipiert, das in den folgenden Abschnitten näher vorgestellt wird. Grundlage dieses *Hybridobjekts* ist ein gerichteter Graph, dessen Knoten, im folgenden *Objektknoten* genannt, einzelne Netze darstellen, die über Kanten, im folgenden als *Verbindungsobjekte* bezeichnet, kommunizieren.

Der Aufbau der Netze und Lernverfahren innerhalb der *Objektknoten* erfolgt mit dem Simulatorkonzept aus Abschnitt A.2, das somit vollständig in das hybride Konzept integriert wird. Die Struktur des Hybridobjekts ist in Abbildung A.2 dargestellt. Dieses Graphenkonzept soll es dem Benutzer ermöglichen, verschiedene Netze miteinander zu verschalten und somit die verschiedensten hybriden Lernarchitekturen aufzubauen. Auf die Komponenten des Graphen wird in Abschnitt A.1.2 detailliert eingegangen.

Um auch die Möglichkeit zu haben, andere Module außer Neuronalen Netzen in den hybriden Graphen zu integrieren, soll dem *Hybridobjekt* die interne Funktionalität der Objektknoten verborgen bleiben. Dies wird dadurch erreicht, daß alle Knoten von außen nur über eindeutig spezifizierte Schnittstellenfunktionen angesprochen werden können. Auf diese speziellen Schnittstellenfunktionen wird in Abschnitt A.1.2.4 noch genau eingegangen. Die Kanten wiederum, die die einzelnen Knoten miteinander verbinden, haben ebenfalls, je nach Verbindungstyp, unterschiedliche Eigenschaften. Diese Eigenschaften sind ebenfalls nach außen hin transparent, da auch die *Verbindungsobjekte* über eindeutig vorgegebene Schnittstellenfunktionen verfügen. Die einzelnen Knotentypen werden in den Abschnitten A.1.2.1 und A.1.2.2 detailliert behandelt.

Für ein sinnvolles Training hybrider Lernarchitekturen sind umfangreiche Steuerungs- und Synchronisationsmechanismen notwendig. In Abschnitt A.1.3 werden deshalb zwei Steuerungskonzepte für hybride Lernarchitekturen vorgestellt und miteinander verglichen.

A.1.2 Die einzelnen Komponenten des Hybridobjekts

A.1.2.1 Die Objektknoten

Fest vorgegeben sind ein Ein- und ein Ausgabeobjekt. Diese sind direkt mit einer Quelle von Lernbeispielen verbunden, von der aus Eingaben in das Eingabeobjekt und *Targets*[1] in das Ausgabeobjekt geladen werden. Die Quelle der Lernbeispiele kann vom Benutzer ausgewählt werden. Mögliche Quellen sind Dateien, Sensoren oder auch Datenbanken. Die Ein- und Ausgabeknoten gehören definitionsgemäß bereits zum Graphen dazu. Damit einzelne Knoten nicht jedesmal mit dem Ein- und Ausgabeobjekt verbunden werden müssen, falls keine hybride Lernarchitektur verwendet wird, sondern beispiels-

[1] Als *Targets* werden die Sollwerte bezeichnet, die ein Netz liefern soll.

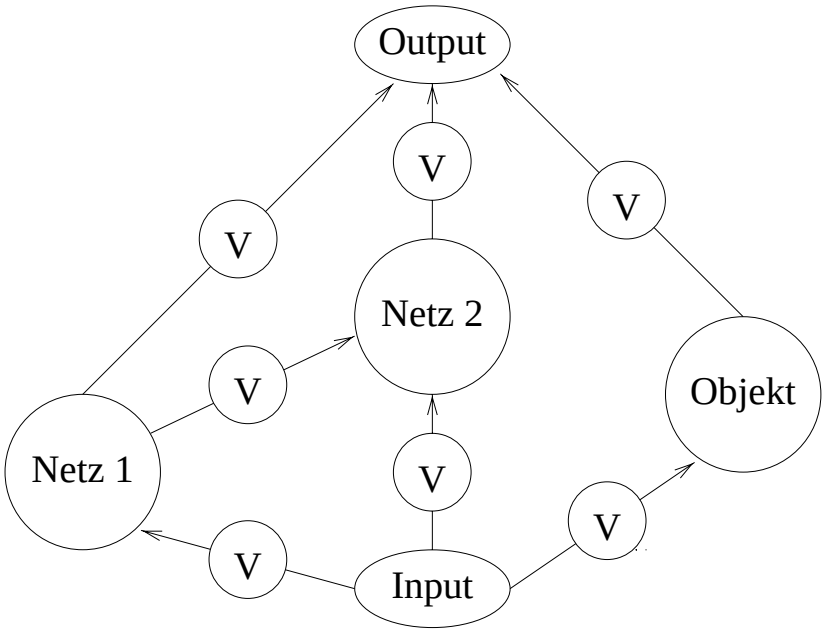


Abbildung A.2: Der prinzipielle Aufbau des Hybridobjekts. Es ist als Graph mit zwei Arten von Knoten, den Objektknoten und den mit V gekennzeichneten Knoten, die den Typ der Verbindung zwischen den Objekten angibt, aufgebaut.

weise nur ein Netz isoliert eingelernt werden soll, ist vom Konzept her vorgesehen, jeden einzelnen Knoten weiterhin mit einer lokalen Quelle für Lernbeispiele auszustatten, die im Falle eines isolierten Objektknotens verwendet werden kann.

Als Objektknoten kommen weiterhin alle Arten von Netzen in Frage, die mit dem *Connectionist Model Simulator* aus [Sun94a] simuliert werden können, wie Backpropagation-Netze, insbesondere die RPROP-Netze aus [RB93], verschiedene selbstorganisierende Karten, die in [Giz94] näher beschrieben worden sind, sowie Radial Basis Functions und der *Dynamic Bounds*-Algorithmus.

Das Hybridobjekt ist konzeptionell derart ausgelegt, daß zukünftig auch Objekte, die keine Netze darstellen, eingebunden werden können. Dabei handelt es sich um alle Arten von Vorverarbeitungsalgorithmen für Lerndaten[2] sowie Module, die eine statistische Vorverarbeitung ermöglichen, oder aber auch um Monitore, die den Zustand einer Anwendung nach außen hin anzeigen.

Alle Objektknoten sind mit sog. Konfigurationsparametern ausgestattet, die die spezifischen Eigenschaften des Knotens wiedergeben.

A.1.2.2 Die Verbindungsknoten

Um sinnvoll zusammenwirken zu können, müssen die einzelnen Objektknoten aus Abschnitt A.1.2.1 miteinander verbunden werden. Diese Verbindungen können von ganz unterschiedlicher Art und je nach Anwendung mehr oder weniger komplex sein. Jede Verbindung wird durch einen Verbindungsknoten V beschrieben (Abbildung A.2). In diesem Objekt V wird festgelegt, was und wie etwas übergeben werden soll.

[2]Darunter sind Module zur Normierung von Lerndaten oder zur Auswahl bestimmter Teilmengen aus Lerndatensätzen wie zum Beispiel der MIFS Algorithmus von [Bat94], aber auch Mathematikmodule wie Maple, zu verstehen.

Mögliche Verbindungstypen sind *eins-zu-eins* Verbindungen, Backpropagationverbindungen, die eine Verbindung in beide Richtungen zulassen, Multiplexer und Demultiplexer sowie Spezialtypen, die nur für bestimmte hybride Netzarchitekturen gebraucht werden. In den *Verbindungsobjekten* werden, wie bei den Objektknoten, die spezifischen Eigenschaften der Verbindungen durch Konfigurationsparameter angegeben. Über diese Parameter wird z.B. auch festgelegt, wie die Übergabe von Lernbeispielen technisch erfolgt, d.h. ob zum Beispiel nur kopiert wird oder Zeiger in den Datenstrukturen umzuhängen sind. In Abbildung A.3 ist ein Objekt dargestellt, das eine *eins-zu-eins* Verbindung zwischen zwei Objekten herstellt.

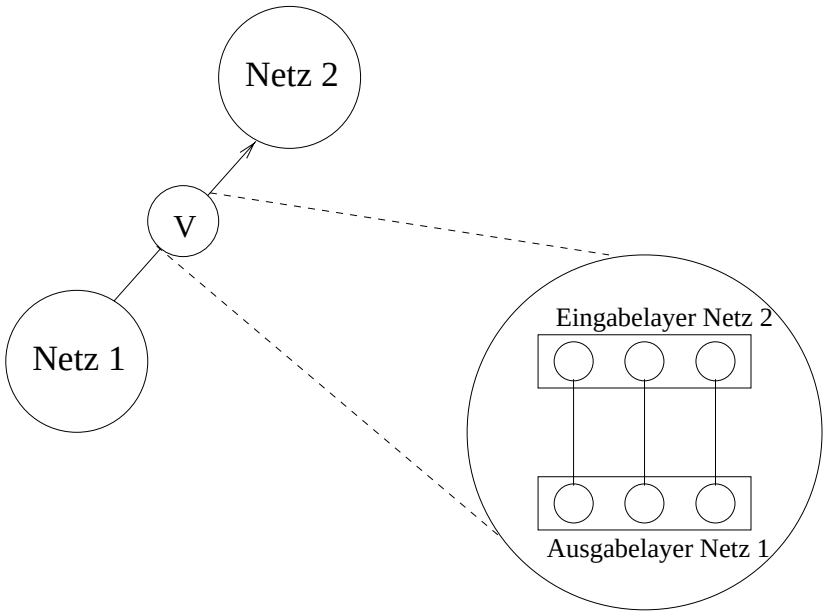


Abbildung A.3: Darstellung einer *eins-zu-eins* Verbindung zwischen zwei Netzen im Detail.

A.1.2.3 Die Objektkonfigurationen

Für die Konsistenz der gesamten hybriden Lernarchitektur ist es wichtig, daß sowohl die einzelnen Objektknoten untereinander, als auch die Objektknoten und die Verbindungsknoten zwischen diesen kompatibel sind. Deshalb müssen sowohl die Objektknoten als auch die Verbindungsknoten konfiguriert werden. Anhand sogenannter Objektkonfigurationen wird dann entschieden, ob zwei Objektknoten miteinander verbunden werden können, und welche Verbindungsknoten gegebenenfalls dafür in Frage kommen. Für diese Objektkonfigurationen sind verschiedene Konfigurationsparameter herausgearbeitet worden, auf die im folgenden näher eingegangen wird.

Die Konfigurationsparameter lassen sich grob in drei Klassen einteilen. Auf der einen Seite gibt es statische Konfigurationsparameter, die beispielsweise durch ein bestimmtes Lernverfahren fest vorgegeben sind, auf der anderen Seite freie Parameter, die sich noch einmal in *interne* und *externe* Parameter unterteilen lassen.

Interne Parameter werden einmal vom Benutzer festgelegt und sind dann für einen Knoten während seiner gesamten Lebensdauer nicht mehr änderbar.

Externe Parameter hingegen sind vom Benutzer jederzeit je nach Anwendung in

einem vorgegebenen Rahmen modifizierbar. Dieser Rahmen hängt wiederum teilweise von der internen Funktionalität des entsprechenden Knotens ab. Ein Beispiel einer vollständigen Objektkonfiguration ist in Abbildung A.4 angegeben. Auf die einzelnen Parameter und deren Repräsentation wird in den folgenden Abschnitten näher eingegangen.

```
//------------------------------------------------------------------------
//  HybridNode: tRPROP
//------------------------------------------------------------------------
//
//  Description: RPROP Lernverfahren
//
//  Parameters
//   propagation mode: Pattern
//   learn mode: Epoch
//   supervised: yes
//   input definition: intern
//   output definition: intern
//   accept targets: yes
//   deliver backpropagation error: yes
//   num input: {1..j}
//   input config: {[-inf,inf]:{0..inf}}
//   num output: {1..k}
//   output config: {[-inf,inf]:{0..inf}}
//
//------------------------------------------------------------------------
//  HYBRID CONFIG END:
//------------------------------------------------------------------------
```

Abbildung A.4: Beispiel einer Objektkonfiguration.

Eine Sonderstellung bei der Konfiguration nehmen die Ein- und Ausgabeknoten ein. Diese werden nicht durch die Eingabe von Konfigurationsparametern festgesetzt, sondern indirekt durch die Konfigurationen der Lerndaten, die geladen werden. Während der Lebensdauer einer hybriden Architektur werden die Konfigurationen des Ein- und Ausgabeknotens durch den ersten Lerndatensatz, der geladen wird, festgelegt. Es können danach auch nur noch Datensätze geladen werden, die dieselbe Konfiguration besitzen wie der zuerst geladene Datensatz.

A.1.2.4 Die Schnittstellenfunktionen der Objekt- und Verbindungsknoten

Um die innere Funktionalität der einzelnen Objekt- und Verbindungsknoten nach außen hin verborgen zu halten, sind die Knoten des Hybridgraphen mit einer Reihe fester Schnittstellenfunktionen versehen, über die die Objekte untereinander bzw. mit der hybriden Steuerung kommunizieren. Für die Objektknoten wurden folgende Schnittstellenfunktionen festgelegt:

Die Funktionen *Propagate* und *BackPropagate* stoßen die innere Funktionalität eines Knotens an. Hierbei muß es sich nicht um einen Lernalgorithmus handeln. Da aber hauptsächlich verschiedene Lernverfahren in der hybriden Struktur zum Einsatz kommen, wurde die Funktion entsprechend genannt.

Configure initialisiert die Anzahl der Ein- und Ausgaben eines Objektknotens.

GetConfig und *SetConfig* werden bei der Konsistenzprüfung der einzelnen Knoten des Graphen benutzt. Hierbei liefert *GetConfig* die komplette Konfiguration eines Knotens. *SetConfig* setzt die Konfigurationsparameter eines Knotens, die vom Benutzer ausgewählt werden können.

Eine wesentliche Rolle spielen die Funktionen *GetOutput* und *SetInput*. Über diese Funktionen wird es für ein Verbindungsobjekt erst möglich, auf die Ausgabe bzw. Eingabe eines Objektknotens zuzugreifen. Mit Hilfe dieser Funktionen erfolgt dann die tatsächliche Übergabe der Daten.

Analog gibt es für mögliche Rückverbindungen die Zugriffsfunktionen *GetTarget* und *SetTarget*. Ein Verbindungsknoten kann mit *GetTarget* aus dem Zielknoten ein *Target* holen und es mit Hilfe von *SetTarget* an den entsprechenden Quellknoten weitergeben.

Auch die Verbindungsobjekte verfügen über festgelegte Schnittstellenfunktionen. Dies sind im folgenden:

- GetIface
- UpdateIface
- Propagate
- BackPropagate
- Init
- Configure
- Load
- Save

Die Funktionen *Configure* und *Init* haben die selben Aufgaben wie bei den Objektknoten. Sie werden ebenfalls bei der Konsistenzprüfung und zum Setzen von Parametern verwendet.

Angestoßen wird eine Verbindung über die Funktionen *Propagate*, was eine Vorwärtsverbindung auslöst, und *BackPropagate* für eine Rückwärtsverbindung. Bei Verbindungsobjekten, die keine Rückwärtsverbindung zulassen, hat die Funktion *BackPropagate* keine interne Funktionalität, sie ist also leer. Die Funktion *Propagate* benutzt die oben bereits erwähnten Schnittstellenfunktionen *GetOutput* und *SetInput* der entsprechenden Objektknoten, die sie verbinden. Analog verwendet die Funktion *BackPropagate* die Schnittstellenfunktionen *GetTarget* und *SetTarget* der Objektknoten, um auf die *Targets* zuzugreifen bzw. sie zu liefern. Es werden hier also explizit die Schnittstellenfunktionen der Objektknoten von Schnittstellenfunktionen der Verbindungsknoten benutzt. Dies ist noch einmal in Abbildung A.5 schematisch dargestellt. Abbildung A.6 schließlich zeigt den vollständigen Zugriff eines Verbindungsknotens auf den Quell- und Zielknoten mit Hilfe der Schnittstellenfunktionen.

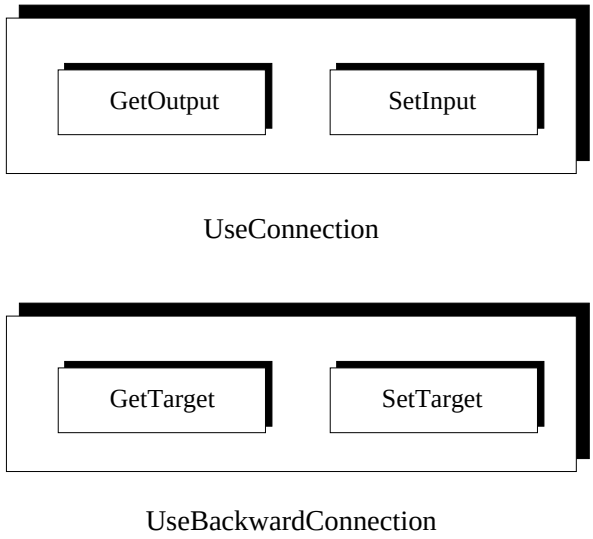


Abbildung A.5: Interner Aufbau der Funktionen *Propagate* und *BackPropagate* der Verbindungsknoten. Diese benutzen explizit einige der Schnittstellenfunktionen der Objektknoten.

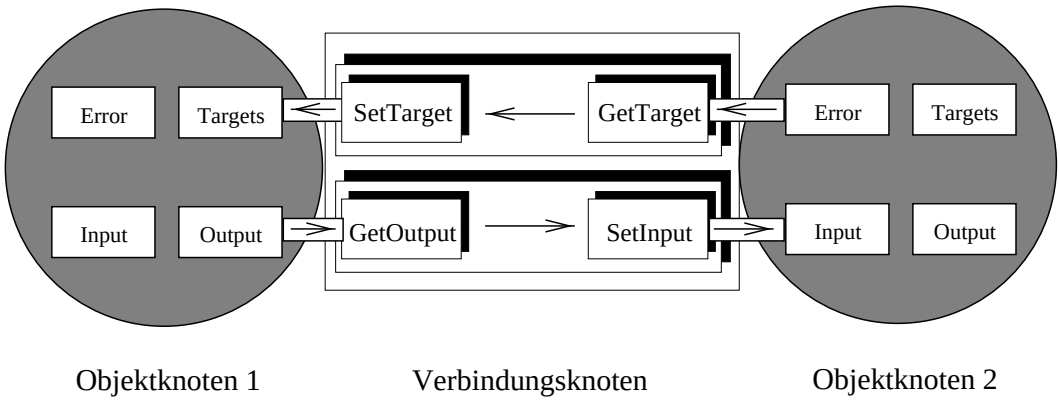


Abbildung A.6: Vollständiger Zugriff eines Verbindungsknotens auf die Objektknoten über die Schnittstellenfunktionen.

A.1.3 Steuerungskonzept für hybride Lernarchitekturen

Unter der Steuerung einer hybriden Lernarchitektur versteht man die Synchronisation der einzelnen Objekt- und Verbindungsknoten. Die Steuerung einer hybriden Lernarchitektur ist sehr aufwendig und kompliziert. Die einzelnen Komponenten des *Hybridgraphen* müssen mit Lerndaten versorgt werden, und der zeitliche Ablauf der einzelnen Aktionen muß exakt gemäß den Anforderungen der einzelnen Objektknoten synchronisiert werden.

A.1.3.1 Steuerung der hybriden Lernarchitektur in den Hybridknoten

Ein erster Ansatz zur Steuerung einer hybriden Lernarchitektur liegt darin, die Steuerung nur über die Hybridknoten zu organisieren. Hauptaufgabe der Verbindungsknoten bleibt es, ihre Vorgänger mir *Targets* bzw. ihre Nachfolger mit Lernbeispielen zu versorgen. Hierzu muß jeder Verbindungsknoten aber genau wissen, wann er ein Lernbeispiel

oder *Target* liefern darf. Um einem Verbindungsobjekt zu signalisieren, daß ein neues Lernbeispiel verarbeitet werden kann, muß jeder Objektknoten über eine Funktion verfügen, die der Verbindung anzeigt, daß neue Daten erwartet werden. Diese Funktion wird im folgenden mit *GetPattern* bezeichnet.

Bei Objektknoten, die nur eine eingehende Verbindung haben, ist die Synchronisation relativ unproblematisch. Hier kann der Verbindungsknoten auf die Anfrage *GetPattern* des Nachfolgeknotens warten und dann sofort ein Lernbeispiel liefern, wenn der Vorgänger eine Ausgabe produziert hat. Abbildung A.7 zeigt diesen Fall.

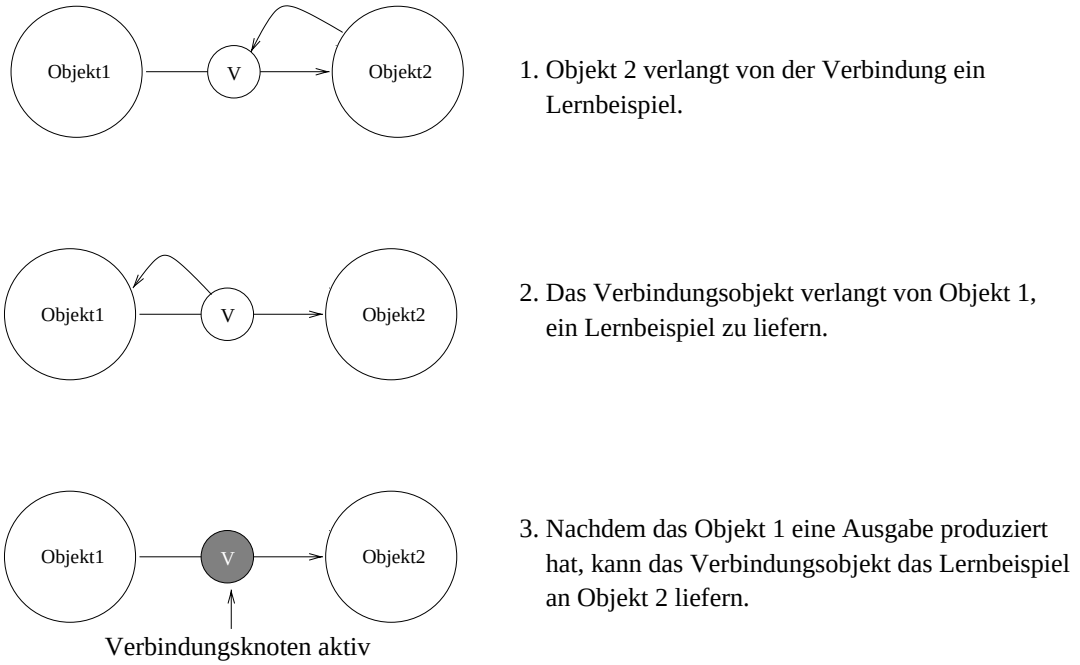


Abbildung A.7: Ablauf der Übergabe von Lernbeispielen zwischen zwei Objektknoten, die über einen Verbindungsknoten gekoppelt sind.

Schwierigkeiten treten dann auf, wenn entweder mehrere Verbindungen denselben Zielknoten haben, oder im Hybridgraphen mehrere Zweige parallel laufen und in einem gemeinsamen Zielknoten enden[3]. Hier darf das Verbindungsobjekt nicht sofort liefern, wenn der Quellknoten eine Ausgabe produziert hat. Der Übergabevorgang der Lernbeispiele aller Vorgängerverbindungen des Zielknotens muß synchronisiert werden.

Dies geht nur im Zielknoten selbst, da nur er weiß, wann alle Verbindungsobjekte geliefert haben. Hierzu wird im Zielknoten ein Zähler geführt, der mit jedem *GetPattern*-Aufruf an einen Verbindungsknoten um eins hochgezählt wird. Liefert die entsprechende Verbindung dann ein Lernbeispiel, wird der Zähler wieder um eins dekrementiert. Wenn der Zähler wieder bei Null angelangt ist, weiß der Knoten, daß jetzt ein vollständiges Lernbeispiel zur Verarbeitung bereitsteht. Abbildung A.8 zeigt ein Beispiel.

Analog verläuft der Prozeß auf der Ausgabeseite eines Objektknotens. Auch hier zeigt ein Zähler an, wann alle abgehenden Verbindungen aktiviert sind und die aktuelle Ausgabe übernommen haben. Ist dies geschehen, kann wieder eine neue Ausgabe

[3]Im folgenden soll nur noch der Propagierungsfall, also die Vorwärtsverbindung, betrachtet werden. Alle Überlegungen gelten aber auch analog für Rückwärtsverbindungen.

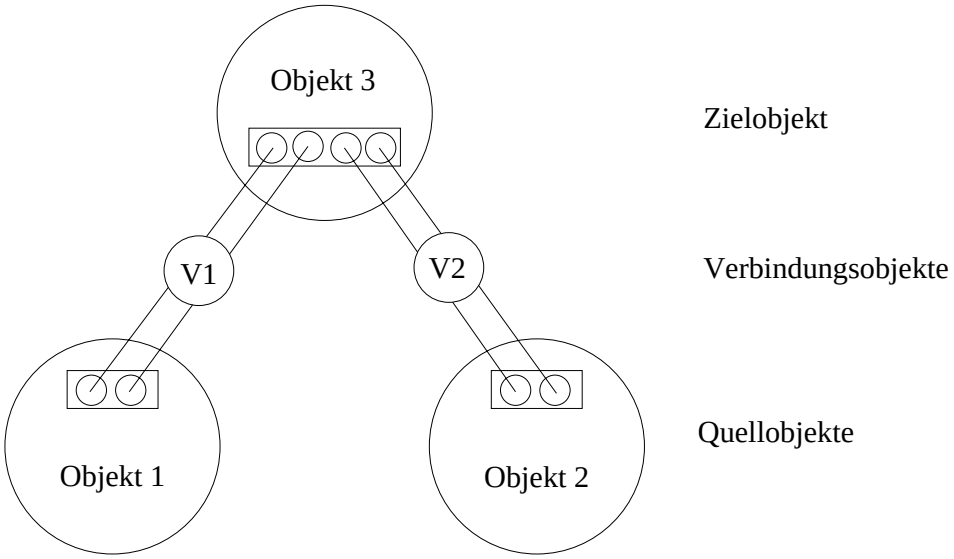


Abbildung A.8: Darstellung eines Zielobjektes, dessen Eingabe sich aus den Ausgaben verschiedener Quellobjekte zusammensetzt. Hierbei ist eine Synchronisation der Verbindungsobjekte *V1* und *V2* notwendig.

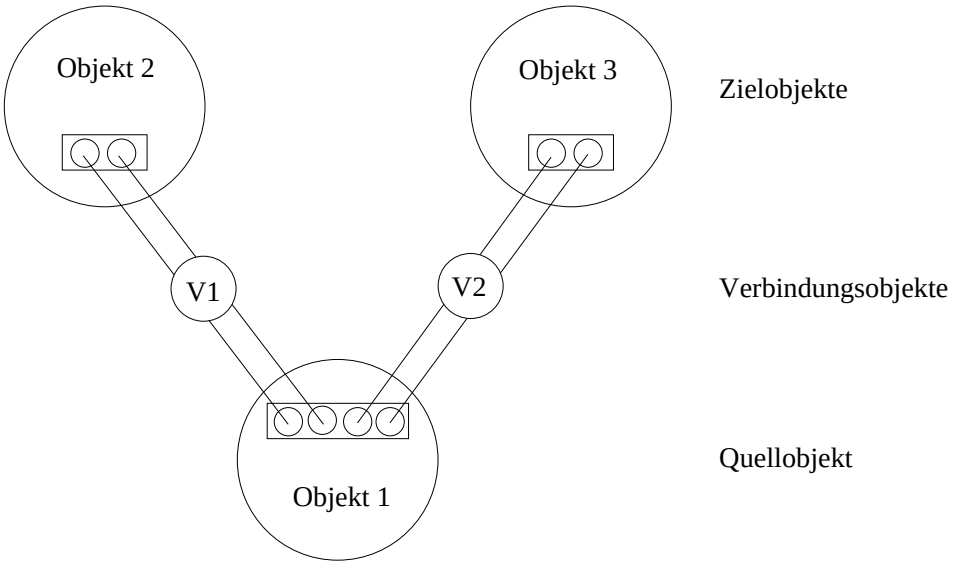


Abbildung A.9: Ein Quellknoten, der zwei Zielknoten versorgt. Hier wird eine Synchronisation der Verbindungsobjekte *V1* und *V2* benötigt, da ein neues Lernbeispiel erst dann geliefert werden darf, wenn beide Zielknoten das vorherige Lernbeispiel verarbeitet haben.

produziert werden. Dies ist in Abbildung A.9 dargestellt.

Zusätzlich zur lokalen Synchronisation in den Knoten selbst ist es noch nötig, den gesamten Ablauf des hybriden Lernvorgangs zu synchronisieren. Begonnen wird beim Ausgabeknoten des Graphen. Dieser setzt einen *GetPattern*-Aufruf an alle seine direkten Vorgängerknoten ab.

Diese wiederum benötigen, um eine Ausgabe liefern zu können, ebenfalls ein Lernbeispiel und versuchen, dies über einen erneuten *GetPattern*-Aufruf an ihre direkten

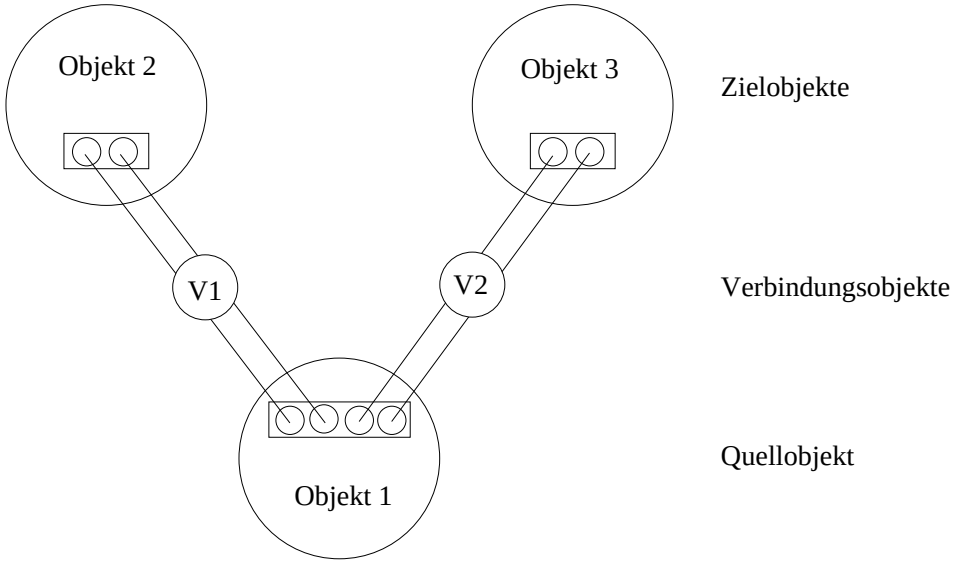


Abbildung A.10: Der Ablauf eines Propagierungsschrittes für einen hybriden Graphen, der mit einer lokalen Steuerung arbeitet. Die aktivierten Knoten oder Verbindungen sind jeweils hervorgehoben dargestellt.

Vorgänger zu erhalten. Der Vorgang wiederholt sich so lange, bis der Eingabeknoten erreicht wird.

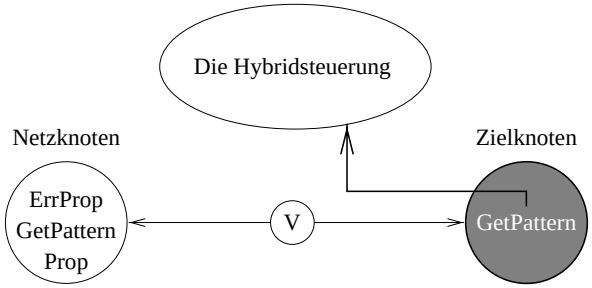
Dieser ist direkt mit einer Quelle von Lernbeispielen verbunden und kann das gewünschte Beispiel sofort liefern. Dazu aktiviert er der Reihe nach sämtliche seiner abgehenden Verbindungen. Alle Nachfolgeknoten können jetzt das Lernbeispiel verarbeiten und ebenfalls alle ihre abgehenden Verbindungen anstoßen. So setzt sich der Prozeß bis zum Zielobjekt fort, wo er von neuem startet. Abbildung A.10 zeigt den Ablauf eines Propagierungsschrittes eines hybriden Graphen, der mit einer lokalen Steuerung arbeitet. Die Aufrufe der Verbindungsobjekte werden hierbei lokal in den Objektknoten synchronisiert.

A.1.3.2 Zentrale Steuerung der hybriden Lernarchitektur

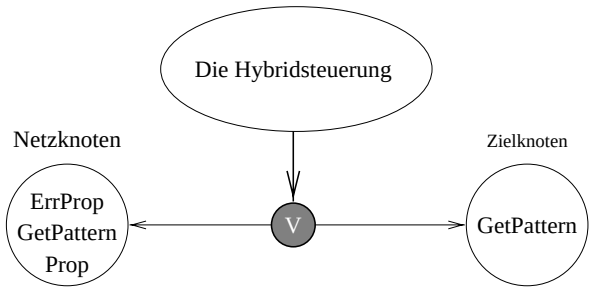
Motivation für das Konzept dieses Steuerungsansatzes ist das Fernziel, über die hybride Lernarchitektur eine Regelbasis zur Automatisierung des Klassifikationsprozesses zu setzen, die dann die Auswahl der hybriden Lernarchitekturen steuern soll. Hierfür ist es von Vorteil, daß die gesamte Steuerung und Synchronisation der Lernarchitektur an einer zentralen Stelle durchgeführt wird, auf die dann von der Regelbasis aus jederzeit direkt zugegriffen werden kann.

Beim hier vorgestellten Steuerungsansatz laufen die Anfragen nach Lernbeispielen bzw. *Targets* sämtlicher Objektknoten bei einer zentralen Steuerungseinheit ein, die diese dann serialisiert und so für die Synchronisation der Lernarchitektur sorgt.

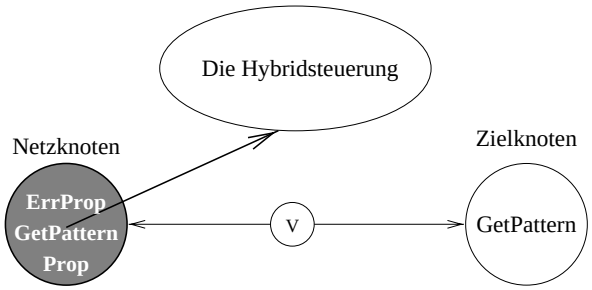
Gestartet wird wieder beim Ausgabeknoten. Dieser setzt an die zentrale Steuerung den Aufruf *GetPattern* ab. Daraufhin aktiviert die Steuerung zuerst einmal alle Rückwärtsverbindungen, die vom Ausgabeknoten wegführen. Dies geschieht aus Gründen der Effizienz. Um die Anzahl der Funktionsaufrufe an die Steuerung zu minimieren, wird bei diesem Konzept die Fehlerpropagierung des $n-1$-ten Schrittes erst im n-ten Schritt durchgeführt. Danach werden in allen Vorgängerknoten des Zielkno-



(a) Der Zielknoten setzt einen GetPattern–Aufruf an die Hybridsteuerung ab.



(b) Die Hybridsteuerung aktiviert die Rückwärtsverbindungen.



(c) Der Hybridknoten führt eienen BackPropagate–Schritt durch und setzt einen erneuten GetPattern–Aufruf an die Hybridsteuerung ab.

Abbildung A.11: Darstellung der zentralen Steuerung des Hybridobjektes. Alle Funktionsaufrufe erfolgen an die Steuerung, die auch die Verbindungen aktiviert.

tens die oben erwähnten Fehlerpropagierungen veranlaßt oder, falls es sich nicht um einen Netzknoten handelt, andere Aktionen durchgeführt. Dieser erste Schritt ist in Abbildung A.11 dargestellt.

Nach dem Abschluß dieses Vorgangs setzen alle Knoten wiederum einen *GetPattern* Aufruf an die Steuerung ab, um ein neues Lernbeispiel zum Propagieren zu erhalten[4]. Dieser Vorgang setzt sich solange fort, bis schließlich der Eingabeknoten erreicht wurde.

Dieser erhält bei seinem *GetPattern* Aufruf dann wieder direkt ein Lernbeispiel

[4]Es wird hier der Lernvorgang des Simulators detailliert dargestellt. Bei einem Recallvorgang, also der Anwendung der eingelernten Architektur, entfällt der Fehlerpropagierungsschritt.

aus einer vom Benutzer vorher ausgewählten Quelle (Datei, Datenbank, Sensor). Jetzt kehrt sich der gesamte Vorgang um, d.h. die Steuerung aktiviert der Reihe nach alle Vorwärtsverbindungen zwischen zwei Knotenschichten und liefert so die Lernbeispiele zur Propagierung. Der Prozeß setzt sich dann wieder bis zum Ausgabeknoten fort.

Der Vorteil bei dieser Art der Steuerung liegt darin, daß die zentrale Steuerung vor Beginn des Trainings den gesamten Graphen analysiert und alle Knoten- und Verbindungsaufrufe serialisiert. Diese Analyse kann bereits während der Konsistenzprüfung der hybriden Lernarchitektur vorgenommen werden.

A.2 Das Konzept der Neuronalen Netze

Um in möglichst umfangreichem Rahmen Neuronale Netze zu simulieren, ist ein Simulatorkonzept notwendig, das dem Benutzer ein breites Spektrum an Variationsmöglichkeiten hinsichtlich der einsetzbaren Lernverfahren bietet und zudem leicht erweiterbar ist. Grundlage des hier vorgestellten Konzeptes ist der *Connectionist Model Simulator* aus [Sun94a]. Die Idee des Konzeptes ist, den Simulator als eine Vererbungshierarchie aufzubauen, beginnend bei den elementarsten Bestandteilen eines Neuronalen Netzes.

Diese elementarsten Bestandteile sind die Neuronen und die Gewichte. Aus den Neuronen und Gewichten wird die Netzstruktur der einzelnen Neuronalen Netze aufgebaut. Die Kontrollinstanz des Simulators für die Organisation des Trainings ist von dieser Netzstruktur abgeleitet.

Jedes Lernverfahren hat spezifische Propagierungs- und Fehlerpropagierungsfunktionen. Diese hängen wiederum von der Kontrollinstanz ab. So entsteht Schritt für Schritt eine Vererbungshierarchie, die schließlich in der vollständigen Spezifikation eines bestimmten Lernverfahrens im Simulator gipfelt. Diese Vererbungshierarchie ist in Abbildung A.12 dargestellt.

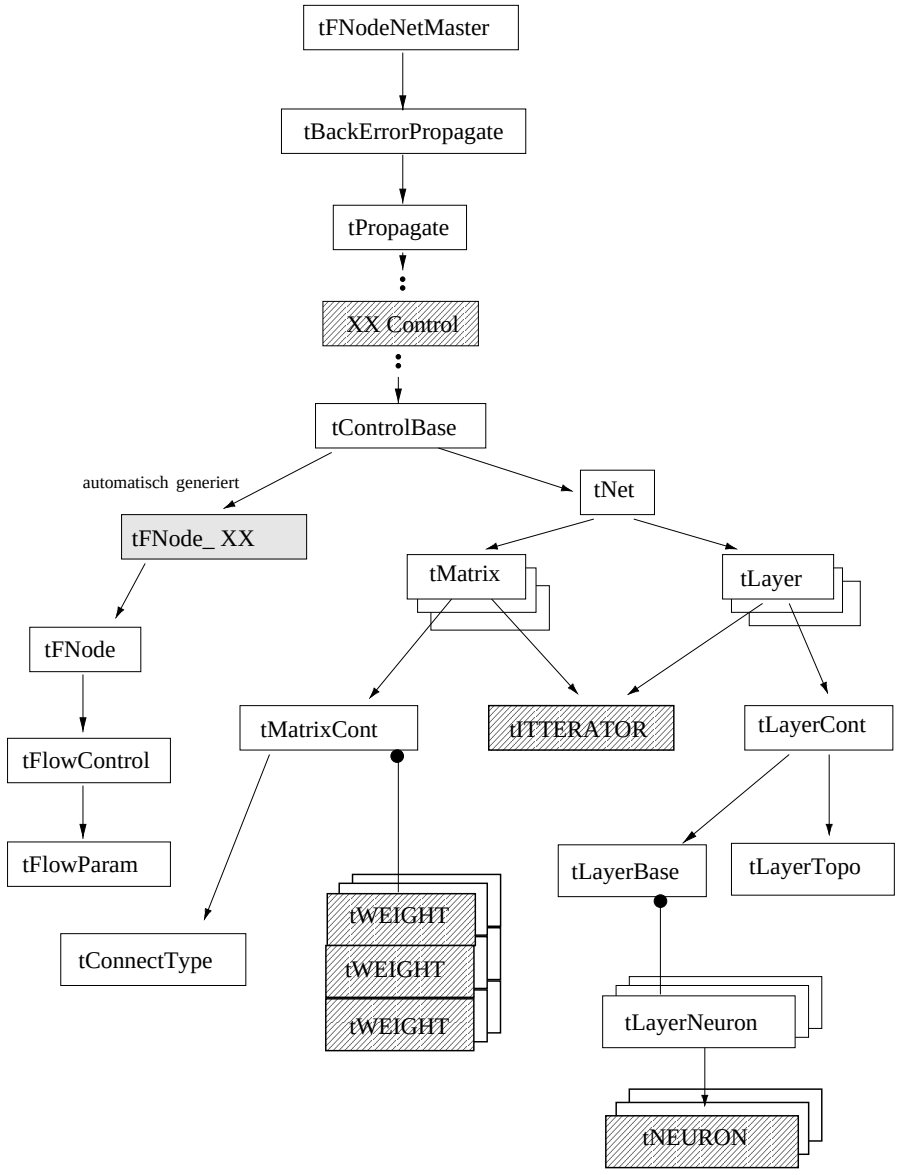


Abbildung A.12: Die Vererbungshierarchie des ***CMS***. Ein vollständiges Lernverfahren setzt sich aus verschiedenen voneinander abhängigen Klassen–Templates zusammen.

Anhang B

CMS-Datenbank

Die ***CMS***-Datenbank dient der Archivierung von Versuchsergebnissen, derem Vergleich und der geordneten Versuchsdurchführung. Zu diesem Zweck ist sie in Projekte gegliedert. Jedes Projekt bildet dabei eine abgeschlossene Problemstellung. So ist beispielsweise die Längsnahtproblematik ein Projekt (siehe E.2). Das Projekt dient dazu, alle diesbezüglichen Informationen zu bündeln (siehe Abbildung 6.15).

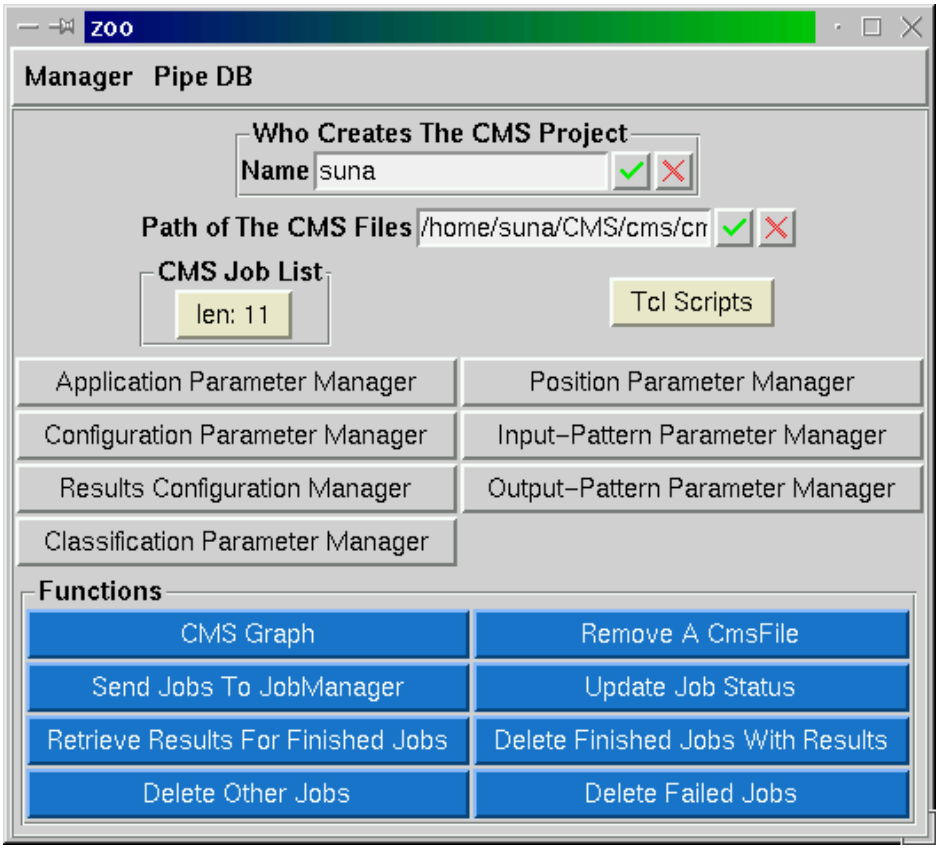


Abbildung B.1: Sicht auf ein Projekt aus der CMS-Datenbank. Von hier können neue Versuche gestartet, Ergebnisse abgefragt und visualisiert werden.

Bei der interaktiven Diagnose besteht die zeitaufwendigste Aufgabe im Sammeln der Beispieldaten. Aus diesem Grund ist die ***CMS***-Datenbank im hohen Maße darauf ausgerichtet, Beispiele zu verwalten. Die Beispiele werden durch die sogenannte

Positionsinformation beschrieben. Diese Position ist für jedes Projekt unterschiedlich definiert, beschreibt aber eindeutig die Herkunft des Beispiels. Mit Hilfe dieser Positionsinformation ist es möglich, Beispiele in den aufgezeichneten Daten wiederzufinden.

Alle zu einem Projekt bekannten Beispiele mit Positionsinformation werden im sogenannten *Masterfile* gespeichert. Ausgehend von diesem *Masterfile* wird ein Ableitungsbaum gebildet (siehe Abbildung 6.15), wobei nur eine Selektion der vorhandenen Beispieldaten erlaubt ist. Während der Ableitung in der Ableitungshierarchie werden die Beispiele durch Informationen ergänzt. Dabei existieren zwei Stufen; die Ergänzung durch die Application-Stufe, bei der die Beispiele um die Merkmale ergänzt werden, und die CMS-Stufe, bei der die Beispiele durch unterschiedliche Verfahren ausgewertet werden (siehe auch Abschnitt 6.5).

Die ***CMS***-Datenbank ist sowohl für die Verwaltung der Beispieldaten und der erzielten Ergebnisse verantwortlich, als auch für die reibungslose Ablaufkontrolle der dazu notwendigen Berechnungen. Diese Berechnungen werden als *Jobs* spezifiziert und durch durch den *JobManager* (siehe Abbildung B.2) auf einem Workstationcluster abgearbeitet. Dabei überwacht er den Prozeß- und Rechnerstatus. Bei Ausfällen oder Abstürzen der Systeme sorgt er für ein ordnungsgemäßes Wiederaufsetzen der Jobs.

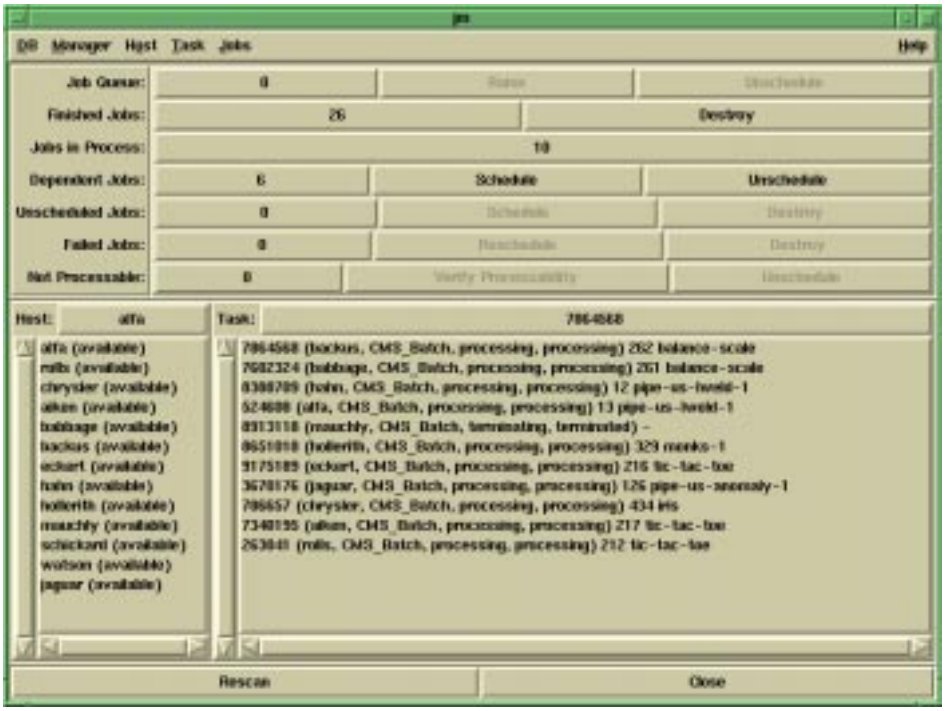


Abbildung B.2: Überwachung der verteilten Lernprozesse über ein Workstation-Cluster.

Das gesamte System ist auf der Basis der am FZI entwickelten objektorientierten Datenbank OBST konzipiert. Mittels des in Abbildung B.3 dargestellten Monitors ist es dem Benutzer möglich, einen Einblick in den jeweiligen Zustand der Datenbank und der aus ihr arbeitenden Prozesse zu gewinnen.

Darüber hinaus hat der Benutzer die Möglichkeit, unterschiedlichste Auswertungen mittels der Tcl-Skriptsprache und der vorgefertigten PERL-Skripten durchzuführen.

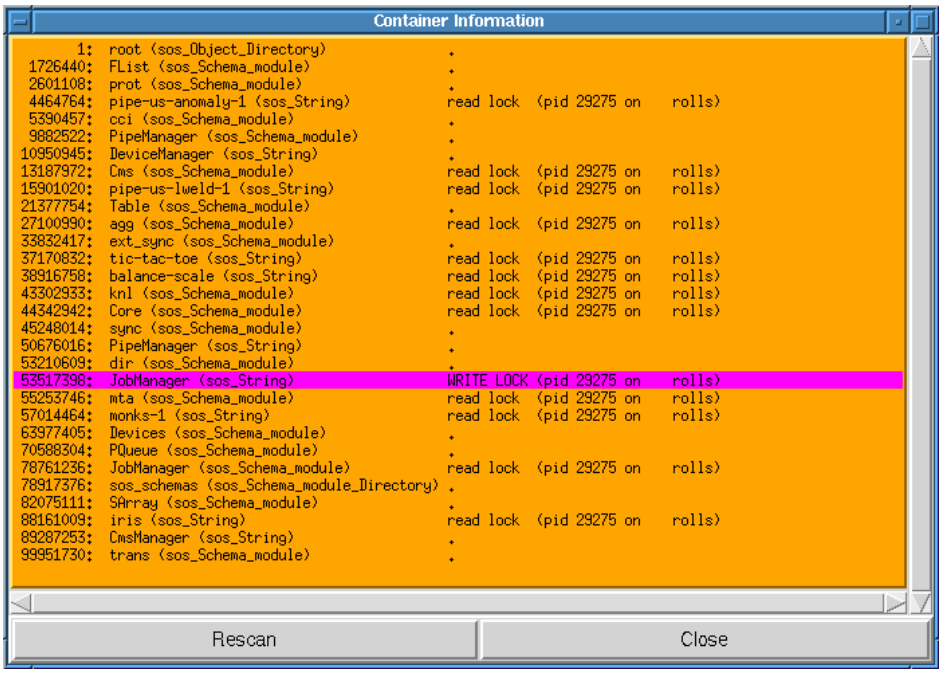


Abbildung B.3: Darstellung des Datenbankstatus der unterschiedlichen Projekte mit Referenzen auf die gerade zugreifenden Prozesse und Workstations.

Anhang C

Generische graphische Schnittstelle *Iface*

Iface ist ein System zur Unterstützung des applikationsunabhängigen Benutzerschnittstellenentwurfs. Mit seiner Hilfe lassen sich automatisch graphische Oberflächen nur anhand der Applikationsstruktur erzeugen. Zum ***Iface***–System gehören verschiedene Teilkomponenten, wie eine online Konfiguration, ein Hilfesystem, eine Makroverarbeitung und das Batchprocessing, die unabhängig von der Applikation durch die Benutzung von ***Iface*** automatisch angebunden werden (siehe Abbildung C.1).

C.1 Kurze Einführung in *Iface*

Die Applikation ist der Darstellung der Schnittstelle nicht bekannt. Dadurch kann das GUI individuell (sogar während der Laufzeit) angepaßt oder automatisch generiert werden, ohne daß dafür die Applikation modifiziert werden muß. Für die konkrete Implementierung der graphischen Schnittstelle wurde das Tcl/Tk verwendet (siehe [Ous94]).

Die Oberfläche ***Iface*** weist folgende besondere Kennzeichen auf (vgl. dazu auch [Sun95a], [Alb96]):

Objektorientiert Die Schnittstelle zu ***Iface*** ist objektorientiert (C++ und iTcl).

Trennung zwischen Oberfläche und Anwendung Die Oberfläche (***Iface***) ist von der eigentlichen Anwendung komplett getrennt.

Plattformunabhängig Die Oberfläche ***Iface*** ist auf kein Betriebssystem oder eine bestimmte Hardware festgelegt.

Callback-Funktionen Die Aktionen des Anwenders werden durch Callback-Funktionen an die Anwendung weitergereicht.

Generische Oberfläche Beim Entwurf mit Iface wird die Oberfläche nicht durch den Entwickler erzeugt, sondern dieser gibt lediglich das Aussehen der Oberfläche vor.

Iface-Widget-Typen Die Widget-Typen in ***Iface*** sind auf wenige Typen beschränkt: FRAME, LIST, GRAPH, FLOAT, INT, STRING, SELECTBUTTON, OUTPUT und ACTION. Die Anordnung der ***Iface***-Widget-Typen erfolgt als Baumstruktur:

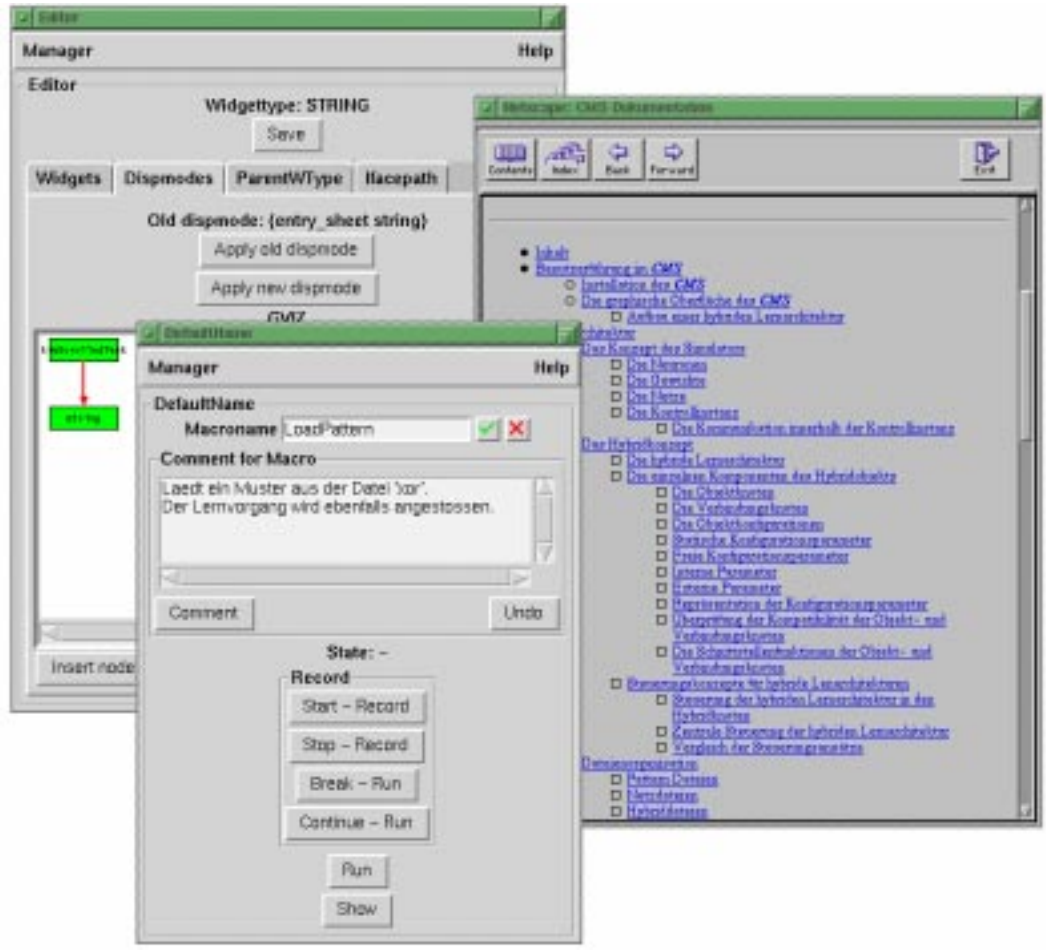


Abbildung C.1: Verschiedene Komponenten des ***Iface***.

```
.eingaben.eingabe1
.eingaben.eingabe2
```

Dabei handelt es sich bei *eingaben*, *eingabe1*, *eingabe2* um ***Iface***-Widget-Typen. Die Widget-Typen werden durch Punkte getrennt. *eingabe1* und *eingabe2* haben als Vorgänger-Widget *eingaben*.

Display-Modus Jeder ***Iface***-Widget-Type besitzt einen Display-Modus, (kurz: Dispmode) mit dem er seine Darstellung auf der Oberfläche bestimmt. Der Display-Modus für ein ***Iface***-Widget kann sich aus mehreren Display-Modi zusammensetzen. Die Display-Modi sind in einer Baumstruktur angeordnet.

Abbildung C.2 zeigt zwei verschiedene Display-Modi für den ***Iface***-Widget-Type STRING. In Abbildung C.2(a) ist ein ***Iface***-Widget-Type STRING mit dem Displaymode {*entry_sheet string*} zu sehen. Dieser Displaymode ermöglicht nur eine einzeilige Stringeingabe.

Reicht jedoch irgendwann eine einzeilige Stringeingabe nicht mehr aus, wird der Displaymode gewechselt. Dabei ist keine Änderung der eigentlichen Anwendung, d.h. vor allem des Programmcodes, notwendig. Abbildung C.2(b) zeigt den Dis-

playmode {*tixScrollText string*}. Dieser Displaymode ermöglicht nun eine mehrzeilige Stringeingabe.

(a) Displaymode {*enty_sheet string*}. **(b)** Displaymode {*tixScrollText string*}.

Abbildung C.2: Zwei verschiedene Displaymode-Darstellungen des ***Iface***-Widget-Type *STRING*.

Konfigurationsdatenbank Mit welchem Displaymode ein ***Iface***-Widget-Type dargestellt werden soll, wird in der Konfigurationsdatenbank, die im ASCII-Format vorliegt, angegeben. Ein Beispiel hierfür ist:

```
.eingabe.eingabe1 : FRAME.STRING;*;Dispmode;: \
                                {entry_sheet string}
```

Dabei bedeutet *.eingabe.eingabe1* den ***Iface***-Pfad, *FRAME* den ***Iface***-Widget-Type von *eingabe*. *eingabe1* ist das ***Iface***-Widget, das im Displaymode {*entry_sheet string*} dargestellt werden soll.

C.2 Herkömmliche Verfahren der Anwendungsanbindung

Im folgenden werden zunächst herkömmliche Ansätze zur Erstellung interaktiver Systeme beschrieben.

Bei Interaktionssystemen steht die Frage im Vordergrund, wie die Anwendung bzw. die Applikation an eine Oberfläche gekoppelt ist. Zur Erstellung interaktiver Systeme werden im wesentlichen die folgenden drei Ansätze verwendet:

abstract-link-view (ALV-System): Dieses System trennt die Applikation in **einen** Anwendungsteil (*abstraction*) und **mehrere** Oberflächenelemente (*links* und *views*).

Zur Implementierung benutzt das ALV-System `LISP`-Code und setzt mehrere Schnittstellen (*links*) zur Oberfläche (*views*) ein. Dabei läßt sich feststellen, daß die verwendeten `LISP`-Konzepte sich nur umständlich in andere Programmiersprachen übertragen lassen. Schwerwiegender macht sich jedoch der Einsatz vieler Schnittstellen zur Oberfläche bemerkbar, da dies die physikalische Trennung dieser beiden Applikationsteile erschwert.

generic-user-interface-constructor (GUIC-System): Im Gegensatz zum ALV–System benutzt das GUIC-System nur **eine** Schnittstelle zur Oberfläche. Hierbei wird die Interaktion an die eigentliche Oberfläche (*agents*) verlagert. Es wird jedoch auf seiten der Anwendung keine Modellierung der Oberflächenelemente angeboten. Darüber hinaus wird eine Daten-Interaktion ebenfalls nicht ermöglicht.

taps: Ein anderer Ansatz zur Kommunikation zwischen Anwendung und Oberfläche wird in [Ber92] verfolgt. In diesem Konzept werden die beiden Applikationsteile **vollständig** voneinander getrennt. Die Aktualisierung zwischen den beiden Programmteilen erfolgt dabei durch sogenannte *taps*. Darunter werden kleine Programmstücke verstanden, die dafür sorgen, daß zwischen einer Programmvariablen und einem oder mehreren Oberflächenelementen Konsistenz herrscht. Jeder *tap* legt bei seiner Erzeugung fest, bei welchen Ereignissen er aktiviert wird. Tritt dieses Ereignis dann ein, werden alle *taps*, die damit in Verbindung stehen, aufgerufen. In diesem Vorgehen liegt aber auch der Hauptschwachpunkt dieses Systems: je komplizierter die Oberfläche wird, desto mehr *taps* existieren.

Keiner dieser Ansätze bietet jedoch die Möglichkeit einer direkten Daten-Interaktion mit der Applikation. Aus diesem Grund wird im folgenden das Konzept zur graphischen Interaktion vorgestellt, zu dessen Modellierung das generische Benutzerinterface ***Iface*** verwendet wird.

C.3 Architektur von *Iface*

Die modulare Architektur des ***Iface*** wird in Abbildung C.3 dargestellt.

Obwohl die ALV- und GUIC-Modelle eine gute und sinnvolle Strukturierung der Anwendung bieten, ist es sehr problematisch, daß sie keine physikalische Trennung der Anwendungs- und Oberflächenkomponenten ermöglichen. Durch eine Aufteilung der *link*-Komponente des ALV-Modells entsprechend des `PICASSO`-Systems (siehe []) läßt sich dieses Problem umgehen. Dies ist bei dem System ***Iface*** realisiert. (Vgl.Abb.C3)

Eine solche Trennung ermöglicht es, daß Anwendung und Oberfläche komplett getrennt werden und im Prinzip auch asynchron voneinander arbeiten können (siehe Abbildung C.4).

Die Anwendung selbst erzeugt nur allgemeine logische Interaktionselemente, wie z.B. Eingabe, Ausgabe oder Auswahl von Alternativen. Die Ausprägung dieser Interaktionselemente in die eigentliche Oberfläche obliegt dann allein dem Oberflächencode.

Die Interaktion des Benutzers mit den Oberflächenelementen, bzw. den Darstellungsmodi als deren Oberflächenrealisierung, erfolgt analog zum GUIC-Modell durch die Oberfläche. Damit läßt sich ein Großteil der Interaktion, zu deren Ausführung die Anwendung nicht benötigt wird, durch den Oberflächencode bewältigen. Lediglich diejenigen Interaktionsmuster, die die Anwendung betreffen, wie z.B. das Auslösen einer Aktion oder die Selektion von Daten, werden an diese weitergeleitet.

Im nächsten Abschnitt wird die Modellierung der Oberfläche dazu näher erläutert.

C.3.1 Modellierung der Oberfläche

Entsprechend den oben beschrieben Konzepten werden in einem Systemkonzept die Anwendungs- und Oberflächencodes komplett getrennt. Die Kommunikation zwischen

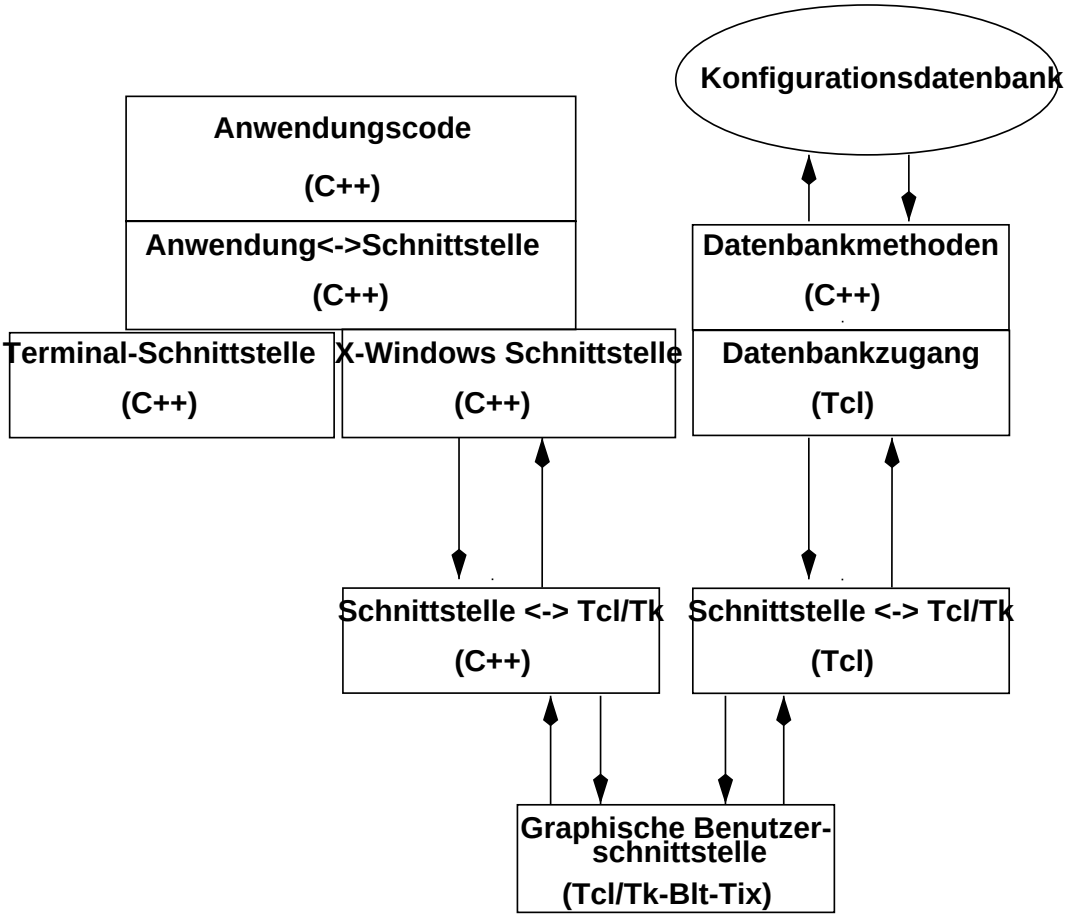


Abbildung C.3: Modulstruktur des ***Iface*** Systems.

den beiden Programmteilen erfolgt über **ein** Link-Objekt. Dieses Objekt kapselt dann die Abhängigkeiten zur verwendeten Oberfläche.

Das Anwendungsprogramm erzeugt also nur logische Interaktionselemente, die es zu *interfaces* gruppiert. Über diese Elemente kommunizieren dann mittels eines Schnittstellenobjekts (*IO*) die Oberfläche und die Anwendung miteinander.

Für jedes Interaktionselement gibt es auf der Oberflächenseite ein entsprechendes Gegenstück, das primär die Informationen über die graphische Gestaltung dieses Elements enthält. Die aktuelle Darstellung des Elements auf der Oberfläche wird durch diverse Darstellungsmodi realisiert. Insgesamt ergeben sich damit folgende Komponenten:

- *Logische Interaktionselemente* verwalten auf der Anwendungsseite die Oberfläche. Sie stellen die **alleinige** Schnittstelle der Applikation zur Oberfläche dar.

- *Physikalische Interaktionselemente* speichern auf der Anwendungsseite die benötigten Informationen für ein Ausgabedarstellungselement. Die abgespeicherten Informationen gliedern sich in einen allgemeinen Teil, der für alle Ausgabeelemente benötigt wird, und einen darstellungsspezifischen Teil, der die Informationen für eine spezifische Darstellung enthält. Dies umfaßt z.B. die Selektion und die Art der Ausgabendarstellung.

  Dabei sind noch die beiden folgenden Ausnahmesituationen zu beachten:

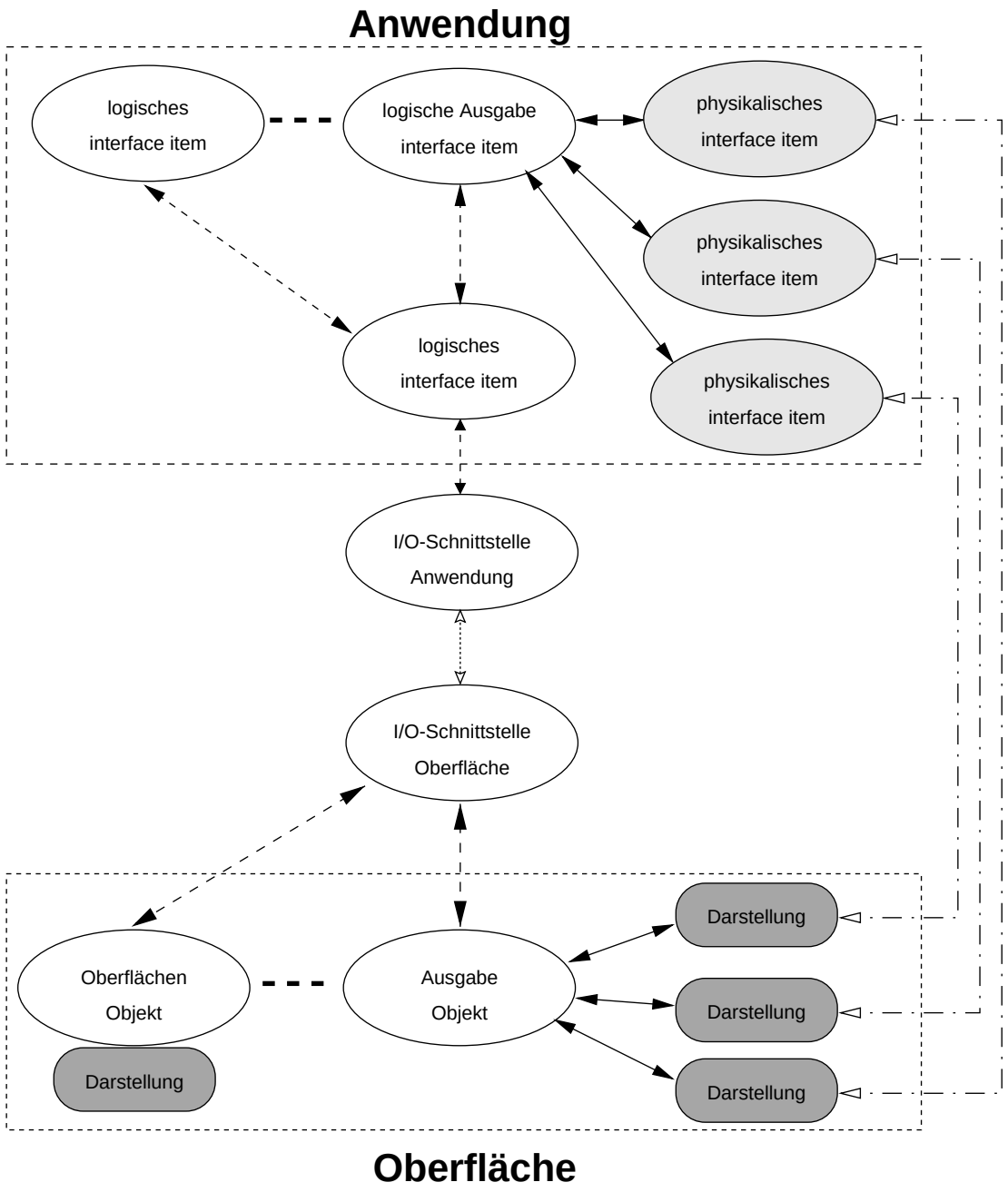


Abbildung C.4: Architektur des *Iface*-Systems ohne Ausgabe- und Interaktionsobjekte.

- **Verwalten der Ausgaberestriktionen:** Aus Effizienzgründen sollten nur die zur Darstellung benötigten Daten an die Oberfläche geschickt werden. Die dazu benötigten Informationen werden von den physikalischen Interaktionselementen verwaltet.
- **Behandeln von Selektionen:** Dies schließt sowohl die Selektionen des Benutzers, als auch die von der Anwendung bereits vorgegebenen Selektionen ein.

Auf beide Maßnahmen wird im folgenden noch näher eingegangen.

- Das *Schnittstellenobjekt* regelt die Kommunikation zwischen den logischen Interaktionselementen und den Oberflächenelementen. Es existiert nur genau ein solches Objekt.

- *Oberflächenelemente* stellen die Gegenstücke der logischen Interaktionselemente auf der Oberflächenseite dar. Sie selbst legen jedoch noch keine Darstellung des logischen Interaktionselementes fest.

- *Ausgabeobjekte* werden benötigt, um die Ausgabedaten an die Oberfläche zu schicken. Dabei handelt es sich um rein temporäre Objekte.

  Sie kapseln die zur eigentlichen Ausgabe notwendigen Daten in einem von der Ausgabedarstellung abhängigen Format. Zur Ausgabe werden diese Objekte in einen binären Datenstrom umgewandelt, über das Schnittstellenobjekt an die Oberfläche geschickt und dort wieder in ein Ausgabeobjekt zurückverwandelt.

  Aus diesem Grund erscheint es zweckmäßig, für jede verwendete Art der Ausgabe einen eigenen Ausgabeobjekttyp vorzusehen.

- Die eigentliche Darstellung der Ausgabe erfolgt durch die *Darstellungsmodi*. Erst diese Objekte legen fest, mit welchen Oberflächenwidgets die Oberflächenelemente, und damit auch die logischen Interaktionselemente, dargestellt werden. Weiterhin wird die Interaktion mit dem Benutzer durch diese Modi realisiert.

- *Ausgabedarstellungen* sind für die Ausgabe spezialisierte Darstellungsmodi. Sie sind mit ihrer Schnittstelle auf die Bedürfnisse der physikalischen und logischen Ausgabeinteraktionselemente abgestimmt. Sie implementiert die tatsächliche Visualisierung der Daten und die Selektion durch den Benutzer.

Damit sind alle Komponenten beschrieben. Die spezielle Behandlung der Selektion und die Notwendigkeit von Ausgaberestriktionen in diesem Zusammenhang wird anschließend eingehender erläutert.

Ausgaberestriktionen

Um in einer Ausgabedarstellung, die n Dimensionen visualisieren kann, einen m-dimensionalen Datensatz darzustellen, müssen für $m-n$ Dimensionen bestimmte Werte vorgegeben werden. Die Information darüber, welche Dimensionen mit welchen Werten vorgegeben sind, kann jedoch nur vom Benutzer festgelegt werden und muß bei der Erzeugung der Ausgabedarstellungen bzw. bei der Änderung von Parametern übermittelt werden.

Dies trifft sowohl bei der Erzeugung von Ausgabedarstellungen, als auch bei der Änderung des sichtbaren Ausschnitts zu, da dies am einfachsten über eine Änderung der Ausgaberestriktionen zu erreichen ist.

Anhand der Ausgaberestriktionen sollte das physikalische Interaktionselement diejenigen Daten ermitteln, die für die Ausgabe benötigt werden und ein entsprechendes Ausgabeobjekt erzeugen.

Für Benutzerselektionen bestimmen die Restriktionen diejenigen Dimensionswerte, die nicht in der Darstellung angezeigt werden, und damit auch nicht vom Benutzer selektiert wurden. Bei Programmselektionen hingegen stellen die Restriktionen einen Filter dar, der festlegt, welche Programmselektionen dem Benutzer vorgegeben werden.

Selektionen

Da Selektionen lediglich einen Sonderfall der Interaktion des Benutzers mit der Anwendung darstellen, werden sie primär von der Oberfläche aus abgehandelt. Die Behandlung der Selektion aus Benutzersicht, also z.B. das Aufziehen eines Selektionsrahmens, wird von jeder Ausgabedarstellung getrennt behandelt. Diese getrennte Behandlung für jede Ausgabedarstellung wird aus Effizienzgründen gewählt, da die Ausgabedarstellungen damit nur die minimal benötigten Daten erzeugen.

Diese Selektionsinformationen hängen damit auch immer von der gewählten Darstellung ab. Damit aber die Anwendung diese Daten verarbeiten kann, müssen die fehlenden Informationen wieder hinzugefügt werden. Analog zu den Ausgaberestriktionen, die die nicht mehr benötigten Dimensionen entfernen, findet jetzt der entgegengesetzte Prozeß statt: die Selektionsinformationen werden um diejenigen Dimensionen erweitert, die in den Ausgaberestriktionen angegeben werden. Dieser Prozeß ist in den physikalischen Interaktionselementen angesiedelt, da hier die Ausgaberestriktionen für die entsprechenden Ausgabedarstellungen existieren.

In der anderen Richtung müssen Selektionen auch von der Anwendung her vorgegeben werden können, z.B. um den Benutzer auf interessante Bereiche im Datensatz hinzuweisen. Analog zur Ausgabe werden hier die Daten entsprechend den Ausgaberestriktionen wieder reduziert. Zusätzlich zur Ausgabe muß jedoch geprüft werden, ob diese Selektionsdaten in dem Bereich liegen, der durch die Ausgaberestriktionen bestimmt wird. Die Restriktionen legen für bestimmte Dimensionen Werte fest, die für die Ausgabe benutzt werden, d.h. die vorgegebenen Selektionen enthalten für diese Dimensionen dieselben Werte.

Die oben beschriebenen Richtlinien treffen nur für Selektionen in Domänenbereich der Daten zu, d.h. Selektion über die Indizes. Selektionen über den Wertebereich hingegen sind einfach zu behandeln, da hier nur zwei Werte auftreten.

C.3.2 Entwurfskonzept unter *Iface*

In der Applikation wird nur eine kleine Menge von Funktionen verwendet, die die Interaktionen zwischen Benutzer und Applikation in einer darstellungsunabhängigen Weise definieren. Die zur Verfügung stehenden Komponenten werden in Abbildung C.5 aufgeführt. Die eigentliche Darstellung dieser Interaktionselemente wird von einer applikationsunabhängigen Komponente übernommen.

Wie bereits erwähnt, werden alle Konfigurationen des graphischen Aussehens in einer applikationsspezifischen Konfigurationsdatenbank abgespeichert. Wenn das Aussehen nicht den Erfordernissen entspricht, kann mit Hilfe eines online Editors die ***Iface***-Konfigurationsdatenbank direkt manipuliert werden (siehe Abbildung C.12).

Die Interaktion mit der Anwendung wird darüber hinaus durch ein Hilfesystem und eine applikationsunabhängige Makroverarbeitung unterstützt.

C.3.3 Veränderung der Darstellung in *Iface*

Unter einer Veränderung der Oberfläche in ***Iface*** ist die Möglichkeit zu verstehen, einem ***Iface***-Widget einen anderen Displaymode zuzuordnen. Wird einem ***Iface***-Widget ein anderer Displaymode zugeordnet, dann muß der Programmcode der Callback-Funktion nicht verändert werden. Dies ist nur aufgrund einer strikten Trennung von

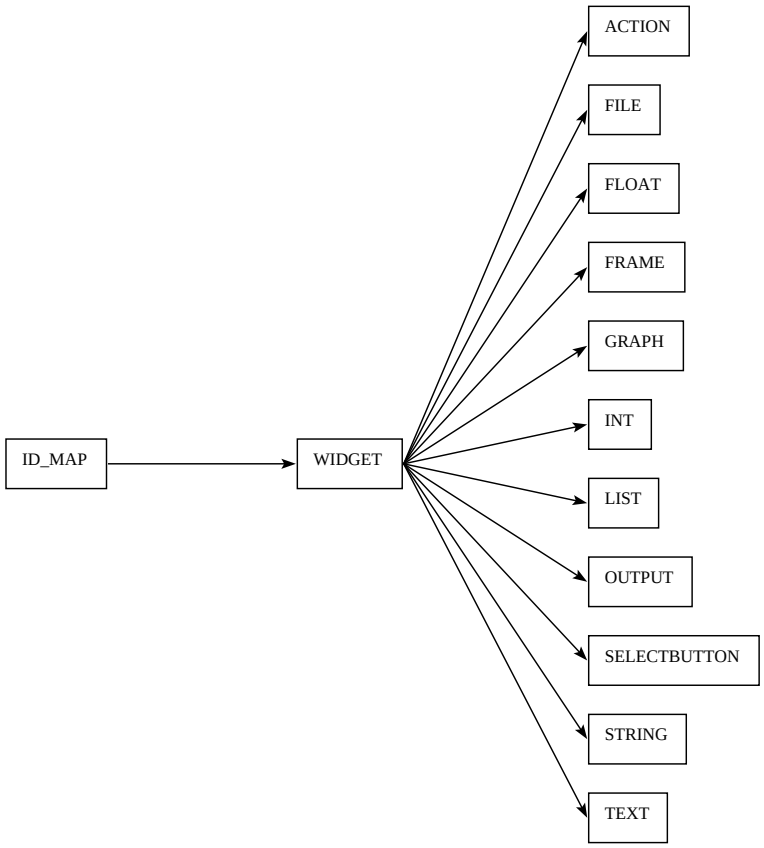


Abbildung C.5: Basiselemente des ***Iface***.

Oberfläche und Anwendung möglich.

Beispielsweise kann ein Push-Button, der nach Betätigung ein neues Fenster öffnet, in dem dann der eigentliche Displaymode dargestellt wird unabhängig von der Applikationsprogrammierung verwendet werden. Dies ist ein Displaymode, der für alle ***Iface***-Widget-Typen gewählt werden kann.

Die nachfolgende Auflistung zeigt die Zuordnung von einigen Displaymodes zu ihren ***Iface***-Widgets:

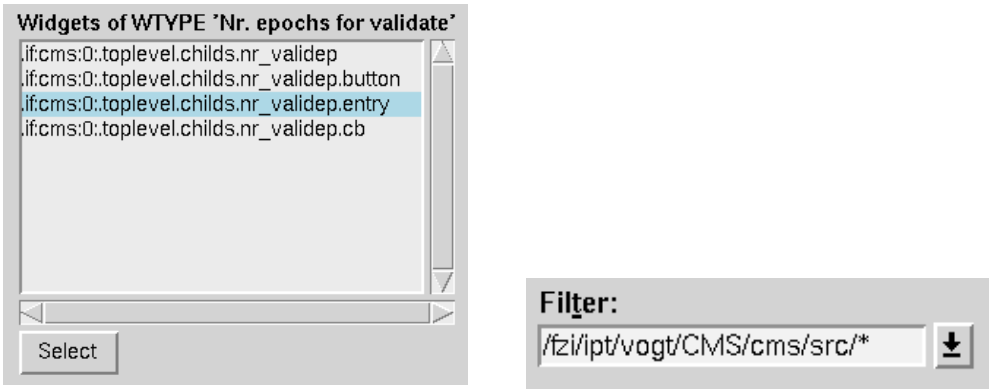
FRAME Ein Frame kann aus einem Rahmen mit einer übergeordneten Überschrift bestehen. Eine andere Form eines Frames ist ein Rahmen, in dem die Überschrift im Rahmen steht (siehe Abbildung C.6(a)). Bei einem Notebook handelt es sich ebenfalls um ein Frame. Dabei sind die Oberflächenelemente, die sich im Frame befinden, in mehrere Frames aufgeteilt (siehe Abbildung C.6(b)).

LIST Eine normale Listbox und eine Combobox sind die möglichen Displaymodes für diesen ***Iface***-Widget-Type (siehe die Abbildungen C.7(a) und C.7(b)). Au-

(a) Frame **(b)** Notebook

Abbildung C.6: Zwei verschiedene Displaymode-Darstellungen des ***Iface***-Widget-Type *FRAME*.

ßerdem können auch alle Nachfolger-***Iface***-Widgets von diesem ***Iface***-Widget gleich in der Listbox dargestellt werden (siehe die Listbox "Configurations" in der Abbildung C.6(b)).



(a) Listbox **(b)** Combobox

Abbildung C.7: Zwei verschiedene Displaymode-Darstellungen des ***Iface***-Widget-Type *LIST*.

GRAPH Ein GRAPH kann mittels zweier Listboxen oder einer graphischen Darstellung realisiert werden (siehe Abbildung C.8).

INT Ein editierbares Eingabefeld bildet den Displaymode für diesen ***Iface***-Widget-Type (vgl. Abbildung C.9(a)). Außerdem kann auch ein *Spin-Button* als Displaymode gewählt werden (vgl. Abbildung C.9(b)).

FLOAT Die selben Displaymodes wie bei INT.

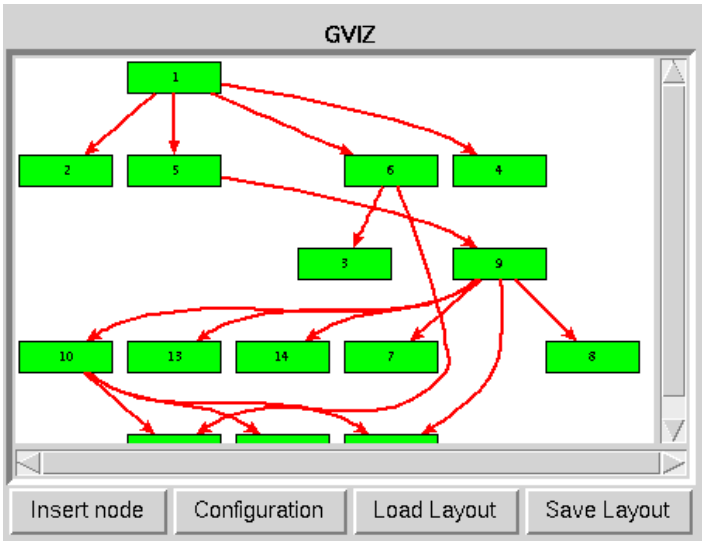


Abbildung C.8: Displaymode-Darstellungen des ***Iface***-Widget-Type *GRAPH*.

(a) Einzeiliges Eingabefeld **(b)** Spin-Button

Abbildung C.9: Unterschiedliche Darstellungen des ***Iface***-Widget-Type *INT*.

STRING Ein einzeiliges Editierfeld oder ein mehrzeiliges Editierfeld werden den Displaymodes für diesen ***Iface***-Widget-Type zur Verfügung gestellt (siehe dazu die Abbildungen C.2(a) und C.2(b)).

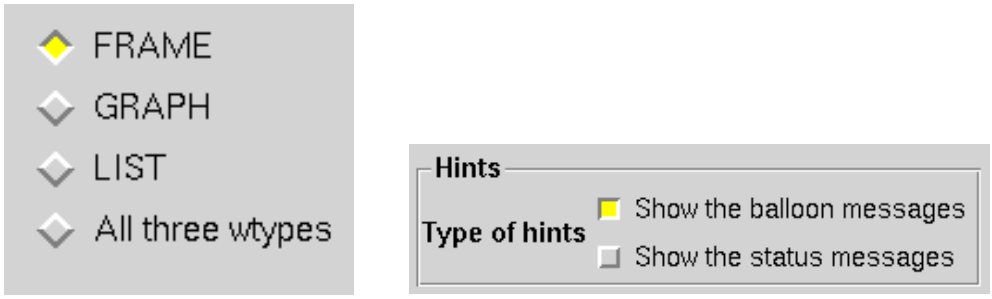
SELECTBUTTON Dieser kann mit den üblichen Radio- oder Check-Buttons (vgl. die Abbildungen C.10(a) und C.10(b)) oder mit einfachen Buttons dargestellt werden. Zusätzlich ist eine Darstellung mit einem Pop-Up-Menü möglich.

ACTION Der übliche Displaymode für den ***Iface***-Widget-Type ACTION besteht aus einem Push-Button. Aber auch ein Pop-Up Menü kann als Displaymode gewählt werden.

FILE Der Widget-Type FILE wird als Push-Button dargestellt, bei dessen Betätigung ein Datei-Auswahlfenster geöffnet wird (vgl. Abbildung C.11).

Konfiguration der Widgets unter *Iface*

Die Konfiguration der Tk-Widgets in ***Iface*** erfolgt über den Tk-Befehl *configure*. Die Optionen für eine Konfiguration unterscheiden sich je nach Tk-Widget. Die entsprechenden Optionen in der ***Iface***-Konfigurationsdatenbank werden über den Operationscode *Config* angegeben. Beispiele für Konfigurationen sind die Hintergrundfarbe, die Größe oder die 3-dimensionale Darstellung der Tk-Widgets.



(a) Radio-Button **(b)** Check-Button

Abbildung C.10: Verschiedene Displaymode-Darstellungen des ***Iface***-Widget-Type *SELECTBUTTON*.

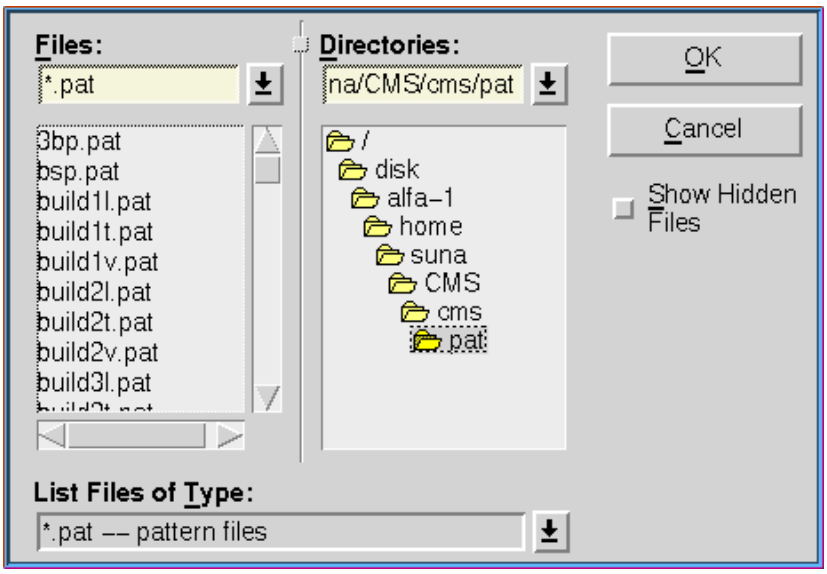
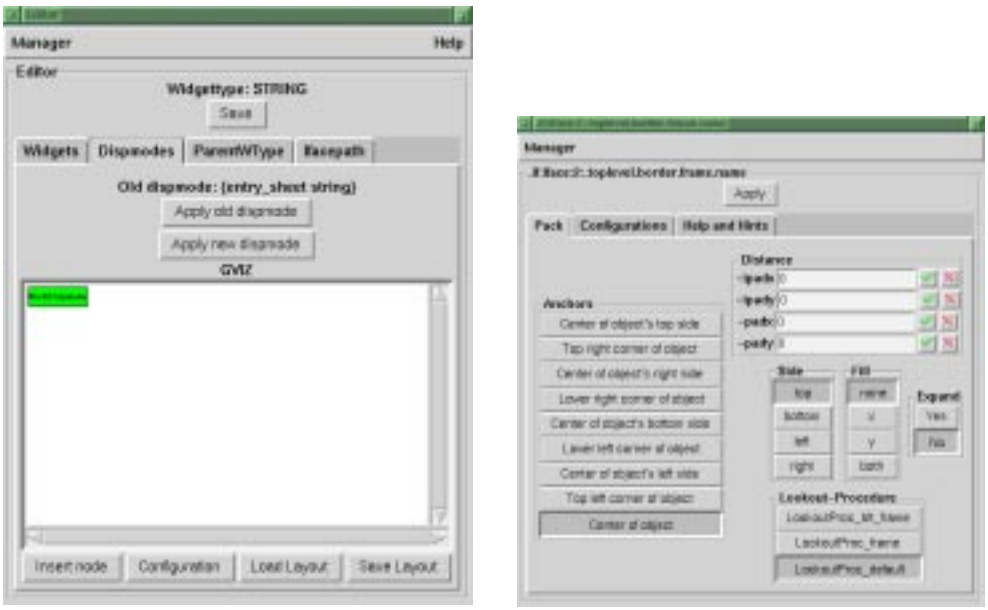


Abbildung C.11: Displaymode-Darstellung des ***Iface***-Widget-Type *FILE*.

C.4 Konfigurationseditor

Der Entwurf einer graphischen Oberfläche ist mit hohem Aufwand verbunden. Neben dem eigentlichen Entwurf ist es oft notwendig, die Oberflächenelemente neu zu konfigurieren oder sie neu anzuordnen. Zwar stellt der Entwurf einer Oberfläche unter ***Iface*** gegenüber dem Entwurf einer Oberfläche mit den APIs der verschiedenen Fenstersysteme bereits eine Vereinfachung dar, doch bleibt immer noch der Eintrag in die Konfigurationsdatenbank. Durch einen graphischen Editor kann der Entwurf, die Konfiguration und die Plazierung von Oberflächenelementen erleichtert werden. Die Konzeption wird dadurch vereinfacht, daß der Zugriff auf die ***Iface***-Widgets immer über den ***Iface***-Pfad erfolgt. Auf diese Weise wird die Plattformunabhängigkeit von ***Iface*** beibehalten. Durch die ***Iface***-Konfigurationsdatenbank können, auch noch auf bereits bestehenden Anwendungen, Änderungen durchgeführt werden.



(a) Display–mode selektion **(b)** Darstellungskonfiguration

Abbildung C.12: Ausschnitt aus der Funktionalität des interaktiven online Konfigurierungseditors.

C.4.1 Ziele eines Oberflächeneditors unter *Iface*

Neben den schon genannten Möglichkeiten bietet der Editor unter ***Iface*** auch noch weitere Möglichkeiten:

1. Auswahl eines entsprechenden Displaymodes für ein ***Iface***-Widget.
2. Auswahl des Eltern-***Iface***-Widget für das ***Iface***-Widget, das bearbeitet wird.
3. Einstellung des ***Iface***-Pfades, um auf die ***Iface***-Widgets zugreifen zu können.
4. Konfiguration der einzelnen Tk-Widgets in einem ***Iface***-Widget.
5. Plazierung der Tk-Widgets in einem ***Iface***-Widget und Plazierung der ***Iface***-Widgets untereinander.
6. Die Änderungen durch den Editor sind sofort sichtbar. Bei einem nochmaligen Starten der Anwendung soll die Oberfläche genauso aussehen, wie es durch den Editor herbeigeführt wurde.
7. Der Editor kann einfach über eine Shortcut-Kombination gestartet werden. Der Anwender muß für ein beliebiges Oberflächenelement deshalb nicht lange nach dem ***Iface***-Widget suchen, das er verändern möchte.

Eine weitere Anforderung an den Editor bildet die Einstellung von Sprungmarken für das Hilfesystem und die Ausgabe von Balloon- bzw. Statusnachrichten. Diese Anforderungen betreffen allerdings nicht das Oberflächendesign.

Bisher war es üblich, die ***Iface***-Konfigurationsdatenbank zu editieren, und dann die Veränderungen vorzunehmen. Die Veränderungen, die durch den Editor herbeigeführt werden, sollen in der ***Iface***- Konfigurationsdatenbank unter einem Dateinamen abgelegt und in eine ASCII-***Iface***- Konfigurationsdatei abgespeichert werden können.

C.4.2 Veränderbarkeit der Oberfläche

In den gängigen Oberflächensystemen ist eine einfache Veränderung der Oberflächenelemente nicht möglich, da sie statisch sind. Unter der Veränderbarkeit einer Oberfläche versteht man den Austausch von Oberflächenelementen, ohne daß dies irgendwelche Auswirkungen auf den Anwendercode hat. Ein Beispiel hierfür ist der Austausch einer einzeiligen durch eine mehrzeilige Stringeingabe. Dabei ist der Austausch unter ***Iface*** besonders einfach. Zwar ist eine Veränderung der Oberfläche unter anderen Systemen auch möglich, aber mit höherem Aufwand verbunden. Vor allem erfordert sie Kenntnisse über das jeweilige Oberflächensystem.

Besonders einfach ist eine Veränderung in ***Iface***, über einen Editor. Durch diesen Editor ist es möglich, die Oberfläche während der Laufzeit eines Programms zu verändern. Der Anwender muß nur seinen gewünschten Displaymode über den Editor einstellen.

Die Displaymodes in ***Iface*** sind als Baumstruktur aufgebaut. Es ist deshalb günstig, wenn der Displaymode über einen Graphen eingegeben werden kann.

C.4.3 Plazierung und Konfiguration von *Iface*-Widgets

Der Anwender muß nicht mühsam die Konfigurationen im Programmcode bestimmen oder sie in komplexen Resourcendateien eintragen (vgl. vor allem Motif).

Die Optionen für die Plazierung durch den *Packer* sind immer dieselben und können deshalb vom Anwender über ein Auswahlmenü eingestellt werden. Die anderen Plazierungsmöglichkeiten *grid* und *blt_table* konnten aus Zeitgründen nicht konzipiert werden. Die Optionen für die Tk-Konfiguration unterscheiden sich häufig, deshalb ist eine Auflistung in einer Listbox sinnvoll. Der Anwender eines Editors kann diese Optionen dann entsprechend einstellen. Dieses Auswahlmenü ist im Editor integriert.

C.4.4 Einstellung der Hilfeoptionen

Mit dem Editor werden auch die Optionen für das Hilfesystem (Sprungmarke, Text der Balloon- und Status-Nachricht) eingestellt. Der Anwender kann diese Optionen in ein Eingabefeld eintragen.

C.4.5 Vorteile des Editors als Oberflächengestalter

Ein Editor unter ***Iface*** bringt als Gestalter von Oberflächen zu den genannten Vorzügen noch folgende Vorteile:

- Der Editor erspart den sonst üblichen Kreislauf der Programmentwicklung, also den Eintrag in die ***Iface***-Konfigurationsdatenbank und das Starten der Anwendung.

- Der Editor erfordert fast keine Kenntnisse der graphischen Oberfläche und der Anwendung (vor allem des ***Iface***-Pfades). Dadurch ist es möglich, daß jeder Anwender seine Oberfläche ergonomisch anpassen kann. Der Anwender sieht nur die Oberfläche und kann sich die Ergebnisse seiner Veränderungen während der Laufzeit sofort ansehen.

- Da die Veränderungen der ***Iface***-Konfigurationen (Displaymode, Plazierung, Tk-Konfigurationen und Hilfeeinstellungen) durch den Editor automatisiert sind, sind sie weniger fehleranfällig als durch den Eintrag in die Konfigurationsdatenbank.

- Wird ein neuer Displaymode für ein ***Iface***-Widget erstellt, kann aufgrund der ***Iface***-Architektue auch dieser über den Editor eingestellt werden.

- Die Konfigurationsmöglichkeiten der Displaymodes sind Transparent an einer Stelle zusammengefaßt, und müssen nicht in Handbüchern oder sonstigen Dokumentationen gesucht werden.

Doch dieser graphische Editor weist zur Zeit noch folgende Nachteile auf:

- Die Displaymodes für ***Iface*** müssen erst entworfen werden, um sie dem Anwender zur Verfügung zu stellen.

- Die Einstellungen über den Tk-Packer sind gewöhnungsbedürftig.

C.5 Makroverarbeitung

Das ***Iface*** bietet die Möglichkeit, unabhängig von der Applikation Makros anzugeben, die interaktiv die Aktionen des Benutzers aufzeichnen und später modifiziert werden können (siehe Abbildung C.13).

Das ergonomische Ziel der Makroverarbeitung ist es, daß eine Sequenz von Benutzeraktivitäten auf der graphischen Benutzeroberfläche ***Iface***, nach einmaligem Aufzeichnen, beliebig oft wiederholt werden kann. Sie bewirkt dadurch die Fehleranfälligkeit und entlastet den Benutzer. Damit die Makroverarbeitung eine weitere Zeitersparnis gegenüber der interaktiven Benutzung aufweist, werden Fenster, die nur temporär während der Makroverarbeitung benötigt werden, nicht geöffnet. Die Motivation für eine Makroverarbeitung besteht deshalb darin, dem Anwender eine Möglichkeit zur Zeitersparnis gegenüber der manuellen Eingabe zu bieten.

C.5.1 Voraussetzung für eine Makroverarbeitung unter *Iface*

Die Möglichkeit, Makros durch "Vormachen" zu erzeugen, ist benutzerfreundlicher als durch eine textuelle Programmierung. Bei ***Iface*** können wegen der Display-Modi z.B. vier Buttons auf unterschiedliche Arten Dargestellt werden, unter anderem auch als ein Pull-down-Menü. Trotzdem ist ein Makro, das bei der Darstellung der vier Buttons aufgezeichnet wurde, bei jeder anderen beliebigen Darstellungsart lauffähig. Die Makros unter ***Iface*** sind unabhängig von der Darstellungsart.

Als nächstes werden alle Aktionen bzw. Eingaben aufgelistet, die ein Anwender unter ***Iface*** ausführen kann. Diese werden Benutzt um die Anzahl der Eingaben und

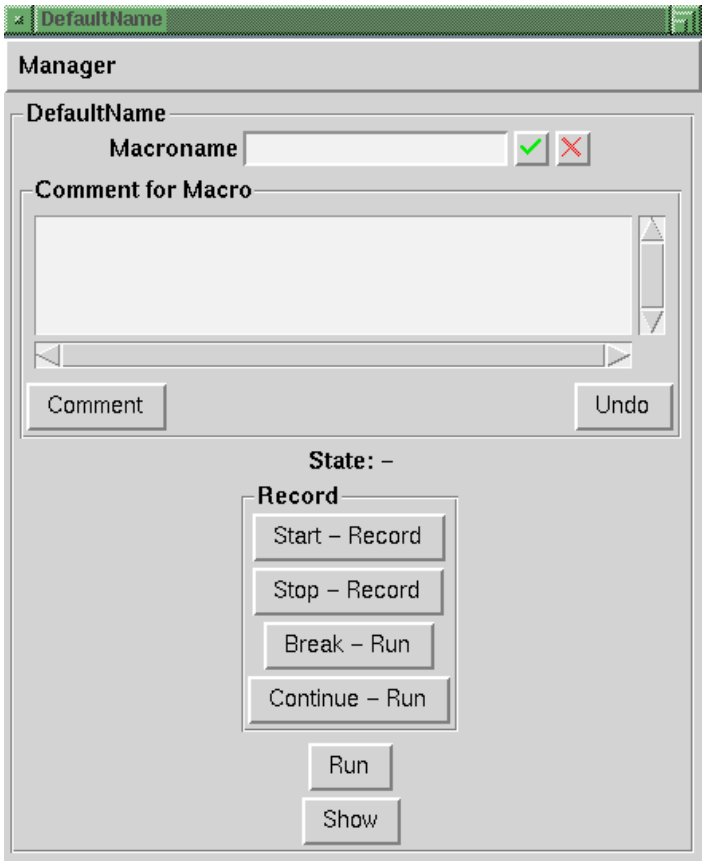


Abbildung C.13: Verwaltung eines einzelnen Makros.

die zu protokollierenden Parameter des Makros festzulegen. Folgende Eingaben bzw. Aktionen der einzelnen ***Iface***-Widgets sind möglich (in den Klammern stehen jeweils die betroffenen Widget-Typen):

- Bei Eingabe einer Float-, Integer-Zahl, String und Dateinamen: Vom Benutzer kann entweder ein fester Wert definiert werden, oder bei der späteren Makroausführung ein Wert eingegeben werden (FLOAT, INT, STRING und FILE).

- Beim Einfügen eines Elements in eine Liste: Der Benutzer kann die Stelle bestimmen, an der das Element in die Liste eingefügt werden soll. Wählt der Benutzer keine Stelle in der Liste aus, dann wird das Element immer an das Ende der Liste angehängt (LIST).

- Beim Löschen eines Elements aus einer Liste: Es wird immer das Element gelöscht, das der Benutzer aus der Liste ausgewählt hat (LIST).

- Beim Auswählen eines Elements aus einer Liste: Es wird immer das Element ausgewählt, das vom Benutzer bestimmt wird (LIST).

- Beim Einfügen eines Knotens in einen Graphen: Ein Knoten wird zu einem Graphen hinzugefügt. Es ist keine bestimmte Stelle für das Einfügen im Graphen vorgesehen (GRAPH).

- Beim Löschen eines Knotens aus einem Graphen: Es wird immer der Knoten gelöscht, der vom Benutzer ausgewählt wurde (GRAPH).
- Beim Auswählen eines Knotens aus einem Graphen: Es wird immer der Knoten ausgewählt, der vom Benutzer bestimmt wurde (GRAPH).
- Beim Einfügen einer Kante in einen Graphen: Die Knoten, die zur neuen Kante gehören, müssen vom Benutzer eingegeben werden (GRAPH).
- Beim Löschen einer Kante aus einem Graphen: Es wird immer die Kante gelöscht, die vom Benutzer ausgewählt wurde (GRAPH).
- Beim Auswählen einer Kante aus einem Graphen: Die Kante, die ausgewählt werden soll, muß vom Benutzer bestimmt werden (GRAPH).

Die hier aufgelisteten Eingaben bzw. Aktionen zeigen, daß die Erzeugung des Makrocodes durch Vormachen unter ***Iface*** mit relativ wenig Aufwand verbunden ist. Schließlich ist es möglich, da die Aktionen bzw. Eingaben in den ***Iface***-Widgets abgefangen werden, ein Makro unter einem anderen Displaymode ablaufen zu lassen, als unter dem er aufgezeichnet wurde.

Unter ***Iface*** gibt es eine Trennung von Anwendung und Oberfläche. Deshalb kann sich eine Anwendung die Angaben, die sie sich normalerweise vom Anwender durch die Oberfläche ***Iface*** holt, auch direkt geben lassen. Bei der Makroverarbeitung bedeuten dies, daß die Parameter dem Makro direkt übergeben werden. Damit kann ein Makro auch ohne Oberfläche laufen.

Eine "Undo"-Funktion zu entwerfen, ist aufgrund des Aufbaus von ***Iface*** mit der bereits erwähnten Trennung von Anwendung und Oberfläche nicht allgemein möglich. Eine Voraussetzung dafür ist, daß die jeweilige Anwendung ebenfalls immer eine "Undo"-Funktion zur Verfügung stellt.

C.5.2 Beeinflussung des Makroablaufs

Ein Anwender ist in der Lage, eine andere Eingabe bzw. Aktion auf ein ***Iface***-Widget beim Ablaufen des Makros zu machen, als dies beim Aufzeichnen des Makros geschehen ist. Ein Beispiel hierfür ist die Eingabe einer Zahl. Angenommen, die Eingabe der Zahl ist bei der Makroaufzeichnung "5". Die Eingabe beim Ablaufen des Makros soll aber den Wert "10" annehmen. Um dies zu ermöglichen, muß das Makro seinen Ablauf solange anhalten, bis der Anwender die Zahl "10" eingegeben hat. Erst danach setzt es seine Ausführung fort.

C.6 Hilfesystem

Wie die Makroverarbeitung ist auch das Hilfesystem so konzipiert, daß es die Möglichkeiten von ***Iface*** nutzt, und dessen Vorteile durch das Hilfesystem nicht eingeschränkt werden. Die Aufgabe des Hilfesystems ist die Vereinfachung und Vereinheitlichung der Hilfe unter unterschiedlichen Applikationen. Das realisierte Hilfesystem bietet sowohl aktive als auch passive kontextsensitive Hilfe an.

Die Aktive Hilfe bedient sich der sogenannten Balloon-nachrichten, die unmittelbar neben dem Mauszeiger erscheinen, und einer Hilfszeile, die am unteren Rand des

Fensters längere Nachrichten präsentieren kann. Hingegen wird die passive Hilfe durch den Benutzer direkt aktiviert, und es erschient ein HTML-Dokument zu Kontext in dem sich der Mauszeiger befindet oder der Benutzer kann im Dokument selbst nach Stichworten oder Indizes Suchen.

Kontextsensitiv ist die Hilfe wegen der Berücksichtigung der Position des Mauszeigers. Durch die hierarchische Anordnung von ***Iface***, kann zu jedem Kontext in der Hierarchie die nächste Hilfe ermittelt werden.

C.6.1 Layout des Hilfesystems

Als Hilfesystem wird der plattformunabhängige Hypertext HTML verwendet. Mit dem Tool *LaTeX2HTML Translator* (vgl. [Dra95]) werden die bereits vorhandenen LaTeX-Dateien in interpretierbare HTML-Dateien übersetzt. Mit Hilfe der im LaTeX-Dokument definierten Marken, kann jede beliebige Stelle über die daraus generierten Sprungmarken im HTML-Dokument angesprungen werden. Dadurch wird ein kontextsensitives Hilfesystem gebildet.

C.6.2 Vorteile des *Iface*-Hilfesystems

Das ***Iface***-Hilfesystems bietet folgende Vorteile:

- Das Hilfesystem ist kontextsensitiv.
- Durch die optionale Verfügbarkeit der Balloon- und Statusnachrichten kann eine aktive Hilfe erfolgen.
- Die Vorteile von Hypertext, durch die einbeziehung von Bildern, Ausführbaren Programmteilen (CGI's, Servlets, Applets, ...).
- Bereits vorhandenes LaTeX-Manual zu einem Anwenderprogramm, das als graphische Oberfläche ***Iface*** verwendet, kann in HTML-Dateien umgewandelt werden und die dort definierten Marken können durch das ***Iface***-Hilfesystem angesprungen werde. Im Gegensatz dazu muß zum Beispiel unter Windows ein bereits geschriebenes Manual in eine für den Compiler verständliche Syntax umgewandelt werden.
- Die Angabe, welche Stelle in den HTML-Dateien angesprungen wird, erfolgt über eine sprechende textuelle Sprungmarke (nicht über eine Nummer wie in Windows üblich).
- Mit Hilfe des Editors kann das Hilfesystem benutzerfreundlich konfiguriert werden.
- Die Konfigurationen für ein ***Iface***-Widget müssen bei der Darstellung eines bestimmten Displaymodes nur einmal angeben werden. Danach gelten sie für alle ***Iface***-Widgets, die mit demselben Displaymode ausgegeben werden.
- Durch den Eintrag in die ***Iface***-Konfigurationsdatenbank können auch fertige Anwendungen um das Hilfesystem erweitert werden.

- Der Benutzer kann den Grad der Hilfe Spezifizieren, und dadurch kann das Hilfesystem an unterschiedlich versierte Benutzergruppen individuell angepaßt werden.
- Das Hilfesystem ist Plattformunabhängig.

Das Hilfesystem hat zur Zeit folgende Nachteile:

- Ein Stichwort- und Inhaltverzeichnis muß bereits im LaTeX angelegt werden.
- Die Qualität der Aufbereitung des LaTeX Dokuments in das HTML-Format ist von dem Programm latex2html abhängig.

Anhang D

Klassifikatorperformanz

Um die Leistungsfähigkeit eines Klassifikators zu bewerten, muß die Klassifikationsbeurteilung auf einer objektiven Grundlage basieren. Insbesondere im Fall der ILD stellt sich die Frage, wie gut der aufgestellte Klassifikator K für die unklassifizierten und zukünftigen Daten ist.

Wie bereits angesprochen, steht normalerweise nicht die gesamte Datenmenge $\mathcal{X}$ als Beispiele zur Verfügung, noch deckt sie den Merkmalsraum $\mathcal{M}$ vollständig ab. Aus der Datenmenge $\mathcal{X}$ liegen die Daten meistens nur zu einem Bruchteil wiederum in Form von klassifizierten Beispielen $\mathcal{B}$ vor. Daraus ergeben sich folgende Beziehungen:

$$\mathcal{B} \subseteq \mathcal{X} \subseteq \mathcal{M} \tag{D.1}$$

Hieraus folgt, daß zunächst zwei Fragen zu lösen sind: wie gut ist der Klassifikator K auf bekannten klassifizierten Beispielen $\mathcal{B}$ (siehe Definition 3.1), und welche Ergebnisse können aus dieser Kenntnis für die Menge aller Merkmale $\mathcal{M}$ erwartet werden?

Um diese Fragen zu beantworten, wird zuerst der Fehler des Klassifkators untersucht.

D.1 Fehler

Der Fehler eines Klassifikators ist definiert als die Zuordnung eines Beispiels $\vec{b} \in \mathcal{B}$ zu einer falschen Klasse $\vec{c_f} \in \mathcal{C}$. Aufgrund der Tatsache, daß die Menge $\mathcal{X}$ sich nicht nur aus den Beispielen $\mathcal{B}$ zusammensetzt, sondern auch noch eine Vielzahl unklassifizierter Daten enthält, kann die Entscheidung, ob ein Datum richtig oder falsch klassifiziert wurde, lediglich auf den Beispielen $\mathcal{B}$ getroffen werden.

Um die Mächtigkeit des Klassifikators abschätzen zu können, werden Instrumente zu seiner Beurteilung benötigt. Als einfachstes Meßinstrument steht dafür die Fehlerrate E_{emp} zur Verfügung (vgl. [WK91]):

$$E_{emp} := \frac{\text{Anzahl der Fehlklassifikationen}}{\text{Anzahl der Beispiele}} \tag{D.2}$$

Dieses Maß zur Bestimmung der Klassifikationsgüte hängt stark von den ausgewählten Beispielen ab. Mit steigender Anzahl der Beispiele konvergiert diese Fehlerrate gegen die *wahre Fehlerrate* $E_{\mathcal{M}}$.

Definition D.1 (wahre Fehlerrate)
Die wahre Fehlerrate eines Klassifikators K ist definiert als:

$$E_{\mathcal{M}} := \lim_{\{\vec{x} \mid (\vec{x},\vec{c}) \in \mathcal{B}\} \to \mathcal{M}} \frac{|\{\vec{c} \mid (\vec{x},\vec{c}) \in \mathcal{B} \text{ und } K(\vec{x}) \neq \vec{c}\}|}{|\mathcal{M}|}$$

In der Praxis besteht das Problem, daß oft nur eine geringe Anzahl von Beispielen existiert, und deshalb die *wahre Fehlerrate* nicht direkt berechnet werden kann. Zudem müssen die Beispiele repräsentativ und in einer für den Klassifikationsalgorithmus ausreichenden Anzahl vorhanden sein (siehe Abschnitt D.3).

Um das Fehlerverhalten eines Klassifikators in einer Anwendung zu beurteilen, wird weiterhin eine Übersicht über seine Schwierigkeiten mit der Problemstellung benötigt. Die Beziehung zwischen der Klassifikation und der tatsächlichen Klasse wird durch die *Konfusionsmatrix* $\mathcal{F}$ ausgedrückt. Sie stellt die vom Klassifikator ausgegebenen Klassen $K(\vec{x})$ den tatsächlichen Klassen $\vec{c}$ gegenüber. Die Anzahl korrekt klassifizierter Beispiele wird in der Diagonale N aufgetragen, und die falsch klassifizierten werden entsprechend dem Fehler in die Matrixfelder eingetragen.

$$\mathcal{F} := \begin{pmatrix} \mathcal{F}_{11} & \mathcal{F}_{12} & \dots & \mathcal{F}_{1n} \\ \mathcal{F}_{21} & \mathcal{F}_{22} & \dots & \mathcal{F}_{2n} \\ \vdots & & \ddots & \vdots \\ \mathcal{F}_{n1} & \mathcal{F}_{n2} & \dots & \mathcal{F}_{nn} \end{pmatrix} \quad \text{mit} \quad \mathcal{F}_{ij} = |\{K(\vec{x}) = \vec{c}_i \mid (\vec{x},\vec{c}_j) \in \mathcal{B}\}| \tag{D.3}$$

Ausgabe des Klassifikators	Tatsächliche Klasse		
	A	B	C
A	100	0	0
B	0	97	5
C	0	3	75

Tabelle D.1: Beispiel einer Konfusionsmatrix für ein Drei-Klassen-Problem. Die Fälle von Klasse B wurden nur in 97 Fällen korrekt zu B gehörig erkannt, 3 Fälle wurden fälschlicherweise C zugeordnet. Entsprechendes gilt für die Fälle der Klasse C – nur die Klasse A wurde komplett korrekt identifiziert.

Ein Beispiel für eine solche Konfusionsmatrix wird in Tabelle D.1 dargestellt. Wie aus der Beispieltabelle hervorgeht, wurde Klasse A korrekt klassifiziert. Im Gegensatz dazu wurden die Fälle aus Klasse B nur in 97 Fällen korrekt als zur Klasse B gehörig erkannt, während drei fälschlicherweise der Klasse C zugeordnet wurden. Bei der Klassifikation der Fälle der Klasse C wurden 75 korrekt und fünf falsch klassifiziert, da sie der Klasse B zugeordnet wurden.

D.2 Kosten und Risiken

Des Weiteren ist für den realen Einsatz eines Klassifikators nicht nur die Fehlerrate von Interesse, sondern auch die Kosten und Risiken, die durch eine Klassifikation entstehen

bzw. entstehen können (vgl. [Sch96, WK91, DH73]).

Beispielsweise ist die Entscheidung, einen Patienten bei unsicherer Diagnose eines Krebsleidens prophylaktisch zu operieren und mit dieser Entscheidung falsch zu liegen, vertretbarer, als den Patienten nicht weiter zu untersuchen und seinem Schicksal zu überlassen. D.h. bei den Kosten "Tod oder Leben" tendiert man eher in die Richtung "Leben", was bedeutet, daß eine Falschklassifikation eines gesunden Menschen in Richtung krank eher toleriert wird, als einen Kranken für gesund zu erklären.

Durch eine *Kostenmatrix* lassen sich die auftretenden Kosten bei einer Falschklassifikation quantisieren. Dabei werden die Kosten für eine korrekte Klassifikation auf Null gesetzt (die Einheitsdiagonalelemente sind 0), und die restlichen Elemente werden auf Werte gesetzt, die die Kosten einer Falschklassifikation in Abhängigkeit von der Applikation widerspiegeln:

$$E_{Kosten} := \sum_{i,j=1}^{|\mathcal{C}|} \mathcal{F}_{ij} \, \mathcal{K}osten_{ij} \tag{D.4}$$

Mit Hilfe der Kosten- und Fehlermatrix lassen sich die Kosten, die durch eine Fehlklassifikation auftreten, darstellen. Die Leistungsfähigkeit eines Klassifikators läßt sich durch E_{Kosten} wesentlich besser erfassen als mit der reinen Anzahl der Fehlklassifikationen E_{emp}, da sie die anwendungsspezifischen Gesichtspunkte in die Bewertung mit einbezieht.

Die Bewertung eines Klassifikators läßt sich noch verbessern, wenn das Risiko mit in die Kalkulation einbezogen wird. Dabei ist das Risiko als eine Abschätzung zwischen der Möglichkeit einer korrekten Klassifikation und einer falschen zu verstehen. Es handelt sich dabei, wie bei der Kostenmatrix, nicht um die Wahrscheinlichkeiten, sondern um zugewiesene fiktive Größen, die aus der jeweiligen Applikation abgeleitet werden müssen.

Die *Risikomatrix* $\mathcal{R}$ wird anhand der Kostenmatrix $\mathcal{K}osten$ gebildet, indem in die Einheitsdiagonale die negativen Risikokoeffizienten[1] eingetragen werden. In Tabelle D.2 werden der Konfusionsmatrix aus Beispiel D.1 Risikofaktoren hinzugefügt.

Ausgabe des Klassifikators	Tatsächliche Klasse		
	A	B	C
A	-10	2	2
B	4	-7.5	1
C	3	1	-16

Tabelle D.2: Beispiel einer Risikomatrix für ein Drei-Klassen-Problem.

Mit Hilfe der Risiko- und Konfusionsmatrix läßt sich der Fehler $E_{\mathcal{R}}$ ermitteln, der die Auswirkung einer (Fehl-)Klassifikation im Applikationskontext wesentlich besser erfassen kann, als mit den vorher angesprochenen Methoden:

[1] In der Literatur wird die Risikomatrix mit positiven Diagonal- und negativen Nebendiagonalelementen verwendet. Aus Gründen der Vereinfachung wird hier $\mathcal{R}$ anders definiert, damit alle Fehlermaße E minimiert werden können.

$$E_{\mathcal{R}} := \frac{1}{|\mathcal{B}|} \sum_{i,j=1}^{|\mathcal{C}|} \mathcal{F}_{ij}\, \mathcal{R}_{ij} \tag{D.5}$$

Durch die Berechnung des Fehlermaßes $E_{\mathcal{R}}$ kann die Klassifikationsgüte eines Klassifikators im Applikationskontext sehr genau und objektiv beurteilt werden. In dem oben durchgeführten Beispiel D.2 beträgt das Risiko $E_{\mathcal{R}} = -10,4267$. Das bedeutet: je kleiner das Risiko $E_{\mathcal{R}}$ ist, desto besser arbeitet der Klassifikator. Vergleicht man zwei Klassifikatoren miteinander, so kann derjenige Klassifikator mit dem kleineren Fehlermaß $E_{\mathcal{R}}$ als besser betrachtet werden.

D.3 Repräsentative Beispiele

Um sicherzustellen, daß die *Güte* eines Klassifikators korrekt ermittelt werden kann, muß die auf den Beispielen basierende Bewertung fair sein, d.h. die für die Bewertung verwendeten Beispiele $\mathcal{B}$ müssen den Merkmalsraum $\mathcal{M}$ möglichst gut wiedergeben. Dies bedeutet statistisch gesehen, daß die gewählte Stichprobe $\mathcal{B}$ mit einer zufälligen Gleichverteilung aus der Menge $\mathcal{M}$ gewählt wird.

Die Zufälligkeit ist eine wesentliche Voraussetzung für die Wahl der Beispiele, insbesondere muß bei der ILD das System dafür sorgen, daß dieses Kriterium erfüllt ist. Es muß dafür gesorgt werden, daß der Mensch nicht in die Auswahl der Beispiele involviert ist, da er auf Grund gewisser Vorstellungen oder Vorlieben dem Problem nicht unbefangen entgegentritt, wodurch wiederum die Zufälligkeit beeinflußt und dadurch auch das Ergebnis verfälscht werden kann.

Es gibt nur eine Ausnahme für dieses Vorgehen und zwar, wenn die Wahrscheinlichkeit für das Auftreten einer bestimmten Klasse sehr gering ist, oder das Klassifikationsverfahren nicht mit unterschiedlichen Häufigkeiten von Klassen zurechtkommt. Dann geht man dazu über, die Wahrscheinlichkeit für diese Klassen in den Beispielen zu erhöhen (siehe [WK91]).

Bei der ILD wird generell davon ausgegangen, daß potentiell genügend Beispiele vorhanden sind, und aus diesem Grund wird die Auswahl der Beispiele hier streng durch einen Zufallsprozeß gesteuert.

Wird der Klassifikator K auf der Basis von Beispielen $\mathcal{B}$ erstellt, werden die Parameter des Klassifikators so gewählt, daß sie die gegebenen Beispiele möglichst gut klassifizieren können. In der Praxis ist die Anzahl der Beispiele wesentlich geringer als die des Merkmalsraums:

$$|\mathcal{B}| \quad \ll \quad |\mathcal{M}| \tag{D.6}$$

Dies hat zur Folge, daß der Klassifikator dazu tendiert, die Beispiele zu repräsentieren und dadurch die Generalisierungsfähigkeit, also die Klassifikationsgüte, auf dem Merkmalsraum $\mathcal{M}$ abnimmt. Dieses Phänomen ist unter dem Begriff *Overfitting* bekannt. Die erzielten Resultate des Klassifikators sind dann sehr optimistisch, und wird dieser Klassifikator mit noch nicht präsentierten Daten konfrontiert, bricht die Klassifikationsgüte dramatisch ab.

Aus diesem Grund wird das sogenannte *Train-and-Test* Verfahren angewendet, bei dem die Beispiele $\mathcal{B}$ zufällig in zwei disjunkte Mengen aufgeteilt werden:

$$\begin{aligned} \mathcal{B} &= \mathcal{B}_{train} \cup \mathcal{B}_{test} \\ &\text{mit } \mathcal{B}_{train} \cap \mathcal{B}_{test} = \emptyset \end{aligned} \tag{D.7}$$

$\mathcal{B}_{train}$ ist die Menge der Trainingsdaten, mit denen der Klassifikator aufgebaut wird, und $\mathcal{B}_{test}$ enthält die Testbeispiele, die ausschließlich zur Bewertung des Klassifikators herangezogen werden sollen. Bei der Erstellung des Klassifikators wird dann versucht, das oben genannte Overfitting zu vermeiden.

Diese Vorgehensweise bringt das Problem des *Trainierens auf den Testdaten* mit sich, da die Testdaten, die eigentlich zur letztendlichen Bewertung herangezogen werden sollten, mit in den Lernprozeß eingebunden werden. Dies hat zur Folge, daß der Klassifikator bezüglich der Lernbeispiele $\mathcal{B}_{train}$ und der Testbeispiele $\mathcal{B}_{test}$ optimiert wird.

Um eine derartige Optimierung zu vermeiden, werden die Beispiele im sogenannten *Holdout*–Verfahren in drei disjunkte Mengen aufgeteilt:

$$\begin{aligned} \mathcal{B} &= \mathcal{B}_l \cup \mathcal{B}_v \cup \mathcal{B}_t \\ \text{mit } \mathcal{B}_l \cap \mathcal{B}_v &= \emptyset,\ \mathcal{B}_l \cap \mathcal{B}_t = \emptyset,\ \mathcal{B}_v \cap \mathcal{B}_t = \emptyset \end{aligned} \tag{D.8}$$

Dabei ist $\mathcal{B}_l$ die Menge der Lerndaten, mit denen der Klassifikator aufgebaut wird. Mit dem Satz $\mathcal{B}_v$ wird der Klassifikator während des Lernvorgangs bewertet, und die Menge $\mathcal{B}_t$ wird ausschließlich zur Bewertung des Klassifikators herangezogen.

D.4 Schätzung des wahren Fehlers

Wie bereits erwähnt, besteht das Hauptinteresse in der Ermittlung des *wahren Fehlers* $E_{\mathcal{M}}$ (siehe Definition D.1). Dieser Fehler kann auf der Basis der Klassifikationsergebnisse der Testbeispiele $\mathcal{B}_t$ geschätzt werden. Die Beispiele aus $\mathcal{B}_t$ sind unabhängige Ereignisse. Sei k die Anzahl der Falschklassifikationen auf $\mathcal{B}_t$, dann ist die unbekannte Wahrscheinlichkeit $P(k)$ binomial verteilt [Spi90]:

$$P(k) = \binom{|\mathcal{B}_t|}{k} p^k (1-p)^{|\mathcal{B}_t|-k} \tag{D.9}$$

Mit dem Wissen, daß die Falschklassifikationen binomialverteilt sind und der Kenntnis des Maximum-Likelihood-Schätzers, kann $\hat{p}$ geschätzt werden:

$$\hat{p} = \frac{k}{|\mathcal{B}_t|} \tag{D.10}$$

Die angestrebte Aussage, wie groß die geschätzte Fehlerwahrscheinlichkeit $\hat{p}$ von der wahren Fehlerwahrscheinlichkeit p abweicht, läßt sich durch einen Konfidenzintervall ausdrücken. Jedes Konfidenzintervall entspricht einem Konfidenzniveau (z.B. 0.95, bzw. 95%), das die Wahrscheinlichkeit ausdrückt, mit der $\hat{p} = p$ ist. Diese Konfidenzniveaus

Konfidenzniveau	99.73%	99%	98%	96%	95.45%	95%	90%	80%
z_k	3.0	2.58	2.33	2.05	2.0	1.96	1.645	1.28

Tabelle D.3: Sicherheitskoeffizienten z_k für die wichtigsten Konfidenzniveaus.

werden in sogenannte Sicherheitskoeffizienten z_k umgerechnet, die den Koeffizienten der Standardabweichung darstellen (siehe Tabelle D.3).

Das Konfidenzintervall für gegebenes $\hat{p}$, z_k und $|\mathcal{B}_t|$ berechnet sich nach der Formel:

$$p = \frac{\hat{p} + \frac{z_k^2}{2|\mathcal{B}_t|} \pm z_k \sqrt{\frac{\hat{p}(1-\hat{p})}{|\mathcal{B}_t|} + \frac{z_k^2}{4|\mathcal{B}_t|^2}}}{1 + \frac{z_k^2}{|\mathcal{B}_t|}} \tag{D.11}$$

Diese Beziehung zwischen der geschätzten Fehlerwahrscheinlichkeit $\hat{p}$ und der wahren Fehlerwahrscheinlichkeit p in Abhängigkeit von der Anzahl der Testbeispiele läßt sich graphisch veranschaulichen (siehe Abbildung D.1). Um die Fehlerwahrscheinlichkeit p durch $\hat{p}$ besser einschätzen zu können, d.h. das Konfidenzintervall zu verkleinern, wird eine größere Anzahl von Testbeispielen benötigt.

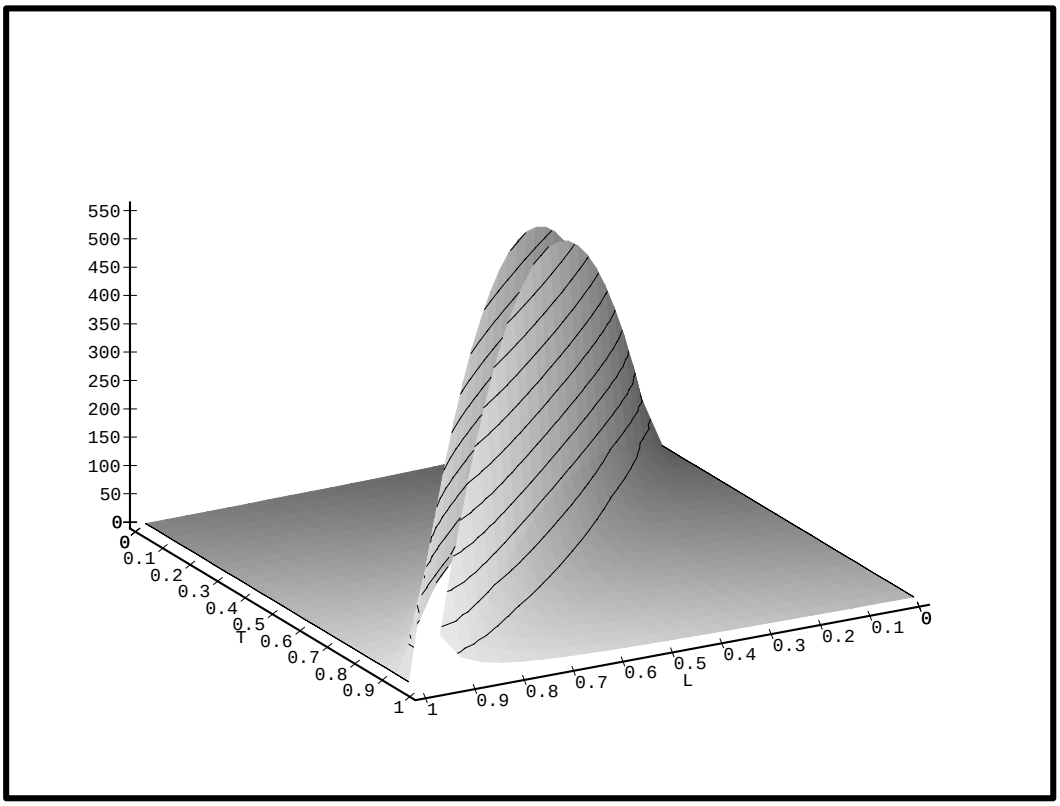


Abbildung D.1: Visualisierung der Gleichung D.11, mit der die Anzahl der Testbeispiele (nach oben gerichtete Achse), die benötigt werden, um in Abhängigkeit von einem Konfidenzniveau (hier konstant bei 0.95) für den *wahren Fehler* p (T-Achse) und $\hat{p}$ (L-Achse), benötigt wird, um eine Aussage zu treffen.

Neben der Möglichkeit für gegebene $\hat{p}$, z_k und $|\mathcal{B}_t|$ die dazugehörigen Konfidenzgrenzen zu ermitteln, stellt sich die Frage nach der Bestimmung der Anzahl der not-

wendigen Testbeispiele, um eine bestimmte Anforderung bezüglich der Abweichung Δp zum *wahren Fehler* p zu bestimmen.

Durch Umformung der Gleichung D.11 und Elimination der Variablen $\hat{p}$, wird die maximale Abweichung Δp zum *wahrem Fehler* p nach folgender Formel bestimmt:

$$\Delta p = \frac{z_k}{2\sqrt{|\mathcal{B}_t|}} \tag{D.12}$$

Mit der Kenntnis von $\hat{p}$ und der Anzahl der Beispiele läßt sich das Konfidenzintervall genau angeben. In Abbildung D.2 sind die Konfidenzintervalle bei einer Wahrscheinlichkeit von 0.95 bei unterschiedlichen Größen von $|\mathcal{B}_t|$ aufgetragen.

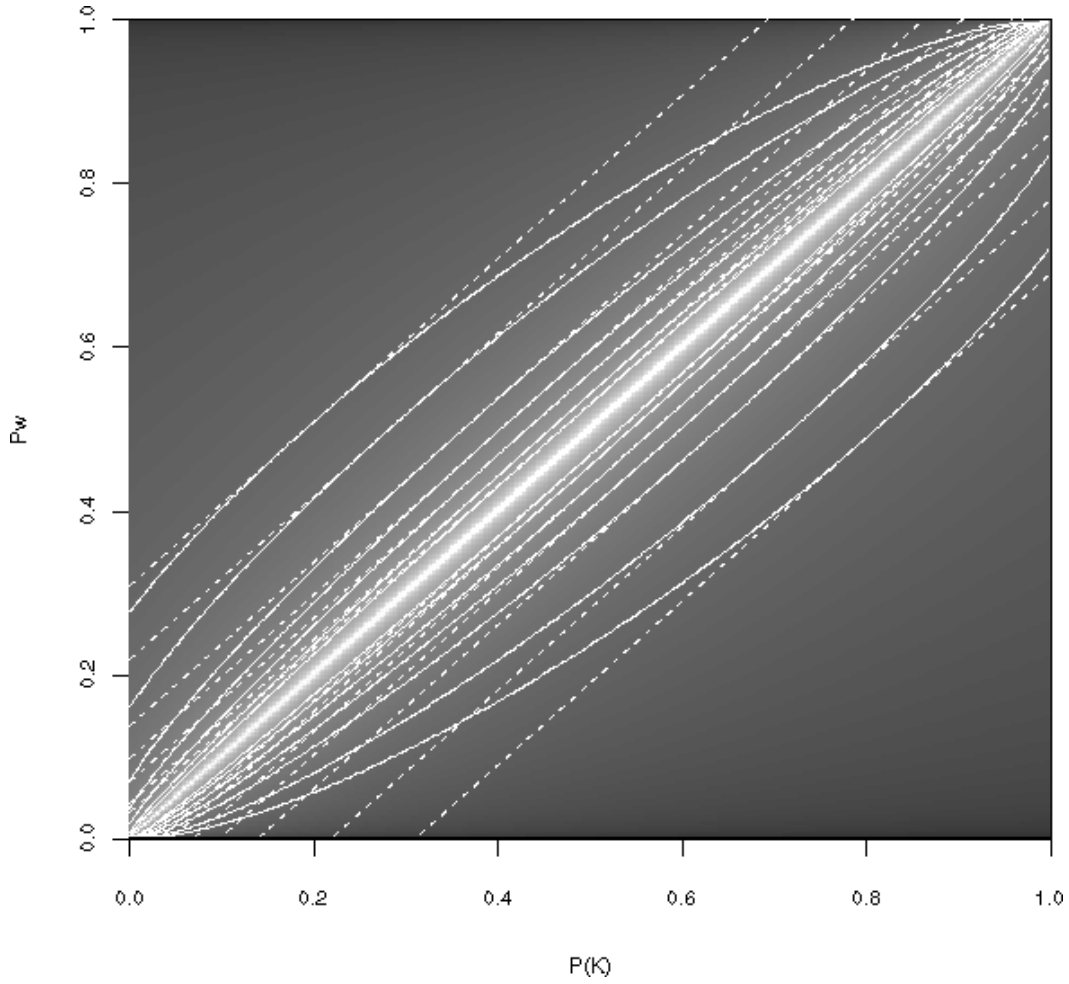


Abbildung D.2: Konfidenzintervalle für die Abschätzung des *wahren Fehlers* P_w in Abhängigkeit von der Fehlerwahrscheinlichkeit $P(K)$ des Klassifikators K. Das Konfidenzniveau liegt hier bei 0.95, so daß P_w zwischen den Intervallgrenzen für die jeweilige Anzahl von Beispielen liegt. Die durchgezogenen Linien markieren die Intervalle für $|\mathcal{B}_t| = \{10, 20, 50, 100, 200, 500, 1000\}$ von außen nach innen. Die gestrichelten Linien markieren die maximale Abweichung Δp der Intervalle (siehe Gleichungen D.11, D.12).

Durch Umformung der Gleichung D.12 kann die Aussage über die maximal notwendige Anzahl von Testbeispielen abgeleitet werden, die a priori notwendig sind, um eine

Abweichung vom *wahren Fehler* p um nicht mehr als ein durch den Sicherheitskoeffizienten z_k bestimmtes Konfidenzniveau einzuhalten (siehe auch Abbildung D.3):

$$|\mathcal{B}_t| = \frac{1}{4}\frac{z_k^2}{\Delta p^2} \tag{D.13}$$

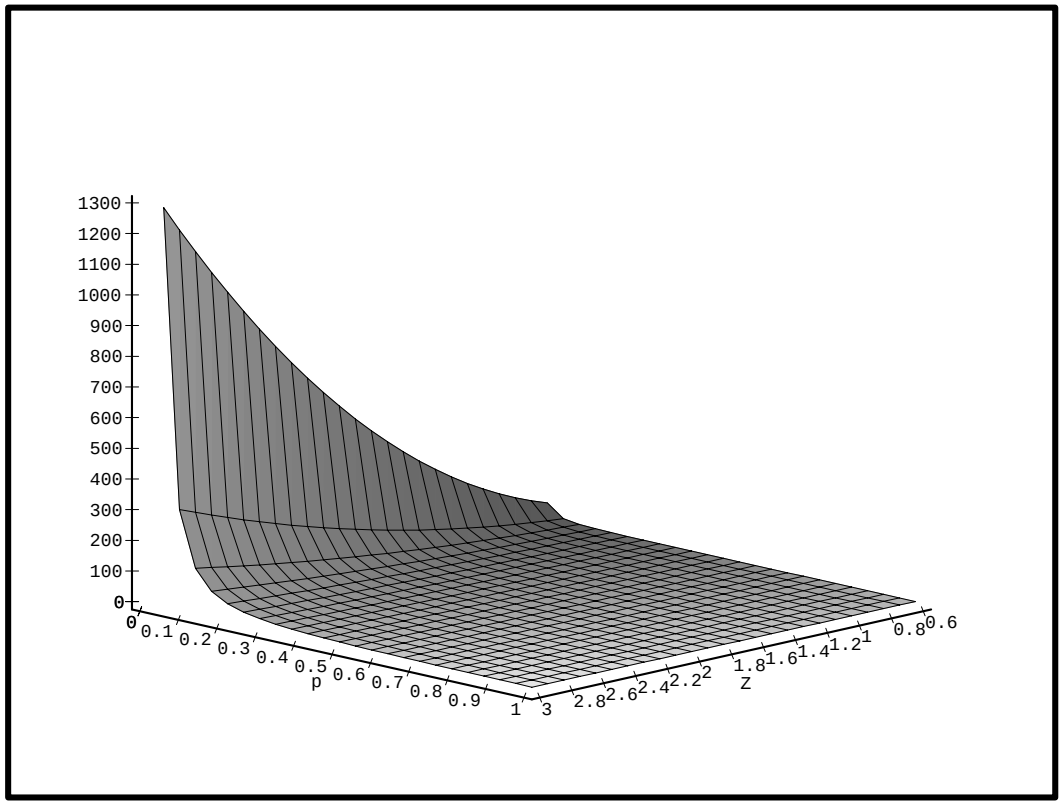


Abbildung D.3: A-priori-Ermittlung der maximal notwendigen Anzahl an Testbeispielen (nach oben gerichtete Achse), die notwendig ist, um von dem *wahren Fehler* p um nicht mehr als Δp (P-Achse) bei gegebenem Konfidenzniveau (Z-Achse) abzuweichen (siehe Gleichung D.13)).

Aus den Ausführungen geht deutlich hervor, daß die Genauigkeit der Schätzung des *wahren Fehlers* lediglich von der Anzahl der Beispiele abhängt. Es ist sogar a priori möglich zu bestimmen, wieviele Testbeispiele mindestens benötigt werden, um sicherzustellen, daß die Abweichung vom *wahren Fehler* mit einer bestimmten Wahrscheinlichkeit innerhalb vorgegebener Grenzen bleibt.

Diese Aussage ist zentral für die Anwendung der ILD, da von vorneherein die Anzahl der Beispiele in der Testmenge abschätzbar wird. Dies kann wiederum dazu ausgenutzt werden, diejenigen Beispiele, die nicht für die Abschätzung benötigt werden, dem Lernprozeß zuzuführen.

D.4.1 Resampling

Bei geringen Mengen von Beispielen ist es schwierig oder gar unmöglich, diese in Trainings- und Testdaten aufzuteilen, da dadurch wesentliche Informationen verloren gehen können. Durch die mehrfache Partitionierung der Beispielmenge sowie die

Bildung des dazugehörigen Klassifikators lassen sich unter bestimmten Bedingungen bessere Abschätzungen erreichen als durch *Train-and-Test-* oder die *Holdout-* Methode. Die nachfolgend aufgeführten Methoden haben sich dabei als die erfolgreichsten herauskristallisiert:

Random subsampling: Es werden k unabhängige und zufällige Unterteilungen aus $\mathcal{B}$ in Test- und Trainingsbeispiele vorgenommen. Diese Methode besteht aus der mehrmaligen Anwendung von *Holdout* auf unterschiedlich partitionierte Test- und Trainingsbeispiele. Für jede der Unterteilungen wird ein Klassifikator erstellt. Der geschätzte wahre Fehler ist dann der Mittelwert der Fehler aller erstellten Klassifikatoren.

leaving 1 out: Jeweils ein Beispiel aus $\mathcal{B}$ wird zur Testmenge und der Rest wird zum Einlernen des Klassifikators verwendet. Auf diese Weise entstehen $|\mathcal{B}|$ Klassifikatoren, deren Fehler wiederum aufaddiert und durch $|\mathcal{B}|$ dividiert werden.

Das *leaving 1 out* Verfahren ist nur ein Spezialfall von *k–fold CV*, da hier $k = n$ gesetzt ist.

Cross–Validation (k–fold CV): Die Beispiele $\mathcal{B}$ werden zufällig in k ungefähr gleich große Mengen $\mathcal{B}_i$ aufgeteilt. Es wird jeweils eine Menge $\mathcal{B}_i$ zur Testmenge. Die restlichen dienen als Trainingsmenge. Der wahre Fehler wird als der Mittelwert aller k Klassifikatoren ermittelt.

Bootstrapping: Es existieren mehrere Ansätze, die unter diesem Namen zusammengefaßt werden. Das für die Schätzung des Fehlers wichtigste Verfahren ist das sogenannte *e0-* Verfahren (siehe [ET93]). Hier werden die Trainingsmengen $\mathcal{B}_{train_i}$ aus $\mathcal{B}$ gebildet, indem zufällig $n = |\mathcal{B}|$ Beispiele mit Zurücklegen ausgewählt werden. Die Testmenge $\mathcal{B}_{test_i}$ besteht aus allen Beispielen, die nicht in $\mathcal{B}_{train_i}$ vertreten sind. Die Wahrscheinlichkeit eines Beispiels in $\mathcal{B}_{train_i}$ beträgt 0.632 und demzufolge die Wahrscheinlichkeit für das Enthaltensein in der Testmenge 0.368. Der Mittelwert aus den Fehlern in $\mathcal{B}_{test_i}$ ergibt den wahren Fehler.

D.4.2 Vergleich der Schätzverfahren

In Tabelle D.4 sind die Aufteilungen der Beispiele, die Anzahl der Iterationen und die Eignung der einzelnen Verfahren für unterschiedliche Größen der Beispielmenge zusammengefaßt. Der Einsatz der verschiedenen Verfahren hängt stark von der Anzahl der vorhandenen Beispiele ab.

Bei der *Holdout-* Methode kann von vornherein die notwendige Anzahl der Beispiele in der Testmenge bestimmt werden. Eine sehr gute Schätzung wird mit Testmengen von ca. 1000 Beispielen erreicht. Damit der Klassifikator erfolgreich eingelernt werden kann, werden nach der Faustregel 2/3 1/3 noch mindestens 2000 Beispiele für den Lerndatensatz benötigt. Diese Anzahl wird allerdings stark von der Komplexität der Aufgabe beeinflußt. Andererseits ist der Rechenaufwand sehr gering, da nur ein Klassifikator gebildet werden muß.

Durch die Anwendung von *Random subsampling* sind bessere Schätzungen als durch die *Holdout*-Methode zu erwarten, da auch die Beispiele, die bei der *Holdout*-Methode in der Testmenge verbleiben, hier zur Bildung des Klassifikators beitragen. Wie auch

	Holdout	Random subsampling	leaving 1 out	k–fold CV	Bootstrapping
Trainingsbeispiele	j	j	$n-1$	$\frac{k-1}{k}n$	n, davon j einmalig
Testbeispiele	$n-j$	$n-j$	1	$\frac{1}{k}n$	$n-j$
Iterationen	1	$k \ll n$	n	k	$k \approx 25-200$

Tabelle D.4: Komplexität der einzelnen Fehlerschätzverfahren. n ist die Anzahl aller Beispiele, k die Anzahl der gebildeten Mengen.

bei den restlichen Methoden, steigt die Komplexität der Berechnung linear zur Anzahl der Iterationen k an.

Die *leaving 1 out* Methode ist weitestgehend vorurteilsfrei bei der Schätzung des *wahren Fehlers*. Allerdings erweist sie sich für eine größere Anzahl von Beispielen als sehr rechenintensiv.

Die Rechenkomplexität bei *k–fold CV* kann direkt durch die Anzahl k der Partitionen beeinflußt werden. In der Praxis hat sich gezeigt, daß eine *10–fold CV* gute Ergebnisse liefert. Cross-Validation ist bei gleicher Anzahl von Iterationen und Partitionierung dem *Random subsampling* vorzuziehen, da hier sichergestellt wird, daß alle Beispiele zum Lernen benutzt werden.

Das *Bootstrapping* Verfahren ist die modernste Methode zur Ermittlung von statistischen Aussagen auf Stichproben. Der Rechenaufwand ist hier allerdings ebenfalls sehr hoch, er ist jedoch unabhängig von der Anzahl der Beispiele. Dieser Ansatz hat im Gegensatz zur der *k–fold CV* Methode eine geringere Varianz bei der Schätzung des *wahren Fehlers* für ein kleines n.

Auf Grund der vorangegangenen Betrachtungen kann für die jeweiligen Anforderungen bezüglich der Güte der Schätzung, der Anzahl der zur Verfügung stehenden Beispiele und des Rechenaufwands für jede Iteration die geeignetste Methode ausgewählt werden. Bei der ILD kann davon ausgegangen werden, daß die Anzahl der Beispiele hunderte bzw. tausende beträgt. Des Weiteren ist das Einlernen Neuronaler Netze derzeit immer noch mit erheblichem zeitlichen Aufwand verbunden. Aus diesem Grund ist die Anwendung des *k–fold CV* bei kleineren Datensätzen (mehrere hundert) und der *Holdout*-Methode bei größeren Datensätzen (mehrere tausend) vorzuziehen.

Durch die Anwendung der Holdout-Methode ist eine genaue Angabe des Konfidenzintervalls des wahren Fehlers $E_{\mathcal{M}}$ möglich und das Fehlerintervall ist a priori durch die Anzahl der Testbeispiele bestimmt (siehe Abschnitt D.4).

Die Ermittlung des Konfidenzintervalls wird durchgeführt, bevor der Klassifikator zum Einsatz kommt, um dessen potentielle Möglichkeiten festzustellen. Der erste Baustein des Klassifikators besteht aus dem folgenden *Dynamic Bounds* Verfahren.

Anhang E

Datensätze

In den vorliegedned Arbeit wurden im Rahmen der Versuche mit Ultraschalldaten hauptsächlich die folgenden zwei Datensatzmengen gesammelt und untersucht. Diese werden im folgenden näher erläutert.

E.1 defects1

Diese *defects1* Beispielmenge wurde im Rahmen des ***NeuroPipe***-Projekts gesammelt und repräsentiert die Anomalien, die bei der Inspektion mit dem UltraScan-Molch der Fima Pipetornix GmbH aufgenommen wurden. Jede Anomalie wird durch 41 Merkmale repräsentiert und eindeutig einer von 5 Klassen zugeordnet.

Die 1211 Beispiele des Datensatzes wurden derart vorselektiert, daß möglichst wenig Widersprüche auftreten. Die Klassen sind nicht linear separierbar.

E.2 lweld

Der *lweld* Datensatz wurde im Rahmen des ***NeuroPipe***-Projekts gesammelt und repräsentiert die Klassifikation von Längsnähten in Pipelines. Dabei wurden für jeden Sensor 75 Merkmale berechnet. Jeder Sensor wurde dann als zur Längsnaht zugehörig oder nicht zugehörig klassifiziert, was durch zwei Klassen repräsentiert wurde.

Es wurde keinerlei Vorselektierung der 9880 Beispieldaten vorgenommen. Es ist auch nicht ausgeschlossen, daß widersprüchliche Beispiele existieren. Das Problem ist auf jeden Fall nicht linear separierbar.