

Wortsinndisambiguierung durch hierarchische Kontextabstraktion

Magisterarbeit im Studiengang Informationsverarbeitung

bei Prof. Dr. Jürgen Rolshoven
Sprachliche Informationsverarbeitung
Universität zu Köln

vorgelegt von

Fabian Steeg
Matrikelnummer 3598900
steeg@netcologne.de
Liebigstr. 43
50823 Köln

Köln,
31. August 2007

Inhalt

Vorbemerkung	vi
1. Wortsinndisambiguierung durch Kontextabstraktion	1
1.1. Motivation und Überblick	1
1.2. Wortsinndisambiguierung	2
1.3. Wortsemantik	2
1.3.1. Sinn und Bedeutung	3
1.3.2. Wortsinn als Kontextabstraktion	4
1.3.3. Arten von Mehrdeutigkeit	6
1.3.4. Abgrenzungsproblematik	6
1.4. Forschungsstand	7
2. Abstrahierte Kognition und kognitive Abstraktion	9
2.1. Computerlinguistik als Kognitionswissenschaft	9
2.2. Abstrahierte Kognition	9
2.2.1. Simulation kognitiver Prinzipien	9
2.2.2. Hierarchische Wissensstrukturierung	11
2.2.3. Erwartung und bedingte Wahrscheinlichkeit	12
2.3. Forschungsstand	13
3. Korpuslinguistik und maschinelles Lernen	15
3.1. Maschinelles Lernen in der Sprachverarbeitung	15
3.1.1. Korpuslinguistik	15
3.1.2. Korpusannotation	16
3.2. Klassifikation	17
3.2.1. Überwachtes und unüberwachtes Lernen	17
3.2.2. Überwachtes Lernen zur Klassifikation	17
3.2.3. Numerische Kontextrepräsentation	20
4. Implementierung	22
4.1. Merkmalsberechnung	22
4.1.1. Wörter	23
4.1.2. Wortlängen	24
4.1.3. Trigramme und Heptagramme	24
4.1.4. Paradigmen	25
4.2. Training und Klassifikation	28
4.2.1. Abstraktion in den Baumknoten	28
4.2.2. Generierung der Sprachelemente	29

4.2.3. Lernalgorithmus	31
4.2.4. Klassifikationsalgorithmus	33
4.3. Datenstruktur	37
4.4. Parallelisierung	38
5. Evaluierung	40
5.1. Korpora zur Evaluierung	40
5.2. Evaluationskriterien	41
5.2.1. Precision und Recall	41
5.2.2. Vergleichswerte	42
5.2.3. Bedeutungsgranularität	42
5.2.4. Anwendungsintegration	43
5.3. Experimente	44
5.3.1. Variablen	44
5.3.2. Ergebnisse	49
5.3.3. Diskussion	50
6. Anwendung	52
6.1. Modularisierung und Integration	52
6.1.1. Komponentenbasierte Softwareentwicklung	53
6.1.2. Komponentensysteme in der Computerlinguistik	53
6.2. Semantische Annotation in Tesla	54
6.2.1. Wortlisten zur Annotation	55
6.2.2. Komponenten beim Training	56
6.2.3. Komponenten bei der Klassifikation	58
7. Schluss: Maschinelles Lernen mit Komponenten	60
7.1. Zusammenfassung	60
7.2. Ausblick	61
7.2.1. Korpora	61
7.2.2. Merkmalsberechnung	62
7.2.3. Klassifikation	63
7.2.4. Anwendungen	64
A. Implementationsdetails	I
A.1. Umsetzungen der Algorithmen	I
A.2. Konfigurationsdateien	IV
A.3. Inhalt des beiliegenden Datenträgers	IX
B. Erklärung	X
Literaturverzeichnis	XI
Index	XVII

Abbildungsverzeichnis

2.1. Hierarchische Abstraktion sprachlicher Symbole	12
3.1. Generische Architektur eines Klassifikationssystems	18
3.2. Regelkreismodell der Didaktik	19
4.1. Ausschnitte des Suffix- und des Präfixbaums mit Paradigmen	26
4.2. Symbolische Abstraktion der Merkmale	28
4.3. Baum mit allen gesuchten Permutationen	30
4.4. Minimales Aktivierungsbeispiel	32
4.5. Tabellen in den Baumknoten nach dem Training	34
4.6. Gesamtstruktur des Lexikons	34
4.7. Vorwärts- und Rückwärtskommunikation im Baum	35
4.8. Klassendiagramm der umgesetzten Datenstruktur	37
5.1. Hierarchisch strukturierte Bedeutungen im Senseval-Korpus	43
5.2. Vergleichswerte und beste Ergebnisse bei feiner Körnung	49
5.3. Vergleichswerte und beste Ergebnisse bei grober Körnung	49
5.4. Entwicklung der Ergebnisse für Kontextfenster ± 2 und ± 4	50
5.5. Entwicklung der Ergebnisse für Kontextfenster ± 8 und ± 16	50
6.1. Komponenten bei der Eigennamenerkennung	55
6.2. Komponentendiagramm der entwickelten Komponenten	57
6.3. Komponenten beim Training	58
6.4. Komponenten bei der Anwendung	59
7.1. Modellierung kognitiver Informationsverarbeitung	61
7.2. Austauschbarkeit von Komponenten	62
A.1. Konfigurationsdatei im graphischen Editor von Tesla	VIII

Tabellenverzeichnis

4.1. Beispiel für bedingte Wahrscheinlichkeiten in einem Blattknoten	32
4.2. Beispiel für bedingte Wahrscheinlichkeiten in einem inneren Knoten	33
5.1. Überblick über das Bedeutungsinventar	41
5.2. Variablen und Werte für die Experimente	44
5.3. Ergebnisse und Netzstrukturen mit Kontextfenster ± 2	45
5.4. Ergebnisse und Netzstrukturen mit Kontextfenster ± 4	46
5.5. Ergebnisse und Netzstrukturen mit Kontextfenster ± 8	47
5.6. Ergebnisse und Netzstrukturen mit Kontextfenster ± 16	48

Liste der Algorithmen

1. Numerische Abstraktion für Tri- und Heptagramme	25
2. Auslesen von Paradigmen aus einem Suffixbaum	27
3. Sortieren von Paradigmen	27
4. Abstraktionsalgorithmus für Blattknoten	29
5. Rekursive Generierung der Sprachelemente	30
6. Generierung der Sprachelemente über eine Bitmaske	31
7. Parallelisiertes Lernen	38

Vorbemerkung

Im Laufe meines Studiums der sprachlichen Informationsverarbeitung war ich immer mehr zu der Ansicht gelangt, dass so stark mit Aspekten wie Äußerungssituation, Vorwissen, logischem Denken und Weltwissen verknüpfte Probleme wie das der Wortsinndisambiguierung (WSD) erst dann vollständig automatisiert lösbar sein werden, wenn eine umfassende maschinelle Umsetzung der menschlichen Biologie verfügbar ist, d.h. ein Roboter, der unter den gleichen Voraussetzungen wie der Mensch lernen und handeln kann. Die Lektüre von Jeff Hawkins Buch *On Intelligence* hat mich wieder mit dem vielleicht naiven Enthusiasmus erfüllt, den man wohl braucht, um dem Versuch nachzugehen, das ungeheuer komplexe menschliche Sprachvermögen auf einem Computer umzusetzen. Die Motivation, die Konzepte aus *On Intelligence* auf ein Problem der maschinellen Sprachverarbeitung anzuwenden, bildete so den Ausgangspunkt dieser Arbeit. Das Problem der WSD verbindet einige der interessantesten Fragen von Philosophie, Linguistik und Kognitionswissenschaft mit empirisch und experimentell ausgerichteten Ansätzen des maschinellen Lernens. Es ist ein prinzipiell ungelöstes Kernproblem der Computerlinguistik und eine wesentliche Voraussetzung für die Entwicklung von Computerprogrammen, die Sprache verstehen. So stellt die WSD ein äußerst spannendes Forschungsgebiet dar, weshalb ich meinem Betreuer für den Vorschlag sehr dankbar bin, mich in meiner hier vorliegenden Abschlussarbeit mit diesem Problem zu befassen.

Kapitel 1.

Wortsinndisambiguierung durch Kontextabstraktion

1.1. Motivation und Überblick

Mehrdeutige Wörter existieren seit Beginn der menschlichen Schriftkultur (s. Abs. 1.3, S. 2). Wortsinndisambiguierung (WSD, engl. *word sense disambiguation*), der Prozess der Auflösung der Mehrdeutigkeit eines Wortes anhand seines Kontextes (Kap. 1) fällt Menschen leicht; maschinell ist dieser Prozess jedoch bislang nicht in vergleichbarer Form durchführbar. Dies ist letztendlich ein wesentlicher Grund dafür, dass Computer Sprache nicht verstehen können und macht so die WSD zu einem Kernproblem der Computerlinguistik.

Der Mensch abstrahiert beim kognitiven Prozess der WSD von konkreten Kontexten der ambigen Wörter, vermutlich auf Grundlage eines “einheitlichen Modus [...] der Informationsverarbeitung” (Singer 2002:145), mit dem Daten unterschiedlicher Herkunft (d.h. die verschiedenen Sinneswahrnehmungen) verarbeitet werden (Kap. 2, S. 9). Diese Verbindung aus domänenspezifischen Daten, die mit einem domänenübergreifenden Mechanismus verarbeitet werden, entspricht Prinzipien des maschinellen Lernens, dessen Datenbasis in der Sprachverarbeitung Korpora bilden (Kap. 3, S. 15).

Diese Konzepte werden in der vorliegenden Arbeit mit hierarchischer *Belief Propagation* in Bäumen implementiert (Kap. 4, S. 22) und auf Daten des British National Corpus (BNC) evaluiert (Kap. 5, S. 40). Die Bestandteile des Verfahrens werden modular in einer *Software Architecture for Language Engineering* (SALE) umgesetzt, um das WSD-Verfahren für unterschiedliche Anwendungen in der maschinellen Sprachverarbeitung zugänglich zu machen (Kap. 6, S. 52). Eine solche Umsetzung eröffnet zudem zahlreiche Möglichkeiten zur Weiterentwicklung des Verfahrens selbst sowie darüber hinaus, etwa durch die Nutzung einzelner Bestandteile des WSD-Verfahrens in anderen Zusammenhängen (Kap. 7, S. 60).

1.2. Wortsinndisambiguierung

Gegenstand der WSD ist die Auswahl der im Kontext passenden Lesart mehrdeutiger Wörter (Agirre & Edmonds 2006b), etwa in Beispiel (1) die zu aktivierende Lesart von *Bank*, z.B. als ‘Möbel’ oder als ‘Gebäude’.

(1) Hast du die neue Bank gesehen?

WSD ist eine kognitive Leistung bei der menschlichen Sprachverarbeitung und ein zentrales Problem der Computerlinguistik, da sie eine Voraussetzung für verschiedene Aufgaben der maschinellen Sprachverarbeitung darstellt, etwa der maschinellen Übersetzung (MÜ)¹ oder der Informationsextraktion² (IE, s. etwa Neumann 2001). WSD wird von Menschen in der Regel³ problemlos, von Computern aber bislang, insbesondere für Polysemie (s. Abs. 1.3, S. 2), nicht vollständig geleistet. Aufgrund seiner Abhängigkeit etwa von logischem Denken und Weltwissen (Ide & Véronis 1998:2, Agirre & Edmonds 2006b:1) kann WSD als kognitives Problem charakterisiert werden.

Kognitive Modelle der menschlichen Informationsverarbeitung (s. Kap. 2, S. 9) stimmen mit Annahmen der lexikalischen Semantik und Erkenntnissen der Philosophie im Bereich der WSD insofern überein, als dass die Bedeutung eines Wortes eine Abstraktion seines Kontextes ist (s. Abs. 1.3.2, S. 4). Auf dieser Grundlage soll in der vorliegenden Arbeit ein sowohl semantisch wie kognitiv plausibles Verfahren zur WSD entwickelt und beschrieben werden.

1.3. Wortsemantik

Ein Wort ist eine orthographisch abgetrennte Sinneinheit in einem Text. Wörter existieren schon in den frühesten Schriftsystemen, z.B. im Altsumerischen (Haarmann 1991:52) und sind als orthographisches Konzept von Schriftsystem und morphologischem Status der Sprache abhängig; so können Sinneinheiten, die in flektierenden Sprachen ein einziges Wort sind (etwa dt. *Wortsinndisambiguierung*), in isolierenden Sprachen mehrere Wörter umfassen (etwa engl. *word sense disambiguation*). Grundsätzlich kann sich dabei WSD sowohl auf einzelne Wörter, als auch auf Wortgruppen oder Wortteile beziehen (vgl. Abs.

1 Wenn etwa ein MÜ-System in einem englischen Text auf das Wort *bank* trifft, so muss dies je nach Lesart etwa im Deutschen als *Bank* oder aber als *Ufer* übersetzt werden.

2 Werden etwa in englischen Texten Begriffe des Finanzwesens gesucht, sollte eine Fundstelle von *bank* je nach Lesart ausgewählt oder übergangen werden.

3 Eine Vielzahl von Witzen basiert auf falsch oder nicht aufgelöster Ambiguität, z.B.: *Richter zur Angeklagten*: “Wie heißen sie?”, *darauf sie*: “*Elisabeth Meier*”. *Richter*: “*Und ihr Alter?*” *Sie*: “*Wartet draußen.*”

6.2, S. 54).

Die Mehrdeutigkeit von Wörtern ist wie das Konzept des Wortes selbst schon in den ersten Schriftsystemen vorhanden, so verwendet das Altsumerische etwa für ‘Schilf’ und ‘zurückkehren’ das gleiche Wort (Haarmann 1991:153). Wörter können also schon immer je nach Kontext verschiedene Bedeutungen haben.

1.3.1. Sinn und Bedeutung

Wenn man von WSD und Mehrdeutigkeit spricht, kommt die Frage nach einer Unterscheidung und einer Definition der Begriffe *Sinn* und *Bedeutung* auf. *Sinn* wird dabei in unterschiedlichen Kontexten gebraucht. Neben der biologischen Lesart (wie in *die fünf Sinne* oder *feinsinnig*, für ‘Schnittstelle des Menschen zur Welt’) und der teleologischen Lesart (wie in *Sinn des Lebens*, für ‘das Ziel oder der Zweck eines Vorgangs’) existieren mindestens drei weitere Lesarten in der sprachwissenschaftlichen Domäne.

Erstens, *Sinn* als Synonym zu *Bedeutung*, als der Teil eines sprachlichen Zeichens, der nicht seine Form ist, sondern das, was der Benutzer durch die Verwendung der Form ausdrücken will (vgl. Wilson & Keil 1999:513), im Kontext eines zweigeteilten Zeichenbegriffs aus Bezeichnung und Bezeichnetem (*signifiant* und *signifié* bei Saussure). Zweitens, *Sinn* unterschieden von *Bedeutung*: *Sinn* ist bei Frege (1892) der nicht deutende oder nicht-referenzielle Teil von Zeichen im Kontext eines dreigeteilten Zeichenbegriffs aus Form, Sinn und Bedeutung (s.u.). Drittens, *Sinn* als ‘eine Lesart eines mehrdeutigen Wortes’ (wie in *im wörtlichen Sinn*, *im übertragenen Sinn*, *im wirtschaftlichen Sinn*). Dies ist die Lesart von *Sinn* im Kontext der WSD: die Ermittlung der passenden Lesart, des richtigen Wortsinns. Unter *Wortsinn* wird daher im folgenden, analog zum englischen *word sense*, ‘eine der Lesarten eines mehrdeutigen Wortes’ verstanden.

Der Sinn eines Wortes ist bei Frege die Menge der Eigenschaften, die ein Begriff umfasst, im Gegensatz zur Bedeutung, die eine Referenz d.h. Deutung⁴ auf einen Gegenstand ist. Grundgedanke ist dabei, dass sich etwa die Aussagen *der Abendstern ist schön* und *der Morgenstern ist schön* in ihrem Informationsgehalt unterscheiden. Wenn jedoch die Bedeutung von Zeichen ausschließlich eine Referenz wäre, würden sich die Aussagen nicht unterscheiden, denn in beiden Fällen wird die Venus referenziert. Die Aussagen werden aber in sehr unterschiedlichen Kontexten verwendet, da hier einmal von einem Stern am Abendhimmel, das andere Mal von einem Stern am Morgenhimmel die Rede ist. Es muss daher neben Gegenstand und Form einen weiteren Faktor im Zeichen geben, und

⁴ Entsprechend der Konzeption von *Bedeutung* als ‘Referenz’ lautet der Titel von Freges Artikel *Über Sinn und Bedeutung* im Englischen *On sense and reference*.

diesen nennt Frege *Sinn*. So haben also *Morgenstern* und *Abendstern* dieselbe Bedeutung (nämlich die Venus), nicht aber denselben Sinn (Frege 1892:24).

In Freges Terminologie haben Eigennamen (“Wort, Zeichen, Zeichenverbindung, Ausdruck”, Frege 1892:24), d.h. Bezeichnungen von einzelnen Gegenständen, immer einen Sinn, aber nicht immer eine Bedeutung, so hat etwa *der gegenwärtige König von Frankreich* einen Sinn, aber keine Bedeutung, da kein Objekt von dem Ausdruck referenziert wird. Eine solche Unterscheidung ist hilfreich für die Aufgabe der metalinguistischen Beschreibung, des Sprechens über Sprache, da sie helfen kann, den Bereich der Wortsemantik begrifflich weiter zu gliedern. So schlägt Frege (1892:30) etwa vor, Zeichen, die nur einen Sinn und keine Bedeutung haben, als *Bild* zu bezeichnen.

1.3.2. Wortsinn als Kontextabstraktion

Wittgenstein richtet sich in vergleichbarer Weise dagegen, als Wortbedeutung nur die Referenz anzunehmen und setzt dem entgegen: “Die Bedeutung eines Wortes ist sein Gebrauch in der Sprache” (Wittgenstein 1953:40). Wenn man jedoch, Frege folgend, die Bedeutung als einen Teil des sprachlichen Zeichens betrachtet und anerkennt, dass es neben dieser (sowie der Form und der subjektiven Vorstellung) noch einen Faktor im Zeichen gibt, könnte man sagen: Die Bedeutung eines Wortes ist eine Referenz auf einen Gegenstand, der Sinn ist sein Gebrauch in der Sprache. Eine solche Konzeption von Wortsinn als Kontextabstraktion⁵ wird auch der Vorstellung Freges eines unterschiedlichen Wortsinns für unterschiedliche Sprecher gerecht:

Bei einem eigentlichen Eigennamen wie *Aristoteles* können freilich die Meinungen über den Sinn auseinandergehen. Man könnte z.B. als solchen annehmen: der Schüler Platos und Lehrer Alexanders des Großen. Wer dies tut, wird mit dem Satze *Aristoteles war aus Stagira gebürtig* einen anderen Sinn verbinden als einer, der als Sinn dieses Namens annähme: der aus Stagira gebürtige Lehrer Alexanders des Großen (Frege 1892:24).

In der Konzeption von Sinn als Kontextabstraktion kennt hier etwa der zweite Nutzer *Aristoteles* aus Kontexten, die etwa die Information enthielten, dass er aus Stagira gebürtig

⁵ Der Gedanke, dass Wortbedeutung durch den Gebrauch bestimmt wird, ist als solcher sehr alt, so lässt Platon im *Kratylos* (2) Hermogenes im Gespräch mit Sokrates äußern: “Denn kein einziges Ding hat seinen Namen von Natur aus, sondern jedes hat ihn durch Gebrauch und durch Gewohnheit derer, die ihn anzuwenden pflegen.” Dabei erklärt Hermogenes auf Nachfrage von Sokrates aber, dass er es so meint, dass ein Einzelner willkürlich die Bedeutung einzelner Wörter durch seinen persönlichen Gebrauch der Wörter umbestimmt, dass etwas “im allgemeinen Sprachgebrauch als *Mensch* bezeichnet [wird], in meinem persönlichen dagegen als *Pferd*”. Diesen Standpunkt lässt Platon Sokrates dann widerlegen.

ist, während der erste Nutzer *Aristoteles* aus Kontexten kennt, die ihm unter Anderem vermittelt haben, dass er der Schüler Platons war.

So unterscheidet Frege auch eine Mehrsinnigkeit von einer Mehrdeutigkeit, wie in der Fortsetzungen des obigen Zitats deutlich wird:

Solange nur die Bedeutung dieselbe bleibt, lassen sich diese Schwankungen des Sinnes ertragen, wiewohl auch sie in dem Lehrgebäude einer bewiesenen Wissenschaft zu vermeiden sind und in einer vollkommenen Sprache nicht vorkommen dürften (Frege 1892:24).

Im Sinne Freges wäre so WSD die Umwandlung der natürlichen Sprache in diese “vollkommene Sprache” ohne “Schwankungen des Sinnes”, konkret in Form von mit eindeutigem Sinn und eindeutiger Bedeutung annotierten Daten (vgl. Abs. 6.2, S. 54).

Obwohl in der WSD in der Regel Zeichen mit ambiger Bedeutung disambiguiert werden, ist auch eine Disambiguierung von ambigem Sinn (bei eindeutiger Bedeutung) über die Kontexte möglich; so ist etwa wahrscheinlich, dass ein Autor *Aristoteles* unterschiedlich verwendet, je nachdem, ob er weiss, dass Aristoteles auch Lehrer Alexanders des Großen war oder nicht. Wenn also der Sinn oder die Bedeutung nicht eindeutig sind, können beide durch Abstraktion des Kontextes erkannt werden. So ist also mit einer WSD durch Kontextabstraktion sowohl Mehrdeutigkeit als auch Mehrsinnigkeit fassbar.

Entsprechend einer Konzeption von Wortsinn als Kontextabstraktion kann man sagen, dass mehrdeutige Wörter in unterschiedlichen Kontexten verwendet werden, und dass genau dies die Bedeutung von *mehrdeutig* ist. Dieses Verständnis von Bedeutung wird in der jüngsten WSD-Literatur ausgedrückt (Agirre & Edmonds 2006a:1, Kilgarriff 2006:44).

Ein solcher Ansatz erklärt viele Phänomene, etwa dass Wörter für unterschiedliche Menschen unterschiedliche Bedeutungsnuancen haben (sie kennen sie aus unterschiedlichen Kontexten), oder dass sich die Bedeutung von Wörtern durch neue Verwendungsmuster wandelt (Kilgarriff 2006:35-6).

Eine solche Konzeption von Wortsemantik bildet in der maschinellen Sprachverarbeitung einen gangbaren Weg, da zumindest die sprachlichen Kontexte⁶ der ambigen Symbole verfügbar und maschinell erschließbar sind; so bilden Korpora eine etablierte Grundlage der maschinellen Sprachverarbeitung, die Kontexte zum Lernen der Bedeutung von Wörtern, d.h. zur Abstraktion ihrer Verwendung, zur Verfügung stellen (s. Abs. 3.1.1, S. 15).

6 Rein sprachliche Kontexte reichen jedoch wohl nicht immer, man stelle sich etwa Beispiel (1), S. 2 vor, geäußert von jemandem, der gerade zu Tür hereinkommt.

1.3.3. Arten von Mehrdeutigkeit

Eine nähere Betrachtung von Ambiguität wirft einige Fragen auf. So stellt sich etwa für die Beispiele in (2) die Frage, ob man es hier mit fünf verschiedenen Bedeutungen von *Laufen* zu tun hat oder mit einer einzigen, die sehr wandelbar ist. Zudem sind einzelne Bedeutungen offenbar enger verwandt als andere (Saeed 2003:60), hier etwa die Beispiele (2-a), (2-b), und (2-c).

- (2)
- a. Das war ein toller Lauf.
 - b. Sie läuft dreimal die Woche.
 - c. Er lief so schnell er konnte.
 - d. Die Kugel schoss aus dem Lauf.
 - e. Die Kühlflüssigkeit lief aus.

Mehrdeutigkeit wird daher häufig auf zweierlei Arten unterschieden: zum Einen wird Polysemie von Homonymie unterschieden, zum Anderen Ambiguität von Vagheit. Homonymie und Polysemie werden danach unterschieden, ob die unterschiedlichen Bedeutungen “überhaupt nichts gemeinsames” (Vater 1999:148), also “keinen semantischen Zusammenhang” (Schiehlen & Klabunde 2001:293, vgl. Saeed 2003:63) haben (Homonymie), oder ob die Bedeutungen verwandt sind (Polysemie, s. Saeed 2003:64).

Ambiguität und Vagheit werden danach unterschieden, ob es sich überhaupt um mehrere Bedeutungen handelt (Ambiguität), oder vielmehr um Variationen einer einzigen, vagen Bedeutung (Vagheit) (Langacker 2002:269). Eine andere Möglichkeit zur Definition von Polysemie im Gegensatz zu Homonymie ist die etymologische Verwandtschaft, sowie zur Definition von Vagheit im Gegensatz zu Ambiguität die Notwendigkeit einer näheren Bestimmung (wie bei *Bruder* oder *schnell*).

1.3.4. Abgrenzungsproblematik

Beide Formen der Abgrenzung sind problematisch (Saeed 2003:60, Langacker 2002:268), weil ihnen keine objektiven, einfach nachvollziehbaren Kriterien zugrunde liegen. Es existieren einige Formen von Tests zur Unterscheidung der genannten Eigenschaften, jedoch sind diese nicht eindeutig (Saeed 2003:60). Ebenso ist die grundsätzliche, mit Homonymie verbundene Vorstellung zweifelhaft, dass es zwischen zwei Lexemen mit der gleichen Form in bestimmten Fällen *gar keine* Verbindung gibt (Langacker 2002:268).

Homonymie und Polysemie, wie auch Ambiguität und Vagheit, stellen so wohl keine diskret zu analysierenden Eigenschaften von mehrdeutigen Wörtern dar, sondern eher die Enden einer Skala (Langacker 2002:270). Allgemeiner formuliert lässt sich feststellen, dass

die Erscheinungen, für die wir dasselbe Wort verwenden “in vielen verschiedenen Weisen verwandt” (Wittgenstein 1953:56) sind.

Bei einer Konzeption von Wortsinn als Kontextabstraktion macht es keinen Unterschied, ob es sich bei den Beispielen in (2) um fünf unterschiedliche Bedeutungen oder um fünf Varianten einer einzigen Bedeutung, und damit um Ambiguität oder Vagheit handelt, es handelt sich in jedem Fall um mehr oder weniger unterschiedliche Kontexte. Letztendlich stellt eine solche Sicht auch eine Orientierung auf eine datenorientierte Beschreibung (der Unterschiedlichkeit oder Ähnlichkeit der Kontexte) statt einer stärker auf Introspektion basierenden Deutung (der Beschaffenheit von Bedeutung) dar.

Ähnlichkeit und Unterschiedlichkeit der Bedeutungen sollte sich bei Verfahren, die auf maschinellem Lernen (vgl. Kap. 3, S. 15) basieren, durch unterschiedlich starke Unterscheidungen im gelernten Modell ausdrücken. In diesem Zusammenhang ist so etwa nachvollziehbar, dass Homonymie einfacher automatisch aufgelöst werden kann als Polysemie (s. Abs. 1.4): sehr unterschiedliche Kontexte sind leichter durch maschinelle Verfahren zu unterscheiden als sehr ähnliche Kontexte (wie etwa Beispiele (2-b) und (2-c), S. 6, vgl. Abs. 4.1, S. 22 zur Problematik, Ähnlichkeit im sprachlichen Bereich zu fassen).

1.4. Forschungsstand

Es ist nicht immer leicht, dem beschriebenen, vielschichtigen Bild dessen, was Bedeutung ist, in der Praxis gerecht zu werden. Ein Lexikograph etwa muss täglich konkrete Entscheidungen zur Bedeutung von Wörtern treffen (Wilks *et al.* 1996:vii), etwa wie viele Lesarten von *Laufen* (s.o.) angenommen werden, und wie diese beschrieben werden. Durch die Arbeit von Lexikographen und durch die Digitalisierung von Lexika stehen Ressourcen für die maschinelle Sprachverarbeitung zur Verfügung, in denen die Bedeutung eines Wortes auf Grundlage dieser Entscheidungen enthalten ist. Die Arbeitshypothese in der maschinellen Sprachverarbeitung und speziell der WSD, lautete daher für den Bereich der Bedeutung bislang meist: Bedeutung ist das, was im Lexikon steht.

Dies führt jedoch zu verschiedenen Problemen, die aber in der WSD-Community aus pragmatischen Gründen⁷ lange ignoriert wurden (Agirre & Edmonds 2006a:9, vgl. Kilgarriff 2006). Es stellt sich jedoch die Frage nach der Allgemeingültigkeit von Verfahren und dem Wert von Erkenntnissen auf Basis bestimmter Lexika, da z.B. unterschiedliche Anwendungen der WSD eine unterschiedliche Granularität der möglichen Bedeutungen erfordern (Ide & Wilks 2006:58).

⁷ Die Natur des WSD-Problems erfordert ein Bedeutungsinventar, aus dem die passende Bedeutung ausgewählt wird (vgl. Abs. 1.2, S. 2).

Forschungsbedarf besteht allgemein im mit diesen Fragen zusammenhängenden Bereich der Integration mit der eigentlichen Anwendung, denn entsprechend seiner Natur als Mittel sollte WSD im Kontext einer konkreten Anwendung evaluiert werden, was bisher jedoch kaum geschieht (Stevenson 2003, Agirre & Edmonds 2006a, vgl. Abs. 6.1, S. 52). Unterschiedliche Verfahren zur isolierten WSD sind dagegen inzwischen ausführlich beschrieben (Wilks *et al.* 1996, Ide & Véronis 1998, Manning & Schütze 1999, Stevenson 2003, Agirre & Edmonds 2006a).

Die maschinelle Disambiguierung von Homonymie ist erheblich einfacher als die von Polysemie, so wird automatische WSD für Homonymie mit Ergebnissen über 95% schon bei kleinen Trainingsinputs als gelöstes Problem angesehen⁸ (Agirre & Edmonds 2006a:14). WSD für Polysemie ist dagegen deutlich schwieriger, da eine Abgrenzung (vgl. Abs. 1.3.4, S. 6) der polysemen Lesarten häufig nicht einfach ist (je nach Korpus Leistungen von 60-70%, vgl. Agirre & Edmonds 2006a:14); dies gilt dabei nicht nur für maschinelle Verfahren, sondern auch für menschliche Leistung: so beträgt etwa die Übereinstimmung der menschlichen Annotatoren (*inter-annotator agreement*, ITA) für die Daten des *English Lexical Sample Task* im Rahmen von Senseval-3 (s. Kap. 5) lediglich 67,3% (Mihalcea *et al.* 2004; Agirre & Edmonds 2006a:14).

8 Eine solche Aussage, dass WSD für Homonymie gelöst sei (Agirre & Edmonds 2006a:14), heisst dabei im Grunde aber, dass WSD bisher nur für die einfachsten Fälle von Ambiguität (vgl. Abs. 1.3.4, S. 6) funktioniert.

Kapitel 2.

Abstrahierte Kognition und kognitive Abstraktion

2.1. Computerlinguistik als Kognitionswissenschaft

Die menschliche Sprachverarbeitung ist Teil der Kognition und damit Gegenstand des interdisziplinären Forschungsbereichs der Kognitionswissenschaft (Strube 2001, Jordan & Russell 1999, Chierchia 1999). Schon das einleitende Zitat von Platon drückt die Vorstellung aus, dass “reden und denken” (Platon, Phaidros 266 b, s. oben) auf den gleichen Prinzipien basiert. In den vergangenen Jahren besteht eine verstärkte Tendenz zur Integration von Linguistik (s. Schwarz 1996, Langacker 2002) und Computerlinguistik (s. Crocker 2002, Schade & Eikmeyer 2002) in die Kognitionswissenschaft.

Dabei besteht die Rolle der Computerlinguistik darin, “Simulationen von Modellen sprachverarbeitender kognitiver Prozesse” (Schade & Eikmeyer 2002:247) bereitzustellen, die eine experimentelle Evaluierung solcher Modelle ermöglichen. Aufgrund seiner Abhängigkeit etwa von logischem Denken und Weltwissen (Ide & Véronis 1998:2, Agirre & Edmonds 2006b:1) scheint WSD besonders geeignet für die Simulation allgemein-kognitiver Modelle und damit zur Umsetzung von holistischen Ansätzen der kognitiven Linguistik (vgl. Schwarz 1996:52).

2.2. Abstrahierte Kognition

2.2.1. Simulation kognitiver Prinzipien

Bei dem Bestreben, kognitive Modelle in der Computerlinguistik, und damit auf digitalen Rechnern, zu simulieren, ergibt sich das grundsätzliche Problem, dass kaum so weit von der biologischen Funktionsweise des Gehirns abstrahierte Modelle existieren, die auf so andersartiger Hardware, wie es digitale Rechner sind, umsetzbar wären:

The familiar complaint that nearly all mathematicians, physical scientists or computer scientists have about biology: too many details, too much fragmentation, and not enough questing for simple unifying principles and theories (Goertzel 2004).

Dies ist zum Teil in der Natur der biologischen Gegenstände begründet, die nicht so formal zu ordnen sind wie etwa die Gegenstände der Mathematik (Goertzel 2004), doch ist diese Fragmentierung nicht immer berechtigt:

In some ways the discipline of biology has overreacted to the diverse messiness of its subject matter and developed a culture of detail-focus that has trouble finding beautiful biological generalizations in the rare cases where they do exist (Goertzel 2004).

Der Mensch vollbringt seine höheren kognitiven Leistungen, also auch die Sprachverarbeitung, auf Grundlage seiner Großhirnrinde (engl. *neocortex*), dem “neuronale[n] Substrat aller höheren kognitiven Leistungen” (Singer 2002:145). Die Großhirnrinde kann mit einer einheitlichen Struktur unterschiedliche Funktionen ausüben; so übernehmen etwa Hirnregionen, die bei Sehenden visuelle Impulse verarbeiten, bei Blinden die Verarbeitung von akustischen Signalen. Das bedeutet, dass der menschlichen Kognition ein einheitlicher Algorithmus zugrunde liegen muss¹ (Mountcastle 1978), ein “Modus der Informationsverarbeitung [...], der zu unterschiedlichen Aufgaben genutzt werden kann” (Singer 2002:156). Wie dieser grundlegende kognitive Mechanismus funktioniert, ist für die Kognitionswissenschaft von höchstem Interesse, da zur Einordnung und Simulation kognitiver Aktivitäten eine allgemeine Theorie nötig ist:

Without a core explanation to guide inquiry, neuroscientists don’t have much to go on as they try to assemble all the details they’ve collected into a coherent picture. [...] With the proper framework, the details will become meaningful and manageable (Hawkins 2004:34).

Auf dieser Basis entwickelt Hawkins (2004) eine Theorie der menschlichen Kognition, die von Erkenntnissen der Biologie abstrahiert, um diese maschinell umsetzbar zu machen. Kerngedanke ist dabei, dass auf der Basis von gespeichertem, hierarchisch strukturier-tem Wissen (s. Abs. 2.2.2, S. 11) eine kognitive Erwartungshaltung (s. Abs. 2.2.3, S. 12) gegenüber den Wahrnehmungen aus der Umwelt entsteht, die so klassifiziert und damit erkannt werden. Hawkins (2004:104) bezeichnet diese Theorie als *memory-prediction framework* (MPF) der Kognition.

Konzeptuell bestehen dabei viele Ähnlichkeiten zu allgemeinen und kognitiven Ansätzen in Linguistik und Computerlinguistik (s. etwa Schwarz 1996, Langacker 2002), sowie anderen Bereichen der Kognitionswissenschaft (s. etwa Markert *et al.* 2005).

¹ Diese Erkenntnisse widersprechen der Vorstellung einer Universalgrammatik, bzw. eines speziellen *Language Acquisition Device* (s. etwa Jackendoff 2002:68), sondern entsprechen vielmehr holistischen Ansätzen der kognitiven Linguistik (Schwarz 1996:52).

Da WSD ein Klassifikationsproblem ist (vgl. Abs. 3.2.2, S. 17), eignet es sich zu einer Anwendung von Konzepten des MPF, das besondere Stärken in der Erkennung mehrdeutiger Muster für sich in Anspruch nimmt (z.B. Hawkins & George 2006:3-6); im MPF wird Ambiguität durch eine Suche nach passenden Mustern beim Aufsteigen in einer hierarchischen Repräsentation eines Ausschnitts der Welt aufgelöst. Eine solche Umsetzung der WSD kann als kognitiv bezeichnet werden, da sie auf allgemein kognitiven und nicht auf speziell linguistischen Prinzipien² basiert.

2.2.2. Hierarchische Wissensstrukturierung

Eine grundlegende (Wilson & Keil 1999:104, Strube 2001), wenn nicht *die* grundlegende (Harnad 2005) kognitive Aktivität des Menschen ist Kategorisierung bzw. Abstraktion.

Abstraktion ist der Prozess der Generalisierung eines Konzeptes oder beobachtbaren Phänomens durch eine Reduzierung seines Informationsgehaltes (Strube 2001). Unter Kategorisierung versteht man den Prozess der Behandlung unterschiedlicher Objekte als äquivalent (Wilson & Keil 1999:104). Kategorisierung und Abstraktion beschreiben damit denselben Vorgang aus unterschiedlicher Perspektive.

Der Begriff der Kategorien geht auf Aristoteles zurück. Der Gedanke eines generalisierten Konzeptes findet sich bereits früher, in der Ideenlehre³ von Platon. Der Prozess der Kategorisierung erfolgt dabei meist hierarchisch, d.h. in einem System von Elementen, die einander über- bzw. untergeordnet sind (vgl. Abb. 2.1, S. 12). Eine solche hierarchische Systematik (oder Klassifikation) wird zur Einteilung der Dinge in der Welt (in Taxonomien) verwendet. Eine Klassifikation wird dabei mittels Klassifizierung der Gegenstände erstellt und kann dann zur Klassierung verwendet werden. Die Tendenz geht zu einer Verwendung des Begriff *Klassifikation* analog zum englischen *classification*, für *Klassifizierung* und *Klassierung*.

Bei der Struktur in Abbildung 2.1 (S. 12) handelt es sich dabei um einen Baum, eine in bestimmter Form restringierte Form einer solchen hierarchischen Systematik, bei der jedem Element nur ein Element übergeordnet, dagegen mehrere Elemente untergeordnet werden können. Bäume sind eine bewährte Datenstruktur in Linguistik und Informatik, etwa in Form von Syntaxbäumen in linguistischen Theorien, zur hierarchischen Strukturierung von linguistischem Wissen (De Smedt & de Graaf 1990, Rolshoven 2007), sowie in Parsern, binären Suchbäumen oder XML und darauf basierenden Markup-Sprachen.

2 Es werden allerdings dennoch speziell linguistische Verfahren benötigt, um die sprachlichen Daten dem allgemein-kognitivem Mechanismus zur Verfügung zu stellen (vgl. Abs 3.2.3, S. 20).

3 Der Unterschied besteht bei Platon und Aristoteles im Status, den sie der Generalisierung geben, so betrachtet Platon die *Idee* als einen in einem jenseitigen Reich der Ideen existenten Gegenstand, während für Aristoteles die *Substanz* der Dinge in ihnen selbst liegt.

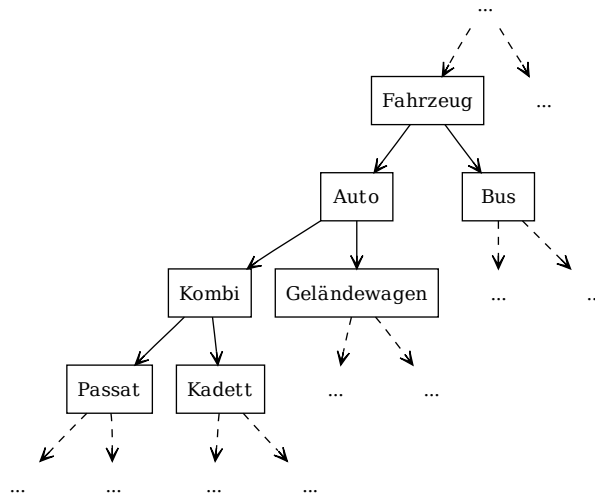


Abbildung 2.1.: Hierarchische Abstraktion sprachlicher Symbole

Auf diese Weise strukturierte Repräsentationen ermöglichen, von konkreten Wahrnehmungen zu abstrahieren, und damit, etwa unterschiedliche Kombis alle als Kombi, unterschiedliche Autos alle als Auto und unterschiedliche Fahrzeuge alle als Fahrzeug zu erkennen (vgl. Abb. 2.1).

In solchen Hierarchien findet dabei im MPF eine Kommunikation sowohl vorwärts wie rückwärts statt (vgl. Abs. 4.2.4, S. 33). Diese Ansicht gründet Hawkins auf die messbare Vorwärts- und Rückwärtskommunikation⁴ in der Großhirnrinde. Die Rückwärtskommunikation deutet Hawkins (2004:113) als Erwartung, welche die Großhirnrinde zurück an die Sinne sendet.

2.2.3. Erwartung und bedingte Wahrscheinlichkeit

Zur Veranschaulichung der hohen Bedeutung der Erwartung für die Kognition beschreibt Hawkins (2004) folgendes Beispiel:

Suppose while you are out, I sneak over to your home and change something about

⁴ Vorwärtskommunikation erfolgt dabei von direkt mit den Sinnesorganen verbundenen Regionen zu konzeptuell tieferen Regionen; dies entspricht der klassischen Vorstellung von Wahrnehmung. Rückwärtskommunikation erfolgt in Form von Signalen konzeptuell tieferer Schichten zurück zu den mit den Sinnen verbundenen Regionen.

your door. It could be almost anything. [...] When you come home that day and attempt to open the door, you will quickly detect that something is wrong. As your hand reaches for the moved knob, you will realize that it is not in the correct location. [...] Or if the door's weight has been changed, you will push with the wrong amount of force and be surprised. The point is that you will notice any of a thousand changes in a very short period of time (Hawkins 2004:87).

Diese Leistung der menschlichen Kognition ist für Hawkins nicht mit bisherigen Ansätzen der künstlichen Intelligenz (*artificial intelligence*, AI) erklärbar:

The AI or computer engineer's approach to this problem would be to create a list of all the door's properties and put them in a database [...]. The AI strategy is implausible. First, it is impossible to specify in advance every attribute a door can have. The list is potentially endless. Second, we would need to have similar lists for every object we encounter every second of our lives. Third, nothing we know about brains and neurons suggest that this is how they work. And finally, neurons are just too slow to implement computer-style databases. It would take twenty minutes instead of two seconds to notice the change as you go through the door (Hawkins 2004:88).

Bisherige Umsetzungen des MPF basieren zur Modellierung der Erwartung auf dem *Belief Propagation* (BP) Algorithmus. BP kann als eine Form von von Bayes-Klassifikation charakterisiert werden, bei der auf Grundlage des Bayesschen Theorems die wahrscheinlichste Klasse zu einem gegebenen Kontext ermittelt wird (vgl. Abs. 3.2.2, S. 17 und 4.2.4, S. 33). Die BP findet dabei hier in Bäumen statt, was zwei zentrale Anforderung des MPF, die hierarchische Wissenstrukturierung und die Erwartung, miteinander verbindet.

Durch die Modellierung der Erwartung mit BP, und damit letztendlich mit bedingten Wahrscheinlichkeiten, erreichen solche Umsetzungen des MPF eine Verbindung von statistischen, scheinbar kognitiv nicht plausiblen, aber funktionierenden Ansätzen, mit kognitiven Erklärungen: was erwartet wird ist das Wahrscheinlichste unter den gegebenen Umständen. Dabei ist eine statistische Methode in einem solchen Kontext kein an sich plausibles Verfahren, sondern eine Modellierung des plausiblen Konzepts, welches zukünftig auch anders umgesetzt werden könnte, als wie hier mit BP.

2.3. Forschungsstand

Konzepte und Implementationen des MPF sind ausführlich (Hawkins 2004, Hawkins & George 2006, George & Jaros 2007) beschrieben und wurden im Bereich der visuellen

Mustererkennung experimentell eingesetzt (Hawkins & George 2005, Thornton *et al.* 2006, Garalevicius 2007), etwa mit vielversprechenden Ergebnissen zur OCR (Thornton *et al.* 2006). Implementationen existieren speziell zur visuellen Mustererkennung in C++ (Garalevicius 2007) sowie in Ansätzen für einen domänenunabhängigen Einsatz in Python und C++ (durch die Firma Numenta, Inc., vgl. George & Jaros 2007).

BP in hierarchischen Netzen wurde schon vereinzelt im Bereich der maschinellen Sprachverarbeitung eingesetzt, etwa zur Textklassifikation (Vaithyanathan *et al.* 2000). Konzeptuell sind neben Bayesschen Netzen und hierarchischer Wissensstrukturierung auch auto-assoziative Speicher (nach Kohonen 1984, s. etwa Lämmel & Cleve 2001:239) und eine Erweiterung der Klassifikation um zeitliche Sequenzialität (die für die Kognition allgemein sehr wichtig ist, vgl. Markert *et al.* 2005) konzeptuelle Grundlagen des MPF.

Im Folgenden sollen in diesem Sinn allgemeine Konzepte der Kognition zur Erkennung sprachlicher Sachverhalte genutzt werden. Dies erfolgt konzeptuell durch eine Anwendung des grundsätzlich in der beschriebenen Weise simulierten kognitiven Apparats auf den Ausschnitt der Welt, in dem die gesuchten Informationen (der Wortsinn, vgl. Abs. 1.3.2, S. 4) enthalten sind, nämlich den Kontext des zu disambiguierenden Wortes.

Konkret orientiert sich das in dieser Arbeit umgesetzte Verfahren dabei an der Beschreibung in Thornton *et al.* (2006). Die Umsetzung als Klassifikationsverfahren mit numerischer Eingabe (vgl. Abs. 3.2.2, S. 17) ermöglicht dabei die direkte Umsetzung der beschriebenen Konzeption eines einheitlichen Algorithmus für die Verarbeitung von verschiedenen Sinneswahrnehmungen, in Thornton *et al.* (2006) zur OCR mit optischen Daten, hier zur WSD mit sprachlichen Daten (Kap. 4, vgl. Abs. 7.1, S. 60).

Kapitel 3.

Korpuslinguistik und maschinelles Lernen

Zu der in Kapitel 2 beschriebenen Vorstellung eines einheitlichen kognitiven Verfahrens, das verschiedene Sinneswahrnehmungen verarbeitet, gibt es es eine direkte Entsprechung im Bereich des maschinellen Lernens, wo einheitliche Verfahren zum Lernen auf der Basis von Daten verschiedener Bereiche (z.B. Text, Audio- und Videodaten) verwendet werden. Dieses Kapitel beschreibt auf dieser Grundlage die Anwendung von maschinellem Lernen in der Sprachverarbeitung, basierend auf annotierten Korpora.

3.1. Maschinellen Lernen in der Sprachverarbeitung

Das überwachte (s. Abs. 3.2.1, S. 17) maschinelle Lernen aus Korpora ermöglicht die Reproduktion menschlicher Intuition anhand empirischer Daten (s. Abs. 3.1.1, S. 15) und bildet daher eine etablierte Grundlage des maschinellen Lernens in der Computerlinguistik, die der in Abschnitt 1.3.2 (S. 4) beschriebenen Konzeption von Wortbedeutung als Kontextabstraktion gerecht wird. Da WSD ein Klassifikationsproblem ist, ist es für das maschinelle Lernen besonders geeignet (s. Abs. 3.2.2, S. 17).

3.1.1. Korpuslinguistik

Korpuslinguistik ist eine sprachwissenschaftliche Arbeitsweise und umfasst verschiedene Tätigkeiten, die mit der Erstellung und der Auswertung von Textkorpora (Köhler 2005) sowie mit der Erstellung von Werkzeugen für die beiden ersten Tätigkeiten zu tun haben. McEnery (2003) definiert ein Textkorpus als “a large body of linguistic evidence¹”. Damit sind Korpora eine sehr wertvolle Quelle für die linguistische Arbeit.

Grundlage für Korpora können sowohl gesprochene Sprache (z.B. IBM/Lancaster Spoken English Corpus; gesprochener Teil des British National Corpus, BNC), als auch schriftliche Veröffentlichungen (z.B. schriftlicher Teil des BNC) sein.

1 vgl. Labov (1975, 1996) zur Debatte um die Rolle der Daten in der Linguistik.

Korpuslinguistische Methoden werden nicht nur in der maschinellen Sprachverarbeitung angewendet, sondern auch in anderen Bereichen der Sprachwissenschaft, etwa in der allgemeinen Sprachwissenschaft oder den Philologien. Somit ist Korpuslinguistik kein Teilbereich der Linguistik, sondern eine Arbeitsweise, die innerhalb der genannten Teilbereiche verwendet wird und auf verschiedene Ebenen der Sprache (wie Morphologie, Syntax, Semantik oder Pragmatik) angewendet werden kann (McEnery & Wilson 1996:2). Da ein großer Teil der Auswertung von Korpora mithilfe von statistischen Verfahren erfolgt, wird diese Art der Arbeit mit Korpora auch als *Quantitative Linguistik* bezeichnet (s. Köhler 2005, Manning & Schütze 1999).

McEnery (2003:448) bezeichnet Textkorpora als “the raw fuel of NLP” und damit als Grundlage und Voraussetzung der maschinellen Sprachverarbeitung. Der Grund hierfür besteht darin, dass annotierte (s.u.) Korpora eine maschinelle Reproduktion menschlicher Intuition ermöglichen (edb.).

3.1.2. Korpusannotation

Annotierte Korpora sind Korpora, die mit verschiedenen Arten linguistischer Information angereichert sind (McEnery & Wilson 1996:24). Ein Beispiel für Korpusannotationen ist etwa die Kennzeichnung von Wortarten durch POS-Tags. Andere Annotationen enthalten etwa Informationen über Stammformen (als Ergebnis einer Stammformenreduktion, auch *Stemming* oder *Lemmatisierung* genannt), über die syntaktische Struktur (solche Korpora werden auch *Baumbanken* genannt) sowie semantische, stilistische oder Informationen zur Diskursstruktur. Dabei sind morphologische Annotationen verbreiteter als syntaktische Annotationen und diese wiederum verbreiteter als semantische oder pragmatische Annotationen.

Eine solche Anreicherung basiert immer auf einer bestimmten Interpretation der Daten (McEnery 2003). Beim Annotieren ist daher zu beachten, dass der ursprüngliche Text auch nach einer Annotierung noch in seiner Rohform verfügbar sein sollte, etwa durch *dynamische Annotation* (s. Benden & Hermes 2004, Hermes & Benden 2005, vgl. Kap. 6, S. 52).

Annotierte Korpora ermöglichen es Computerprogrammen, die Intuitionen von Experten, die die Annotationen erstellt haben, in Bezug zu den Texten zu setzen und so menschliche Intuitionen zu reproduzieren. Ein Beispiel hierfür sind etwa auf Hidden-Markov-Modellen (HMM) basierende POS-Tagger, die aus annotierten Korpora lernen und dadurch in die Lage versetzt werden, neue Texte zu annotieren (s. etwa Manning & Schütze 1999). So bilden annotierte Korpora die Grundlage für maschinelles Lernen in der Sprachverarbeitung (McEnery 2003:459). Das Lernen aus Korpora entspricht dabei der in Kapitel 1

beschriebenen Konzeption vom Wortsinn als Abstraktion der Kontexte eines Wortes.

3.2. Klassifikation

3.2.1. Überwachtes und unüberwachtes Lernen

Eine grundsätzliche Unterteilung von Verfahren des maschinellen Lernens erfolgt häufig danach, ob zu den Trainingsbeispielen eine Klasse aus einem festgelegten Inventar angegeben wird (überwachtes Lernen), oder ob das Verfahren unannotierte Texte verarbeitet und die Klassenbildung selbst vornimmt (unüberwachtes Lernen, häufig auch als *Clustering* bezeichnet). In der WSD gibt es insofern eine terminologische Verwirrung, als dass häufig Verfahren als unüberwacht bezeichnet werden, die zwar unannotierte Korpora verarbeiten, jedoch auf lexikalischen Ressourcen basieren, in welchen die möglichen Klassen spezifiziert sind, und damit als überwachte Verfahren einzustufen sind (vgl. Màrquez *et al.* 2006:171; Pedersen 2006:134-5).

Das in dieser Arbeit entwickelte Verfahren basiert auf annotierten Korpora (s.o.) und ist daher als überwachtes Verfahren zu kennzeichnen. Gleichzeitig besteht keine Abhängigkeit von lexikalischen Ressourcen, es handelt sich daher um ein grundsätzlich sprachunabhängiges Verfahren, das die zugrunde liegenden Klassen allein aus den Trainingsbeispielen lernt. Die Abhängigkeit von speziell annotierten Trainingskorpora stellt ein Problem solcher Ansätze des überwachten Lernens dar, das auch als *knowledge aquisition bottleneck* bezeichnet wird (Màrquez *et al.* 2006:172; 196). Von Hand disambiguiertes Material erfordert viel Arbeit, bildet aber bei ausgewogenem, repräsentativem Korpus solide, empirische Daten (McEnery 2003); bei guter Integration ist eine manuelle Annotation zudem durchaus realistisch, wie die in modernen Email-Clients integrierten Spam-Filter zeigen. Es gibt verschiedene Ansätze zur Umgehung dieses Problems, etwa durch eine automatische Erstellung von Trainingsbeispielen (z.B. mit dem *Yarowski Bootstrapping Algorithmus*, s. Màrquez *et al.* 2006:181) oder durch die Verwendung von Übersetzungen eines Wortes als Klasse für dieses, basierend auf bilingualen, Wort für Wort ausgerichteten Korpora (Màrquez *et al.* 2006:172).

3.2.2. Überwachtes Lernen zur Klassifikation

Besonders für das maschinelle Lernen geeignet sind Klassifikationsprobleme, unter anderem, weil diese gut erforscht und in verschiedenen Ausprägungen beschrieben sind (Màrquez *et al.* 2006:168). WSD kann unmittelbar als Klassifikationsproblem formuliert werden: eine Instanz eines ambigen Wortes soll einer seiner möglichen Lesarten (den Klas-

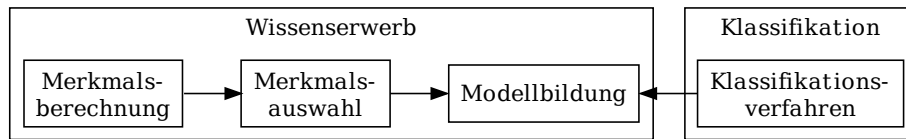


Abbildung 3.1.: Generische Architektur eines Klassifikationssystems (aus: Brückner 2001)

sen) zugewiesen (und damit klassifiziert) werden. Zahlreiche Probleme, die nicht unmittelbar als Klassifikationsproblem beschrieben werden können, lassen sich, etwa durch eine Verallgemeinerung des Klassifikationsmechanismus oder durch ein Aufbrechen der Aufgabe in kleinere Teile, als Klassifikationsprobleme beschreiben (Màrquez *et al.* 2006:168).

Klassifikationssysteme bestehen konzeptuell aus zwei Hauptkomponenten: dem Wissenserwerb und der eigentlichen Klassifikation. Der Wissenserwerb gliedert sich dabei in drei weitere Teile: erstens der Merkmalsberechnung, bei der die Merkmale ermittelt werden, anhand derer klassifiziert werden soll; zweitens der Merkmalsauswahl, bei der die relevanten Merkmale ausgewählt werden und schließlich drittens der Modellbildung. Das so im Wissenserwerb gebildete Modell wird dann bei der eigentlichen Klassifikation verwendet (s. Abb. 3.1, vgl. Brückner 2001).

Formal lässt sich (nach Màrquez *et al.* 2006:168) das Ziel des überwachten Lernens zur Klassifikation als Induktion einer Hypothese h der unbekanntem Funktion f beschreiben, die einer Eingabe X eine Ausgabe Y zugeordnet. Die Trainingsmenge enthält m Beispiele $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$, wobei x Teil von X ist und $y = f(x)$, d.h. x ist Teil der Eingabemenge und y ist die korrekte Klasse für x . Das Element x eines jeden Beispiels ist üblicherweise ein Vektor $x = (x_1, \dots, x_n)$, dessen als *Merkmale* bezeichneten, reellwertigen Elemente die Form sind, in der das Trainingsbeispiel dargestellt wird.

Ein Vorteil der Verwendung einer solchen standardisierten Schnittstelle zwischen Merkmalsberechnung und Klassifikation besteht in der Ermöglichung des Einsatzes und Vergleichs verschiedener Klassifikationsalgorithmen unter gleichen Voraussetzungen (in Form von gleichen Merkmalen), sowie umgekehrt der Umsetzung eines bestimmten Klassifikationsalgorithmus für verschiedene Aufgaben – im Kontext dieser Arbeit etwa die Umsetzung des Verfahrens zur OCR aus Thornton *et al.* (2006) für die WSD.

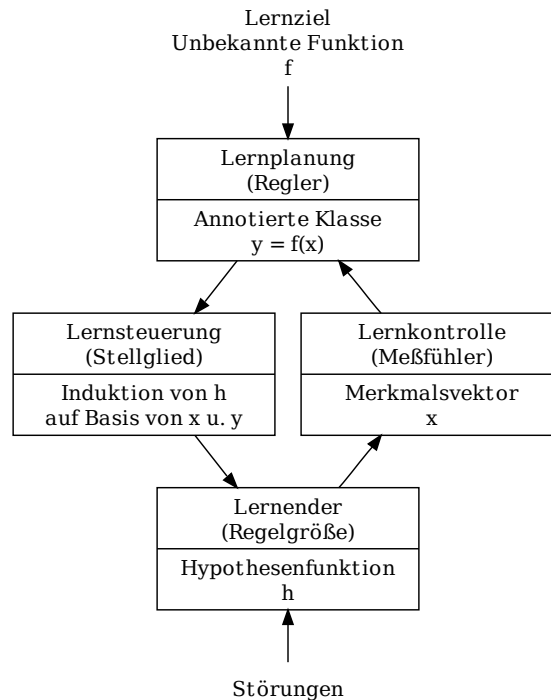


Abbildung 3.2.: Regelkreismodell der Didaktik aus Von Cube (1965:187), im unteren Teil jeweils ergänzt für die Terminologie aus Màrquez *et al.* (2006:168)

Die Hypothesenfunktion h entstammt dem Hypothesenraum H . Verschiedene Lernalgorithmen unterscheiden sich danach, welchen Teil von H sie berücksichtigen sowie in ihrer internen Repräsentation und in der Strategie zur Auswahl von h (Màrquez *et al.* 2006:168-9). Das beschriebene Verfahren kann als wahrscheinlichkeitstheoretische Methode (*probabilistic method*, vgl. Màrquez *et al.* 2006:175) gekennzeichnet werden, da hier die korrekte Klasse y als diejenige mit der höchsten bedingten Wahrscheinlichkeit unter Voraussetzung des beobachteten Kontextes x , repräsentiert durch m Merkmale, also $\max P(y|x_i, \dots, x_m)$, angenommen wird (vgl. Abs. 4.2.4, S. 33).

Maschinelles Lernen basiert auf generellen Konzepten des Lernens, die bereits vor einer maschinellen Anwendung in der Kybernetik – der Wissenschaft von der Struktur komplexer Systeme – beschrieben wurden. So lässt sich das hier umgesetzte maschinelle Lernen zur WSD im Rahmen des Regelkreismodells der Didaktik aus Von Cube (1965) beschreiben (vgl. Abb. 3.2).

3.2.3. Numerische Kontextrepräsentation

Die Frage, welche Merkmale zur Repräsentation der Kontexte gewählt werden sollten, und wie diese numerisch repräsentiert werden, ist der domänenspezifische Teil der Klassifikation. Hier erfolgt die Umwandlung der sprachlichen Informationen zum Kontext eines ambigen Zielwortes in eine numerische Darstellung. Dies stellt den ersten Schritt einer Abstraktion der sprachlichen Daten dar (Márquez *et al.* 2006:174).

Verfahren zur numerische Repräsentation für *Types* sind im Bereich des unüberwachten Lernens in unterschiedlicher Ausprägung beschrieben (Pedersen 2006); solche Verfahren repräsentieren semantische verwandte *Types*, hypothetisch etwa *Dogge* und *Pudel*, mit ähnlichen Merkmalsvektoren. Zur WSD wird jedoch eine Repräsentation des Kontexts unterschiedlicher Vorkommen eines *Token* benötigt, da ja unterschiedliche Lesarten identischer Wortformen erkannt werden sollen. Die *Type*-basierten Ansätze für überwachte Verfahren (Pedersen 2006) wurden zum Teil für unüberwachte (Márquez *et al.* 2006) Verfahren der WSD angepasst.

Das in der vorliegenden Arbeit umgesetzte Verfahren aus Thornton *et al.* (2006) basiert, entsprechend der in Kapitel 2 beschriebenen Konzepte, darauf, dass ähnliche Merkmale gruppiert werden. Dazu müssen ähnliche Merkmale durch ähnliche numerische Werte repräsentiert werden. In anderen Domänen, in denen Verfahren des maschinellen Lernens eingesetzt werden, etwa der optischen Mustererkennung, ist eine Ähnlichkeit leichter auszudrücken als in der Sprachverarbeitung; so kann etwa für jeden Pixel eines Piktogramms auf einer Skala von 0 (ganz dunkel) bis 1 (maximale Helligkeit) ausgedrückt werden, wie ähnlich zwei Pixel sind.

Im Bereich sprachlicher Daten ist die Frage der Ähnlichkeit hingegen schwerer zu fassen; so sind zwar Wörter nach morphologischen Kriterien ähnlich, wenn sie ähnliche Buchstaben enthalten (etwa *Geher* und *gehen*), eine semantische Ähnlichkeit (etwa von *gehen* mit *laufen*) ist dagegen nur schwer zu ermitteln und auszudrücken – abgesehen von dem seltenen Fall², dass ein Lexikon vorhanden ist, das solchen Zusammenhänge beschreibt (wie etwa WordNet).

Benötigt würde also eine numerische Repräsentation des Kontextes eines *Token*, dessen Werte für ähnliche Symbole im Kontext ähnlich sind. Zur Annäherung an dieses Ziel werden in Kapitel 4 (Abs. 4.1, S. 22) verschiedene simple Verfahren implementiert, die in Kapitel 5 (S. 40) vergleichend evaluiert werden. Der Grund für die Umsetzung verschiedener simpler Verfahren zur Merkmalsberechnung anstelle der direkten Umsetzung eines einzelnen, komplexen Verfahrens ist die so gegebene Möglichkeit einer systemati-

² Es existieren neben dem englischsprachigen WordNet einige sprachspezifische semantische Netze, etwa GermaNet oder EuroWordNet, diese sind jedoch im Umfang nicht mit WordNet vergleichbar.

schen Annäherung an ein geeignetes Verfahren. Auf Grundlage dieser ersten Erkenntnisse könnte dann ein geeignetes komplexes Verfahren ausgewählt oder angepasst werden (vgl. Abs 7.2.2, S. 62).

Kapitel 4.

Implementierung

Aufbauend auf die Konzepte des maschinellen Lernens aus Kapitel 3 (S. 15) folgt nun die Beschreibung der Implementierung des WSD-Verfahrens. Die Datenbasis der Hypothesenfunktion h (vgl. Abs. 3.2.2, S. 17) wird dabei, den kognitiven Konzepten aus Kapitel 2 (S. 9) folgend, in Bäumen und auf Basis von Korpora induziert.

Die Umsetzung orientiert sich an der Beschreibung in Thornton *et al.* (2006), wo die Anwendung im Bereich der OCR liegt. Hierbei ist zu untersuchen, inwieweit das Verfahren auf die Sprachdomäne übertragbar ist und welche Voraussetzungen dafür möglicherweise noch zu erfüllen sind. Bevor auf den eigentlichen Klassifikationsmechanismus eingegangen werden kann (Abs. 4.2, S. 28), erfolgt eine Beschreibung der Merkmalsberechnung (Abs. 4.1, S. 22).

Konkret soll ein WSD-Verfahren entwickelt werden, das unabhängig von einer bestimmten lexikalischen Ressource ist, sondern Bedeutung allein durch die Verwendung der Wörter erfasst; bisher wurden WSD-Verfahren meist auf Basis bestimmter Lexika entwickelt, in der Regel WordNet; eine Übertragbarkeit solcher Verfahren für andere Anwendungsfälle oder Sprachen ist aber nicht gegeben (vgl. Abs. 1.4, S. 7). Eine solche ressourcenunabhängige Umsetzung erfordert korpusbasierte Verfahren des maschinellen Lernens (vgl. Kap. 3, S. 15).

4.1. Merkmalsberechnung

In den bisherigen Umsetzungen des MPF hat der Merkmalsvektor (vgl. Abs. 3.2.2, S. 17) eine Größe von 64 (Hawkins & George 2005) bis 100 (Thornton *et al.* 2006) Merkmalen. Die Verwendung von wortbasierten Merkmalen in Vektoren solcher Größen ergäbe dabei Kontextfenster von ± 32 bzw. ± 50 Wörtern, einer Größe, die für die WSD nicht sinnvoll ist (dies wird durch die Experimenten in Kapitel 5 bestätigt), da durchaus wahrscheinlich ist, dass sich die Kontexte unterschiedlicher Lesarten, insbesondere bei polysemen Wörtern, bei solchen Kontextfenstern überlappen.

Zur Annäherung an ein geeignetes Verfahren zur Merkmalsberechnung und damit zur

Kontextrepräsentation (vgl. Abs. 3.2.3, S. 20) werden hier verschiedene Methoden der Merkmalsberechnung implementiert und evaluiert: die Wörter (Abs. 4.1.1, S. 23), Wortlängen (Abs. 4.1.2, S. 24), buchstabenbasierten Tri- und Heptagramme (Abs. 4.1.3, S. 24) sowie Paradigmen¹ (Abs. 4.1.4, S. 25) im Kontext des Zielwortes.

Dabei wird bei allen Verfahren zur Berechnung der numerischen Repräsentation der Merkmale nur der Teil des Korpus verwendet, der in den zu verwendenden Kontextfenstern vorkommt, da sonst die berücksichtigten Daten nur einen geringen Anteil des Gesamtkorpus bilden, und dadurch alle Merkmale zu ähnlich repräsentiert würden (s.u.).

4.1.1. Wörter

Zur Verwendung der Wörter als Merkmale wird die Menge aller Wörter in allen Kontexten (und damit alle *Types*) alphabetisch sortiert und anschließend fortlaufend durchnummeriert. Dann werden die Werte normalisiert, indem jeder Wert durch die Anzahl der Wörter geteilt wird. Der erste *Type* erhält dadurch den Wert 0, der letzte den Wert 1. Eine Ähnlichkeit der numerischen Werte entspricht damit einer Ähnlichkeit im Sinne einer alphabetischen Sortierung, wie etwa im folgenden Ausschnitt der sortierten Liste:

```
...
offend      0.63102
offended    0.63104
offender    0.63106
offenders   0.63108
offending   0.63110
offense     0.63113
offensive   0.63115
...
```

Da verschiedene Formen eines Wortes bei alphabetischer Sortierung nah beieinander stehen (s.o.) und gemäß Konzepten des MPF ähnliche Merkmale gruppiert werden (s. Abs. 4.2, S. 28), leistet eine solche Merkmalsauswahl und -berechnung im beschriebenen Verfahren eine Form von sprachunabhängiger Stammformenreduzierung – unter der Voraussetzung, dass die Möglichkeit einer sprachspezifische Sortierung vorhanden ist, wie dies etwa für Java mittels der Klasse `RuleBasedCollator` für eine Vielzahl von Sprachen der Fall ist.

¹ *Paradigma* bezeichnet hier und nachfolgend eine Menge von Wörtern, die zueinander in paradigmatischer Relation stehen, d.h. die in gleichen Kontexten vorkommen und damit austauschbar sind (vgl. Abs. 4.1.4, S. 25).

4.1.2. Wortlängen

Bei der Verwendung von Wortlängen als Merkmale wird prinzipiell so vorgegangen, wie oben für Wörter beschrieben, allerdings entspricht hier der Wert 0 dem kürzesten Wort und der Wert 1 dem längsten Wort, alle anderen Werte werden durch die Division der Länge eines Wortes durch die Länge des längsten Wortes berechnet. Auf diese Weise erhalten Wörter ähnlicher Länge ähnliche Werte, z.B. für den gleichen Ausschnitt wie oben:

...	
offend	0.2
offended	0.27
offender	0.27
offenders	0.3
offending	0.3
offense	0.24
offensive	0.3
...	

Eine solche Merkmalsrepräsentation drückt durchaus numerisch eine Ähnlichkeit der repräsentierten Wörter aus, so sind etwa Funktionswörter in vielen Sprachen tendenziell kürzer als Inhaltswörter. Allerdings ist das Spektrum der möglichen Werte sehr gering, so dass zu erwarten ist, dass nur wenige unterschiedliche Werte zur Repräsentation verwendet werden, und so die Möglichkeiten von auf solchen Merkmalen aufbauenden Klassifikationsverfahren nicht ausgereizt werden.

4.1.3. Trigramme und Heptagramme

Wie in Abschnitt 4.1 (S. 22) angedeutet, wäre erstrebenswert, größere Merkmalsvektoren für lokale Kontexte zu verwenden, als dies mit den beschriebenen, wortbasierten Verfahren möglich ist. Eine Möglichkeit dazu bietet die Verwendung von buchstabenbasierten N-Grammen, d.h. Gruppen einer bestimmten Anzahl (N) von Buchstaben, die kürzer als Wörter sein können und so größere Vektoren für gleichbleibend kleine Kontexte ermöglichen. Eine weitere positive Eigenschaft von buchstabenbasierten N-Grammen ist, dass sie (wie die oben beschriebene Sortierung von Wörtern) eine Form von ressourcen- und sprachunabhängiger Stammformenreduktion leisten, so ist etwa das Trigramm *bau* in unterschiedlichen Formen wie *Aufbau*, *bauen*, *Abbau*, *Bauten* oder *baut* enthalten. Konkret werden Trigramme (3-Gramme) und Heptagramme (7-Gramme) zur Evaluierung in Kapitel 5 verwendet.

Die Berechnung der numerischen Repräsentation der N-Gramme erfolgt entsprechend Algorithmus 1 (S. 25). Eine solche Umsetzung wird der Forderung nach ähnlichen numerischen Werten für ähnliche Eingaben insofern gerecht, als dass bei fortlaufend nummerier-

Eingabe : Eine Zeichenkette s , die sortierte Liste aller möglichen Zeichen l

Ausgabe : Ein ganzzahliger Wert für s

```

1  $n \leftarrow 0$ ;
2 foreach Zeichen  $z$  in  $s$  do
3    $i \leftarrow$  Position von  $z$  in  $l$ ;
   // Da die Zeichenketten immer gleich lang sind, ergeben sich aus
   // der folgenden Berechnung für ähnliche Zeichenketten ähnliche
   // Werte:
4    $n \leftarrow 31 * n + i$ ;
5 return  $n$ ;
```

Algorithmus 1 : Numerische Abstraktion für Tri- und Heptagramme

ten, sortierten Buchstaben die Berechnung² in Zeile 4 von Algorithmus 1 für Zeichenketten gleicher Länge mit ähnlichen Zeichen ähnliche Werte ergibt.

Die Normalisierung der durch Algorithmus 1 berechneten Werte erfolgt wie in den zuvor geschilderten Fällen durch Division aller Werte durch den maximalen Wert, so würde prinzipiell etwa dem Trigramm *aaa* der Wert 0 und dem Trigramm *zzz* der Wert 1 zugewiesen.

Wie die Merkmalsberechnung für Wörter (vgl. Abs. 4.1.1, S. 23), ist auch dieses Verfahren zur Merkmalsberechnung – eine sprachspezifische Sortierung vorausgesetzt – prinzipiell sprachunabhängig.

4.1.4. Paradigmen

Er wäre erstrebenswert, dass nicht nur bestimmte Wörter im Kontext eines Zielwortes berücksichtigt werden, sondern dass semantisch verwandte Wörter gleichermaßen berücksichtigt werden. Würde etwa ein Paradigma wie [Finanzen, Geld, Aktien, Börse] anstelle eines einzelnen Wortes wie *Geld* als Merkmal von *Bank* in der Lesart ‘Geldinstitut’ verwendet, sollte das Verfahren in der Lage sein, mehr neue Kontexte korrekt zu erkennen.

Es gibt semantische Netze, d.h. Lexika, die semantische Relationen paradigmatischer Art enthalten. Das bekannteste Beispiel hierfür ist das englischsprachige WordNet, welches vielfach zur WSD eingesetzt wird. Das Problem eines solchen Ansatzes besteht jedoch in der Abhängigkeit von solchen, nur unter hohem Aufwand zu erstellenden, Ressourcen; so existieren in anderen Sprachen keine semantischen Netze, die im Umfang mit WordNet

² Diese entspricht der Berechnung des Hashcodes für Strings in Java, vgl. die Implementation in Anh. A.1, S. I

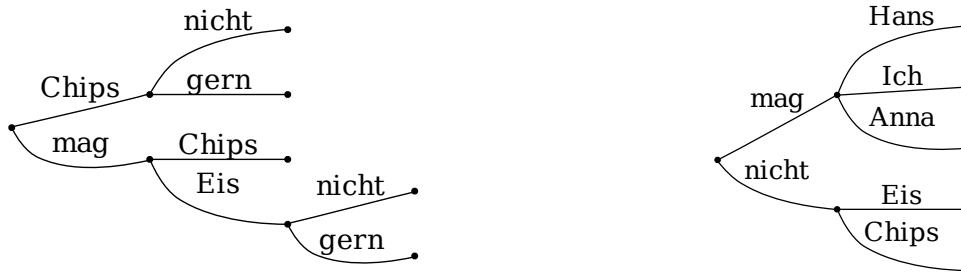


Abbildung 4.1.: Ausschnitte des Suffix- und des Präfixbaums mit Paradigmen (für den Text *Anna mag Chips nicht. Ich mag Chips gern. Hans mag Eis nicht. Ich mag Eis gern.*)

vergleichbar wären. Aussichtsreicher wäre daher eine automatische Ermittlung paradigmatischer Relationen.

Eine solche Analyse, auch von großen Datenmengen, ist etwa mit Suffixbäumen möglich, einer bisher vor allem in der Bioinformatik verwendeten, vielseitigen³ und effizient⁴ nutzbaren Datenstruktur. Ein Suffixbaum enthält Symbolsequenzen, wobei mehrfach auftretende Teilsequenzen nur einmal im Baum enthalten sind (vgl. Gusfield 1997:89). So enthält ein Suffixbaum Informationen über die paradigmatische und die syntagmatische Struktur des Eingabetextes (s. Abb. 4.1, links).

Ein Suffixbaum mit Buchstaben als Symbole enthält so Informationen über potentielle Morpheme und hat so möglicherweise Potential zur sprachunabhängigen Stammformenreduktion. Ein Suffixbaum mit Wörtern als Symbole enthält Informationen über sich potentiell semantisch nahestehende (da in gleichen Kontexten auftretende) Wörter. Durch die Verwendung von Suffix- und Präfixbäumen (d.h. für den umgekehrten Eingabetext aufgebaute Suffixbäume, vgl. Abb. 4.1, rechts) werden im hier umgesetzten Verfahren dabei Paradigmen am Anfang und am Ende von Sätzen ausgelesen (s. Abb. 4.1, vgl. Geertzen 2003). Ein solches Vorgehen stellt eine Umsetzung von grundlegenden strukturalistischen Vorstellungen der Linguistik dar (s. etwa Lyons 1968:70). Zur Ermittlung der Paradigmen werden die Beschriftungen der Kanten von inneren Knoten zu ihren Kindern im Suffix- und im Präfixbaum ausgelesen (s. Algorithmus 2, S. 27).

3 Gusfield (1997) nennt eine Vielzahl von Anwendungsfällen für Suffixbäume zum exakten (1997:122-93) und weichen (1997:196-207,279) Stringvergleich.

4 Ein Suffixbaum ist mit linearer Zeit- und Speicherplatzkomplexität konstruierbar, also $O(n)$ für einen Text der Länge n und ermöglicht danach etwa die Suche nach einem Muster der Länge m linear zur Länge des Musters, also $O(m)$ (Gusfield 1997:122).

Eingabe : Suffixbaum s

Ausgabe : Paradigmen in s

```

1  $a \leftarrow$  leere Liste von Paradigmen;
2 foreach innerer Knoten  $n$  do
3    $p \leftarrow$  leere Liste von Strings;
   // Zur Ermittlung der Paradigmen werden die Beschriftungen von
   // Kanten innerer Knoten zu ihren Kindern im Suffixbaum ausgelesen:
4   foreach  $k$ , Kind von  $n$  do
5      $m \leftarrow$  Beschriftung der eingehenden Kante von  $k$ ;
6     Füge  $m$  zu  $p$  hinzu;
7   Füge  $p$  zu  $a$  hinzu;
8 return  $a$ ;
```

Algorithmus 2 : Auslesen von Paradigmen aus einem Suffixbaum

Zur Anwendung im implementierten Verfahren wird nun eine numerische Repräsentation der Paradigmen benötigt. Grundüberlegung hierzu ist, dass, eine sinnvolle Sortierung der Paradigmen vorausgesetzt, die numerischen Werte durch Normalisierung wie für die zuvor geschilderten Merkmale erfolgen kann.

Erreicht wird dies durch eine alphabetische Sortierung der Mitglieder eines Paradigmas sowie einer anschließenden Sortierung aller Paradigmen nach ihrem ersten sich unterscheidenden Mitglied (vgl. Algorithmus 3 und die Implementation in Anh. A.1, S. I). Eine solche Sortierung führt dazu, dass Paradigmen mit gemeinsamen Mitgliedern nacheinander erscheinen, und so – nach einer wie oben vorgenommenen Normalisierung – eine ähnliche numerische Repräsentation ähnlichen Paradigmen entspricht.

Eingabe : In sich sortierte Paradigmen a und b

Ausgabe : Sortierfolge für a und b oder 0 wenn $a = b$

```

1 foreach korrespondierende Position  $i$  in  $a$  und  $b$  do
2   if  $a_i \neq b_i$  then
3     // Wir sortieren die Paradigmen nach ihrem ersten sich
     // unterscheidenden Element:
     return Sortierfolge für  $a_i$  und  $b_i$ ;
   // Wenn alle Elemente gleich sind, sind die Paradigmen gleich:
4 return 0;
```

Algorithmus 3 : Sortierung von Paradigmen

0.000 bis 0.125	→	10000000	}	1xxx
0.125 bis 0.250	→	01000000		
0.250 bis 0.375	→	00100000	}	x1xx
0.375 bis 0.500	→	00010000		
0.500 bis 0.625	→	00001000	}	xx1x
0.625 bis 0.750	→	00000100		
0.750 bis 0.875	→	00000010	}	xxx1
0.875 bis 1.000	→	00000001		

Abbildung 4.2.: Symbolische Abstraktion der Merkmale bei acht Positionen auf der ersten und vier Positionen auf der zweiten Ebene

4.2. Training und Klassifikation

Nach der Beschreibung der Verfahren zur Berechnung des Merkmalsvektors folgt nun die Beschreibung des Lernverfahrens, durch das die Datenbasis der Hypothesenfunktion h induziert wird. Das Ziel besteht dabei in einer hierarchischen Abstraktion der Merkmalsvektoren, und damit der Trainingsbeispiele (vgl. Kap. 3, S. 15).

4.2.1. Abstraktion in den Baumknoten

Das Grundprinzip der Abstraktion im umgesetzten Verfahren besteht, der Beschreibung in Thornton *et al.* (2006) folgend, in der Gruppierung ähnlicher Werte auf den verschiedenen Ebenen einer Hierarchie, konkret in einem Baum. Die Knoten des Baums enthalten auf jeder Ebene Muster einer bestimmten Länge, deren Positionen abhängig von den hereinkommenden Merkmalen (für Blattknoten) bzw. Aktivierungen der Kinder (für innere Knoten) aktiviert werden. Die Länge der Muster halbiert sich dabei von unten (den Blattknoten) nach oben (zur Wurzel) auf jeder Ebene.

In einem Minimalbeispiel mit acht Sensoren und Aktivierungsmustern mit acht Positionen in den Blattknoten würden etwa Werte zwischen 0 und 0.125 zu einer Aktivierung der ersten Position führen, Aktivierungen zwischen 0.125 und 0.25 zu einer Aktivierung der zweiten Position etc. (s. Abb. 4.2). Eine Umsetzung dieser Gruppierung der numerischen Eingangswerte für eine variable Anzahl von Positionen findet sich in Algorithmus 4 (S. 29), eine Implementation des Algorithmus in Java findet sich in Anhang A.1 (S. I).

Im Unterschied zu Blättern müssen innere Knoten im Baum nicht von reellen Zahlen abstrahieren, sondern von Mustern der tieferen Ebenen. Analog zur numerischen Abstraktion werden entsprechend der Aktivierung einer tieferen Ebene (etwa mit 8 Positionen) in der

Ebene darüber Muster mit der Hälfte der Positionen (hier also 4) aktiviert (s. Abb. 4.2, S. 28).

Eingabe : Komplette nicht aktiviertes Muster m der gewünschten Länge l und der zu klassifizierende reelle Wert n

Ausgabe : Das Muster m mit genau einer aktivierten, der mit n korrespondierenden, Position

```

1 foreach Position  $i$  in  $m$  do
2    $v \leftarrow \frac{1}{l} * i$ ;
3   if  $n \leq v$  oder aktuelle Position ist die letzte then
4     // Zu klassifizierender Wert stimmt bei der gewählten Länge mit
5     // der aktuellen Position überein, also ist die aktuelle die zu
     // aktivierende Position:
4      $m[i] \leftarrow 1$ ;
5     return  $m$ 

```

Algorithmus 4 : Abstraktionsalgorithmus für Blattknoten

4.2.2. Generierung der Sprachelemente

Jeder Blattknoten speichert die Häufigkeit von Kookkurrenzen von Mustern seiner eigenen Ebene mit Mustern der übergeordneten Ebene (s. Abs. 4.2.3, S. 31). Da diese Häufigkeiten in Tabellen gespeichert werden, in denen beim Auftreten eines Musters an der entsprechenden Position hochgezählt wird, müssen vor dem Beginn des Trainings alle möglichen Sprachelemente, d.h. die möglichen Musteraktivierungen jeder Ebene, bekannt sein.

Gesucht werden hier alle Permutationen von Zeichenketten einer bestimmten Länge mit einer Anzahl von Aktivierungen von 1 bis n , wobei n die Anzahl der Kinder des Knotens ist, da jedes Kind eine Position im Elternknoten aktivieren kann (s. Abs. 4.2.3, S. 31 und Abb. 4.4, S. 32). Zur Darstellung der Permutationen können aktivierte Positionen durch 1, nicht aktivierte Positionen durch 0 dargestellt werden.

Eine Möglichkeit zur Erzeugung der Permutationen ist ein rekursiver, auf Backtracking basierender Algorithmus (s. Algorithmus 5, S. 30), der einen Baum (s. Abb. 4.3, S. 30) traversiert, in dem jeweils der Pfad von der Wurzel zu einem Blatt eine der gesuchten Permutationen bildet.

Da dieser Baum auf jeder Ebene der Länge n in jedem Knoten höchstens 2 mal verzweigt (1 oder 0), durchläuft eine solche Berechnung der Permutationen im schlechtesten Fall $1 + 2^1 + 2^2 + \dots + 2^n$ Knoten und hat so mit $O(2^n)$ eine exponentielle Laufzeit. Eine Umsetzung des Algorithmus in Java findet sich in Anhang A.1, S. I.

Eingabe : Die Länge des zu generierenden Musters l , der Zähler der bereits aktivierten Stellen c , die Anzahl der zu aktivierenden Stellen n , das aktuelle Muster m und die Liste aller fertigen Muster M

Ausgabe : Füllt M mit allen Permutationen der Länge l mit 1 bis n aktivierten Positionen

```

1 if aktuelles Muster hat die gewünschte Länge then
2   | Muster der Liste hinzufügen;
3 else
4   | if Die gewünschte Anzahl aktivierter Elemente ist aktiviert then
5     | Fülle Muster mit 0 auf;
6     | Muster der Liste hinzufügen;
7   | else
8     | // Aktuelles Element aktivieren:
9     | Rekursiver Aufruf mit  $m + 1$  und  $c + 1$ ;
    | // Aktuelles Element nicht aktivieren:
    | Rekursiver Aufruf mit  $m + 0$  und  $c$ ;

```

Algorithmus 5 : Rekursive Generierung der Sprachelemente

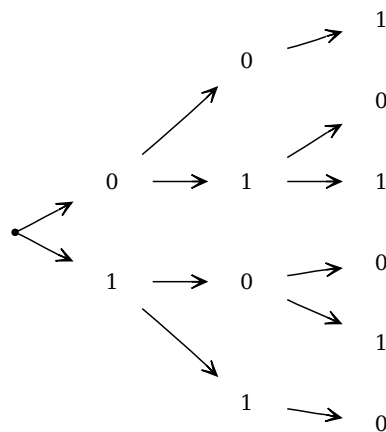


Abbildung 4.3.: Baum mit allen Permutationen der Länge 3 mit 1 bis 2 aktivierten Elementen, generiert durch Algorithmus 5; die gesuchten Permutationen sind die Beschriftungen von der Wurzel zu jedem Blatt, auf jeder Rekursionsstufe verzweigt der Algorithmus maximal 2 mal, zu 1 oder 0 (Zeilen 8 und 9 in Algorithmus 5)

Eine Alternative zum rekursiven Algorithmus besteht in der Berechnung der Permutationen über eine Bitmaske; da hier nur die Aktivierungszustände 1 oder 0 gegeben sind, lassen sich alle Permutationen als Teilmengen einer Gesamtmenge (der komplett aktivierten Bitmaske) formulieren. Bei einer Darstellung der Gesamtmenge als Bitmaske der gesuchten Länge, bei der alle Positionen aktiviert sind, lassen sich durch einfaches Herunterzählen der Binärzahl alle Permutationen der Länge l mit 0 bis l aktivierten Positionen in linearer Laufzeit erzeugen.

Da jede Permutation, ebenfalls in linearer Laufzeit, auf die Anzahl der aktivierten Elemente überprüft werden muss (s. Algorithmus 6 sowie die Umsetzung in Anhang A.1), hat dieser Algorithmus mit $O(n^2)$ eine quadratische Laufzeit, ist damit aber effizienter als der rekursive Algorithmus mit exponentieller Laufzeit.

Eingabe : Länge des zu generierenden Musters l , Anzahl der maximal zu aktivierenden Stellen n

Ausgabe : Alle Permutationen der Länge l mit 1 bis n aktivierten Positionen

```

1  $a \leftarrow$  vollständig aktivierte Bitmaske der Länge  $l$ ;
  // Ausgehend von der aktivierten Bitmaske der gesuchten Länge,
  // erhalten wir durch Substraktion von 1 jede Teilmenge der
  // Ausgangsmenge, dargestellt als Binärzahl:
2 foreach ganze Zahl von  $a$  bis 0 do
  | // Wenn die Permutation unseren Anforderungen hinsichtlich der
  | // aktivierten Positionen entspricht, ist sie Teil der
  | // Ergebnismenge:
3   if aktuelle Binärzahl hat 1 bis  $n$  aktivierte Positionen then
4   | Füge Binärzahl als Permutation hinzu;
```

Algorithmus 6 : Generierung der Sprachelemente über eine Bitmaske

4.2.3. Lernalgorithmus

Beim Training werden Kontexte in numerischer Repräsentation zusammen mit der korrekten Klasse (z.B. *Bank* als ‘Möbel’) verarbeitet (vgl. Abs. 3.2.2, S. 17). Entsprechend der beschriebenen symbolischen Abstraktion würde für ein minimales Netz mit einem Eingabevektor aus 4 Elementen die Aktivierung nach der Verarbeitung aussehen wie in Abbildung 4.4 (S. 32).

Durch die Betrachtung mehrerer solcher Kontexte lernt der Baum in jedem seiner Blattknoten (L1) die Wahrscheinlichkeit des Auftretens von Mustern auf Ebene der Blattknoten mit Mustern der übergeordneten Ebene (L2), und speichert diese in Form der bedingten Wahrscheinlichkeit für das Auftreten eines Musters auf L2 unter der Bedingung eines

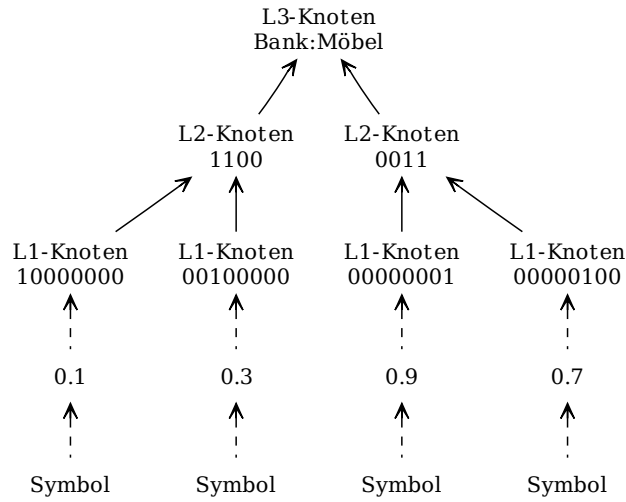


Abbildung 4.4.: Minimales Aktivierungsbeispiel mit einem Eingabevektor aus 4 Elementen

	10000000	01000000	00100000	00010000	00001000	00000100	00000010	00000001
0011	0.18	0.16	0.08	0.12	0.06	0.12	0.16	0.11
0101	0.16	0.14	0.08	0.16	0.05	0.11	0.16	0.15
0110	0.17	0.16	0.09	0.12	0.06	0.13	0.17	0.11
1001	0.13	0.07	0.09	0.13	0.09	0.17	0.17	0.15
1010	0.22	0.09	0.13	0.13	0.07	0.13	0.13	0.09
1100	0.20	0.16	0.08	0.11	0.04	0.15	0.17	0.09
0001	0.16	0.15	0.07	0.13	0.07	0.13	0.15	0.13
0010	0.17	0.13	0.08	0.14	0.08	0.14	0.12	0.13
0100	0.16	0.17	0.09	0.12	0.06	0.13	0.16	0.11
1000	0.16	0.15	0.09	0.14	0.05	0.12	0.16	0.13

Tabelle 4.1.: Beispiel für bedingte Wahrscheinlichkeiten von Kookkurrenzen von Mustern in einem Blattknoten (L1) und Mustern eines übergeordneten Knotens (L2) in jedem Blattknoten, d.h. $P(L2|L1)$, vgl. Abb. 4.4

	0011	0101	0110	1001	1010	1100	0001	0010	0100	1000
S1	0.04	0.08	0.06	0.08	0.16	0.12	0.00	0.12	0.31	0.02
S2	0.09	0.22	0.04	0.04	0.04	0.22	0.09	0.04	0.13	0.09
S3	0.00	0.28	0.06	0.17	0.06	0.06	0.00	0.06	0.17	0.17
S4	0.00	0.00	0.00	0.00	0.20	0.00	0.00	0.40	0.20	0.20
S5	0.00	0.25	0.25	0.12	0.00	0.25	0.00	0.00	0.12	0.00
S6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00

Tabelle 4.2.: Beispiel für bedingte Wahrscheinlichkeiten von Kookkurrenzen von Mustern in einem inneren Knoten (L2) und Mustern des übergeordneten Knotens (L3, hier der Wurzelknoten), d.h. $P(L3|L2)$, hier bei 6 Lesarten, vgl. Abb. 4.4 (S. 32)

eingetretenen Musters auf L1, d.h. $P(L2|L1)$. Diese bedingte Wahrscheinlichkeit wird in jedem Knoten in einer Tabelle (s. Tab. 4.1, S. 32), der *conditional probability table* (CPT), festgehalten.

Analog lernt der Baum in seinen inneren Knoten (L2) die Beziehung von L2-Mustern zu L3-Mustern, hier den gesuchten Lesarten, d.h. die bedingten Wahrscheinlichkeiten $P(L3|L2)$, die ebenso in einer CPT (s. Tab. 4.2) festgehalten werden.

Im Baum hat so nach dem Training jeder Knoten auf L1 eine CPT $P(L2|L1)$ und jeder Knoten auf L2 eine CPT $P(L3|L2)$. Zum Klassifizieren kann nun für eine Eingabe, abhängig von den Werten in den Tabellen, das wahrscheinlichste L3-Element, d.h. die gesuchte Lesart, ermittelt werden (durch BP, s. Abs. 4.2.4).

Das Lexikon besteht so also insgesamt aus einem Baum für jedes ambige Wort, in dessen Knoten Tabellen enthalten sind, die die bedingten Wahrscheinlichkeiten von Musterkookkurrenzen enthalten (s. Abb. 4.5, S. 34). Zur Disambiguierung eines Vorkommens werden die Blattknoten an die (gewissermaßen als Sensoren verwendeten) Merkmale des Kontextes eines Vorkommens angelegt, um die passende Lesart zu ermitteln (s. Abb. 4.6, S. 34).

4.2.4. Klassifikationsalgorithmus

Nach dem Training der Netze, d.h. der Induktion der Datenbasis für die Hypothesenfunktion h (vgl. Abs. 3.2.2, S. 17), können die Netze zum Klassifizieren eingesetzt werden. Dazu wird, Thornton *et al.* (2006) folgend, *Belief Propagation* (BP) in Bäumen⁵ eingesetzt.

⁵ BP wurde in exakter Form zuerst für Bäume (Pearl 1982), dann für gerichtete azyklische Graphen (Kim & Pearl 1983) sowie später als Approximationsalgorithmus für allgemeine Graphen (Pearl 1988) beschrieben.

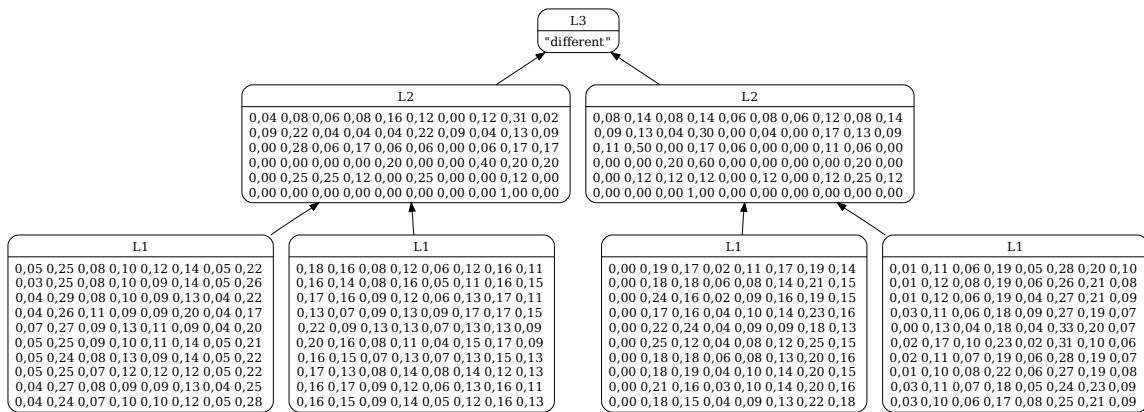


Abbildung 4.5.: Tabellen in den Baumknoten eines minimalen Beispielnetzes nach dem Training (hier für das Lemma *different*)

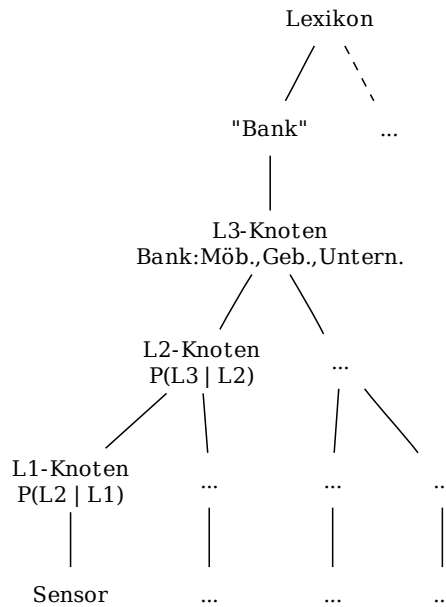


Abbildung 4.6.: Gesamtstruktur des Lexikons: Für jedes ambige Wort existiert ein Baum, dessen Blattknoten an die Merkmale (die quasi als Sensoren dienen) eines Kontextes angelegt werden, um die passende Lesart zu ermitteln

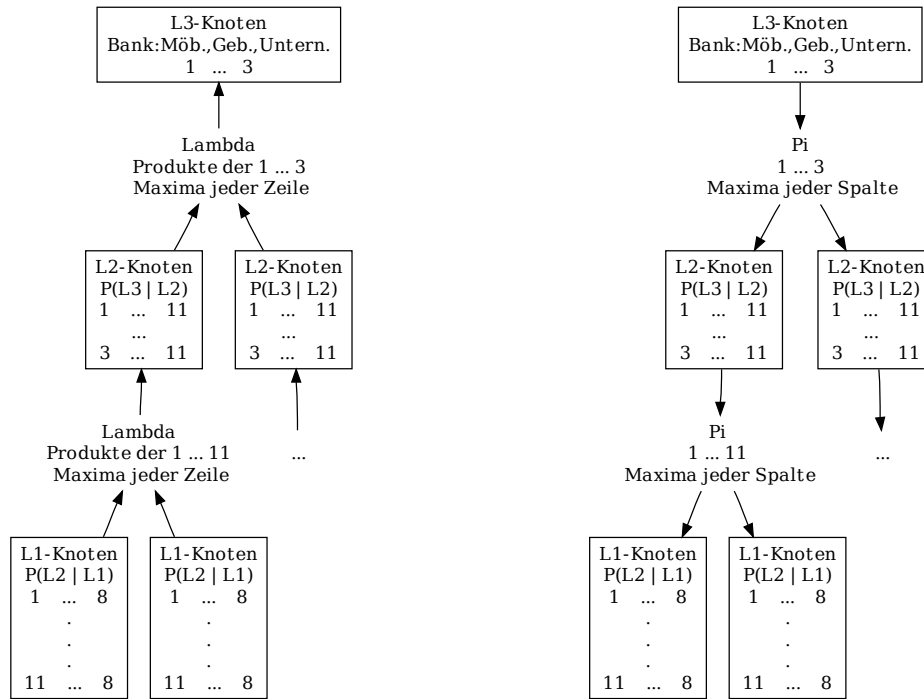


Abbildung 4.7.: Vorwärts- und Rückwärtskommunikation im Baum

BP ist ein wahrscheinlichkeitstheoretisches Verfahren, d.h. das Ergebnis ist prinzipiell die wahrscheinlichste Klasse y unter der Bedingung des Kontextes x , bestehend aus m Merkmalen, d.h. $\max P(y|x_i, \dots, x_m)$ (vgl. Abs. 3.2.2, S. 17).

Die Ermittlung von y erfolgt dabei bei BP durch einen Nachrichtenaustausch zwischen den Knoten. In den Bäumen, in denen hierarchisch von den Merkmalen abstrahiert wird, besteht über die bedingten Wahrscheinlichkeiten in den Tabellen auf den verschiedenen Ebenen ein indirekter, abstrahierter Bezug der möglichen Klassen zu den Merkmalen in x .

Ein Knoten schickt in der Umsetzung von Thornton *et al.* (2006) seine aktuelle Vorstellung (*belief*), wie die gemessenen Daten zu interpretieren sind, im Baum sowohl abwärts (in Form des Vektors π) wie aufwärts (in Form des Vektors λ). Die Elemente der gesendeten Vektoren werden dabei mit den CPT in den Knoten multipliziert⁶ und manipulieren so die Erwartung der Knoten. So kommuniziert jeder Knoten mit seinem Elternknoten und seinen Kindern (s. Abb. 4.7). Diese Kommunikation entspricht der in der Hirnforschung beobachteten Kommunikation von funktional sinnesnahen Regionen mit funktional ent-

ferneren Regionen in beiden Richtungen (vgl. Abs. 2.2, S. 9).

Die Eingabe für Blattknoten bilden λ -Vektoren, die an der Position, die mit dem gemessenen Wert übereinstimmt, eine Aktivierung von 1, an den anderen eine Aktivierung von 0,004 haben. Nicht aktivierte Elemente erhalten nicht 0, um beim Multiplizieren die Werte der CPT nicht zu neutralisieren; der gewählte Wert hat sich dabei in Experimenten als Optimum erwiesen. In einem weiteren Ausbau des Verfahrens wäre so denkbar, entsprechend Konzepten des MPF, bei der Klassifikation eines Vorkommens mehrere Vektoren hintereinander anzulegen, etwa für jedes Wort im Kontext einen, und so einen initialen λ -Vektor zu erhalten, der aus sequentiellm Input entsteht (vgl. Abs. 7.2.3, S. 63).

Zur Berechnung des eingehenden λ -Vektors (λ_{in}) für innere Knoten wird erst für alle Kinder je ein ausgehender λ -Vektor (λ_{out}) erstellt, der die Maxima jeder Zeile der CPT des Kindes enthält. Durch Multiplikation der korrespondierenden Positionen aller dieser Vektoren wird dann λ_{in} für den inneren Knoten berechnet. λ_{in} hat so die gleiche Anzahl von Elementen wie der empfangende Knoten Spalten in seiner CPT.

Zur Beeinflussung der Erwartung des empfangenden Knotens wird nun jedes Element in λ_{in} mit jedem Element der korrespondierenden Spalte multipliziert. In gleicher Weise setzt sich dieser Vorgang der Vorwärtsaktivierung im Baum bis zur Wurzel fort. Jeder Knoten schickt also einen Vektor λ_{out} mit den Maxima der Zeilen in seiner CPT aufwärts, welche dann, kombiniert mit den Vektoren der Geschwisterknoten den Vektor λ_{in} für den Elternknoten bilden. Das heisst λ_{in} des Knotens x ist das Produkt der λ_{out} -Vektoren aller $1 \dots n$ Kinder z : $\lambda_{in}(x) = \prod_{k=1}^n \lambda_{out}(z_k)$.

Wenn nun der λ_{in} für den Wurzelknoten berechnet wird, entspricht jedes Element in λ_{in} einer Klasse, hier einer Lesart des ambigen Wortes. Auf diese Weise bildet das Netz eine Annahme über wahrscheinlichste Klasse für die verarbeiteten Merkmale.

Die Rückwärtsaktivierung läuft analog ab, indem der π -Vektor eines jeden Knotens durch die Berechnung der Maxima jeder Spalte gebildet wird und mit jedem Element jeder Zeile der Kinder multipliziert wird. Anschließend bildet dieser auf gleiche Weise einen π -Vektor, der weiter abwärts im Netz geschickt wird, usw. bis zu den Blattknoten (s. Abb. 4.7, S. 35), wodurch diese insofern beeinflusst werden, als dass der nächste ankommende Merkmalsvektor auf Basis der so veränderten CPT weiterverarbeitet wird. Um diesen Mechanismus zu nutzen, wäre für jede zu disambiguierende Fundstelle die Eingabe mehrerer Vektoren nötig, weshalb im implementierten Verfahren die Rückwärtskommunikation nicht wirksam ist und erst durch ein Verfahren zur Kontextrepräsentation durch mehrere Vektoren (vgl. Abs. 7.2.3, S. 63) ausgenutzt würde.

6 Nach der Multiplikation werden die Werte normalisiert, um Präzisionsprobleme mit Fließkommazahlen infolge immer kleiner werdenden Werte zu umgehen. Dabei entspricht die höchste vorhandene Wahrscheinlichkeit 100%, die anderen Werte werden entsprechend angepasst.

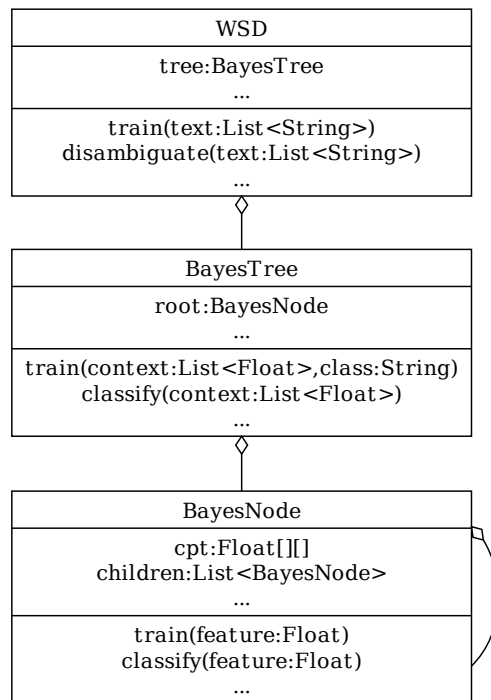


Abbildung 4.8.: Klassendiagramm der umgesetzten Datenstruktur in UML-Notation

4.3. Datenstruktur

Der beschriebene Baum ist als rekursive, objektorientierte Datenstruktur implementiert. Die Klasse `WSD` ist dabei mit einer Instanz der Klasse `BayesTree` assoziiert, welche den Wurzelknoten der rekursiven Struktur enthält, eine Instanz der Klasse `BayesNode`. Ein `BayesNode` enthält Referenzen auf seine Kinder, ebenfalls Instanzen der Klasse `BayesNode`. Konzeptuell ist die Implementation so angelegt, dass die `WSD` Methoden zum Training und zur Disambiguierung eines ganzen Textes enthält, die Klasse `BayesTree` entsprechende Methoden für den Kontext eines Vorkommens und `BayesNode` schließlich Methoden für einzelne Zahlen und Muster (vgl. Abb. 4.8).

Zur Umsetzung verschiedener Netzstrukturen können nicht nur Bäume mit unterschiedlichen Sensorenzahlen (d.h. Blattknoten), sondern auch Bäume mit einer unterschiedlichen Anzahl von Ebenen sowie einer unterschiedlichen Anzahl von Kindern auf den unterschiedlichen Ebenen erstellt werden. Die in den Knoten gespeicherten Tabellen unterscheiden sich abhängig von diesen Parametern, da diese einen direkten Einfluss auf die Anzahl der

möglichen Sprachelemente auf einer Ebene haben (vgl. Abs. 4.2.2, S. 29). Zur Evaluierung in Kapitel 5 (S. 40) werden unterschiedliche Netzstrukturen verglichen.

4.4. Parallelisierung

Angesichts der rasanten Entwicklung der Anzahl von Prozessorkernen⁷ in Computern ist die Frage der Parallelisierbarkeit von Algorithmen und Verfahren heute von zentraler Bedeutung, da diese eine Voraussetzung zur Ausnutzung der so verfügbaren Rechenleistung ist. Daher beschreibe ich im Folgenden eine Möglichkeit zur Parallelisierung des implementierten Verfahrens.

```

1 foreach Lemma do
  // Paralleler Subprozess, der nur auf seinen eigenen Daten
  // operiert, dem Baum für ein Lemma:
2   begin
3     foreach Lesart l do
4       foreach Vorkommen v von l do
5         Generiere Merkmale m für den Kontext von v;
6         Trainiere Baum mit m;
7         // Training für eine Lesart ist abgeschlossen, Kontinuität
           // des Inputs endet hier, daher werden die Zähler im Baum
           // zurückgesetzt:
           Baum zurücksetzen;
8     end

```

Algorithmus 7 : Parallelisiertes Lernen

Eine grundlegende Unterscheidung paralleler Algorithmen kann danach erfolgen, ob aus den parallel laufenden Subprozessen auf gemeinsame Daten zugegriffen wird oder nicht. Wird auf gemeinsame Daten zugegriffen, wird der Prozess nondeterministisch, d.h. das gleiche Programm kann bei unterschiedlichen Durchläufen unterschiedliche Ergebnisse liefern (Van Roy & Haridi 2004:20). Daher muss in einem solchen Fall der parallele Zugriff koordiniert werden. Verfahren ohne gemeinsame Daten sind dagegen deutlich einfacher zu realisieren, da Gleichläufigkeit ohne gemeinsamen Datenzugriff und damit ohne die Notwendigkeit, diesen zu koordinieren, ein sehr simples Konzept ist (Van Roy & Haridi 2004:14-5).

⁷ So haben einzelne Standardprozessoren inzwischen bis zu 8 Kerne; der Sun UltraSPARC T2 etwa unterstützt dabei pro Kern hardwaremässig 8 Threads und damit auf Hardwareebene bis zu 64 parallele Prozesse, und bietet so eine parallele Leistung von 89,6 GHz (http://blogs.sun.com/jonathan/entry/sun_enters_the_commodity_silicon) in einem einzelnen Prozessor, auf dem Standardanwendungen (etwa auf Basis von Java, wie die hier beschriebene Implementation) laufen.

Das beschriebene Verfahren lässt sich ohne gemeinsamen Datenzugriff (und damit einfach) parallelisieren, indem alle Kontexte eines jeden Lemmas in einem separaten Thread trainiert bzw. klassifiziert werden (s. Algorithmus 7, S. 38). Da für jedes Lemma ein eigener Baum existiert (vgl. Abb. 4.6, S. 34), operieren die parallelen Prozess nicht auf gemeinsamen Daten. Der Algorithmus kann dennoch stark parallelisiert werden, da zu erwarten ist, dass die zu bearbeitende Gesamtmenge im Normalfall gleichmässig auf verschiedene Lemmata verteilt ist. Das in Kapitel 5 (S. 40) zur Evaluierung des Verfahrens verwendete Korpus etwa hat 57 Lemmata, wodurch hier 57 separate Threads gestartet werden könnten, sowohl im Training wie auch beim Disambiguieren. Im gleichen Maße, in dem bei einem Anstieg der verwendeten Lemmata der Aufwand für Training und Disambiguierung zunehmen würde, würde so zugleich zunehmend parallele Rechenleistung ausgenutzt.

Kapitel 5.

Evaluierung

Wie in allen Bereichen der maschinellen Sprachverarbeitung existieren auch in der WSD verschiedene Ansätze; dabei stellt sich die Frage, welches der beste Ansatz ist, weshalb WSD-Systeme evaluiert werden. Dies erfolgte in der Vergangenheit meist isoliert, was einen Vergleich der Ergebnisse sehr schwierig gemacht hat. Die im Folgenden umgesetzte Evaluierung mit gemeinsamen Korpora bietet eine inzwischen etablierte Grundlage zur Evaluierung von Systemen zur maschinellen Sprachverarbeitung, auch im Bereich der WSD (Palmer *et al.* 2006:75-6).

5.1. Korpora zur Evaluierung

Bevor annotierte Korpora zur gemeinsamen Evaluierung verfügbar waren, wurden häufig Korpora mit automatisch generierter Ambiguität in Form sogenannter Pseudowörter verwendet. Dabei wird jedes Auftreten von zwei Wortformen der gleichen Wortart durch eine Zusammensetzung der Wörter ersetzt (etwa alle Vorkommen von *banana* und *door* durch *banana-door*, vgl. Manning & Schütze 1999:233).

Gegen eine solche Evaluierung spricht, dass es sich bei den Pseudowörtern um kein echtes sprachliches Phänomen handelt (Palmer *et al.* 2006:86) und daher gute Ergebnisse nur bedingt Rückschlüsse auf Phänomene der lexikalischen Semantik erlauben. So entsprechen Pseudowörter etwa nie Polysemie (vgl. Abs. 1.3.3, S. 6), sondern stets eher Homonymie (Palmer *et al.* 2006:86).

Mit den Senseval-Workshops¹ existiert eine Reihe von Veranstaltungen, die (nach dem Vorbild der *Message Understanding Conferences* in der Informationsextraktion, s. etwa Grishman & Sundheim 1996) eine gemeinsame Evaluierung von Systemen zur WSD auf der Basis gemeinsamer Korpora zum Gegenstand haben (vgl. Palmer *et al.* 2006:86). Für diese Workshops wurden gemeinsame Trainings- und Teskorpora bereitgestellt, für die auch das *inter-annotator agreement* (ITA, s. Abs. 5.2, S. 41) ermittelt wurde.

¹ Senseval, Evaluation Exercises for the Semantic Analysis of Text (s. <http://www.senseval.org/senseval3>)

Wortarten	Lemmata	Lesarten (fein)	Lesarten (grob)
Nomen	20	5,8	4,4
Verben	32	6,3	4,6
Adjektive	5	10,2	9,8
Gesamt	57	6,5	5,0

Tabelle 5.1.: Überblick über das Bedeutungsinventar im Korpus: Wortarten und durchschnittliche Anzahl von Lesarten pro Wort bei feiner und grober Körnung (aus: Mihalcea *et al.* 2004)

Im Folgenden wird das beschriebene Verfahren anhand der Korpora des *english lexical sample task* im Rahmen von Senseval-3 (vgl. Mihalcea *et al.* 2004) evaluiert, der aktuellsten Veranstaltung, für die Daten öffentlich verfügbar² waren. Das Senseval-3 Korpus enthält Ausschnitte des BNC – für das Training 8529 Vorkommen 57 ambiger Lemmata, zur Disambiguierung 5693 Vorkommen. Einen Überblick über das Inventar an Bedeutungen im Korpus gibt Tabelle 5.1. Die Bedeutungen im annotierten Korpus stammen für Nomen und Adjektive aus WordNet 1.7.1, die von Verben aus Wordsmyth (Mihalcea *et al.* 2004).

5.2. Evaluationskriterien

Im Folgenden wird die Genauigkeit der Ergebnisse des beschriebenen Verfahrens zur WSD evaluiert. Es gibt auch andere Kriterien, nach denen Verfahren bewertet werden können (Kiss 2002:163), etwa auf Grundlage der kognitiven Plausibilität des Verfahrens (vgl. Kap. 2, S. 9) oder auf technischer Ebene nach Kriterien wie Laufzeit- und Speicherplatzkomplexität (vgl. Abs. 4.2.2, S. 29) sowie Modularität der Umsetzung (vgl. Kap. 6, S. 52).

5.2.1. Precision und Recall

Zur Evaluierung der Genauigkeit von WSD-Verfahren wird bewertet, wie viele Wörter richtig disambiguiert wurden. Ein richtig disambiguiertes Vorkommen erhält dabei die Wertung 1 bzw. 100%, ein falsch disambiguiertes Wort dagegen 0 bzw. 0%. Zur Ermittlung der *Precision* des Verfahrens wird der Durchschnitt m der n ermittelten Ergebnisse x

² Senseval-4 fand während der Erstellung dieser Arbeit statt, weshalb die Daten zum Zeitpunkt der Entstehung nur für Teilnehmer zugänglich waren.

berechnet: $m = \frac{1}{n} \sum_{i=1}^n x_i$. Die Berechnung des *Recall* entspricht bei der WSD grundsätzlich der Berechnung der *Precision*, wobei jedoch Vorkommen, die ausgelassen wurden, als 0 zählen. Wurden alle Vorkommen versucht, entspricht also der *Recall* stets der *Precision* (Palmer *et al.* 2006:79).

5.2.2. Vergleichswerte

Bei der Ermittlung der *Precision* stellt sich die Frage, wie die gemessenen Ergebnisse zu bewerten sind, d.h. insbesondere, womit sie verglichen werden sollten. Üblicherweise wird mit einer Obergrenze (*upper bound*, s. etwa Manning & Schütze 1999:232), etwa der menschlichen Leistung (in Form des ITA, s. oben) und einer Untergrenze (*lower bound*) verglichen; für Letztere ist eine verbreitete Methode, immer die häufigste Lesart im Trainingskorpus zur Disambiguierung zu wählen (Palmer *et al.* 2006:79). Eine Alternative bestünde in einer reinen Zufallsauswahl oder in anderen simplen Verfahren, wie dem Lesk-Algorithmus (vgl. Palmer *et al.* 2006:79-88). Schließlich kann die Leistung auch mit der anderer WSD-Systeme verglichen werden, wenn für diese Ergebnisse und die Daten verfügbar sind, wie im Fall der Senseval-Korpora.

In der folgenden Evaluierung werden die Ergebnisse mit ITA und der Leistung des besten Systems bei Senseval-3 (*upper bound*), sowie mit der beschriebenen Wahl der häufigsten Lesart (*lower bound*) verglichen. Die Evaluierung erfolgt dabei mit der Senseval-Bewertungssoftware³, welche die Ergebnisse des WSD-Systems in Form einer Textdatei einliest.

5.2.3. Bedeutungsgranularität

Wie die Bedeutungen nicht-ambiger Wörter können auch die verschiedenen Bedeutungen ambiger Wörter einander konzeptuell über- (Hyperonymie) oder untergeordnet (Hyponymie) sein, sowie eine gemeinsame, übergeordnete Bedeutung haben (Kohyponymie). So ist etwa in WordNet⁴ *bet* mit der Lesart ‘the money risked in a gamble’ den weiteren Lesarten ‘the initial contribution that each player makes to the pot’ und ‘the combined stakes of the betters’ übergeordnet.

Bei einer solchen hierarchischen Strukturierung möglicher Bedeutungen können bei der Evaluierung drei Stufen von Bedeutungsgranularität zugrunde gelegt werden: feine (*fine*), gemischte (*mixed*) und grobe (*coarse*) Körnung (Palmer *et al.* 2006:79).

³ Senseval-3 Scoring: <http://www.senseval.org/senseval3/scoring>. Das Programm liegt im C-Quelltext vor und muss zum Einsatz kompiliert werden.

⁴ WordNet kann unter <http://wordnet.princeton.edu/perl/webwn> online abgefragt werden.

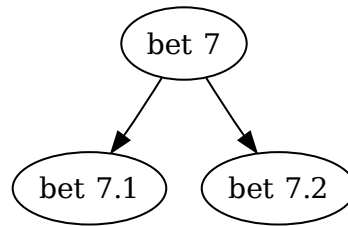


Abbildung 5.1.: Hierarchisch strukturierte Bedeutungen im Senseval-Korpus (aus: Palmer *et al.* 2006:79)

Bei feiner Körnung sind nur exakte Treffer korrekt, so wäre etwa für die Bedeutungen in Abbildung 5.1, wenn Lesart 7 korrekt wäre, die Wahl von 7.1 oder 7.2 nicht korrekt. Bei grober Körnung dagegen wären sowohl 7, 7.1 als auch 7.2 korrekt, d.h. Hyponyme, Hyperonyme und Kohyponyme der korrekten Lesart werden auch als korrekt gewertet. Bei gemischter Körnung dagegen zählen Kohyponyme der korrekten Lesart nicht als korrekte Ergebnisse, und für Hyperonyme wird die Wertung abhängig von der Anzahl der Kinder verringert, z.B. würde, wenn Lesart 7.1 korrekt wäre, die Wahl von 7.1 mit 1.0, die von 7 mit 0.5 und die von 7.2 mit 0 bewertet. Bei den folgenden Ergebnissen sind stets die Werte für die drei Granularitäten angegeben, welche mit der Senseval-Bewertungssoftware (s.o.) ermittelt wurden.

5.2.4. Anwendungsintegration

In Anbetracht seiner Natur als Mittel sollte WSD nicht nur isoliert, sondern auch im Kontext einer konkreten Anwendung evaluiert werden – unter der Fragestellung, ob und wie sehr die WSD die Ergebnisse der Anwendung verbessert (Agirre & Edmonds 2006a:17).

Bei einer Anwendung zur Eigennamenerkennung (einer Form von Informationsextraktion) etwa, bei der ambige Wörter mit *allen* möglichen Lesarten annotiert werden, beträgt ohne WSD der Recall 100%, während die Precision durch die ambigen, mehrfach ausgezeichneten Eigennamen verringert ist. Die Frage, die sich zum Einsatz der WSD in einem solchen Fall stellt, ist ob die WSD präzise genug ist, um die Ergebnisse zu verbessern, d.h. bleibt der Recall hoch und steigt die Precision oder bleibt die Precision gering und der Recall sinkt zusätzlich durch falsch disambiguierte Wörter? Hier ist je nach Disambi-

Variable	Werte
Kontextfenster	$\pm 2, \pm 4, \pm 8, \pm 16$ Symbole
Symb. Abstr. (Symbol \rightarrow Zahl)	Wörter, Wortlängen, Trigramme Heptagramme, Paradigmen
Num. Abstr. (Zahl \rightarrow Muster)	8, 16, 24, 32, 64 Positionen in den Mustern der Blattknoten
Netzstruktur	Mit und ohne Zwischenebene

Tabelle 5.2.: Variablen und Werte für die Experimente

guierungsleistung der WSD zu erwarten, dass die Ergebnisse im schlechtesten Fall durch die WSD verschlechtert statt verbessert werden.

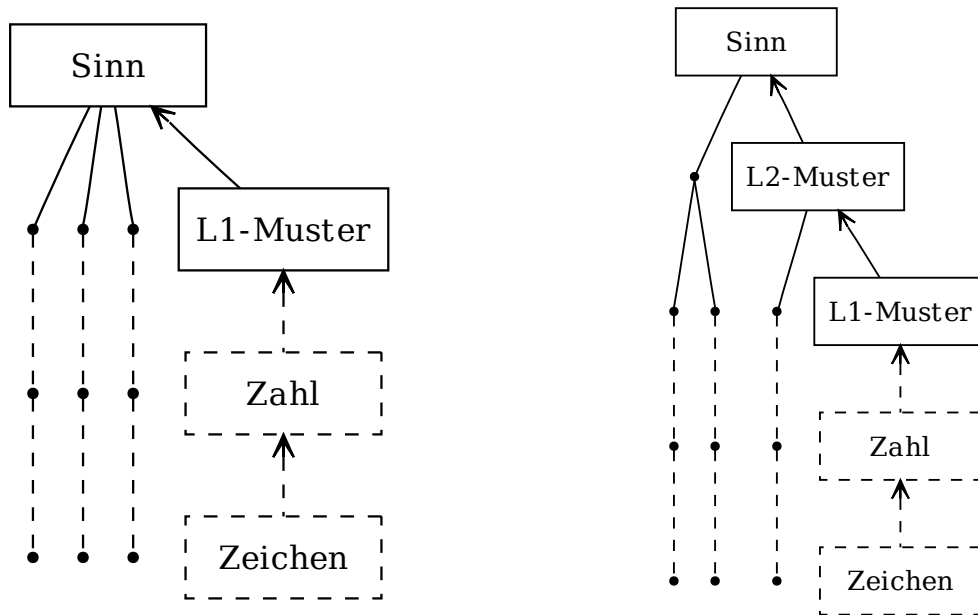
Zur Ermöglichung einer solchen integrierten Evaluierung wird in Kapitel 6 (S. 52) die Umsetzung des Verfahrens im *Text Engineering Software Laboratory* (Tesla), einer *Software Architecture for Language Engineering* (SALE) beschrieben.

5.3. Experimente

5.3.1. Variablen

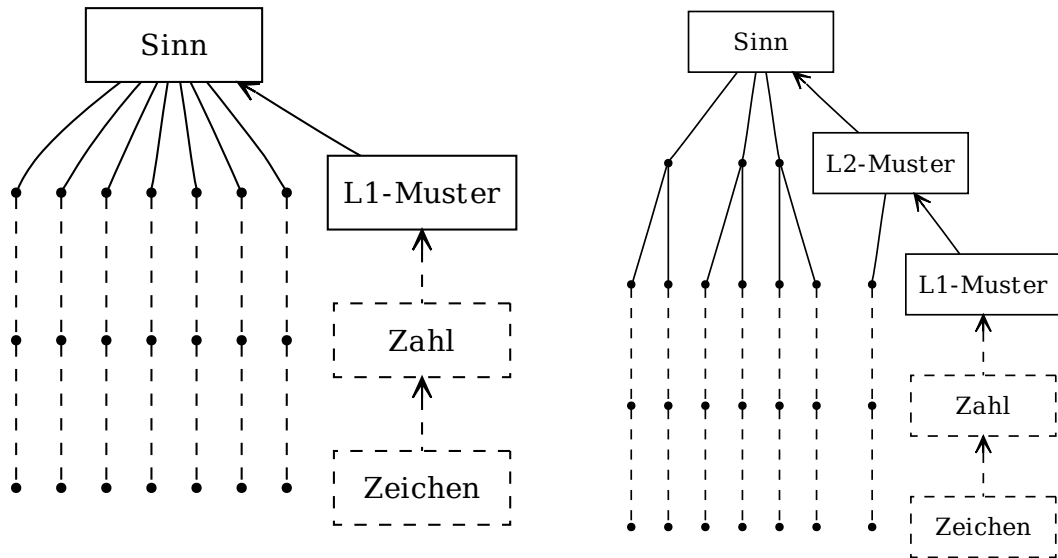
Entsprechend der in Kapitel 4 (S. 22) beschriebenen Umsetzung des Verfahrens ergeben sich verschiedene Variablen, die für die durchzuführenden Experimente unterschiedliche Werte annehmen können (s. Tab. 5.2). Dabei habe ich aus Gründen der Laufzeit des Verfahrens auf die Evaluierung der mehrschichtigen Netze mit den höheren Musterfaktoren verzichtet ($4, 4 * 2$; $4, 4 * 4$; $4, 8 * 1$; $4, 8 * 2$)⁵, da eine Vergrößerung der L1-Muster bei zugleich steigender Anzahl von Kindern einen exponentiellen Anstieg (vgl. Diskussion der Laufzeit von Algorithmus 5, S. 30 in Kap. 4.2.2, S. 29) der Größe von Tabellen in den Knoten des Baums zur Folge hat und so zu hohem Speicherbedarf und langer Laufzeit führt (bis zu mehreren Tagen pro Einstellung). Hier wäre zu untersuchen, wie das Verfahren effizienter umsetzbar wäre, wobei hier (s. Abs. 5.3.2) die besten Ergebnisse mit Einstellungen erreicht werden, die nur wenige Sekunden benötigen.

5 Zur Beschreibung der Netzstrukturen gebe ich für jede Ebene unter der Wurzel mit Kommata getrennt an, wie viele Kinder jeder Knoten auf der entsprechenden Ebene hat, z.B. “4,2” für 4 Kinder der Wurzel mit je 2 Kindern oder “8” für 8 Kinder der Wurzel. Anschließend folgt hier der Musterfaktor, z.B. “4, 4 * 2” für einen Baum mit 4 Kindern unter der Wurzel, die je wieder 4 Kinder haben, d.h. einen Baum mit 16 Blättern. Diese Zahl wird mit 2 multipliziert, dem Musterfaktor (die Anzahl der Blattknoten wird zur Berechnung der Länge der Muster in L1-Knoten mit diesem Musterfaktor multipliziert), und ergibt so für “4, 4 * 2” 32 ($16 * 2$) Positionen in den Mustern der Blattknoten.



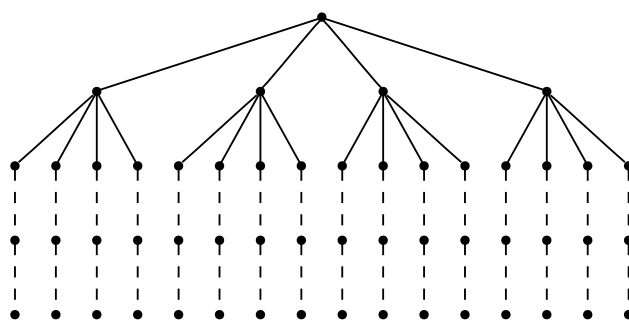
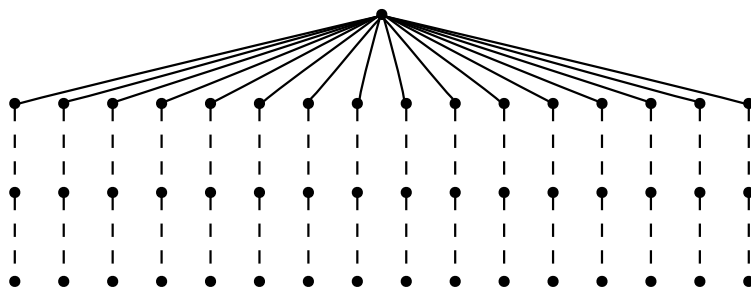
M	N	Wörter	Wortlängen	Trigramme	Heptagramme	Paradigmen
		fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob
2	4	40,8/44,7/47,5	30,3/33,5/36,6	42,9/46,9/49,5	34,7/38,6/41,4	41,7/45,3/47,8
4	"	48,2/52,0/54,3	35,5/39,3/41,9	47,6/51,3/53,7	37,3/41,1/43,8	46,5/50,0/52,3
6	"	51,1/54,2/56,1	43,0/46,7/49,2	49,6/53,4/55,5	37,6/41,2/43,9	48,6/52,1/54,3
8	"	51,5/54,8/56,8	42,7/46,7/49,2	50,2/53,8/55,8	37,6/41,4/44,2	48,9/52,2/54,3
16	"	50,7/54,0/55,9	42,7/46,7/49,2	50,6/53,9/55,9	37,8/41,1/43,6	49,2/52,5/54,6
2	2,2	04,2/05,2/06,3	06,5/08,0/09,9	03,2/05,5/06,1	05,1/06,7/08,5	03,9/06,2/07,4
4	"	04,3/06,5/08,0	05,5/07,2/09,2	05,5/08,0/10,6	05,5/07,5/08,8	04,5/06,7/08,6
6	"	07,3/09,8/12,6	03,7/05,1/06,7	05,7/08,1/10,1	05,5/07,9/09,3	06,2/09,4/12,4
8	"	09,2/13,0/16,6	03,7/05,9/07,9	07,7/10,4/13,2	06,8/09,1/10,3	08,4/11,7/14,7
16	"	14,5/18,2/21,1	04,4/06,6/08,0	11,4/14,3/17,0	06,5/08,7/10,1	11,1/14,2/17,9

Tabelle 5.3.: Ergebnisse mit Kontextfenster 4 (± 2) für Netzstrukturen "4" (Abb. links, in Tab. oben) und "2,2" (Abb. rechts, in Tab. unten)



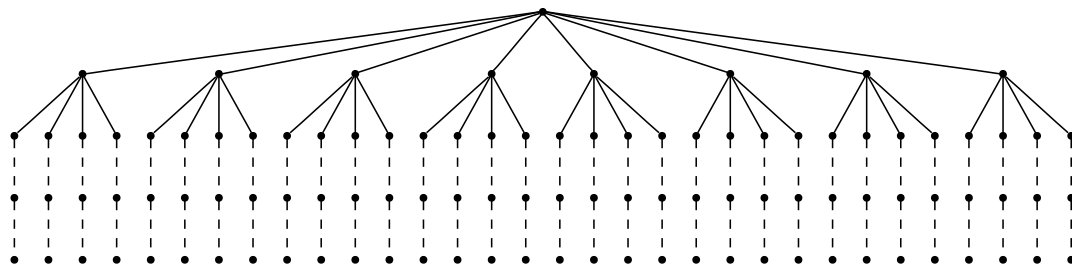
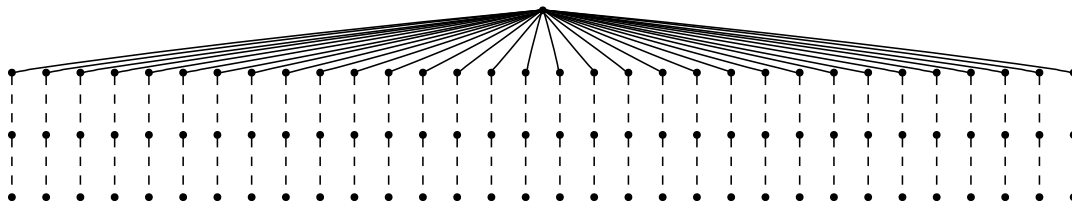
M	N	Wörter	Wortlängen	Trigramme	Heptagramme	Paradigmen
		fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob
1	8	46,1/49,9/52,3	36,7/40,4/43,1	48,2/52,3/54,7	40,1/44,3/47,1	46,0/50,0/52,3
2	"	53,9/57,2/59,0	44,1/48,0/50,1	52,8/56,8/59,2	41,1/46,0/48,1	51,4/54,7/56,6
3	"	55,4/58,9/60,8	48,6/52,2/54,3	53,9/57,4/59,5	42,5/46,9/49,3	53,5/56,7/58,7
4	"	55,2/58,6/60,3	49,5/53,1/55,1	55,2/58,6/60,3	43,3/47,4/50,1	54,3/57,3/59,3
8	"	53,9/57,4/59,2	49,5/53,1/55,1	55,3/58,6/60,5	43,0/47,1/49,7	53,4/56,7/58,5
1	4,2	02,9/04,5/05,7	04,7/06,4/08,4	02,4/04,1/05,5	04,0/05,4/07,2	02,7/04,5/06,0
2	"	03,7/05,8/07,4	04,4/06,5/08,2	04,1/06,6/08,8	04,8/07,0/08,4	03,6/05,5/07,6
3	"	06,1/08,6/11,1	02,9/05,1/06,8	04,7/06,9/08,5	05,0/07,3/09,0	05,3/07,9/10,0
4	"	07,3/10,8/14,3	02,9/04,7/06,5	06,3/09,2/11,9	05,5/07,5/08,6	06,8/09,6/12,5
8	"	11,2/14,7/18,0	04,5/06,0/07,3	08,7/12,0/15,2	05,7/08,0/09,4	09,3/12,5/15,6

Tabelle 5.4.: Ergebnisse mit Kontextfenster 8 (± 4) für Netzstrukturen "8" (Abb. links, in Tab. oben) und "4,2" (Abb. rechts und Tab. unten)



M	N	Wörter		Wortlängen		Trigramme		Heptagramme		Paradigmen	
		fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob		
0,5	16	50,3/54,2/56,1	40,2/43,8/46,1	51,1/54,9/56,9	46,2/50,2/52,6	50,5/54,2/56,4					
1	"	54,2/57,7/59,5	47,7/51,2/53,1	55,2/58,0/60,8	47,9/52,3/54,6	53,8/56,8/58,6					
1,5	"	55,9/59,6/61,2	50,9/54,8/56,7	55,9/59,5/61,3	48,4/52,6/54,8	54,3/57,8/59,6					
2	"	54,1/57,7/59,4	51,2/54,8/56,6	56,0/59,7/61,6	48,3/52,4/54,7	55,4/58,8/60,6					
4	"	41,1/44,8/46,8	51,2/54,8/56,6	52,5/56,1/58,1	47,8/51,9/54,3	51,9/55,3/57,2					
0,5	4,4	02,5/04,1/05,9	06,8/09,4/10,8	02,7/03,7/05,0	03,3/04,2/05,2	02,7/04,7/06,1					
1	"	05,6/08,1/10,5	07,5/09,4/10,5	06,9/09,8/12,7	03,5/05,0/06,4	05,1/08,1/11,0					
1,5	"	15,0/20,4/25,2	03,4/05,3/06,4	12,2/16,6/20,6	04,3/06,0/08,0	11,0/15,2/19,5					
2	"	n/a	n/a	n/a	n/a	n/a					
4	"	n/a	n/a	n/a	n/a	n/a					

Tabelle 5.5.: Ergebnisse mit Kontextfenster 16 (± 8) für Netzstrukturen "16" (Abb. und Tab. oben) und "4,4" (Abb. und Tab. unten)



M	N	Wörter		Wortlängen		Trigramme		Heptagramme		Paradigmen	
		fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob	fein/gem./grob		
0,25	32	08,8/11,7/14,4	31,6/34,5/36,6	09,6/12,4/15,2	31,5/34,0/35,6	10,6/12,9/14,8					
0,5	"	11,0/15,7/19,5	16,0/18,0/19,7	09,5/14,4/18,3	28,7/31,5/33,4	08,0/12,7/16,7					
0,75	"	28,7/33,4/36,5	08,0/11,3/14,4	21,5/26,5/30,2	28,3/31,0/32,8	24,9/29,9/33,2					
1	"	52,6/56,7/58,8	06,8/10,9/14,7	36,1/41,0/43,9	27,8/30,5/32,4	43,9/48,2/50,7					
2	"	39,6/43,5/45,7	07,0/11,4/15,4	53,6/57,6/59,4	22,2/25,0/26,8	52,0/55,7/57,5					
0,25	8,4	02,3/04,4/06,0	11,6/14,2/16,0	02,5/04,5/05,7	02,6/04,2/05,3	02,5/04,5/05,9					
0,5	"	05,8/08,9/11,9	07,0/09,1/10,9	06,9/10,3/13,9	03,3/05,9/08,5	05,2/08,7/11,7					
0,75	"	14,0/19,4/23,9	04,8/06,8/08,1	11,6/16,6/21,0	03,8/06,6/09,3	11,0/16,2/20,7					
1	"	n/a	n/a	n/a	n/a	n/a					
2	"	n/a	n/a	n/a	n/a	n/a					

Tabelle 5.6.: Ergebnisse mit Kontextfenster 32 (± 16) für Netzstrukturen "32" (Abb. und Tab. oben) und "8,4" (Abb. und Tab. unten)

5.3.2. Ergebnisse

Auf den Seiten 45 bis 48 finden sich die unterschiedlichen Netzstrukturen und die dazugehörigen Ergebnisse für Kontextfenster 4 (± 2) (Tab. 5.3, S. 45), 8 (± 4) (Tab. 5.4, S. 46), 16 (± 8) (Tab. 5.5, S. 47) und 32 (± 16) (Tab. 5.6, S. 48), jeweils für verschiedene Merkmale, Musterfaktoren (M) und Netzstrukturen (N), mit Hervorhebung der besten Ergebnisse. Das Verhältnis von M und N ist stets so gewählt, dass sich für alle Kontextfenster und Netzstrukturen in den Blattknoten jeweils Muster gleicher Länge ergeben.

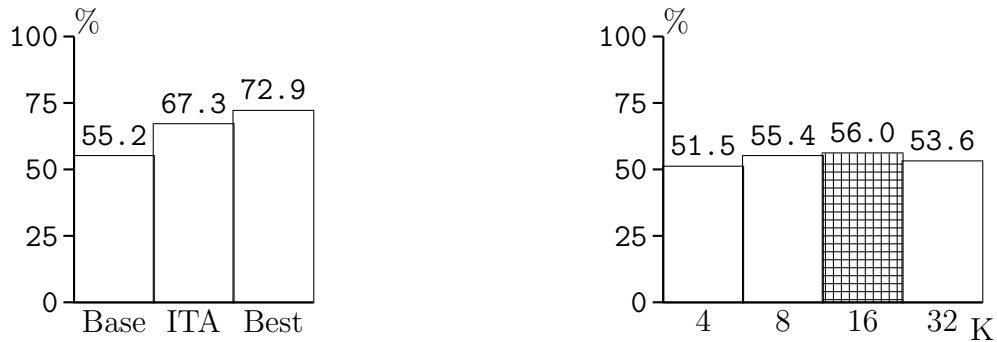


Abbildung 5.2.: Vergleichswerte (links) und beste Ergebnisse (rechts) für unterschiedliche Kontextfenster (K) bei feiner Körnung

Die besten Ergebnisse des beschriebenen Verfahrens für alle Kontextfenster im Vergleich zur Wahl der häufigsten Lesart (Base), der Übereinstimmung menschlicher Annotatoren (ITA) und der Leistung des besten Systems bei Senseval-3 (Best) finden sich in Abbildung 5.2 für feine Körnung (fine-grained) und in Abbildung 5.3 für grobe Körnung (vgl. Agirre & Edmonds 2006a:16; Mihalcea *et al.* 2004). Für die grobe Körnung wird in Mihalcea *et al.* (2004) kein ITA-Wert genannt.

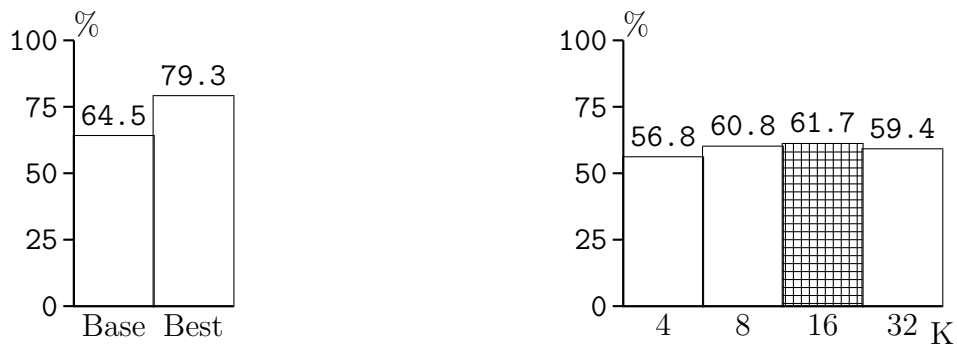


Abbildung 5.3.: Vergleichswerte (links) und beste Ergebnisse (rechts) für unterschiedliche Kontextfenster (K) bei grober Körnung

5.3.3. Diskussion

Bereits für Kontextfenster 4 (± 2) ergeben sich Ergebnisse nahe an der Baseline (bei einer Ebene, mit 4 Kinder und 32 Positionen in Blättern, vgl. Tab. 5.3, S. 45). Das Optimum wird mit einem Wert knapp über der Baseline bei Kontextgröße 16 (± 8), ohne Zwischenebene, mit 32 Positionen in den L1-Mustern und Trigrammen als Merkmale erreicht (vgl. Tab. 5.5, S. 47).

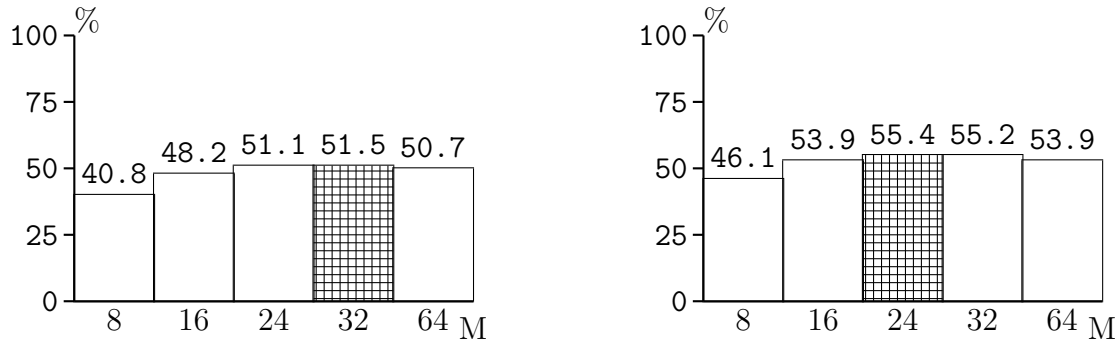


Abbildung 5.4.: Entwicklung der Ergebnisse mit dem besten Aufbau (Wörter, ohne Zwischenschicht) bei Zunahme der Musterlänge (M), für Kontextfenster 4 (± 2 , links) und 8 (± 4 , rechts)

Bei Kontextfenstern von 4 (± 2) und 8 (± 4) werden die besten Ergebnisse mit Wörtern als Merkmale, bei Kontextfenstern von 16 (± 8) und 32 (± 16) mit Trigrammen erreicht. Dies ist nachvollziehbar, da zu erwarten ist, dass bei größeren Kontexten die Zahl der für das Zielwort charakteristischen Symbole abnimmt (vgl. Abs. 4.1, S. 22). Da Trigramme im Schnitt kürzer als Wörter sind, fassen so 16 Trigramme einen kleineren Kontext als 16

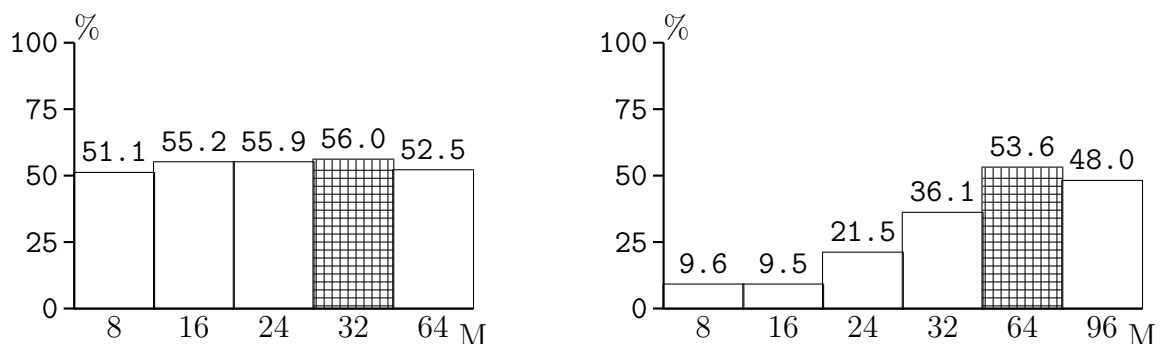


Abbildung 5.5.: Entwicklung der Ergebnisse mit dem besten Verfahren (Trigramme, ohne Zwischenschicht) bei Zunahme der Musterlänge (M), für Kontextfenster 16 (± 8 , links) und 32 (± 16 , rechts)

Wörter und umgekehrt 8 Wörter einen grösseren Kontext als 8 Trigramme. Das für das Verfahren optimale Kontextfenster liegt also im Bereich von 8 Wörtern bis 16 Trigrammen. Als qualitative Tendenz, die sich bei allen Kontextgrößen, Netzstrukturen, Merkmalen (außer der Wortlänge) und unabhängig von den quantitativen Ergebnissen zeigt, ist eine Verbesserung der Ergebnisse mit einer Zunahme der Länge der Muster in den Knoten. Dabei erreichen die Ergebnisse stets ein Maximum und sinken dann bei weiter zunehmender Mustergröße (vgl. Abb. 5.4, S. 50 und 5.5, S. 50). Die optimalen Ergebnisse aller Kontextfenster haben 24 (Fenster ± 4), 32 (Fenster ± 2 und ± 8) oder 64 (Fenster ± 16) Positionen in den Blattmustern (vgl. Abb. 5.4, S. 50 und 5.5, S. 50).

Die Einführung einer Zwischenebene führt zu einer dramatischen Verschlechterung der Ergebnisse, die erst bei zunehmender Größe der Netze langsam wieder besser werden. Mit größeren Netzen werden aber bei der beschriebenen Vorgehensweise auch die Kontexte größer, was angesichts der oben beschriebenen Ergebnisse zur optimalen Kontextgröße nicht wünschenswert ist (vgl. Abs. 7.2.2, S. 62 für einen Lösungsansatz zu diesem Problem).

Die Abstraktionsschritte von sprachlichen Symbolen zu Zahlen und von Zahlen zu Mustern scheinen dagegen trotz der Einfachheit der implementierten Verfahren zur Merkmalsberechnung grundsätzlich zu funktionieren. Mögliche Anpassungen verschiedener Aspekte des Verfahrens zur Verbesserung der Ergebnisse werden in Abschnitt 7.2 (S. 61) beschrieben.

Kapitel 6.

Anwendung

In diesem Kapitel soll die Rolle der WSD in einem konkreten Anwendungsfall der maschinellen Sprachverarbeitung untersucht werden. Dazu wird die Aufgabe der WSD bei der Eigennamenerkennung beschrieben und eine Umsetzung des beschriebenen WSD-Verfahrens im Rahmen von Tesla (einer *Software Architecture for Language Engineering*, SALE) dargestellt, die eine Anwendung der WSD in unterschiedlichen Kontexten ermöglicht.

6.1. Modularisierung und Integration

Sprachverarbeitende Verfahren basieren häufig auf einer Vorverarbeitung der Daten; so erfordern praktisch alle Sprachanalyseverfahren eine Einteilung des Textes in Sätze oder Wörter (durch einen *Sentence Splitter* bzw. *Tokenizer*). Viele Verfahren setzen auch weitergehende Informationen über den Text ein, etwa Wortarten oder syntaktische Strukturen. Häufig bilden Ergebnisse von sprachverarbeitenden Verfahren auch selbst wieder die Grundlage für eine Weiterverarbeitung. Im Fall der WSD ist beides der Fall, so werden etwa POS-Tags der Wörter im Kontext eines Zielwortes als Merkmale verwendet (Márquez *et al.* 2006:175) und die WSD ist ihrer Natur als Mittel entsprechend immer Grundlage für eine weitere Verarbeitung.

Bis vor kurzem wurden sprachverarbeitende Verfahren meist isoliert implementiert und evaluiert, was aber besonders bei Aufgaben, die keinen Selbstzweck darstellen (wie im Fall der WSD) vor dem Hintergrund der Relevanz der so gewonnenen Ergebnisse eine fragwürdige Praxis darstellt (vgl. Ide & Wilks 2006:57, 64; Palmer *et al.* 2006:99; Resnik 2006). Daher wäre eine modulare Umsetzung des Verfahrens wünschenswert, die durch Komponenten und definierte Schnittstellen einen integrierten Einsatz der WSD ermöglicht. Im Folgenden beschreibe ich deshalb eine Umsetzung des WSD-Verfahrens in einer komponentenbasierten Umgebung zur Sprachverarbeitung, einer *Software Architecture for Language Engineering* (SALE).

6.1.1. Komponentenbasierte Softwareentwicklung

Komponentenbasierte Softwareentwicklung ist eine Weiterentwicklung der objektorientierten Softwareentwicklung mit dem Ziel einer verbesserten Wiederverwertbarkeit größerer Softwarebausteine und einer daraus resultierenden Fokussierung auf die Geschäftslogik der zu erstellenden Anwendung anstelle der dazu nötigen Infrastruktur (Gruhn & Thiel 2000:1).

Eine Komponente ist ein gekapseltes Stück Software, das eine bestimmte Funktionalität, welche auch als *Service* bezeichnet wird, anbietet. Analog zur Beziehung von *Instanz* zu *Klasse* ist eine *Komponenteninstanz* das Gegenstück zur statischen Beschreibung einer Komponente. Eine Komponenteninstanz kann auch über ihre Lebensdauer hinaus eine persistente Identität haben. Die Anforderungen an eine Komponente werden durch das *Komponentenmodell* definiert, in dem die Komponente verwendet werden soll (Gruhn & Thiel 2000:15-6).

6.1.2. Komponentensysteme in der Computerlinguistik

In jüngerer Zeit werden Verfahren zur maschinellen Sprachverarbeitung vereinzelt nicht mehr nur isoliert, sondern integriert, in einer *Software Architecture for Language Engineering* (SALE), einer Infrastruktur für die maschinelle Sprachverarbeitung, entwickelt. Eine solche Infrastruktur besteht aus Frameworks, Referenzarchitekturen und einer Entwicklungsumgebung und bildet damit eine Art Werkzeugkasten für die computerlinguistische Arbeit (s. Cunningham & Bontcheva 2006 sowie Köhler 2005). Im Sinne der oben skizzierten Konzepte komponentenbasierter Entwicklung definiert eine SALE so unter anderem das Komponentenmodell, mit dem die sprachverarbeitenden Komponenten verwendet werden sollen.

Die Umsetzung in einer SALE verbindet die Forderung nach einer Integration der Komponenten, hier sowohl der Komponenten der WSD selbst (s.u.), als auch der Komponenten zu ihrer Anwendung, mit dem Anspruch von Modularität und Wiederverwertbarkeit, sowohl computerlinguistisch durch *dynamische Annotation*, bei der das ursprüngliche Signal stets verfügbar bleibt (vgl. Abs. 3.1.1, S. 15, siehe auch Hermes & Benden 2005) als auch softwaretechnisch durch *dynamische Konfiguration* (Hunt & Thomas 2003:135) wiederverwertbarer Komponenten.

Im Folgenden wird eine Umsetzung des beschriebenen WSD-Verfahrens sowie dessen Anwendung zur Eigennamenerkennung im Rahmen von *Tesla*¹, einer solchen SALE, be-

¹ *Tesla* (Text Engineering Software Laboratory) ist eine an der Universität zu Köln an der Abteilung für Sprachliche Informationsverarbeitung entwickelte SALE, die auf Konzepten aus dem von der

schrieben. Kennzeichen von Tesla (vgl. Hermes & Schwiebert 2007) sind die konsequente dynamische Annotation (s. Benden & Hermes 2004, Hermes & Benden 2005), die IDE-Integration in Eclipse sowie eine Client-Server-Architektur zur Durchführung rechenaufwändiger Analysen und zur zentralen Bereitstellung von Komponenten, Korpora und Ergebnissen. Die durchgehend komponentenbasierte² Client-Server-Architektur unterscheidet Tesla von anderen vergleichbaren Systemen³.

6.2. Semantische Annotation in Tesla

Das Ergebnis einer Informationsextraktion kann als semantische Annotation festgehalten werden, bei der Textabschnitte mit ihrer Bedeutung annotiert werden, etwa *Rhein* als ‘Fluss’. Eine solche semantische Annotation ist für verschiedene direkte Anwendungsfälle (z.B. in der Lexikographie, s.u.) hilfreich, sowie für zahlreiche Probleme der maschinellen Sprachverarbeitung nützlich oder sogar notwendig.

Eigennamenerkennung (*named entity recognition*) ist eine Form von Informationsextraktion, bei der in einem Text Eigennamen nach bestimmten Kriterien gesucht werden, z.B. könnten in biologischen Texten bestimmte Spezies, in medizinischen bestimmte Medikamente oder bei Nachrichten bestimmte Personen, Firmen oder Orte erkannt werden. *Eigennamen* bezieht sich dabei hier, wie bei Frege (1892:27), auf die Bezeichnung eines bestimmten einzelnen Gegenstandes, wobei diese Bezeichnung aus einem einzigen oder auch aus mehreren Wörtern⁴ bestehen kann (vgl. Abs. 1.3.1, S. 3).

Fritz-Thyssen-Stiftung geförderten Projekt *SemGen* aufbaut (s. <http://www.spinfo.uni-koeln.de/space/Forschung/Tesla>). Tesla war zum Zeitpunkt der Entstehung der vorliegenden Arbeit noch in Entwicklung und nicht in einer finalen Version verfügbar. Die hier beschriebene Umsetzung ist daher keine vollständige Implementation, sondern eine Untersuchung zur grundsätzlichen Möglichkeit einer solchen Umsetzung in der angestrebten Kapselung. Zugleich werden aber wohl nur wenige Anpassungen nötig sein, um die Komponenten in einer finalen Version von Tesla verfügbar zu machen.

2 Die eingesetzten Komponentenmodelle sind im Client Eclipse-Plugins und damit OSGi (The Dynamic Module System for Java, <http://www.osgi.org/>, s. auch JSR-291, Dynamic Component Support for Java SE, <http://www.jcp.org/en/jsr/detail?id=291>), und im Server EJB3 (Enterprise Java Beans, <http://java.sun.com/products/ejb/>, vgl. Burke & Monson-Haefel 2006)

3 Die bekanntesten SALES sind GATE (General Architecture for Text Engineering, <http://gate.ac.uk/>) und das neuere UIMA (Unstructured Information Management Architecture, <http://www.research.ibm.com/UIMA/>)

4 In diesem Sinn ist WSD, wie in dieser Arbeit beschrieben, eigentlich Eigennamendisambiguierung, da abhängig davon, was als Sinneinheit annotiert wurde, einzelne Wörter, Wortgruppen oder Wortteile disambiguiert werden könnten.

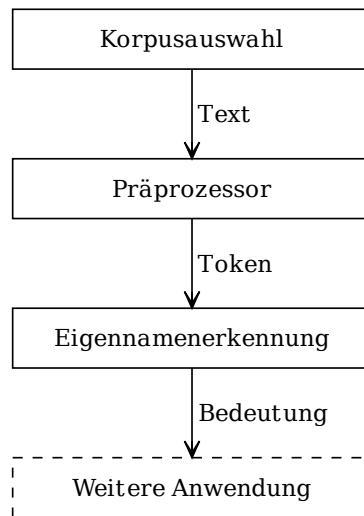


Abbildung 6.1.: Komponenten bei der Eigennamenerkennung ohne WSD

6.2.1. Wortlisten zur Annotation

Eine einfache Umsetzung einer solchen Eigennamenerkennung kann durch Listen mit den Eigennamen⁵ erfolgen. Hierbei wird pro Kategorie eine Liste angelegt, z.B. für alle Flüsse oder alle Städte, welche dann konkrete Einträge enthalten, wie *Rhein* bzw. *Köln*.

Wortlisten bilden eine gut wartbare, flexible Form eines maschinenlesbaren Lexikons, so ist etwa neben der Nutzung zur Informationsextraktion eine Integration von POS-Tags ebenso denkbar wie jede andere Form von Klassifikation; es müssen lediglich entsprechende Wortlisten (manuell oder automatisch) erstellt werden, ohne dass eine Modifikation des Programms nötig wäre. Es handelt sich hierbei um eine Form von Metaprogrammierung, oder dynamischer Konfiguration, bei der das Programm die Abstraktionen und die Metadaten (hier die Listen) die Details enthalten (vgl. Hunt & Thomas 2003:135).

Wäre in menschlicher Sprache jeder Wortform eindeutig eine Bedeutung zugeordnet, wäre eine semantische Annotation von Texten ohne große Schwierigkeiten automatisierbar, einfach durch ein Abrufen der Bedeutung einer Wortform in einer Datenstruktur, die einem Wörterbuch entspricht und die auf Grundlage der Listen gefüllt würde.

⁵ Auf Wortlisten basiert etwa ANNIE, die Informationsextraktionskomponente von GATE (<http://www.gate.ac.uk/ie/annie.html>)

Die Namen der Listen (konkret der Dateien) bilden dabei die semantische Kategorie oder Bedeutung der in der Liste enthaltenen Wörter. Beim Vorfinden einer Wortform (etwa *Rhein* oder *Köln*), lässt sich diese so kennzeichnen (etwa als ‘Fluss’ bzw. als ‘Stadt’).

Mehrdeutige Wörter tauchen dabei in mehreren Listen auf, so könnte etwa *Bank* in den Listen ‘Möbel’ (mit Einträgen wie *Tisch* und *Stuhl*) und ‘Gebäude’ (mit Einträgen wie *Kino* oder *Rathaus*) enthalten sein. Dabei ergibt sich folgendes Problem: Wie sollen Einträge, die in mehreren Listen enthalten sind, annotiert werden? Denn im Kontext des Vorkommens ist nur eine der Lesarten richtig. Hier benötigen wir also einen Mechanismus zur WSD.

In einer SALE könnte die Eigennamenerkennung in der beschriebenen Form, ohne WSD, etwa aus den Komponenten in Abbildung 6.1 (S. 55) bestehen. Ohne WSD würden hier etwa bei Einträgen, die in mehreren Listen sind, alle Kategorien vergeben, etwa für alle Vorkommen von *Bank* sowohl ‘Möbel’ als auch ‘Gebäude’. Das Resultat dieses Aufbaus ist so ein semantisch annotierter, aber nicht disambigierter⁶ Text. Das WSD-Verfahren müsste hier also als nachgelagertes Verfahren die ambigen Annotationen disambiguieren.

6.2.2. Komponenten beim Training

Bei einer komponentenbasierten Umsetzung der WSD stellt sich zunächst die Frage, inwieweit das Verfahren selbst modularisiert werden kann. Hierbei bietet sich für das beschriebene Verfahren eine Umsetzung an, welche die Hauptkomponenten eines generischen Klassifikationssystems (vgl. Abb. 3.1, S. 18), nämlich die Merkmalsgenerierung und die eigentliche Klassifikation, kapselt.

Wie in Abschnitt 3.2.2 (S. 17) beschrieben, ist die Eingabe eines Klassifikationsverfahrens beim Training ein Merkmalsvektor und die zugehörige Klasse, bei der Klassifikation nur ein Merkmalsvektor. Entsprechend habe ich die Kapselung in Komponenten zur Merkmalsgenerierung und zur Klassifikation implementiert. Hinzu kommt eine Komponente zur Bereitstellung der annotierten Korpora beim Training.

Dabei hängt, entsprechend der oben beschriebenen Zusammenhänge der Komponenten, die Klassifikationskomponente von der Merkmalsgenerator Komponente und der bedeutungsliefernden Komponente ab (s. Abb. 6.2, S. 57). Beim Training etwa ergibt sich so ein Zusammenspiel der Komponenten wie in Abbildung 6.3 (S. 58). Die Formulierung eines

6 Eine solche semantische Annotation, die nicht disambiguiert ist, ermöglicht bei der Extraktion einen Recall von 100%, da die richtige Bedeutung in jedem Fall dabei ist. Dass die Precision durch ambige Wörter reduziert ist, ist bei einer Verarbeitung der Annotationen durch Menschen möglicherweise unproblematisch. Für eine maschinelle Weiterverarbeitung allerdings ist eine Disambiguierung nötig (vgl. Abs. 1.2, S. 2).

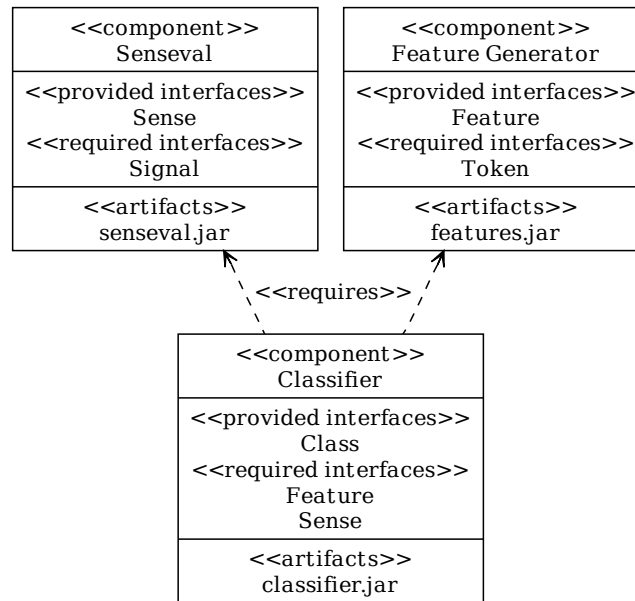


Abbildung 6.2.: Komponentendiagramm der entwickelten Komponenten in UML-Tabellennotation (vgl. Oestereich 2005)

entsprechenden Experiments im Konfigurationsformat von Tesla findet sich in Anhang A.2 (S. IV).

Der Vorteil einer solchen modularisierten Umsetzung der WSD selbst besteht darin, dass die beiden Hauptkomponenten des Verfahrens, die numerische Abstraktion der Symbole im Kontext eines Zielwortes und die Klassifikation der Kontexte (und damit die Disambiguierung des Zielwortes), austauschbar werden: so könnte etwa die entwickelte Komponente zur Merkmalsgenerierung mit anderen Klassifikationskomponenten evaluiert werden oder die entwickelte Komponente zur Klassifikation mit anderen Komponenten zur Merkmalsgenerierung.

Die Umsetzung der Senseval-Annotation als Komponente macht die annotierten Senseval-Korpora zudem für andere Verfahren zugänglich, die keine der anderen zwei Komponenten verwenden. Dies erschließt prinzipiell⁷ auch die Senseval-Korpora in anderen Sprachen für

⁷ Die Daten haben grundsätzlich ein gemeinsames Format und sind so prinzipiell wie hier implementiert nutzbar. Sie sind jedoch z.T. nicht in validem XML (z.B. keine maskierten Sonderzeichen), sowie manchmal in einer einzelnen Datei und manchmal für verschiedene Lemmata auf verschiedene Dateien verteilt. Hier müsste also ein Import der verschiedenen Varianten des Formats implementiert werden.

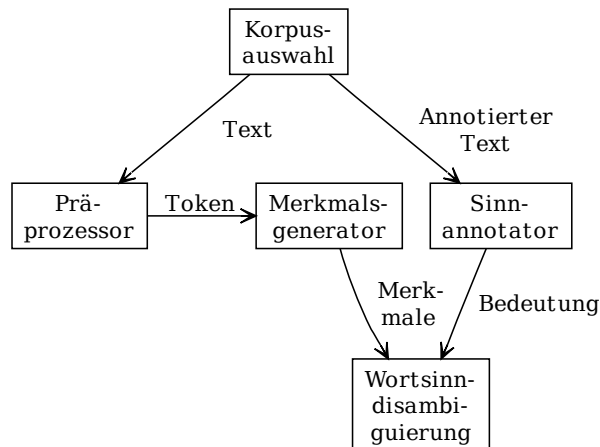


Abbildung 6.3.: Komponenten beim Training

solche Verfahren. Neben der WSD sind auch andere Aufgaben der maschinellen Sprachverarbeitung tokenbezogene Klassifikationsprobleme, etwa das Auszeichnen von Wortarten (*part of speech tagging*, POS-Tagging). Hier würde etwa ein bloßer Austausch des Korpus gegen ein mit POS-Tags ausgezeichnetes Korpus die Nutzung der (eigentlich zur WSD implementierten) Merkmalsberechnung und Klassifikation zum POS-Tagging ermöglichen (vgl. Abb. 7.2, S. 62).

6.2.3. Komponenten bei der Klassifikation

Bei der Anwendung der trainierten Komponente sind die Schnittstellen im Grunde die gleichen wie beim Training, nur dass bei der Anwendung hier die Zielwörter nicht durch die annotierten Korpora gegeben sind, sondern durch die ambigen Annotationen der Eigennamenerkennung. Diese konsumiert zudem nicht (wie die Senseval-Komponente) direkt das Signal, sondern Token des Präprozessors (vgl. Abb. 6.4, S. 59). Bei einer solchen Anwendung wäre zu untersuchen, was sich für Werte für Precision und Recall der Eigennamenerkennung für das ambige und das disambiguierte Korpus ergeben (vgl. Abs. 5.2.4, S. 43).

Wie in Abbildung 6.4 (S. 59) auch deutlich wird, kann der beschriebene Anwendungsfall der Informationsextraktion (wie die WSD) selbst wieder Grundlage für weitere Anwendungen sein, die auf den von der IE-Komponente erstellten und von der WSD-Komponente

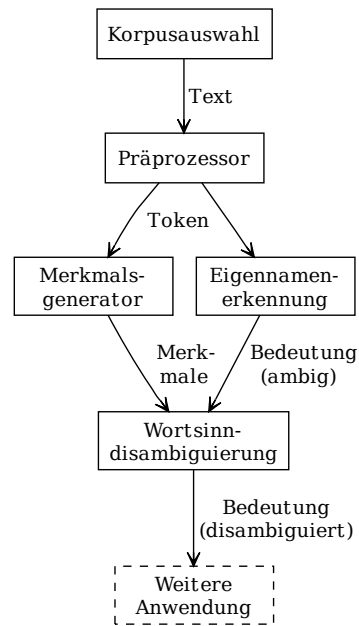


Abbildung 6.4.: Komponenten bei der Anwendung

disambiguierten Annotationen aufsetzen. So stellt die beschriebene semantische Annotation zwar auch einen direkten Anwendungsfall dar, ist zugleich aber auch die Form, in der die Ergebnisse der WSD anderen Komponenten zugänglich gemacht werden können (vgl. Abs. 7.2.4, S. 64).

Kapitel 7.

Schluss: Maschinelles Lernen mit Komponenten

7.1. Zusammenfassung

In der vorliegenden Arbeit wurde, basierend auf einer Konzeption von Wortsinn als Kontextabstraktion (Kap. 1) und Kognition als domänenunabhängige, hierarchische Klassifikation (Kap. 2), mit Konzepten des korpusbasierten maschinellen Lernens (Kap. 3) ein Verfahren zur WSD entwickelt (Kap. 4), auf Daten des BNC evaluiert (Kap. 5) und modular in einer SALE implementiert (Kap. 6).

Die Arbeit stellt die erste mir bekannte Anwendung einer hierarchischen BP zur WSD dar. Die Ergebnisse zeigen eine grundsätzlich vielversprechende Tendenz, könnten aber vermutlich durch einen Ausbau einiger hier nur rudimentär umgesetzter Aspekte deutlich verbessert werden; dies betrifft verschiedene Bereiche, wie die zugrunde gelegten Korpora (s. Abs. 7.2.1, S. 61), die Merkmalsauswahl und -berechnung (s. Abs. 7.2.2, S. 62) sowie die darauf aufbauende Klassifikation (s. Abs. 7.2.3, S. 63).

Durch die beschriebene modulare Umsetzung wird die Übereinstimmung kognitiver Konzepte eines domänenunabhängigen Verarbeitungsverfahrens mit Konzepten des maschinellen Lernens deutlich, wo basierend auf domänenspezifisch fundierten¹ Merkmalen mit domänenunabhängigen Algorithmen klassifiziert wird. Die direkte Umsetzbarkeit dieser Kapselung in einer SALE (vgl. Abb. 7.1, S. 61 sowie Kap. 6, S. 52) halte ich für eine vielversprechende Perspektive, da so durch einen Vergleich verschiedener Komponenten die besten Komponenten je nach Anwendungsfall ausgemacht und miteinander kombiniert werden könnten (s. Abb. 7.2, S. 62). Eine solche Herangehensweise verspricht nicht nur die Ermöglichung hochoptimierter Verfahren (da Möglichkeiten zur Optimierung in jeder gekapselten Komponente gegeben sind und diese miteinander kombiniert werden können), sondern stellt zugleich für bestimmte Aufgaben entwickelte Komponenten zugleich für neue Anwendungsfälle zur Verfügung.

¹ Der domänenspezifische Hintergrund bestimmt, wie und woraus die Merkmale gewonnen werden sollten. Im vorgestellten Verfahren sind dies etwa die in Kapitel 1 beschriebenen Vorstellungen zur Wortsemantik als Kontextabstraktion, und dementsprechend die korpusbasierte Merkmalsberechnung (vgl. Abschnitt 4.1, S. 22).

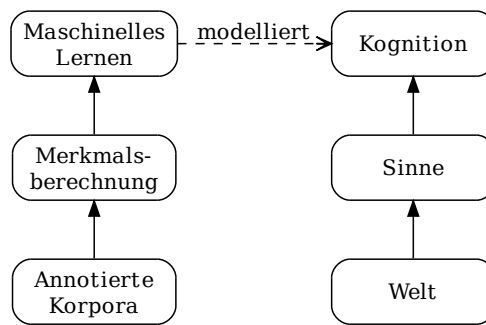


Abbildung 7.1.: Modellierung kognitiver Informationsverarbeitung durch maschinelles Lernen mit den umgesetzten Komponenten

7.2. Ausblick

Mögliche Anknüpfungspunkte ergeben sich unabhängig innerhalb jeder gekapselten Komponente: im Bereich der annotierten Korpora (s. Abs. 7.2.1, S. 61), der Merkmalsberechnung (s. Abs. 7.2.2, S. 62), dem eigentlichen Klassifikationsverfahren (s. Abs. 7.2.3, S. 63), sowie der möglichen Anwendungen der WSD (s. Abs. 7.2.4, S. 64).

7.2.1. Korpora

Im Bereich der Korpora bieten sich Ausbaumöglichkeiten auf unterschiedlichen Ebenen. Eine Möglichkeit ist etwa der Einsatz anderer Korpora mit dem umgesetzten Verfahren; neben den Senseval-Korpora der verschiedenen Veranstaltungen in verschiedenen Sprachen existieren andere, speziell zur WSD zusammengestellte Korpora, die zum Vergleich mit den Ergebnissen dieser Arbeit herangezogen werden könnten (vgl. Palmer *et al.* 2006). Wie in Abschnitt 5.3 (S. 44) beschrieben, liegen die besten Ergebnisse mit dem beschriebenen Verfahren leicht über der Baseline, und auch relativ nah am ITA (vgl. Abb. 5.2, S. 49 und 5.3, S. 49). Da allerdings für das verwendete Korpus Baseline und ITA sehr nah aneinander liegen (Baseline 55%, ITA 67%), wäre hier zu untersuchen, ob die Ergebnisse zur Baseline oder zum ITA tendieren. Da das Verfahren prinzipiell sprachunabhängig ist und Senseval-Korpora für verschiedene Sprachen verfügbar sind, bietet sich hier eine Evaluierung mit einem Korpus an, bei dem Baseline und ITA weiter auseinander liegen, etwa das italienische Korpus (Baseline 18%, ITA 89%, vgl. Agirre & Edmonds 2006a:16).

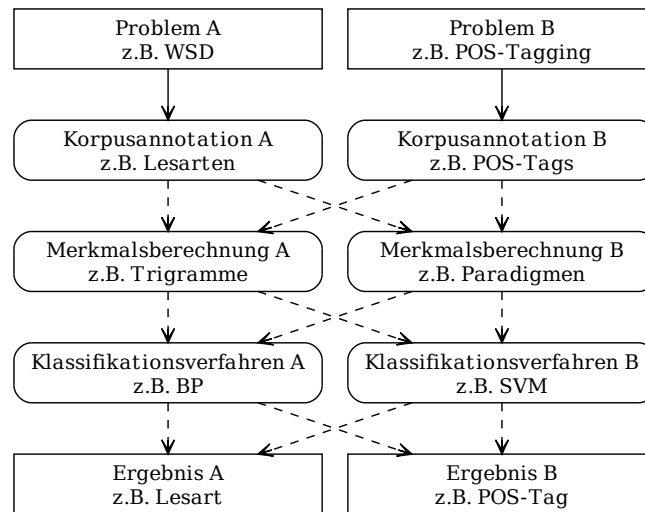


Abbildung 7.2.: Austauschbarkeit von Komponenten zur Merkmalsberechnung und zur Klassifikation innerhalb eines Problems und für andere Probleme, eine problemspezifische Korpusannotation vorausgesetzt

Außer dem Einsatz anderer Korpora für die WSD ist auch der Einsatz mit anders annotierter Korpora denkbar, und damit ein Einsatz für andere Anwendungsfälle. So könnte etwa mit einem Korpus, das anstelle von Lesarten mit Wortarten annotiert ist, das beschriebene Verfahren auch für das POS-Tagging verwendet werden (vgl. Abb. 7.2).

Neben der Verwendung von fertigen, manuell zusammengestellten Korpora wäre hier auch die Entwicklung und Integration von Verfahren zur automatische Erstellung von Trainingskorpora zur Umgehung des *lexical acquisition bottleneck* (s. Abs. 3.2.1, S. 17) denkbar.

7.2.2. Merkmalsberechnung

Da die verschiedenen Merkmale durchweg Ergebnisse nahe der Baseline liefern, wäre eine Kombination der Merkmale, eventuell zu größeren Merkmalsvektoren (für gleichbleibende Kontextfenster) eine Möglichkeit zur Verbesserung des Verfahrens. Eine solche Verbesserung könnte so möglicherweise nicht nur durch die Kombination der Merkmale zu besseren Ergebnissen führen, sondern auch durch die Ermöglichung größerer Netze durch die größeren Merkmalsvektoren (vgl. Abs. 4.1, S. 22).

Darüberhinaus wäre von einer Optimierung der verschiedenen Verfahren zur Merkmals-

berechnung eine Verbesserung der Ergebnisse zu erwarten, etwa durch Vorverarbeitung in Form einer Stammformenreduktion vor der Ermittlung der Wort- oder Paradigmenmerkmale, oder durch die Verwendung von bestehenden Verfahren zur Ermittlung semantisch verwandter Wörter (z.B. SOG, s. Schwiebert 2005, Schwiebert & Rolshoven 2006) mit einer darauf aufbauenden Konvertierung in eine numerische Repräsentation.

Ein solcher, auf Paradigmen basierender Ansatz könnte auch zu einer Umstellung der Klassifikation auf mehrere Vektoren pro Kontext genutzt werden. Dabei könnten etwa für jedes Wort w im Kontext eines Zielwortes alle mit w semantisch verwandten Wörter numerisch repräsentiert werden, z.B. wie die Wörter im beschriebenen Verfahren. Jedem Wort im Kontext würde so ein Vektor zugeordnet, und damit semantisch verwandte Wörter aller Wörter im Kontext als Merkmale für das Zielwort verwendet. Diese Vektoren könnten von einem Baum nacheinander bei Training und Disambiguierung für jeden Kontext verarbeitet werden. Dies würde auch stärker den Vorstellung des MPF entsprechen (vgl. Abs. 2.2, S. 9 und 4.2.4, S. 33).

7.2.3. Klassifikation

Die Voraussetzungen der durchgeführten Experimente unterscheiden sich in einem Punkt deutlich von den in der Literatur beschriebenen Anwendungen des MPR zur optischen Mustererkennung, nämlich der Anzahl der möglichen Klassen. Diese sind im Bereich der WSD deutlich geringer (vgl. Tab. 5.1, S. 41) als bei der optischen Buchstabenerkennung (Thornton *et al.* 2006) oder der Erkennung von Piktogrammen (Hawkins & George 2005). Dies hat eine direkte Auswirkung auf die Beschaffenheit der Tabellen in den Knoten direkt unterhalb der Wurzel (vgl. Abs. 4.2, S. 28) und damit auf den letzten, entscheidenden Klassifikationsschritt. Im Zusammenhang damit steht ein weiterer Unterschied zu den genannten Umsetzungen, nämlich dass nicht der gesamte Problembereich in einem einzigen Netz trainiert und klassifiziert wird, sondern dass separate Netze für jedes ambige Lemma existieren, die jeweils die Lesarten dieses Lemmas unterscheiden. Möglicherweise wäre denkbar, alle Lesarten aller berücksichtigten Lemmata in einem einzigen Netz zu trainieren und damit näher an beschriebenen Umsetzungen der Konzepte des MPF zu sein.

Als Modifikation der BP, die hier in Bäumen erfolgt, bietet sich etwa eine BP in gerichteten, azyklischen Graphen (für die BP in präziser Form beschrieben ist) oder in allgemeineren Graphen (für die BP als Approximationsalgorithmus beschrieben ist) an (vgl. Abs. 4.2.4, S. 33). Daneben wäre hier auch der Einsatz alternativer Klassifikationsverfahren, z.B. *support vector machines* (SVM, vgl. Márquez *et al.* 2006:180) denkbar, etwa zum Vergleich mit BP-Ansätzen.

Als weitergehende Veränderung des Verfahrens ist schließlich eine Weiterentwicklung zu einem völlig unüberwachten Verfahren denkbar, etwa indem die verschiedene Aktivierungen der Netze geclustert werden, oder auf Basis von bestehenden unüberwachten Lernverfahren (vgl. Pedersen 2006, Hawkins & George 2006).

7.2.4. Anwendungen

Als direkter Anknüpfungspunkt bietet sich bei der Anwendung zur IE eine automatische Erstellung der zur IE eingesetzten Wortlisten an, um eine semantische Annotation möglichst vieler Wörter im Text automatisch zu erreichen. Für solche Listen gibt es auch weitere Anwendungsmöglichkeiten, etwa die IE durch gezielte Zusammenfassung (s. etwa Euler 2001b,a, 2002).

Die WSD könnte neben der hier umrissenen Informationsextraktion auch für andere Aufgaben der maschinellen Sprachverarbeitung genutzt werden, etwa zur maschinellen Übersetzung (MÜ). Eine solche MÜ-Komponente würde dann neben dem Text auch auf die Bedeutungen der Wörter, soweit disambiguiert, zugreifen. Eine weitere Anwendungsmöglichkeit liegt im *Information Retrieval* (IR), wobei zum Suchwort hier die gewünschte Lesart angegeben werden könnte und so nur Dokumente, in denen das gesuchte Wort in der gesuchten Lesart vorhanden ist, als Ergebnisse geliefert würden. Eine solche semantische Suchmaschine könnte etwa in der Lexikographie eingesetzt werden: der Lexikograph muss so nicht alle Fundstellen eines ambigen Wortes betrachten, sondern nur jene, die der zu beschreibenden Lesart entsprechen (Agirre & Edmonds 2006a:11).

Weitere Informationen zur Anwendung von WSD in der Sprachverarbeitung finden sich in Resnik (2006). Bei der Nutzung der WSD-Komponenten durch unterschiedliche Anwendungen sollte dabei untersucht werden, wie hilfreich die WSD für die verschiedenen Aufgaben ist (s. Abs. 5.2, S. 41 und 6.2, S. 54, vgl. Kilgarriff 1997).

Anhang A.

Implementationsdetails

A.1. Umsetzungen der Algorithmen

Im Folgenden finden sich Implementationen der in Kapitel 4 (S. 22) beschriebenen Algorithmen in Java. Die vollständige Umsetzung ist auf dem beigelegten Datenträger enthalten (s. Anhang A.3, S. IX).

Numerische Abstraktion für Tri- und Heptagramme (Algorithmus 1, S. 25)

Die Berechnung entspricht funktional der Hashcode-Berechnung für Zeichenketten in der Methode `hashCode` der Klasse `String` von Java. Für das Englische kann, wie hier erfolgt, auf die interne Repräsentation der Zeichen zurückgegriffen werden. Für andere Sprachen würde eine sortierte Liste mit den möglichen Zeichen benötigt (vgl. Algorithmus 1, S. 25).

```
private int code(String ngram) {
    int n = 0;
    char val[] = ngram.toCharArray();
    for (int i = 0; i < val.length; i++) {
        n = 31 * n + val[i];
    }
    return n;
}
```

Auslesen von Paradigmen aus einem Suffixbaum (Algorithmus 2, S. 27)

Sammelt die eingehenden Kantenbeschriftungen von Kindern innerer Knoten:

```
for (Node node : tree.getAllNodes()) {
    if (node.isInternal()) {
        // the current paradigm: all children of an inner node
        Paradigm p = new Paradigm();
        for (Node child : node.getChildren()) {
            /*
             * the first word of the child's incoming edge label is a
             * member of the paradigm:
             */
            String paradigmMember = tree.getIncomingEdgeLabel(
                child).split("_")[0];
            if (!paradigmMember.trim().equals("")) {
                p.add(paradigmMember);
            }
        }
        paradigms.add(p);
    }
}
```

Sortieren von Paradigmen (Algorithmus 3, S. 27)

Die Implementierung des Interface `Comparable` ermöglicht es in Java, für Objekte eine natürliche Sortierfolge zu definieren, die von Sortieralgorithmen (etwa `Arrays.sort` und `Collections.sort`) und sortierten Datenstrukturen (wie `TreeSet`) berücksichtigt wird. Dazu muss die Methode `compareTo` implementiert werden:

```
public int compareTo(Paradigm o) {
    Iterator<String> otherIterator = o.members.iterator();
    for (String m1 : members) {
        String m2 = null;
        if (otherIterator.hasNext())
            m2 = otherIterator.next();
        if (!m1.equals(m2) && m2 != null) {
            // compare the first different member:
            return m1.compareTo(m2);
        }
    } // if no member was different, the paradigms are equal:
    return 0;
}
```

Abstraktionsalgorithmus für Blattknoten (Algorithmus 4, S. 29)

Aktiviert eine Position in einem `StringBuilder` der gewünschten Länge:

```
private String pattern(StringBuilder m, float n) {
    for (int i = 0; i < m.length(); i++) {
        double v = 1.0 / m.length() * i;
        if (n <= v || i == m.length() - 1) {
            m.setCharAt(i, '1');
            return m.toString();
        }
    }
    return null;
}
```

Rekursive Generierung der Sprachelemente (Algorithmus 5, S. 30)

Generiert rekursiv alle gesuchten Permutationen durch Backtracking:

```
private void addElements(double length, int counter, int max,
    StringBuilder builder, List<String> words) {
    if (builder.length() == length) {
        if (builder.toString().contains("1"))
            words.add(builder.toString());
    } else {
        if (counter == max) {
            StringBuilder b = new StringBuilder(builder);
            for (int j = b.toString().length(); j <= length - 1; j++) {
                b.append("0");
            }
            if (b.toString().contains("1")) words.add(b.toString());
        } else {
            addElements(length, counter + 1, max,
                new StringBuilder(builder).append("1"), words);
            addElements(length, counter, max, new StringBuilder(builder)
                .append("0"), words);
        }
    }
}
```

Generierung der Sprachelemente über eine Bitmaske (Algorithmus 6, S. 31)

Optimierte Generierung der Permutationen über eine Bitmaske, implementiert auf Basis der Klasse `BigInteger` zur Unterstützung beliebig großer Masken und Vermeidung direkter Bitoperationen (siehe Kommentar):

```
/**
 * Computation of permutations of strings with 0 to n activated (1) and else
 * unactivated (0) positions of a specified length. Computation is low-level
 * optimized by usings binary representations of integers representing sets. The
 * implementation is based on the BigInteger class to allow arbitrary set sizes
 * (opposed to 32-bit limit for ints and 64-bit limit for longs) and high-level
 * coding style (e.g. by using the testBit method instead of direct bit shifting
 * operators).
 */
public class BigBitwisePermutationGenerator implements PermutationGenerator {
    private Set<String> set = new HashSet<String>();

    private int length;

    public Set<String> permutations(int length, int num) {
        this.length = length;
        /* We create a binary number of the required length:
        */
        BigInteger all = setAll(length);
        /* We compute all subsets with a specified maximum number of activations:
        */
        subsets(all, num);
        return set;
    }

    private BigInteger setAll(int length) {
        char[] a = new char[length];
        Arrays.fill(a, '1');
        return new BigInteger("1" + new String(a), 2);
    }

    void subsets(BigInteger all, int num) {
        /* For a set with n entries, the binary representation of every n-bit
        * value corresponds to a subset, so while we are greater than 0 we
        * always subtract 1 and use the resulting value as the binary
        * representation of a subset:
        */
        for (BigInteger a = all; !a.equals(BigInteger.ZERO);
            a = (a.subtract(BigInteger.ONE))) {
            collect(a, num);
        }
    }

    private void collect(BigInteger a, int num) {
        /*
        * We create a string representation corresponding to the bit mask:
        */
        StringBuilder builder = new StringBuilder();
        for (int i = length; i >= 1; i--) {
            builder.append(a.testBit(i) ? "1" : "0");
        }
        /**
        * If the permutation has the required number of activated positions,
        * add it to the result:
        */
        BigInteger b = new BigInteger(builder.toString(), 2);
        if (b.bitCount() <= num && b.bitCount() > 0) {
            set.add(builder.toString());
        }
    }
}
```

Parallelisiertes Lernen (Algorithmus 7, S. 38)

Jedes Lemma wird, ohne Zugriff auf gemeinsame Daten, in einem eigenen Thread trainiert:

```
for (final String lemma : lists.keySet()) {
    Thread thread = new Thread(new Runnable() {
        public void run() {
            Map<String, List<Ambiguity>> map = lists.get(lemma);
            for (final String sense : map.keySet()) {
                List<Ambiguity> name = map.get(sense);
                // get the applicable tree:
                BayesTree tree = lexicon.get(lemma);
                for (Ambiguity ambiguity : name) {
                    if (tree == null) {
                        // it's the first time we train for the lemma:
                        tree = new BayesTree(structure, patternFactor,
                            classes.get(ambiguity.getLemma()));
                        lexicon.put(ambiguity.getLemma(), tree);
                    }
                    // compute the features for the context:
                    float [] features = feat.getFeatures(ambiguity
                        .getContext().target, ambiguity
                        .getContext().all, tree.getLeafs().size());
                    // train the applicable tree:
                    tree.train(features, ambiguity.getCorrect());
                }
                // done training one sense, reset pattern counters:
                tree.reset(tree.getRoot());
            }
        }
    }); // start the thread for this lemma:
    thread.start();
}
```

A.2. Konfigurationsdateien

Properties

Die Properties-Datei mit der zu verwendenden Konfiguration beim Evaluieren des WSD-Verfahrens mit den Senseval-3 Daten (s. Kap. 5, S. 40):

```
# print lots of stuff, tables etc:
debug=false

# train and classify each tree in its own thread:
parallel=false

# number of digits when formatting double values:
digits=3

# stopword list settings:
filter=false
stopwords=files/stopwords
encoding=utf-8

# tree structure: the number of children of each node on the different levels of the tree:
tree_structure=4,2

# what features to use (n-gram, length, word, paradigms)
features=3-gram

# the factor for multiplying the leaf number to get the level-1 pattern size:
pattern_factor=8
```

Experimentenkonfiguration

Die Konfigurationsdatei eines Tesla-Experiments zum Training in einer Entwicklungsversion von Tesla (auf dem beigelegten Datenträger enthalten, s. Abs. A.3, S. IX); diese enthält Korpusauswahl, Präprozessierung und die drei Komponenten des WSD-Verfahrens (s. Kap. 6, S. 52).

Metadaten zum Experiment:

```
<?xml version="1.0" encoding="UTF-8"?>
<tesla_chain xmlns="http://spinfo.uni-koeln.de/tesla/chain"
  xmlns:component="http://spinfo.uni-koeln.de/tesla/component"
  xmlns:shared="http://spinfo.uni-koeln.de/tesla/shared">
  <metadata>
    <author>
      <shared:name>Fabian Steeg</shared:name>
      <shared:organization>
        Sprachliche Informationsverarbeitung
      </shared:organization>
      <shared:email>fsteeeg@spinfo.uni-koeln.de</shared:email>
      <shared:external_reference>
        http://www.spinfo.uni-koeln.de/space/Forschung/Tesla
      </shared:external_reference>
    </author>
    <description>Training mit dem Senseval-3 Korpus</description>
    <displayname>Training</displayname>
  </metadata>
</tesla_chain>
```

Komponente zur Auswahl des Korpus:

```
<corpuselector
  adapter_class="de.uni-koeln.spinfo.tesla.runtime.signal.SignalAdapter"
  selector_class="de.uni-koeln.spinfo.tesla.runtime.signal.TextCorpusSelector">
  <configuration>
    <shared:item category="query">
      <shared:value>
        SELECT e.documentId FROM CorpusAndDocument e WHERE e.corpusId='197887'
      </shared:value>
    </shared:item>
    <shared:item category="reader">
      <shared:value>
        de.uni-koeln.spinfo.tesla.runtime.signal.reader.TextFileReader
      </shared:value>
    </shared:item>
  </configuration>
  <produces_signals signal_group_id="sig_1" />
  <name />
</corpuselector>
```

Komponente zur Präprozessierung:

```
<component>
  <component:component_description>
    <name>SPre_1</name>
    <role>Preprocessor</role>
    <description>A tokenizer and sentence splitter</description>
    <version>1</version>
    <component_class>
      de.uni-koeln.spinfo.tesla.component.spre.SPre2Component
    </component_class>
  </component:component_description>
  <component:author>
    <shared:name>Christoph Benden, Juergen Hermes</shared:name>
    <shared:organization>
      Sprachliche Informationsverarbeitung, Universitaet zu Koeln
    </shared:organization>
    <shared:email>
      cbenden@spinfo.uni-koeln.de, jhermes@spinfo.uni-koeln.de
    </shared:email>
    <shared:external_reference>
      http://www.spinfo.uni-koeln.de/space/Forschung/SPre
    </shared:external_reference>
  </component:author>
  <component:produces_analysis_reference id="SPre_1" />
  <component:consumes_signals signal_group_id="sig_1" />
  <component:configuration
    editor="de.uni-koeln.spinfo.tesla.client.ui.editors.form.configurations.FallbackConfigurationEditor"
  />
</component>
```

Komponente zur Annotation mit den Senseval-Daten:

```
<component>
  <component:component_description>
```

```
<name>Senseval.1</name>
<role>Senseval Annotation</role>
<description>A component to use Senseval data in Tesla</description>
<version>1</version>
<component_class>
  de.uni-koeln.spininfo.tesla.component.wsd.senseval.SensevalComponent
</component_class>
</component:component_description>
<component:author>
<shared:name>Fabian Steeg</shared:name>
<shared:organization>
  Sprachliche Informationsverarbeitung
</shared:organization>
<shared:email>fsteeeg@spinfo.uni-koeln.de</shared:email>
<shared:external_reference>
  http://www.spininfo.uni-koeln.de/space/Forschung/Tesla
</shared:external_reference>
</component:author>
<component:produces_analysis_reference id="Senseval.1" />
<component:consumes_signals signal_group_id="sig.1" />
</component>
```

Komponente zur numerischen Merkmalsrepräsentation:

```
<component>
<component:component_description>
<name>Features.1</name>
<role>Numerical Text Representation</role>
<description>
  A component to generate a feature vector for an input text; output
  of this component is meant to act as the input for machine learning
  algorithms.
</description>
<version>1</version>
<component_class>
  de.uni-koeln.spininfo.tesla.component.wsd.features.FeatureGeneratorComponent
</component_class>
</component:component_description>
<component:author>
<shared:name>Fabian Steeg</shared:name>
<shared:organization>
  Sprachliche Informationsverarbeitung
</shared:organization>
<shared:email>fsteeeg@spinfo.uni-koeln.de</shared:email>
<shared:external_reference>
  http://www.spininfo.uni-koeln.de/space/Forschung/Tesla
</shared:external_reference>
</component:author>
<component:consumes_analysis_reference id="SPre.1"
  requires="de.uni-koeln.spininfo.tesla.component.spre.data.Token"
  role="Token" />
<component:produces_analysis_reference id="Features.1" />
<component:consumes_signals signal_group_id="sig.1" />
</component>
```

Konfiguration der zu verwendenden Merkmale:

```
<component:configuration
  editor="de.uni-koeln.spininfo.tesla.client.ui.editors.form.configurations.FallbackConfigurationEditor">
<shared:item category="Features">
<shared:description>
  The features to use (n-gram, length, word, paradigms).
</shared:description>
<shared:value>3-gram</shared:value>
</shared:item>
</component:configuration>
</component>
```

Komponente zur eigentlichen Klassifikation:

```
<component>
<component:component_description>
<name>Classifier.1</name>
<role>Classifier</role>
<description>
  A classifier based on hierarchical belief propagation, implementing
  principles of the memory-prediction framework
</description>
<version>0</version>
<component_class>
  de.uni-koeln.spininfo.tesla.component.wsd.classifier.ClassifierComponent
</component_class>
</component:component_description>
<component:author>
<shared:name>Fabian Steeg</shared:name>
<shared:organization>
  Sprachliche Informationsverarbeitung
</shared:organization>
<shared:email>fsteeeg@spinfo.uni-koeln.de</shared:email>
<shared:external_reference>

```

```
    http://www.spinfo.uni-koeln.de/space/Forschung/Tesla
  </shared:external_reference>
</component:author>
<component:consumes_analysis_reference id="Features_1"
  requires="de.uni.koeln.spinfo.tesla.component.wsd.features.Feature"
  role="Feature" />
<component:consumes_analysis_reference id="Senseval_1"
  requires="de.uni.koeln.spinfo.tesla.component.wsd.senseval.data.Sense"
  role="Sense" />
<component:produces_analysis_reference id="Classifier_1" />
```

Konfiguration der zu verwendenden Netzstruktur:

```
<component:configuration
  editor="de.uni.koeln.spinfo.tesla.client.ui.editors.form.configurations.FallbackConfigurationEditor">
  <shared:item category="Structure">
    <shared:description>
      Tree structure, number of children of each node on the levels of the tree.
    </shared:description>
    <shared:value>2,4</shared:value>
  </shared:item>
  <shared:item category="Factor">
    <shared:description>
      The factor for multiplication with the leaf number to get the level-1
      pattern size.
    </shared:description>
    <shared:value>8</shared:value>
  </shared:item>
</component:configuration>
</component>
```

Die Konfigurationsdatei enthält außerdem Angaben zu den Annotationen, über die hier die Komponenten miteinander kommunizieren.

Token des Präprozessors:

```
<analysis
  input_adapter="de.uni.koeln.spinfo.tesla.annotation.adapter.token.CachingTokenInputAdapter"
  output_adapter="de.uni.koeln.spinfo.tesla.annotation.adapter.output.DefaultOutputAdapter"
  id="SPre_1">
  <shared:role>Token</shared:role>
  <shared:annotation_type>
    de.uni.koeln.spinfo.tesla.component.spre.data.Token
  </shared:annotation_type>
  <shared:annotation_impl>Impl</shared:annotation_impl>
  <shared:view_adapter
    class="de.uni.koeln.spinfo.tesla.component.spre.converter.SPreConverter" />
</analysis>
```

Korrekte Lesarten des Trainingskorpus:

```
<analysis
  input_adapter="de.uni.koeln.spinfo.tesla.annotation.adapter.token.CachingTokenInputAdapter"
  output_adapter="de.uni.koeln.spinfo.tesla.annotation.adapter.output.DefaultOutputAdapter"
  id="Senseval_1">
  <shared:role>Sense</shared:role>
  <shared:annotation_type>
    de.uni.koeln.spinfo.tesla.component.wsd.senseval.data.Sense
  </shared:annotation_type>
  <shared:view_adapter
    class="de.uni.koeln.spinfo.tesla.component.wsd.senseval.data.SenseConverter" />
</analysis>
```

Merkmale:

```
<analysis
  input_adapter="de.uni.koeln.spinfo.tesla.annotation.adapter.token.CachingTokenInputAdapter"
  output_adapter="de.uni.koeln.spinfo.tesla.annotation.adapter.output.DefaultOutputAdapter"
  id="Features_1">
  <shared:role>Feature</shared:role>
  <shared:annotation_type>
    de.uni.koeln.spinfo.tesla.component.wsd.features.Feature
  </shared:annotation_type>
  <shared:view_adapter
    class="de.uni.koeln.spinfo.tesla.component.wsd.features.FeatureConverter" />
</analysis>
```

Disambiguierte Lesarten, das Ergebnis der WSD:

```
<analysis
  input_adapter="de.uni.koeln.spinfo.tesla.annotation.adapter.token.CachingTokenInputAdapter"
  output_adapter="de.uni.koeln.spinfo.tesla.annotation.adapter.output.DefaultOutputAdapter"
  id="Classifier_1">
  <shared:role>Class</shared:role>
  <shared:annotation_type>
```

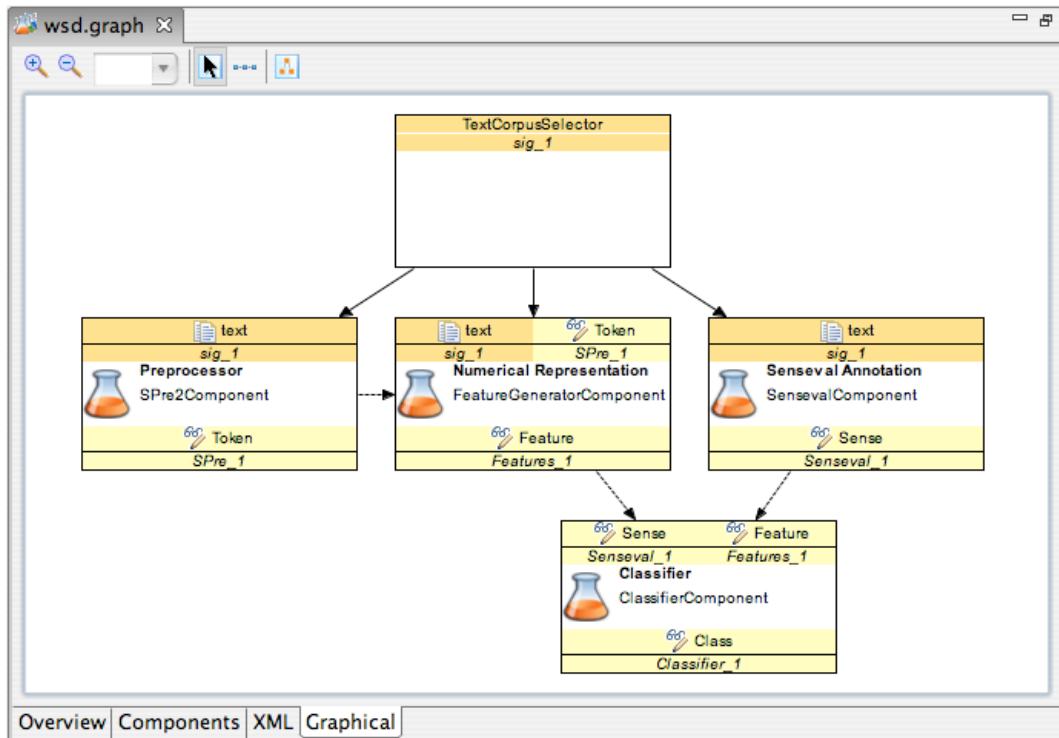


Abbildung A.1.: Konfigurationsdatei im graphischen Editor von Tesla (vgl. Abb. 6.3, S. 58)

```

de.uni.koeln.spinfo.tesla.component.wsd.classifier.Class
</shared:annotation_type>
<shared:view_adapter
  class="de.uni.koeln.spinfo.tesla.component.wsd.classifier.ClassConverter" />
</analysis>

```

Schließlich enthält die Konfigurationsdatei noch Angaben zum Signal, d.h. dem Text, der von den Annotationen ausgezeichnet wird:

```

<signals>
  <shared:signal
    adapter_class="de.uni.koeln.spinfo.tesla.runtime.signal.SignalAdapter"
    type="text" signal_id="sig_1" />
</signals>
</tesla.chain>

```

Die Erstellung der obigen Konfigurationsdatei erfolgt dabei nicht von Hand, sondern im graphischen Editor von Tesla (s. Abb. A.1, S. VIII). Als alternative Form der Formulierung und Darstellung von Experimentenkonfigurationen wurde für Tesla eine *domain-specific language* (DSL) entwickelt (Bilagher 2006), die in einer fertigen Version von Tesla enthalten sein wird.

A.3. Inhalt des beiliegenden Datenträgers

- `arbeit/`: Diese Arbeit, als PDF (`magisterarbeit-fsteeg.pdf`) sowie als TeX-Datei mitsamt Abbildungen (im GraphViz/Dot-Format und als PDFs).
- `code/`: Vollständiger Quelltext des in Kapitel 4 beschriebenen WSD-Verfahrens.
- `eval/`: Ausführbare Jar-Datei zur Durchführung der Experimente aus Kapitel 5 (`sen-seval.jar`), mitsamt Konfigurationsdatei, benötigten Texten und Bibliotheken zur XML-Verarbeitung.
- `tesla/`: Quelltexte und Konfigurationsdateien der Tesla-Komponenten sowie die vollständige Entwicklerversion von Tesla, die für die in Kapitel 6 beschriebene Umsetzung benutzt wurde.

Anhang B.

Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Dasselbe gilt sinngemäß für Tabellen, Karten und Abbildungen.

Köln, den 31. August 2007

.....
(Fabian Steeg)

Literaturverzeichnis

- ABELSON, H., G. J. SUSSMAN & J. SUSSMAN: 2001, *Struktur und Interpretation von Computerprogrammen: Eine Informatik-Einführung (Springer-Lehrbuch)*, Springer.
- AGIRRE, E. & P. EDMONDS: 2006a, 'Introduction', in E. Agirre & P. Edmonds (eds.), *Word Sense Disambiguation: Algorithms and Applications (Text, Speech and Language Technology 33)*, Springer, pp. 1–28.
- AGIRRE, E. & P. EDMONDS (eds.): 2006b, *Word Sense Disambiguation: Algorithms and Applications (Text, Speech and Language Technology 33)*, Springer.
- BENDEN, C. & J. HERMES: 2004, 'Präprozessierung mit Nebenwirkungen: Dynamische Annotation', in E. Buchberger (ed.), *Beiträge zur 7. Konferenz zur Verarbeitung natürlicher Sprache (KONVENS)*, Riegelnik, Wien, pp. 25–28.
- BILAGHER, J.: 2006, *Konfiguration und Steuerung des sprachverarbeitenden Prozesskettensystems TESLA*, Master's thesis, University of Cologne.
- BRÜCKNER, T.: 2001, 'Textklassifikation', in K. U. Carstensen, C. Ebert, E. Endriss, S. Jekat, R. Klabunde & H. Langer (eds.), *Computerlinguistik und Sprachtechnologie*, Spektrum, Heidelberg, Berlin, pp. 442–447.
- BURKE, B. & R. MONSON-HAEFEL: 2006, *Enterprise JavaBeans 3.0 (5th Edition)*, O'Reilly Media, Inc.
- CHIERCHIA, G.: 1999, 'Linguistics and Language', in R. A. Wilson & F. C. Keil (eds.), *The MIT Encyclopedia of the Cognitive Sciences*, A Bradford Book, The MIT Press, Cambridge, Massachusetts and London, England, pp. xci–cix.
- CROCKER, M. W.: 2002, 'Computational Linguistics: Cognitive Science or Language Engineering?', in G. Willée, B. Schröder & H.-C. Schmitz (eds.), *Computerlinguistik: Was geht, was kommt? / Computational Linguistics: Achievements and Perspectives (Sprachwissenschaft, Computerlinguistik und Neue Medien 4)*, gardez!, pp. 48–51.
- CUNNINGHAM, H. & K. BONTCHEVA: 2006, 'Computational Language Systems, Architectures', in K. Brown, A. H. Anderson, L. Bauer, M. Berns, G. Hirst & J. Miller (eds.), *The Encyclopedia of Language and Linguistics*, second edn., Elsevier, München.
- DE SMEDT, K. & J. DE GRAAF: 1990, 'Structured inheritance in frame-based representation of linguistic categories', in W. Daelemans & G. Gazdar (eds.), *Workshop on Inheritance in Natural Language Processing*, ITK, Tilburg, The Netherlands, pp. 39–47.
- EULER, T.: 2001a, 'Informationsextraktion durch gezielte Zusammenfassung von Texten', Universität Dortmund.

- EULER, T.: 2001b, ‘Informationsextraktion durch Zusammenfassung maschinell selektierter Textsegmente’, Universität Dortmund.
- EULER, T.: 2002, ‘Tailoring Text using Topic Words: Selection and Compression’, in *Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA)*, IEEE Computer Society Press.
- FREGE, G.: 1892, ‘Über Sinn und Bedeutung’, in M. Textor (ed.), *Funktion - Begriff - Bedeutung (Sammlung Philosophie 4)*, Vandenhoeck & Ruprecht, Göttingen.
- GARALEVICIUS, S.: 2007, ‘Memory-Prediction Framework for Pattern Recognition: Performance and Suitability of the Bayesian Model of Visual Cortex’, Paper accepted for FLAIRS-20.
- GEERTZEN, J.: 2003, *String Alignment in Grammatical Inference: What Suffix Trees can do*, Master’s thesis, University Tilburg.
- GEORGE, D. & B. JAROS: 2007, ‘The HTM Learning Algorithm’, Numenta Inc. Whitepaper.
- GOERTZEL, B.: 2004, ‘On Biological and Digital Intelligence’, Dynamical Psychology.
- GRISHMAN, R. & B. SUNDHEIM: 1996, ‘Message Understanding Conference: A Brief History’, in *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, Kopenhagen, pp. 466–471.
- GRUHN, V. & A. THIEL: 2000, *Komponentenmodelle. DCOM, Javabeans, Enterprise Java Beans, CORBA*, Addison-Wesley.
- GUSFIELD, D.: 1997, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press.
- HAARMANN, H.: 1991, *Universalgeschichte der Schrift*, second edn., Campus, Frankfurt a. M., New York.
- HARNAD, S.: 2005, ‘Cognition is Categorization’, in C. Lefebvre & H. Cohen (eds.), *Handbook of Categorization*, Elsevier.
- HAWKINS, J.: 2004, *On Intelligence (with Sandra Blakeslee)*, Times Books.
- HAWKINS, J. & D. GEORGE: 2005, ‘A Hierarchical Bayesian Model of Invariant Pattern Recognition in the Visual Cortex’, in *Proceedings of the International Joint Conference on Neural Networks 2004*.
- HAWKINS, J. & D. GEORGE: 2006, ‘Hierarchical Temporal Memory: Concepts, Theory, and Terminology’, Numenta Inc. Whitepaper.
- HERMES, J. & C. BENDEN: 2005, ‘Fusion von Annotation und Präprozessierung als Vorschlag zur Behandlung des Rohtextproblems’, in B. Fisseni, H.-C. Schmitz, B. Schröder & P. Wagner (eds.), *Sprachtechnologie, mobile Kommunikation und linguistische Ressourcen. Beiträge zur GLDV-Tagung 2005 in Bonn (Sprache, Sprechen und Computer 8)*, Lang, Frankfurt a.M., pp. 78–90.

- HERMES, J. & S. SCHWIEBERT: 2007, 'Tesla - Ein Labor für Computerlinguisten', Posterpräsentation und Systemvorführung auf der DGfS-Jahrestagung 2007 in Siegen.
- HUNT, A. & D. THOMAS: 2003, *Der Pragmatische Programmierer*, Hanser, München, Wien.
- IDE, N. & J. VÉRONIS: 1998, 'Word Sense Disambiguation: The State of the Art', *Computational Linguistics* **24**(1).
- IDE, N. & Y. WILKS: 2006, 'Making Sense about Sense', in *Word Sense Disambiguation: Algorithms and Applications (Text, Speech and Language Technology 33)*, Springer, Dordrecht, The Netherlands, pp. 47–74.
- JACKENDOFF, R.: 2002, *Foundations of Language: Brain, Meaning, Grammar, Evolution*, Oxford University Press, New York.
- JORDAN, M. I. & S. RUSSELL: 1999, 'Computational Intelligence', in R. A. Wilson & F. C. Keil (eds.), *The MIT Encyclopedia of the Cognitive Sciences*, A Bradford Book, The MIT Press, Cambridge, Massachusetts and London, England, pp. lxxiiiv–xc.
- KILGARRIFF, A.: 1997, 'What is word sense disambiguation good for?', in *Proceedings of the Natural Language Processing Pacific Rim Symposium. Phuket, Thailand.*, pp. 209–214.
- KILGARRIFF, A.: 2006, 'Word Senses', in *Word Sense Disambiguation: Algorithms and Applications (Text, Speech and Language Technology 33)*, Springer, Dordrecht, The Netherlands, pp. 29–46.
- KIM, J. H. & J. PEARL: 1983, 'A computational model for combined causal and diagnostic reasoning in inference systems', in *Proceedings of the IJCAI-83*, Karlsruhe, Germany, pp. 190–193.
- KISS, T.: 2002, 'Anmerkung zur scheinbaren Konkurrenz von numerischen und symbolischen Verfahren in der Computerlinguistik', in G. Willée, B. Schröder & H. C. Schmitz (eds.), *Computerlinguistik: Was geht, was kommt? / Computational Linguistics: Achievements and Perspectives (Sprachwissenschaft, Computerlinguistik und Neue Medien 4)*, gardez!, pp. 163–171.
- KÖHLER, R.: 2005, 'Korpuslinguistik - zu wissenschaftstheoretischen Grundlagen und methodologischen Perspektiven', *GLDV-Journal for Computational Linguistics and Language Technology* **20**(2), 1–16.
- KOHONEN, T.: 1984, *Self-Organization and Associative Memory*, second edn., Springer.
- LABOV, W.: 1975, *What is a Linguistic Fact?*, Peter de Ridder Press, Lisse.
- LABOV, W.: 1996, 'When Intuitions Fail', in L. McNair (ed.), *Papers from the Parasession on Theory and Data in Linguistics*, CLS 32, University of Chicago, Chicago.

- LÄMMEL, U. & J. CLEVE: 2001, *Lehr- und Übungsbuch Künstliche Intelligenz*, Fachbuchverlag Leipzig.
- LANGACKER, R. W.: 2002, *Concept, Image, and Symbol: the Cognitive Basis of Grammar (Cognitive Linguistic Research 1)*, second edn., Walter de Gruyter.
- LYONS, J.: 1968, *Introduction to Theoretical Linguistics*, University Press, Cambridge.
- MANNING, C. D. & H. SCHÜTZE: 1999, *Foundations of statistical natural language processing*, MIT Press, Cambridge, Mass.
- MARKERT, H., A. KNOBLAUCH & G. PALM: 2005, 'Detecting Sequences and Understanding Language with Neural Associative Memories and Cell Assemblies', in *Biomimetic Neural Learning for Intelligent Robots*, Springer, pp. 107–117.
- MÀRQUEZ, L., G. EXSUDERO, D. MARTÍNEZ & G. RIGAU: 2006, 'Supervised Corpus-Based Methods for WSD', in *Word Sense Disambiguation: Algorithms and Applications (Text, Speech and Language Technology 33)*, Springer, Dordrecht, The Netherlands, pp. 167–216.
- MCENERY, T.: 2003, 'Corpus Linguistics', in R. Mitkov (ed.), *The Oxford Handbook of Computational Linguistics*, Oxford Handbooks in Linguistics, Oxford University Press, pp. 448–463.
- MCENERY, T. & A. WILSON: 1996, *Corpus Linguistics*, Edinburgh University Press.
- MIHALCEA, R., T. CHKLOVSKI & A. KILGARRIFF: 2004, 'The Senseval-3 English Lexical Sample Task', in *Proceedings of Senseval-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text, Barcelona, Spain, July 2004*, Association for Computational Linguistics (ACL).
- MOUNTCASTLE, V.: 1978, 'An organizing principle for cerebral function: the unit model and the distributed system', in G. Edelman & V. Mountcastle (eds.), *The Mindful Brain*, MIT Press, Cambridge, Mass.
- NEUMANN, G.: 2001, 'Informationsextraktion', in K. U. Carstensen, C. Ebert, E. Endriss, S. Jekat, R. Klabunde & H. Langer (eds.), *Computerlinguistik und Sprachtechnologie*, Spektrum, Heidelberg, Berlin, pp. 448–456.
- OESTEREICH, B.: 2005, *Die UML 2.0 Kurzreferenz für die Praxis*, fourth edn., Oldenbourg, München, Wien.
- PALMER, M., H. T. NG & H. T. DANG: 2006, 'Evaluation of WSD Systems', in *Word Sense Disambiguation: Algorithms and Applications (Text, Speech and Language Technology 33)*, Springer, Dordrecht, The Netherlands, pp. 75–106.
- PEARL, J.: 1982, 'Reverend Bayes on inference engines: A distributed hierarchical approach', in *Proceedings of the American Association of Artificial Intelligence National Conference on AI*, Pittsburgh, PA, pp. 133–136.

- PEARL, J.: 1988, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann.
- PEDERSEN, T.: 2006, 'Unsupervised Corpus-Based Methods for WSD', in *Word Sense Disambiguation: Algorithms and Applications (Text, Speech and Language Technology 33)*, Springer, Dordrecht, The Netherlands, pp. 133–166.
- RESNIK, P.: 2006, 'WSD in NLP Applications', in *Word Sense Disambiguation: Algorithms and Applications (Text, Speech and Language Technology 33)*, Springer, Dordrecht, The Netherlands, pp. 299–338.
- ROLSHOVEN, J.: 2007, 'Licensing Strategies in Natural Language Processing', in A. Mehler & R. Köhler (eds.), *Aspects of Automatic Text Analysis (Studies in Fuzziness and Soft Computing 209)*, Springer, Berlin, Heidelberg.
- SAEED, J. I.: 2003, *Semantics (Introducing Linguistics 2)*, second edn., Blackwell, Malden, Oxford, Carlton.
- SCHADE, U. & H. J. EIKMEYER: 2002, 'Computerlinguistik in der Kognitionswissenschaft', in G. Willée, B. Schröder & H. C. Schmitz (eds.), *Computerlinguistik: Was geht, was kommt? / Computational Linguistics: Achievements and Perspectives (Sprachwissenschaft, Computerlinguistik und Neue Medien 4)*, gardez!, pp. 247–250.
- SCHIEHLEN, M. & R. KLABUNDE: 2001, 'Semantik', in K. U. Carstensen, C. Ebert, E. Endriss, S. Jekat, R. Klabunde & H. Langer (eds.), *Computerlinguistik und Sprachtechnologie*, Spektrum, Heidelberg, Berlin, pp. 246–304.
- SCHWARZ, M.: 1996, *Einführung in die Kognitive Linguistik*, second edn., A. Francke, Tübingen, Basel.
- SCHWIEBERT, S.: 2005, 'Entwicklung eines agentengestützten Systems zur Paradigmenbildung', in B. Fisseni, C. B. Schröder & P. Wagner (eds.), *Sprachtechnologie, mobile Kommunikation und linguistische Ressourcen. Beiträge zur GLDV-Tagung 2005 in Bonn (Sprache, Sprechen und Computer 8)*, Lang, Frankfurt a.M., pp. 633–646.
- SCHWIEBERT, S. & J. ROLSHOVEN: 2006, 'SOG: Ein selbstorganisierender Graph zur Bildung von Paradigmen', in Rapp, Reinhard, Sedlmeier & Zunker-Rapp (eds.), *Perspectives on Cognition*, Pabst Science Publishers, Lengerich.
- SINGER, W.: 2002, *Der Beobachter im Gehirn: Essays zur Hirnforschung*, Suhrkamp.
- STEVENSON, M.: 2003, *Word Sense Disambiguation. The Case for Combinations of Knowledge Sources*, CSLI Studies in Computational Linguistics, CSLI Publications.
- STRUBE, G. (ed.): 2001, *Wörterbuch der Kognitionswissenschaft*, Klett-Cotta-Verlag.
- THORNTON, J. R., T. GUSTAFSSON, M. BLUMENSTEIN & T. HINE: 2006, 'Robust Character Recognition using a Hierarchical Bayesian Network', in *Proceedings of the 19th Australian Joint Conference on Artificial Intelligence, AI-2006, Hobart*, pp. 1259–1264.

- VAITHYANATHAN, S., J. C. MAO & B. DOM: 2000, 'Hierarchical Bayes for Text Classification', in *Proceedings of the PRICAI Workshop on Text and Web Mining*, pp. 36–43.
- VAN ROY, P. & S. HARIDI: 2004, *Concepts, Techniques, and Models of Computer Programming*, The MIT Press, Cambridge, Mass.
- VATER, H.: 1999, *Einführung in die Sprachwissenschaft*, third edn., UTB, W. Fink, München.
- VON CUBE, F.: 1965, *Kybernetische Grundlagen des Lernens und Lehrens*, Klett.
- WILKS, Y. A., B. M. SLATOR & L. GUTHRIE: 1996, *Electric Words: Dictionaries, Computers, and Meanings*, ACL-MIT Series in Natural Language Processing, The MIT Press, Cambridge, Mass.
- WILSON, R. A. & F. C. KEIL: 1999, *The MIT Encyclopedia of the Cognitive Sciences (MITECS)*, Bradford Books, The MIT Press, Cambridge, Mass.
- WITTGENSTEIN, L.: 1953, *Philosophische Untersuchungen (Auf Grundlage der Kritisch-genetischen Edition, neu herausgegeben von Joachim Schulte, 2003, Bibliothek Suhrkamp 1372)*, Suhrkamp, Frankfurt a. M.

Index

- Abstraktion
 - als Prinzip der Kognition, 11
 - von Mustern und Zahlen, 27
 - von sprachlichen Symbolen, 21
 - Wortsinn als Abstraktion, 4
- Ambiguität, 2
 - vers. Arten von A., 6
- Annotation
 - in einer SALE, 51
 - Semantische Annotation, 53
 - von Korpora, 5
- Bäume
 - bei der Rekursion, 28
 - Suffixbäume, 24
 - zur Belief Propagation, 36
- Bayes, 12, 32
- Bedeutung, 3, 5, 6, 21, 41
- Clustering, 17, 62
- Evaluierung
 - Kriterien zur E., 40
 - mit Korpora, 39
- Homonymie, 6, 8
- Klassifikation
 - Überwachtes Lernen, 17
 - als Prinzip der Kog., 10, 12
 - für vers. Aufgaben, 18, 61
- Kognition, 10
 - Computerling. als Kognitionswiss., 9
 - Einheitl. Algorithmus d. K., 9
 - Modell. von K. durch masch. Lernen, 59
 - WSD als kognitives Problem, 2
- Komponenten, 52
 - Austauschbarkeit und Wiederverwertbarkeit, 60
- SALE, 52
- Korpora, 21
 - Lernen aus K., 5, 15
 - zur Evaluierung, 39
- Lexical Aquisition Bottleneck, 17, 61
- Lexikon, 20, 24, 32
- Merkmale, 19, 61
 - Merkmalsberechnung, 21
 - Merkmalsvektoren, 19
- Metaprogrammierung, 54
- Paradigmen, 24, 62
- Parallelisierung, 37
- Permutationen, 28
- Polysemie, 6, 8
- Semantik
 - Semantische Relationen, 20, 24, 41
 - Sinn und Bedeutung, 3
- Service, 52
- Sinn
 - als Abstraktion, 4
 - bei Frege, 3
 - bei Wittgenstein, 4, 5
- Sprachunabhängigkeit, 21
- Tesla, VIII, 53
- Vagheit, 6
- Wahrscheinlichkeit
 - Belief Propagation, 13, 34
 - zur Modellierung d. Kog., 12
- Wort, 2
- Wortlisten, 54
- XML
 - für Korpora, 56
 - zur Konfiguration, V