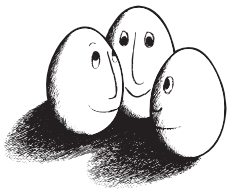
**Diplomarbeit**

# Relation Extraction zur Ergänzung deutschsprachiger Firmendossiers

Martin Had



Diplomarbeit
Fakultät für Informatik
Technische Universität Dortmund

2. März 2009

**Betreuer:**

Prof. Dr. Katharina Morik
Dipl.-Inform. Felix Jungermann

# Danksagung

Mein besonderer Dank gilt meinen Diplombetreuern Prof. Dr. Katharina Morik und Dipl.-Inf. Felix Jungermann, die mich beim Verfassen dieser Diplomarbeit mit vielen guten Ratschlägen und Ideen unterstützt haben. Außerdem danke ich ganz herzlich meinen Korrekturlesern.

# Inhaltsverzeichnis

# 1. Einleitung

Eine Vielzahl von Informationen steht heute im Internet bereit. Täglich kommen neue Nachrichten, Veröffentlichungen und Berichte hinzu. Firmen sind daran interessiert, einen möglichst umfassenden Überblick über die Aktivitäten potentieller Kooperationspartner oder Konkurrenten zu gewinnen. Für einen Menschen ist die Recherche sehr zeitintensiv, Suchmaschinen liefern oft eine lange Liste mit URLs. Diese verweisen auf Seiten, die manuell gelesen und bewertet werden müssen. Dabei ist es schwer, aus den oft zahlreichen Ergebnissen eine sinnvolle Auswahl zu treffen und die Informationen zu erhalten, die wirklich von Bedeutung sind. Wenn es darum geht, Informationen über Firmen und Konzerne zu gewinnen, bietet das Internet vielfältige Möglichkeiten. Eine gute Quelle für wichtige Unternehmensinformationen stellen z. B. die Online-Archive großer Presseanbieter dar. Aber auch diese bieten keinen gut sortierten Überblick und erlauben meist nur eine Suche nach einfachen Schlagworten. Das Ziel ist also, den menschlichen Nutzer bei der Erschließung der verstreuten Fakten innerhalb des unüberschaubar großen World Wide Web zu unterstützen und ihm das Auffinden der gesuchten Information dadurch zu erleichtern. Um dies zu erreichen, müssen Softwaresysteme geschaffen werden, die den Großteil der Arbeit, basierend auf automatischen Lernverfahren, übernehmen können. Da viele der Ursprungsdaten in unstrukturierter natürlicher Sprache vorliegen, also für Menschen lesbar und nicht mit Metadaten versehen sind, hat ein Computer jedoch keine Möglichkeit, diese zu verstehen. Es müssen daher Methoden der *Künstlichen Intelligenz*, genauer des *Natural Languange Processing (NLP)* angewendet werden, um die Texte zu bearbeiten und in eine automatisch auswertbare Form zu bringen.

Die vorliegende Arbeit befasst sich mit der automatischen Extraktion von Relationen zwischen Wirtschaftsunternehmen. Als Datenquelle dient das Internet. Die gefundenen Informationen über Beziehungen zwischen verschiedenen Firmen sollen dem Benutzer in einer übersichtlichen, klar strukturierten Form präsentiert werden. Zu diesem Zweck werden die Daten in Form einer Netzwerkstruktur abgebildet und diese über ein graphisches Interface zugänglich gemacht. Die Anreicherung dieser Netzwerkstruktur um neue Informationen ist für einen Menschen eine mühsame Aufgabe. Sie macht es erforderlich, große Mengen von Texten zu lesen und zu bewerten und die gefundenen Resultate in die Datenbank einzutragen. Herkömmliche Suchmaschinen sind nicht in der Lage, Anfragen zu beantworten wie *„Zeige Konzerne, mit denen Siemens fusioniert hat“*. Mit Hilfe

der *Relation Extraction* kann diese Aufgabe durch automatische Systeme übernommen werden. Auf diese Weise kann ein System selbstständig und gezielt neue Informationen im Internet suchen und diese in das Netzwerk integrieren. Um dieses Ziel zu ereichen, müssen zahlreiche Voraussetzungen geschaffen werden. Daten müssen aus dem Internet gesammelt und abgespeichert werden. Sie müssen anschließend analysiert und in eine geeignete Form gebracht werden, die es einem Lernverfahren erlaubt, Relationen darin zu erkennen. Ein passendes Lernverfahren muss gefunden, implementiert und getestet werden. Abschließend soll ein System entworfen werden, in dem die gefundenen Relationen in einer graphischen Umgebung als Netzwerk – in Verbindung mit weiteren Informationen über die beteiligten Firmen – dargestellt werden können.

# 2. Information Extraction

*Information Retrieval (IR)* beschäftigt sich mit der sehr allgemein gefassten Aufgabe, in großen Datenmengen enthaltene Informationen aufzuspüren. Die *Information Extraction (IE)* lässt sich als Teilbereich des IR betrachten. Ihr Ziel ist es, in Texten gezielt Informationen zu einem fest definierten Themenbereich aufzufinden. Von Interesse ist beispielsweise die Untersuchung großer unstrukturierter Dokumentmengen wie dem Internet. Durch Verfahren der Information Extraction sollen darin enthaltene Fakten aus einem Bereich gefunden und abgespeichert werden. Möchte man die Beziehungen zwischen Wirtschaftsunternehmen betrachten, so kann ein solches Fakt beispielsweise die Information über den Zusammenschluss der Unternehmen **Mars** und **Wrigley** sein. Diese soll aus frei formulierten Sätzen wie ***Mars*** *und der Kaugummihersteller* ***Wrigley*** *fusionieren zum größten Süßwarenhersteller der Welt.* entnommen werden.
Erworbenes Wissen über Fakten wird in eine strukturierte Form überführt und z. B. in einer Datenbank abgelegt. Für den genannten Beispielsatz werden nur die beteiligten Firmen sowie die Beziehung, die zwischen ihnen besteht, abgespeichert. Das Softwaresystem formt den natürlichsprachlichen Satz in die eindeutig definierte Relation *fusion(Mars,Wrigley)* um. Während der unstrukturierte Text vorher nur eine einfache Schlüsselwortsuche erlaubte, ermöglicht diese Repräsentation die Beantwortung von Fragen, automatische Auswertungen und vieles mehr.

Grishman (Gri97) beschreibt die Information Extraction auf folgende Weise: „IE ist im Vergleich zu dem Versuch, einen Text vollständig zu verstehen, deutlich eingeschränkt. Beim vollständigen Textverständnis geht es darum, alle enthaltenen Informationen zu erkennen und abzuspeichern. Bei der IE kommt es nur auf bestimmte, in der Aufgabenstellung vordefinierte Teilinformationen an.“
Im Laufe der Zeit sind die Aufgabenstellungen jedoch immer komplexer geworden, sodass immer mehr Schritte durchgeführt und zahlreiche Teilaufgaben gelöst werden müssen. Während es anfangs nur darum ging, einer kleinen Menge homogener Texte aus einem festgelegten Bereich bestimmte, vordefinierte Informationen zu entnehmen, versuchen heutige Systeme, das komplette Internet als Datenquelle zu nutzen. Basierend auf diesen Daten soll eine Software z. B. in der Lage sein, komplexe Fragen zu einem Themenbereich beantworten zu können. Dies setzt eine umfangreiche Vorverarbeitung und eine genaue Analyse des Datenmaterials voraus.

## 2.1. Entwicklung der IE

Die Idee, Texte in eine tabellarische Struktur zu überführen, wurde von Zellig S. Harris bereits um 1950 vorgeschlagen. Jedoch erst mit der Einführung der *MUC*-Konferenzen im Jahre 1987 kam dem Bereich IE deutlich größere Bedeutung zu. Insgesamt fanden sieben Veranstaltungen in den Jahren 1987-1998 statt. Finanziert von der *DARPA* (Defense Advanced Research Projects Agency) ging es zunächst darum, an militärischen Nachrichten in Textform automatische Analysen durchzuführen. Dabei standen nicht die eigentlichen Konferenzen im Vordergrund, sondern die im Vorfeld stattfindende Forschungsarbeit mit Wettbewerbscharakter (GS96). Jedem Teilnehmer wurde eine Menge von Beispieldokumenten zur Verfügung gestellt, die manuell von Experten mit Zusatzinformationen versehen worden waren. In der Aufgabenstellung war festgelegt, was aus den Dokumenten extrahiert werden sollte. Mit Hilfe dieses Datensatzes sollte jede Gruppe ein dafür speziell angepasstes System entwickeln. Kurz vor der Konferenz wurde ein weiterer Datensatz mit Testdaten herausgegeben, den die entwickelten Softwaresysteme nun ohne weitere Veränderungen verarbeiten sollten. Die Ergebnisse wurden danach mit manuell erstellten Lösungen verglichen. Auf diese Weise konnte die Effizienz der verschiedenen Wettbewerber miteinander verglichen werden.

Sowohl das Interesse der DARPA als auch die guten Fortschritte in der IE nach der *MUC-2* hatten zur Folge, dass in dieser Zeit viele großangelegte Forschungsprojekte ins Leben gerufen wurden. Das Projekt *Tipster* (Gee96) beispielsweise war in den Jahren 1991 bis 1998 ein umfassendes Programm zur Erforschung und Entwicklung in den Bereichen IR, IE und Textzusammenfassung, das von der DARPA und diversen anderen Bundesbehörden finanziert wurde. Dabei wurde erstmals eine Standardarchitektur geschaffen, die portabel und rekonfigurierbar war. Diese Architektur wurde in vielen der damaligen Systeme implementiert. Von 1999 bis 2008 wurden in jährlichem Abstand die *ACE*-Konferenzen abgehalten. Organisiert vom *National Institute of Standards and Technology* (NIST) führten sie die Idee der *MUC* weiter. Ziel des *ACE*-Programms ist die Entwicklung von Softwaresystemen, die Inhalte aus Texten in menschlicher Sprache extrahieren können. Diese haben ihren Ursprung in unterschiedlichen Quellen wie Nachrichtenbeiträgen, Weblogs etc. Es sollen neue und effiziente Technologien entwickelt werden, die diese Aufgaben automatisch durchführen können, und Verfahren, die deren Ergebnisse bewerten. Es wurden mehrere aufeinander aufbauende Phasen definiert; die Komplexität der gestellten Aufgaben steigerte sich von Phase zu Phase. Die Forschung sollte dabei zahlreiche Klassifizierungs-, Filter- und Extraktionsaufgaben untersuchen. Im Jahr 2009 wird die *ACE*-Konferenz eingegliedert in die *Text Analysis Conference* (TAC).

## 2.2. Techniken und Aufgabenstellungen

Ein System, das in der Lage ist, einem natürlichsprachlichen Text Informationen zu entnehmen, erfordert viele Verarbeitungsschritte. Oft bauen die genutzen Verfahren zur Lösung von Teilaufgaben aufeinander auf. Die Qualität der Ergebnisse eines Schrittes ist dadurch von den Ausgaben seines Vorgängers abhängig. Aus diesem Grund kann die Verbesserung einzelner kleiner Teilschritte die Performace eines Gesamtsystems stark beeinflussen. Je komplexer eine Aufgabe gestellt ist, desto mehr Verarbeitungsschritte sind absolut notwendig, um sie lösen zu können. Die gewonnenen Erkenntnisse müssen zusammen mit dem Text in geeigneter Form gespeichert werden, um eine Weiterverarbeitung durch automatische Systeme zu ermöglichen oder sie einem menschlichen Benutzer auf übersichtliche Weise präsentieren zu können.

### 2.2.1. Grundlegende Techniken

Die wichtigen grundlegenden Verabeitungsschritte sollen an dieser Stelle kurz vorgestellt werden. In Kapitel 3 werden die Verfahren dann im Detail erläutert werden.

#### Tokenisierung:

Bei der Tokenisierung wird ein Text in seine Bestandteile zerlegt. Im Normalfall findet zunächst eine Zerlegung in Sätze statt. Anschließend werden diese noch einmal in einzelne Tokens aufgespalten. Tokens sind elementare Bausteine des Satzes, in der Regel also die Wörter sowie die Satzzeichen.

#### Morphologische Analyse:

Die morphologische Analyse beschäftigt sich mit der Struktur der Wörter. Auf diese Weise können Informationen über das Wort, z. B. die Stammform, Tempusform etc. gefunden werden.

#### Part-Of-Speech-Tagging (POS):

Beim Part-of-Speech-Tagging wird jedem Wort eine Wortart wie Nomen, Verb etc. zugeordnet. Dabei wird die Entscheidung sowohl aufgrund des Kontextes als auch aufgrund der Definition des Wortes getroffen.

#### Named Entity Recognition (NER):

Die Named Entity Recognition hat zum Ziel, Entities, also festgelegte atomare Elemente, in einem Text zu markieren. Hierbei werden Kategorien wie *Person*, *Firma*, *Ort* sowie *Zeitangaben*, *Währungsmengen* etc. definiert.

#### Parsing:

Unter Parsing eines Textes versteht man den Versuch, die vollständige grammatische Struktur einer Sequenz von Wörtern zu erkennen. Diese Struktur ist nicht unbedingt eindeutig, was das Parsen durch automatische Systeme sehr erschwert. Zugrunde gelegt wird häufig eine probabilistische kontextfreie Grammatik (*probabilistic context-free grammar, PCFG*). Parsing ist ein zeitintensiver Vorgang. Um den Aufwand und Zeitbedarf zu reduzieren, werden daher nach Möglichkeit Methoden eingesetzt, die zwar eine geringere Menge von Informationen liefern, die dafür aber schneller und genauer durchführbar sind. Ein Ansatz dafür ist das *Shallow Parsing*, bei dem nur Grenzen von größeren Bestandteilen wie z. B. Nominalphrasen und Verbalphrasen erfasst werden.

### 2.2.2. Komplexe Aufgabenstellungen

Die folgenden Aufgabenstellungen sind deutlich komplexer und bestehen oft aus mehreren Teilen. Sie basieren auf den in den grundlegenden Vorverarbeitungsschritten gefundenen Informationen im Text. Durch deren Kombination lässt sich neues Wissen ableiten und verknüpfen. Dadurch kann das Verständniss der Computersysteme für die Inhalte, die in übergreifenden Strukturen wie Sätzen, Dokumenten oder Dokumentsammlungen enthalten sind, schrittweise erweitert werden.

#### Koreferenz-Erkennung (Coreference Analysis)

Es soll erkannt werden, wenn dieselbe Entität in einem Text mehrfach erwähnt wird. Die Erwähnung kann dabei unter Verwendung des gleichen Wortes oder eines Ersatzwortes erfolgen. Im dem Beispiel *Herr Meier arbeitet bei Volkswagen. Er fährt einen Golf.* bezieht sich *„Er“* zu Beginn des zweiten Satzes auf *„Herr Meier“* und beschreibt somit die gleiche Entität. Diese Erkennung kann sowohl nur innerhalb eines Satzes als auch über Satzgrenzen hinweg für ganze Dokumentsammlungen erfolgen.

#### Relationserkennung (Relation Detection and Extraction)

Eine Relation ist eine Beziehung, die zwischen zwei Entities besteht. Die Relation *EMP-ORG* z. B. beschreibt den Fall, dass ein *Mitarbeiter* (EMP) bei einer *Firma* (ORG) beschäftigt ist.

#### Ereignis-Erkennung (Event Extraction)

Die Ereignis-Erkennung ist definiert als die Aufgabe, zu einem Ereignis eine Menge definierter Eckdaten herauszufinden. Ein Ereignis wird definiert als eine Sammlung freier Slots, die gefüllt werden sollen. Zu dem Ereignis *Raketenstart* z. B. könnend die Slots

*Startzeit*, *Startort*, *Name der Rakete* gehören. Welche Slots zu einem Ereignis zu füllen sind, wird bei der Aufgabenstellung festgelegt.

#### Beantwortung von Fragen (Question Answering (QA))

Bei der Fragenbeantwortung werden große Sammlungen von Dokumenten mit dem Ziel durchsucht, eine präzise Antwort auf eine spezifische (natürlichsprachliche) Fragestellung zu erhalten. Die Antwort auf die Frage *Welche Länder planen, neue Atomkraftwerke zu bauen?* wäre eine Liste der entsprechenden Länder. Als Verschärfung der Aufgabe können auch Fragen gestellt werden wie *Warum wollen Länder neue Atomkraftwerke bauen?*. Die Antwort darauf soll ein Satz oder ein kurzer Text mit einer korrekten Begründung sein.

#### Folgern aus Texten (Recognizing Textual Entailment (RTE))

Es sollen Systeme entworfen werden, die überprüfen, ob ein Textteil einen anderen bedingt bzw. den darin enthaltenen Inhalt auf logische Weise stützt. Beispiel: *Die Polizei erschoss den Bankräuber. Der Bankräuber ist tot.* Es werden drei Klassen unterschieden. Diese sind wie folgt definiert: *ENTAILMENT*(Unterstüzung), *CONTRADICTION*(Widerspruch) und *UNKNOWN*(Unbekannt).

#### Zusammenfassung von Texten (Summarization)

Bei der Zusammenfassung sollen ein oder mehrere Texte in eine kurze, schlüssige Zusammenfassung gebracht werden. Der zusammengefasste Text soll deutlich gekürzt sein, die wichtigen Inhalte enthalten und in einer für Menschen verständlichen und angenehm lesbaren Form vorliegen.

#### Anreicherung von Wissensbasen (Knowledge Base Population (KBP))

KBP ist eine stark erweiterte Form der Event Extraction. Es soll eine umfassende Ontologie mit Wissen über Entitäten und deren Beziehungen untereinander aufgebaut werden. Die Grundlagen dafür sollen aus großen Mengen freier Texten extrahiert werden. Im Unterschied zur Fragenbeantwortung, wo nur eine textuelle Information zurückgeben wird, werden Verbindungungen zwischen Entitätsobjekten erkannt und dargestellt. Neues Wissen kann dem Netzwerk hinzugefügt und altes Wissen aktualisiert werden.

# 3. Vorverarbeitungsschritte für IE

In diesem Kapitel wird beschrieben, was die natürliche Sprache ausmacht und welche Vorbereitungen getroffen werden müssen, um mit einem Computersystem erfolgreich Texte untersuchen zu können, die in natürlicher Sprache verfasst sind. Dazu wird zunächst eine kurzer Blick in den Bereich der Linguistik geworfen, danach werden verschiedene grundlegende Schritte betrachtet, die zur Vorbereitung auf die komplexeren Aufgaben der Informationsextraktion hilfreich sein können.

## 3.1. Natürliche Sprache

Natürliche Sprache wird von Menschen zur alltäglichen Kommunikation genutzt. Im Gegensatz zu künstlichen Sprachen wie z. B. Programmiersprachen, die in ihrer Struktur eindeutige Konstrukte bilden müssen, bietet sie zahlreiche Freiheiten und Möglichkeiten zur Interpretation. Die Erforschung der natürlichen Sprache fällt in den wissenschaftlichen Bereich der Linguisik. Diese wird unterteilt in zwei wichtige Themenbereiche: die Erforschung der Struktur, d. h. der Grammatik, und die Suche nach der Bedeutung, also der Semantik.

Die *Grammatik* einer Sprache umfasst mehrere voneinander abhängige Ebenen. Auf Wortebene betrachtet man die Morphologie, d. h. die Struktur der Wörter. Zerlegt man ein Wort in seine bedeutungstragenden Elemente, so erhält man seine Morpheme. Die Syntax beschreibt die Regeln und Vorgehensweisen, wie Wörter zu vollständigen Sätzen kombiniert werden können (Blo33).

Bei der *Semantik* geht es darum, die Bedeutung zu verstehen. Auch hier unterscheidet man zwischen der Bedeutung eines Wortes, eines Satzes oder sogar ganzer Texte. Semantik ist häufig abhängig vom Kontext und benötigt unter Umständen große Mengen an Hintergrundwissen.

Verschiedene natürliche Sprachen haben unterschiedliche Eigenschaften und erfordern zu ihrer Verarbeitung unterschiedlich angepasste Systeme. Ein Programm, welches in der Lage ist, aus einem englischen Text Informationen zu extrahieren, wird dieses nicht ohne weiteres bei einem deutschen oder gar chinesischen Text können. Dies ist durch

generelle strukturelle Unterschiede im Aufbau der Sprachen bedingt.
Dass eine Sprache wie die chinesische Besonderheiten im Vergleich zu anderen Sprachen hat, ist leicht einzusehen. Die chinesische Schriftsprache besteht aus mehreren tausend komplexen Symbolen. Diese entsprechen in der Regel Silben bzw. Bedeutungseinheiten. Durch Kombination können Aussagen und Wörter gebildet werden. Die genaue Bedeutung ergibt sich erst durch die Kombination der Zeichen und deren Stellung zueinander. Feste Wortgrenzen lassen sich hier nicht anhand eines Trennsymbols erkennen.
Aber auch germanische Sprachen, die einen gemeinsamen Ursprung haben und somit eng miteinander verwandt sind, weisen untereinander starke Unterschiede auf. Vertreter der germanischen Sprachen sind das Englische und das Deutsche. Die Struktur der englischen Sprache weist eine feste Wortstellung im Satz auf: Subjekt, Verb, Objekt. Im Deutschen hingegen kann die Wortstellung auch in einer anderen Reihenfolge erfolgen, ohne den Sinn eines Satzes dabei zu verändern: *Peter kauft einen Fernseher → Einen Fernseher kauft Peter*. Versucht man die gleiche Umstellung für den englischen Satz wird der Sinn nicht mehr korrekt wiedergegeben: *Peter buys a TV → A TV buys Peter*. Flexionen, d. h. Veränderungen des Wortes aufgrund seiner Funktion im Satz sind in der englischen Sprache weitestgehend nicht mehr vorhanden (*ein → einen*).
Auch die im Deutschen gegebene Möglichkeit, Wörter direkt zu verbinden und damit ein neues gültiges Wort zu erhalten macht einen Unterschied zu anderen Sprachen: *eiskalt, sonnengebräunt, Türklingelschildbefestigung, Buchdeckelbeschriftung.*
Neben den genannten existieren noch zahlreiche weitere Besonderheiten. Es ist daher erforderlich, die angewendeten Methoden für jede Sprache einzeln zu überprüfen und gegebenenfalls anzupassen. Dubey (Dub06) untersucht z. B. die Rolle, die linguistische Unterschiede zwischen Deutsch und Englisch beim statistischen Parsing spielen.

## 3.2. Korpora und Korpuserstellung

Korpora sind Sammlungen von Texten, die in elektronisch verwertbarer Form vorliegen. Sie können ein- oder mehrsprachig sein, sich nur auf bestimmte Themenbereiche beziehen oder allgemeiner Natur sein, in Schriftform oder als Tondokument vorliegen etc. Sie werden zur Erforschung der Sprache mit Computerhilfe verwendet.

### 3.2.1. Erstellung von Korpora

Die Erstellung guter Korpora ist zeitaufwändig und erfordert eine durchdachte Vorgehensweise. Es kommt zum einen auf eine representative Zusammenstellung der Dokumente an, zum anderen erfordert die passende Vorverarbeitung viel Sorgfalt und Zeit sowie teilweise Expertenwissen. Einen guten allgemeinen Überblick über die Erfordernisse und Vorgehensweisen bietet (Wyn05). Meistens sind zusätzlich zu den rein inhaltlichen Daten

auch Metainformationen eingebettet. Diese können sehr unterschiedlich sein und richten sich danach, für welche Aufgabenstellungen das Korpus konzipiert wurde. Beispielsweise sind häufig Wörter mit ihrer Wortklasse oder ihrer Stammform versehen, Sätze sogar mit der kompletten grammatischen Struktur. Weiterhin können Named Entites gekennzeichnet sein, Relationen und Abhängigkeiten und vieles mehr.

### 3.2.2. Nutzung von Korpora

Im Bereich der Computerlinguistik möchte man häufig Modelle anhand von vorhandenem und bereits passend verarbeitetem Textmaterial trainieren und überprüfen. Das Material verfügt im optimalen Fall bereits über alle Informationen, die ein System später unbekannten Texten entnehmen soll. Im Trainingsfall stehen diese dem Programm sichtbar zur Verfügung. Das System kann daraus dann z. B. die nötigen Muster oder statistische Wahrscheinlichkeiten lernen, die notwendig sind, um später seine Aufgabe erfolgreich erfüllen zu können. Weiterhin bietet sich die Möglichkeit, ein Maß für die Erkennungsrate des Systems festzustellen. Dabei wird ein Testlauf über das Korpus gemacht, bei dem die zu findenden Informationen für das Programm verdeckt sind. Anschließend wird die Abweichung zwischen den erzielten Ergebnissen und den in den Beispielen vorhandenen Annotationen festgestellt. Es ist üblich, die Systeme unterschiedlicher Entwickler an einem einheitlichen Korpus zu messen, um diese miteinander vergleichen zu können. Ein Beispiel dafür sind die zuvor schon erwähnten themenspezifischen Korpora der *MUC*-Konferenzen.

### 3.2.3. Bekannte deutsche Korpora

Für die deutsche Sprache bedeutsam sind z. B. das *NEGRA*-Korpus[1] bzw. als Erweiterung davon die *TIGER Treebank* (DBL$^+$), die auch syntaktische Informationen bereitstellt. Die TIGER Treebank enthält 900 000 Token in 50 000 Sätzen aus deutschen Zeitungstexten. Diese sind mit POS-Tags, morphologischen und Lemma-Informationen sowie syntaktischen Strukturen annotiert.
Die *„Tübinger Baumbank des Deutschen / Schriftsprache“* (TüBa-D/Z) [2] enthält ca. 36 000 Sätze bzw. 640 000 Worte und wird auf Grundlage der deutschen Zeitung „die tageszeitung“ (taz) aufgebaut. Der Datensatz ist von Hand annotiert und enthält Informationen auf vier Ebenen (lexikalisch, phrasal, topologisch, Satzebene). Die Annotation der Baumbank wird stetig erweitert und weitere Versionen mit einem größeren Datenumfang sind geplant.
Das größte ausgewogenen Textkorpus der deutschen Sprache des 20. Jahrhunderts stellt derzeit das *„Digitale Wörterbuch der deutschen Sprache“* bereit (GK01). Das Kernkor-

[1] `http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/negra-corpus.html`
[2] `http://www.sfs.uni-tuebingen.de/de_tuebadz.shtml`

pus umfasst 100 Mio. Textwörter in 79 830 Dokumenten. Des Weiteren stehen dort auch mehrere Zeitungskorpora zur Verfügung, beispielsweise das „ZEIT“-Korpus mit allen ZEIT-Ausgaben von 1996-2008. Er besteht aus 106 Mio. Textwörtern (tokens) in mehr als 200 000 Artikeln und wird täglich aktualisiert. Alle Korpora sind lemmatisiert, mit Wortartinformationen versehen und mit einer linguistischen Suchmaschine online abfragbar. Das Projekt existiert seit 2000 und wird gefördert von der Deutschen Forschungsgemeinschaft.

## 3.3. Textrepräsentation und Verarbeitungsschritte

Im folgenden Abschnitt soll beschrieben werden, welche Schritte der Umgang mit Texten zur Information Extraction erforderlich macht. Die Verarbeitungskette beginnt damit, einen Text in ein einheitliches Format zu konvertieren. Anschließend erfolgt eine Zerlegung der Dokumente in Teilstrukturen. In den späteren Verarbeitungsschritten werden die Teilstrukturen betrachtet und um zusätzliche Merkmale angereichert. Zahlreiche Verfahren stehen dabei zur Verfügung und für jede Aufgabenstellung muss eine geeignete Auswahl getroffen werden, welche Verarbeitungsschritte durchgeführt werden müssen. Die Schritte sind unterschiedlich komplex. Einige sind sehr schnell und ohne großes Hintergrundwissen durchführbar, andere setzen sprachspezifisches Wissen voraus und erfordern zeitintensive Berechnungen.

### 3.3.1. Textrepräsentation und Zerlegung

Das erste Ziel, das durch die Vorverarbeitung von Dokumenten erreicht werden soll, ist, diese so in eine einheitliche Form zu bringen, dass sie mit Hilfe der gleichen Werkzeuge und durch die gleichen Arbeitsabläufe untersucht werden können. Die anschließende Zerlegung in Teilstrukturen schafft die Grundlagen für nachfolgende Verarbeitungsschritte, die nicht mehr den Text im Gesamten betrachten, sondern gezielt Informationen über einzelne Elemente gewinnen wollen.

#### Struktureller Aufbau von Dokumenten

Man unterscheidet zwischen strukturierten, halbstrukturierten und unstrukturierten Dokumenten. Je festgelegter und umfangreicher die Regeln sind, nach denen Inhalte einer Quelle aufgebaut sind, desto einfacher ist es, eine gewünschte Information darin zu finden und zu extrahieren. Um eine problemlose Weiterverarbeitung zu ermöglichen, müssen alle Dokumente im System dem gleichen Standardformat entsprechen, was im Normalfall eine Konvertierung des Ursprungsmaterials erfordert. Dabei ist es wichtig, darauf zu achten, möglichst viele der sinnvoll verwendbaren Strukturen zu übernehmen, da diese

sich bei den späteren Aufgaben als sehr hilfreich erweisen können. Quellformatspezifischer Overhead sollte entfernt bzw. vorhandene Tags für SGML, html oder XML-Markup sollten bei Bedarf in ein eindeutig festgelegtes Format überführt werden.
Für die Zeichendarstellung empfiehlt sich die Verwendung der UTF-8 Codierung. Der komplette Bereich der Unicode-Zeichen kann hiermit abgedeckt werden. Außerdem ist UTF-8 anerkannt und weit verbreitet; es ist beispielsweise die Standardcodierung für XML-Dokumente.

### Tokenisierung

Bei der Tokenisierung wird ein fortlaufender Text in seine Bestandteile zerlegt. Ziel ist es, einzelne Wörter und Satzzeichen zu identifizieren und diese als einzelne Einheiten zu markieren. Eine Trennung des Textes an jedem Leerzeichen zur Worterkennung ist zwar häufig ein guter Ansatz, versagt jedoch bei Sprachen wie Chinesisch, wo es kein eigenes Zeichen für Wortgrenzen gibt. Auch die Erkennung von Satzgrenzen kann in diesem Schritt erfolgen. Hier taucht das Problem auf, dass z. B. ein Punkt zwar ein Satzende markieren, aber auch in einem Datum, einer Zahl oder einer Abkürzung stehen kann. Die Erkennung von Satzgrenzen ist daher keine einfache Aufgabe und es ist mit einer gewissen Fehlerquote zu rechnen (GT94).
Die Zerlegung eines Textes beginnt mit der Zerlegung in seine Sätze. Diese Zerlegung wird in Abb. 3.1 an einem Beispiel dargestellt.

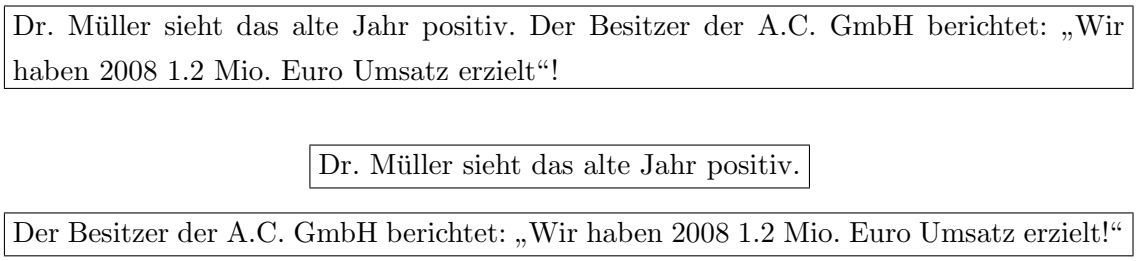



Abbildung 3.1.: Zerlegung eines Textes in Sätze

Die gefundenen Sätze werden dann ein weiteres Mal aufgesplittet. Die entstehenden Tokens sind die elementaren Bausteine des Satzes, also die einzelnen Wörter und Satzzeichen. Der Vorgang wird in Abb. 3.2 illustriert.

### Darstellung in Vektorform

Damit Lernverfahren erfolgreich sein können, setzen sie eine für sie passende Repräsentation der Eingabedaten voraus. Oftmals wird die Darstellung der zu untersuchenden Texte in Vektorform durchgeführt *(Vector Space Model)*. Jedem vorkommenden Wort der Dokumentmenge wird hierbei eine Dimension zugeordnet. Die Werte des Vektors

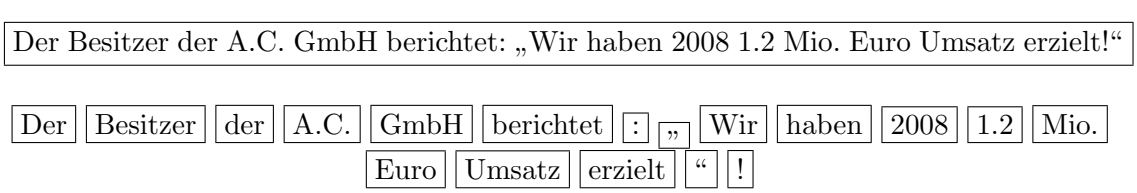



Abbildung 3.2.: Zerlegung eines Satzes in seine Tokens

beschreiben im einfachsten Fall die Häufigkeit eines Wortes im Dokument. Als effizienter hat sich allerdings die Darstellung *term frequency - inverse document frequency (tf-idf)* herausgestellt, bei der jedem Wort in Abhängigkeit von der Häufigkeit des Auftretens in der Gesamtmenge der Dokumente ein Gewicht zugeteilt wird (SB88). Da die Darstellung in dieser Form die Position des Wortes im Dokument nicht mehr berücksichtig, wird sie auch *Bag of Words* genannt. Dementsprechend eignet sie sich auch nur für Aufgaben, in denen die genaue Struktur des Textes unerheblich ist, beispielsweise zur Dokumentklassifizierung oder Kategorisierung. Um die Dimension des Vektors zu verringern, können häufig auftretende Wörter ohne Aussagekraft, die sog. Stopwörter, entfernt werden.

### 3.3.2. Verarbeitung auf Wortebene

Bei der Verarbeitung auf Wortebene stehen das aktuelle Wort und dessen unmittelbare Eigenschaften im Mittelpunkt der Betrachtung.

#### Stemming

Für statistikbasierte Lernverfahren hat sich eine Reduktion auf den Wortstamm oft als sinnvoll herausgestellt. Dabei werden verschiedene Varianten eines Wortes, die z. B. durch Komposition, Flexion oder Derivation entstanden sind, auf eine gemeinsame Form zurückgeführt. Dadurch wird eine Generalisierung erreicht; das Lernverfahren unterscheidet nicht mehr zwischen einzelnen Wörtern, die im Grunde die gleiche Bedeutung haben, aber z. B. in einer anderen Zeitform vorliegen (*gegangen, gehen*). Die Stammform muss nicht zwangsläufig identisch sein mit der morphologischen Grundform. Es geht in erster Linie darum, zusammengehörige Wörter auf eine eindeutige Basis zu reduzieren. Ein Verfahren dazu wurde bereits 1968 von Julie Lovins (Lov68) vorgestellt. Ein bekannter und bis heute verbreiteter Ansatz wurde 1980 von Porter mit dem *Porter-Stemmer-Algorithmus* (Por80) entwickelt, der für verschiedene Sprachen anpassbar ist. Im Deutschen ist das Finden einer Stammform schwieriger als z. B. im Englischen, wo es oftmals reicht, das Ende eines Wortes abzuschneiden *(walked, walk)*. Stattdessen muss z. B. doch auf die schwieriger zu bestimmende Grundform eines Wortes zurückgegriffen

werden (Abb. 3.3).

| | | |
|---|---|---|
| Scania | wurde | übernommen |
| Scania | werden | übernehmen |

Abbildung 3.3.: Ein Satz mit seinen Grundformen

### Morphologische Analyse

Wesentlich detailreicher als das Stemming ist die morphologische Analyse der Wörter. Hierbei wird das Wort in seine Bestandteile zerlegt und untersucht. Wichtige Informationen kann man z. B. durch *Lemmatisierung* gewinnen, bei der ein Wort auf seine Grund- bzw. Lexikonform gebracht wird. Dies ist ein sehr genaues Verfahren, erfordert aber auch sehr umfangreiche Informationen über den Aufbau der Wörter einer Sprache. Für die deutsche Sprache wurde von Wolfgang Lezius das System *Morphy* (Lez96) implementiert. Als kostenpflichtiges Tool wird von der Firma Lingsoft das Programm *gertwol* [3] angeboten, das auch in kommerziellen Softwareprodukten Einsatz findet. *TAGH*[4] von A. Geyken und T. Hanneforth (GH06) ist ein weiteres Softwaresystem zur Zerlegung von Wortformen, das als stand-alone Programm oder als Webservice nutzbar ist. In Abb. 3.4 wurde die Zerlegung der Wörter eines Satzes in ihre Morpheme durchgeführt.

| | | |
|---|---|---|
| Scania | wurde | übernommen |
| Scania | wurd—e | über—nomm—en |

Abbildung 3.4.: Ein Satz mit der morphologischen Zerlegung seiner Wörter

### Generalisierung

Bei der Generalisierung werden bestimmte Eigenschaften eines Wortes in einer verallgemeinerten Form dargestellt. Beispielsweise kann es nützlich sein, nur Informationen über die Groß- bzw. Kleinschreibung zu betrachten. Aus dem Wort „Haus“ wird so die Zeichenkette „Aaaa“ wobei „A“ einem Großbuchstaben und „a“ einem Kleinbuchstaben entspricht (Abb. 3.5).

| | | | | |
|---|---|---|---|---|
| VW | übernimmt | Kontrolle | über | Scania |
| AA | aaaaaaaaa | Aaaaaaaaa | aaaa | Aaaaaa |

Abbildung 3.5.: Ein Satz mit einer einfachen Generalisierung

[3] `http://www2.lingsoft.fi/cgi-bin/gertwol`
[4] `http://tagh.de/demo.php`

#### N-gramme

Als N-gramme bezeichnet man eine Sequenz der Länge $n$ von Objekten. Dies können z. B. Folgen von Buchstaben oder Wörtern sein, die im Schiebefensterverfahren aus einem Text erzeugt werden. Mit Hilfe statistischer Untersuchungsverfahren können aus den so konstruierten Features nützliche Informationen gewonnen werden. N-gramme wurden bereits 1951 von Shannon (Sha51) genutzt, der den Versuch unternahm, für die englische Sprache bei einer gegebenen Folge von Buchstaben das nachfolgende Zeichen vorherzusagen.

| ... Kontrolle ... | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| _K | Ko | on | nt | tr | ro | ol | ll | le | e_ |

Abbildung 3.6.: Bigramme (2-gramme) des Wortes „Kontrolle“

#### Weitere Verarbeitungsschritte

Zahlreiche nützliche Webservices zur Analyse weiterer Eigenschaften von einzelnen Wörtern stellt das *Projekt Deutscher Wortschatz*[5] der Universität Leipzig zur Verfügung. Die Nutzung einfacher Dienste ist kostenlos und erfordert keine Anmeldung, nur für kompliziertere, rechenintensive Anfragen ist eine Registrierung erforderlich. Die Liste der verfügbaren Dienste umfasst u. a. die Bildung von Grundformen, das Auffinden von Synonymen und die Abfrage von häufigen Nachbarwörtern oder Beispielsätzen, die das Wort enthalten.

### 3.3.3. Part of Speech - Tagging

Die Informationen, die mit den folgenden Verfahren gesucht werden, können nur dann gefunden werden, wenn die Umgebung der Wörter mit einbezogen wird. Nur durch den Kontext wird eindeutig festgelegt, welche Eigenschaften dem aktuellen Wort zugeordnet werden können.
Grundsätzlich bedeutet Tagging bei der hier betrachteten Verarbeitung von Sprache und Informationen, dass den Ursprungsdaten zusätzliche Anmerkungen (Tags) als Metadaten hinzugefügt werden. Die Menge der dabei zulässigen Tags bezeichnet man als Tagset.

Beim *Part of Speech* - Tagging im Speziellen geht es darum, jedem Wort im Text eine Wortklasse zuzuordnen. Dieses ist nicht durch eine simple Liste von Zuordnungen zu erreichen, da viele Wörter – je nach Kontext – verschiedene Wortklassen repräsentieren können. Als Standard-Tagset für das Deutsche kann derzeit das *Stuttgart-Tübingen*

[5] `http://wortschatz.uni-leipzig.de/`

*Tagset (STTS)* (Schiller, Teufel, Stöckert, Thielen 1999) angesehen werden. Es ist hierarchisch strukturiert und codiert morpho-syntaktische Eigenschaften. 54 Tags zur Annotation deutscher Korpora stehen darin zur Verfügung. Ein mit dem STTS getaggter Satz ist in Abb. 3.7 dargestellt.

| VW | übernimmt | Kontrolle | über | Scania |
|---|---|---|---|---|
| NE | WFIN | NN | APPR | NE |

Abbildung 3.7.: Ein Satz mit Part of Speech - Tags

Ein automatischer Tagger bekommt als Eingabe eine Folge von Wörtern und ordnet diesen das zugehörige (wahrscheinlichste) Tag zu. Erste Anwendung fand ein solches System als Präzisierungskomponente des *TDAP Parser* (Harris, 1958/59) in einem Projekt der University of Pennsylvania. Wenige Jahre später wurde das Brown-Korpus mit 1 Mio. Token mit einer Erkennungsrate von 70% getagged. (Klein and Simmons, 1963) Diese Systeme beruhten auf handgefertigten Regelsystemen und Lexikoneinträgen.

Moderne Tagger verfolgen unterschiedliche Ansätze. Die Verfahren werden in den folgenden Abschnitten kurz erläutert.

**Überwacht** Überwachte Tagger basieren darauf, dass sie große vorgetaggte Trainingsmengen zum Lernen verwenden. Sofern keine passenden Trainingsdokumente zur Verfügung stehen, müssen diese manuell erstellt werden, was viel Zeit in Anspruch nimmt. Tagger dieser Art können regelbasiert oder stochastisch arbeiten. Regelbasierte Ansätze verwenden typischerweise einfache deterministische Regeln, um mehrdeutige Wörter zu taggen. Diese bezeichnet man als *context frame rules*. Weiterhin können auch morphologische Informationen herangezogen werden, um den Prozess zu verbessern. Beim Training verwendet man häufig halbautomatische Verfahren, bei denen der Tagger auf einen ungetaggten Text angewendet wird. Danach wird die Ausgabe überprüft und bei Fehlern werden Korrekturregeln in das System eingefügt. Populärer Vertreter eines regelbasierten Taggers ist der *Brill-Tagger* (Bri92), der auch für das Deutsche trainiert werden kann.
Stochastische Verfahren bezeichnen alle Ansätze, deren Modelle mit Wahrscheinlichkeiten arbeiten. Um die Wahrscheinlichkeit zu bestimmen, mit der ein Tag einem bestimmten Wort zugewiesen werden kann, betrachtet man die Sequenz der Tags seiner Vorgänger. Realisieren kann man dies z. B. mit dem *Viterbi-Algorithmus*. Auch das *Hidden-Markov-Model (HMM)* eignet sich sehr gut für diese Art von Tagging.
Speziell um den Anforderungen der deutschen Sprache gerecht zu werden und ihre morphologische Vielfalt zu berücksichtigen, entwickelte H. Schmid (Sch95) das Programm *TreeTagger* mit Erweiterungen des Markov-Model-Ansatzes, wodurch damals die Erken-

nungsrate für diese Sprache deutlich verbessert werden konnte. Einen auf Beziehungsnetzen (*Dependency Networks*) basierenden Ansatz findet man beim *Stanford POS Tagger* (TKMS03), der als Java-Quellcode verfügbar ist und ebenfalls ein Modell für die deutsche Sprache bereitstellt.

**Unüberwacht** Beim unüberwachten Verfahren können stochastische Tagger ohne manuell annotierte Texte trainiert werden. Verwendung findet dabei der *Baum-Welch-Algorithmus*. Durch diesen werden iterativ die einzelnen Wahrscheinlichkeitswerte angepasst, um eine möglichst hohe Gesamtwahrscheinlichkeit im Trainingskorpus zu erreichen. Die einzige Eingabe, die ein solches System benötigt, ist eine Liste der erlaubten POS-Tags. Die erreichte Genauigkeit ist geringer als bei den überwachten Verfahren, erspart einem jedoch das Erstellen der Trainingsbeispiele. Heute stehen in der Regel ausreichende Textmengen mit Annotierung zu Verfügung, sodass überwachte Verfahren aufgrund der höheren Genauigkeit zu bevorzugen sind.

### 3.3.4. Named Entity Recognition

Named Entities sind laut *CoNLL2003*-Definition zusammenhängende Phrasen vom Typ *Person*, *Firma* oder *Ort* (TKSDM03). Allgemeiner gefasst wird der Begriff beim *Entity Detection and Tracking-* (EDT) Task der *ACE2004*-Konferenz. Hier lautet die Definition: „Eine Entity ist ein Objekt oder eine Menge von Objekten in der Welt.“. Diese Objekte werden bei ACE in sieben verschiedene Kategorien eingeteilt: *Person* (PER), *Organization* (ORG), *Geo-political Entity* (GPE), *Location* (LOC), *Facility* (FAC), *Vehicle* (VEH) und *Weapon* (WEA) (LDC04a).
Moderne Verfahren zur NER basieren auf verschiedenen statistischen Methoden wie HMM oder MEMM, wie sie in ähnlicher Weise auch zum POS Tagging eingesetzt werden. Diese Systeme werden automatisch auf getaggten Korpora trainiert. Im Gegensatz dazu können auch regelbasierte Systeme verwendet werden, deren auf grammatischen Zusammenhängen beruhende und teilweise handcodierte Regeln in mühevoller Arbeit manuell erstellt werden müssen.

**IOB-Schema** Die Markierung von Named Entities im Text kann mittels Tags erfolgen. Ein Tag wird als zusätzliches Attribut zusammen mit einem Token abgespeichert. Die *IOB*-Repräsentation, die von Ramshaw and Marcus (RM95) vorgestellt wurde, ist für diese Art von Tagging sehr gut geeignet. *IOB* steht für *Inner, Outer, Beginning*. Mit Hilfe der Tags können Grenzen von Entitäten auf folgende Weise markiert werden: Token, die nicht Teil einer Entität sind, also außerhalb stehen, haben das Tag *O*. Worte, die den Beginn einer Entität darstellen, haben das Tag *B-XXX*. *XXX* ist ein Platzhalter für den Typ der Entität. Worte, die innerhalb einer Entität stehen, haben das Tag *I-XXX*.

| Die | Deutsche | Bank | ist | pleite | . |
|---|---|---|---|---|---|
| O | B-ORG | I-ORG | O | O | O |

Abbildung 3.8.: Ein Satz mit IOB Tags für Named Entities

### 3.3.5. Parsing

Die syntaktische Struktur einer Sprache wird durch ihre Grammatik festgelegt. In dieser sind alle Regeln enthalten, durch deren Anwendung gültige Konstrukte einer Sprache aufgebaut werden können. Die häufigste Art einer Grammatik ist die *kontextfreie Grammatik* (Context Free Grammar, CFG) (Cho56).
Eine kontextfreie Grammatik $G$ besteht aus vier Elementen:

$$G = (V, \Sigma, R, S) \tag{3.1}$$

- $V$ ist eine endliche Menge von Nichtterminalsymbolen. Diese entspricht den gröberen Teilstrukturen ($VP$, $NP$ etc.), die ein Satz enthalten kann.
- $\Sigma$ ist eine endliche Menge von Terminalsymbolen, $\Sigma \cap V = \emptyset$. Terminalsymbole sind Wörter und Satzzeichen.
- $R$ ist eine Relation von $V \rightarrow (V \cup \Sigma)^*$, sodass gilt: $\exists w \in (V \cup \Sigma)^* : (S, w) \in R$
- $S$ ist ein Startsymbol. $S \in V$. Es repräsentiert den ganzen Satz.

Basierend auf der CFG wird von Charniak (Cha93) die *probabilistische kontextfreie Grammatik* (Probabilistic Context-Free Grammar (PCFG)) eingeführt. Jeder der Regeln aus $R$ wird eine Wahrscheinlichkeit $P(V \rightarrow (V \cup \Sigma)^*)$ zugeordnet. Die Wahrscheinlichkeit aller Regeln, die von einem Nichtterminalsymbol aus anwendbar sind, muss sich zu 1 aufsummieren. Die PCFG definiert somit eine kontextfreie Sprache, deren Sätze mit

| $V$ | $\rightarrow$ | $(V \cup \Sigma)^*$ | $P(V \rightarrow (V \cup \Sigma)^*)$ |
|---|---|---|---|
| s | $\rightarrow$ | np vp | 0,8 |
| s | $\rightarrow$ | vp | 0,2 |
| np | $\rightarrow$ | n | 0,4 |
| np | $\rightarrow$ | n np | 0,4 |
| np | $\rightarrow$ | n pp | 0,2 |

Tabelle 3.1.: Einige Regeln einer PCFG

berechenbaren Wahrscheinlichkeiten durch eine Abfolge von Regelanwendungen erzeugt werden. Bei einem gegebenen Satz lassen sich basierend auf der PCFG alle möglichen

Folgen von Regelanwendungen bilden, die diesen Satz erzeugt haben könnten. Für jede dieser Strukturen lässt sich ein Wahrscheinlichkeitswert berechnen. Dies erlaubt eine Unterscheidung zwischen gebräuchlichen und weniger gebräuchlichen Konstruktionen, die Regelkette mit der höchsten Wahrscheinlichkeit wird von einem Softwaresystem als „richtige Interpretion“ eines Satzes angenommen.

### Full Parsing

Full Parsing versucht die vollständige Struktur der zugrundeliegenden Grammatik eines Satzes herauszufinden. Die Repräsentation erfolgt in der Regel in Form einer Baumstruktur, die das Startsymbol als Wurzel hat. Ihre Funktionsweise basiert oftmals auf

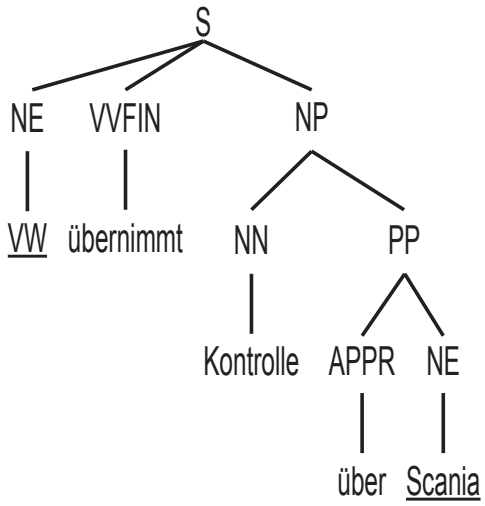



Abbildung 3.9.: Der Parsebaum eines deutschen Satzes

den im vorherigen Abschnitt vorgestellten PCFG. Die aus den statistischen Untersuchungen gewonnen Erkenntnisse können über andere Merkmale des Satzes, die z. B. das Auftreten bestimmter Worte mit einbeziehen, verbessert werden. Während für die englische Sprache mehrere Parser inkl. passender Modelle zur Verfügung stehen, ist für das Deutsche die Auswahl weniger groß. Die Struktur der deutschen Sprache stellt höhere Herausforderungen an die Systeme als die vieler anderer Sprachen. Drei wichtige Gründe dafür werden von Kübler (KHW06) genannt:

- Die unterschiedliche Stellung des finiten Verbs im Satz
- Die freie Stellung von Satzkomponenten
- Unzusammenhängende Konstituenten

Statistische Parser benötigen einen umfassenden Trainingsdatensatz in der Sprache, für die sie angewendet werden sollen. Kübler zeigt auch, dass dieses Trainingskorpus einen

enormen Einfluss auf die Genauigkeit eines Parsers hat. Der *Stanford Parser* (KM02) wurde auf zwei verschiedenen Korpora trainiert. Zum einen wurde das *NEGRA*-Korpus genutzt, zum anderen das *TüBa-D/Z* Korpus. Die Ergebnisse wurden verglichen, mit dem Resultat, dass die Ergebnisse, die mit dem *TüBa-D/Z*-Korpus erzielt werden konnten, deutlich besser ausfielen. Sie waren von ähnlicher Güte wie die eines auf der *Penn*-Treebank trainierten Modells, das für die englische Sprache eingesetzt werden kann. Das zeigt, dass moderne Parser, die auf einer Kombination von PCFG und anderen Verfahren basieren und die auf sorgfältig erstellten Trainingsdaten trainiert werden, auch für die deutsche Sprache ein hohes Maß an Genauigkeit erreichen können. Bei der *ACL*-Konferenz 2008 wurde ein Workshop speziell für die Erforschung des Parsens der deutschen Sprache abgehalten, wodurch die Entwicklung neuer Systeme begünstigt wurde. Die zugehörigen Veröffentlichungen sind zusammengefasst in (KP08).

### Shallow Parsing

Beim *Shallow Parsing* wird ein Satz nur in seine Konstituenten zerlegt. Dies sind größere Satzfragmente wie Nominal- und Verbalphrasen. Eine Nominalphrase ist eine zusammengehörige Gruppe von Wörtern, dessen Kopf ein Nomen oder Pronomen ist. Das Kopfwort kann noch zusätzlich z. B. durch einen Artikel modifiziert werden. Eine Verbalphrase hat dementsprechend ein Verb als Kopfwort. Shallow Parsing wird auch *Chunking* genannt. Da die Verfahren zum Shallow Parsing in der Regel weniger rechenintensiv und leichter zu implementieren sind als Methoden zum vollständigen Parsen, wird es in Bereichen angewendet, wo die komplette Struktur keine Vorteile bringen würde oder eine schnelle Verarbeitung wichtig ist. Ein auf PCFG basierender Chunker wird z. B. von (SSIW00) vorgestellt. Er basiert auf einer von Hand erstellten Grammatik, die halbautomatisch um Regeln ergänzt wurde, um die Zuverlässigkeit zu steigern.

# 4. Relationserkennung

Ein wichtiger Bereich der IE ist die Relationserkennung. Die Aufgabenstellung des *ACE Relation Detection and Characterization*-Teilgebietes (LDC04b) beschreibt eine Relation als eine gültige Kombination zwischen genau zwei Entitäten, die in demselben Satz enthalten sind und zwischen denen eine Beziehung besteht. Im Rahmen dieser Diplomarbeit soll die Relation *Firmenfusion* im Vordergrund stehen. Diese ist definiert als symmetrische Relation zwischen zwei Entitäten vom Typ *Firma*, die ihre Unternehmen zusammenlegen wollen. Diese Art der Relation ist eine klassische Aufgabe der Relationsextraktion. Bereits bei der *MUC* Konferenz (GS96) wurden Fusionen von Unternehmen untersucht.

## 4.1. Eigenschaften von Relationen

Eine Relation ist folgendermaßen definiert:

$$R_i(Sentence\ s) =< Type_m \in relationtypes,$$
$$(Argument_1 \in entities_s, Argument_2 \in entities_s) >$$

$entities_s$ entspricht der Menge der Entitäten im Satz
$relationtypes$ ist die Gesamtmenge der möglichen Relationen im Korpus

Die Entitäten werden die Argumente der Relation genannt. Je nach Typ ist ihre Reihenfolge von Bedeutung. Man spricht dann von einer asymetrischen Relation. Spielt die Reihenfolge keine Rolle, so ist die Relation symmetrisch. Ziel der Relationserkennung ist es, aus natürlichsprachlichen Sätzen Relationen zu extrahieren.

### 4.1.1. Verteilung von Relationsstrukturen

Die Verteilung von unterschiedlichen Sprachkonstrukten, mit deren Hilfe eine Relation ausgedrückt werden kann, entspricht dem Zipfschen Gesetz. Das bedeutet, dass einige Strukturen sehr häufig auftauchen und mit deren Hilfe – wenn sie dem System bekannt sind – bereits eine recht große Anzahl an Relationen gefunden werden kann. Alle übrigen Relationen werden durch eine große Menge unterschiedlich aufgebauter Strukturen beschrieben, die im Vergleich viel seltener auftauchen. Ein System zur Relation-Extraction

wird also zunächst durch die Fähigkeit, die wenigen, sehr populären Relationsstrukturen erkennen zu können, einen Basiswert seiner Performance erreichen können. Sind diese populären Strukuren jedoch abgedeckt so lässt sich die Erkennungsrate nur noch langsam verbessern.

### 4.1.2. Generierung von Relationskandidaten

Eine Relation beschreibt eine Beziehung zwischen zwei Entities. Ein Satz kann im Allgemeinen jedoch mehr als zwei Entities enthalten. Es ist also möglich, dass auch mehrere Relationen in ihm enthalten sind. Für einige Lernverfahren müssen vorab für jede mögliche Kombination von Entities Relationskandidaten aus den Sätzen gebildet werden. Diese bestehen aus einem Paar von zwei unterschiedlichen Entities, einem zugeordneten Relationstyp und einem Label. Das Label des Relationskandidaten wird auf *true* gesetzt, falls er eine wahre Relation beschreibt, ansonsten ist es *false*. Es müssen nur Kandidaten gebildet werden, die theoretisch mögliche Relationen beschreiben. Dies setzt voraus, dass die Typen der Entities gemeinsam als Argumente einer gültigen Relation auftreten können. Eine Firmenfusion kann z. B. per Definition nur zwischen zwei Unternehmen, nicht aber zwischen einem Unternehmen und einer Person stattfinden. Relationskandidaten können zusätzlich durch weitere Features angereichert werden, anhand derer die Lernverfahren entscheiden können, welche Label sie zuordnen müssen.

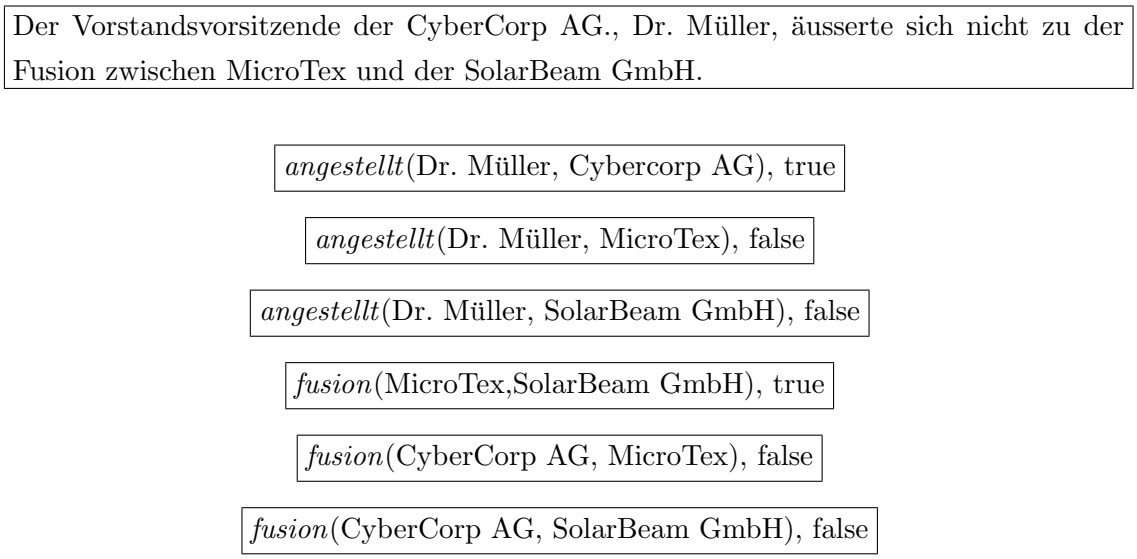



Abbildung 4.1.: Bildung von Relationskandidaten aus einem Satz

## 4.2. Verfahren zur Relationserkennung

Im Laufe der Jahre wurden zahlreiche Verfahren zur Relation Extraction entwickelt. Diese verfolgen unterschiedliche Ansätze. Während frühe Systeme stark abhängig waren von menschlichen Regeldesignern, soll der manuelle Aufwand in neueren Systemen soweit wie möglich reduziert werden.

### 4.2.1. Maschinelle Lernverfahren

Zur Relationserkennung werden oft maschinelle Lernverfahren eingesetzt. Ziel des maschinellen Lernen ist es, ein System zu schaffen, das selbstständig in der Lage ist, aus einer Menge von Daten Muster und Informationen abzuleiten. Man unterscheidet zwischen unterschiedlichen Ansätzen, von denen einige im Folgenden vorgestellt werden:

**Überwachtes Lernen** Das überwachte Lernen (*supervised learning*) bekommt als Eingabe einen Satz von Trainingsbeispielen. Trainingsbeispiele enthalten dabei sowohl die reinen Eingabedaten als auch deren Labels. Die Labels geben an, welche Ausgabe einem Trainingsbeispiel durch das System zugeordnet werden soll. Während der Trainingsphase werden so lange Anpassungen vorgenommen, bis das Ausgabeergebnis des Lernverfahrens möglichst optimal mit den vorgegebenen Labels übereinstimmt. Die dadurch gewonnenen Informationen werden in Form eines gelernten Modells abgespeichert. Ein typisches Beispiel für ein überwachtes Lernverfahren ist die *Support Vektor Machine*, die in 6.2 vorgestellt wird. Für eine Klassifikationsaufgabe erhält sie die Trainingsdaten in Form von Features sowie die zugehörigen Labels. Daraus bildet sie ein Modell. In dieses können später gleichartige Featurevektoren eingeordnet werden.

**Unüberwachtes Lernen** Beim unüberwachten Lernen (*unsupervised learning*) werden keine Trainingsdaten mit Labels zur Verfügung gestellt. Das Lernverfahren entscheidet selbstständig, wie mit den zu untersuchenden Daten zu verfahren ist. Ein Beispiel dafür ist das *Clustering*. Die Eingabe ist eine Menge von Objekten. Das Lernverfahren betrachtet deren Eigenschaften und fasst die Objekte darauf basierend zu Gruppen zusammen. Die Ausgabe des Systems ist somit unabhängig von einer Trainingsphase, in der durch Beispiele vorgegeben wurde, auf welche Weise Objekte zu Gruppen zugeordnet werden müssen.

**Halbüberwachtes Lernen** Halbüberwachte Lernverfahren (*semi-supervised learning*) bekommen zu Beginn sowohl eine kleine Menge gelabelter Daten als auch eine größere Menge ungelabelter Daten. Anhand der gelabelten Daten werden erste Lernschritte durchgeführt. Anschließend versucht das System, sein Modell durch Betrachtung der ungelabelten Daten zu erweitern.

### 4.2.2. Handcodierte Regelsysteme

Die älteste Form, Relationen aus Textsammlungen zu extrahieren, erfolgt mit Systemen, deren Basis eine Sammlung von fest definierten Regeln bildet. Diese Regeln werden von menschlichen Experten von Hand zusammengestellt und speziell auf die Aufgabe und die Art der zugrunde liegenden Dokumente angepasst. Die Erkennungsrate ist zwar sehr gut, der Nachteil ist allerdings, dass die Anpassung an eine andere Aufgabenstellung bzw. an einen anderen Themenbereich den kompletten Neuentwurf des gesamten Regelsystems notwendig macht. Da die Erzeugung solcher Regelsysteme eine sehr langwierige und mühevolle Aufgabe ist, die mehrere Wochen oder Monate in Anspruch nehmen kann, ist diese Lösung auf die Dauer nicht zufriedenstellend.

### 4.2.3. Überwachte Extraktionssysteme

Musterbasierte Systeme beruhen darauf, dass Informationen häufig in ähnlicher Form vorliegen. Anstatt Regeln von Hand in das System einzuprogrammieren, werden Texte vorbereitet, in denen die zu findende Information bereits markiert ist. Diese werden dem Softwaresystem als Trainingsmenge zur Verfügung gestellt. Aufgrund der markierten Informationen versucht das Softwaresystem, wiederkehrende Muster zu erkennen und diese in seiner Regeldatenbank abzuspeichern. Regeln sehen zum Beispiel so aus:

- $\langle Firma \rangle$ fusioniert mit $\langle Firma \rangle$
- gab $\langle Firma \rangle$ die Fusion mit $\langle Firma \rangle$ bekannt
- Fusion von $\langle Firma \rangle$ mit $\langle Firma \rangle$
- Zusammenschluss von $\langle Firma \rangle$ mit $\langle Firma \rangle$

Eines der ersten Beispiele für ein solches System ist das Programm *CRYSTAL* (SFL97). Die Aufgabe des menschlichen Experten wird darauf reduziert, Trainingsdokumente mit Anmerkungen zu versehen und Relationen zu markieren. Ein schwieriger Punkt beim Systementwurf ist zu entscheiden, wie stark gefundene Muster generalisiert, also „aufgeweicht“ werden dürfen. Fasst man ein Muster zu eng auf, verringert das die Wahrscheinlichkeit, mit diesem Muster eine Relation aufzufinden, die sehr ähnlich formuliert wurde. Wird das Muster zu locker aufgefasst, so findet man unter Umständen falsche Informationen. Da nur Muster erkannt werden können, die das System während der Trainingsphase schon mindestens einmal gesehen hat, ist es erforderlich, dass das Trainingskorpus sehr groß ist.

Andere Systeme basieren nicht mehr auf Verfahren, die einfache Textmuster direkt in der zuvor vorgestellten Form verwenden. Dennoch benötigen sie ebenfalls eine Trainingsmenge. Sie verwenden überwachte Lernverfahren, die in der Lage sind, komplexe

Strukturen und Eigenschaften der Sätze zu berücksichtigen. Beispiele dafür sind probabilistische Methoden oder die Kern-Methoden, die in Kapitel 6 vorgestellt werden. Diese Lernverfahren müssen ein Modell bilden, auf dessen Grundlage sie später bei unbekannten Sätzen entscheiden können, ob es wahrscheinlich ist, dass dieser Satz eine Relation beinhaltet bzw. ob die Ähnlichkeit zu der Menge der positiven Beispiele gegeben ist. Eine genügend große Menge von Beispielsätzen, in denen mindestens die Relationspartner sowie die passenden Label markiert sind, müssen daher auch diesen Systemen zur Verfügung stehen.

### 4.2.4. Halbüberwachte Extraktionssysteme

Um zu vermeiden, große manuell annotierte Korpora zum Training vorbereiten zu müssen, verfolgen halbüberwachte Systeme einen noch stärker automatisierten Ansatz. Ausgehend von einer kleinen Basismenge von bekannten Relationsbeispielen wird – in mehreren Durchläufen – die Menge der bekannten positiven bzw. negativen Beispieldaten um die neu gefundenen Relationskandidaten erhöht und das Modell des Systems angepasst. Bei jedem erneuten Durchlauf werden die neu gefundenen Informationen direkt eingesetzt, um mit ihrer Hilfe weitere, neue Lernbeispiele zu finden. Als Variante kann man die gefundenen neuen Relationen auch vor dem Einfügen in das Modell von einem Menschen kontrollieren lassen. Auf diese Weise können grobe Fehler des Programms von vornherein ausgeschlossen werden. Das Verfahren, auch Bootstrapping genannt, ist in (JMNR99) ausführlich dargelegt. Systeme, die auf dieser Vorgehensweise basieren, sind sehr zahlreich (Ril96),(YGTH00),(AG00),(RF07) und an dieser Stelle soll stellvertretend nur ein einfaches Beispiel anhand eines musterbasierten Systems beschrieben werden.

Das von Brin (Bri98) vorgestellte System *DIPRE* erhält als Eingabe nur eine Menge von Tupeln der Form $\langle Buch, Autor \rangle$. Brin durchsuchte eine große Menge von Internetseiten nach Auftreten dieser Tupel. Bei jedem Treffer wird die umgebende Textstelle der Entitäten abgespeichert. Eine Teilkomponente von *DIPRE* versucht im darauffolgenden Schritt, allgemeingültige Muster daraus zu generieren. An dieser Stelle tritt das Problem auf, das bereits im vorherigen Abschnitt erwähnt wurde. Es ist wichtig, die Regeln so zu erzeugen, dass sie nicht zu streng und nicht zu locker gefasst sind. Im nächsten Schritt wird die Dokumentsammlung erneut durchlaufen. Dieses Mal jedoch wird nicht nach den bekannten Tupeln gesucht, sondern die neu erzeugten Muster werden zur Anwendung gebracht. Auf diese Weise werden neue $\langle Buch, Autor \rangle$ - Tupel gefunden, die in der gleichen Form wie die bereits Bekannten auf den Internetseiten aufgeführt sind. Führt man diese beiden Schritte mehrfach im Wechsel hintereinander aus, erhält man eine sich selbstständig erweiternde Menge von Mustern und eine stetig wachsende Datenbank mit Informationen über Bücher und deren Autoren.

### 4.2.5. Unüberwachte Extraktionssysteme

Aktuelle Forschungen versuchen die Relation Extraction noch weiter zu fassen. Anstatt in einer Sammlung von homogenen Texten eine vordefinierte Menge von Relationen und Entitäten zu erkennen, sollen alle möglichen Relationen aus großen Textmengen wie dem Internet gesucht und extrahiert werden. Das universell einsetzbare System *KNOWITALL* (ECD$^+$04) ist bereits sehr flexibel und kann schnell für neue Anwendungsbereiche angepasst werden. Dennoch ist es auf themenspezifische Quellmengen für Bootstrapping-Methoden angewiesen. Im Gegensatz dazu benötigt die Software *TEXTRUNNER* (BaSBE07) keinerlei vorgetaggtes Material.

Das System entscheidet selbstständig – auf Basis einer Quelldatenmenge – welche Sätze als positiv und welche als negativ für eine Lernaufgabe zu betrachten sind. Dabei wird in zwei Schritten vorgegangen: Zunächst werden die Abhängigkeitsgraphen der Wörter für mehrere Tausend Sätze erzeugt. Als nächstes werden alle Basis-Nominalphrasen gesucht. Dies sind Nominalphrasen ohne verschachtelte Nominalphrasen oder Modifikatoren wie Präpositionalphrasen. Zwischen jeweils zwei der Nominalphrasen werden Sequenzen von Wörtern durch das Durchlaufen eines verbindenen Pfades im Abhängigkeitsgraphen gebildet. Diese Wortfolge wird als potentieller Kandidat für eine Relation betrachtet. Der Lerner markiert sie als positiv, wenn sie eine Menge von Bedingungen erfüllt; sonst wird ein negatives Label vergeben. Diese Bedingungen sind so ausgearbeitet, dass es wahrscheinlich ist, das Label richtig zu setzen, auch wenn beim Parsen einige Fehler aufgetreten sind und die Wortfolge daher nicht ganz korrekt aufgestellt wurde.

Nachdem der Lerner eine Menge an Relationen gefunden und mit Labeln versehen hat, wird jeder Relationskandidat in eine Featurevektor-Form überführt. Diese Feature-Vektoren sind unabhängig von einem Anwendungsbereich und können sehr schnell und ohne den Gebrauch eines Parsers auch aus neuen Sätzen erzeugt werden. Dies ist wichtig, wenn große Textmengen wie das Internet durchsucht werden sollen, da das Parsen eines Satzes ein sehr zeitaufwändiger Prozess ist. Die auf diese Weise erzeugten Featurevektoren werden als Eingabemenge für einen *Naive Bayes*-Klassifizierer genutzt. Das Modell des Klassifizierers enthält keinerlei Abhängigkeiten von vordefinierten Relationstypen oder lexikalische Vorgaben für einen Themenbereich. Daher kann er auf domain-unabhängige Weise eingesetzt werden. Anstelle des *Naive Bayes*-Klassifizierers kann ebenfalls die *Conditional Random Fields* (CRF) - Methode angewendet werden (BE08).

## 4.3. Performancemaße

Die Bewertung der Ergebnisse eines Systems zur Relationserkennung kann mit Hilfe der Maße *precision*, *recall* sowie einem Mittelwert über die beiden Werte, der *f-measure* genannt wird, erfolgen. Auch die insgesamte Genauigkeit *accuracy* kann von Interesse sein. Diese Werte sind folgendermaßen definiert:

$$precision = \frac{[true\ positives]}{[true\ positives] + [false\ positives]} \tag{4.1}$$

$$recall = \frac{[true\ positives]}{[true\ positives] + [false\ negatives]} \tag{4.2}$$

$$f - measure = \frac{2 * precision * recall}{precision + recall} \tag{4.3}$$

$$accuracy = \frac{[true\ positives] + [true\ negatives]}{[true\ positives] + [false\ positives] + [true\ negatives] + [false\ negatives]} \tag{4.4}$$

- *true positive*: Ein Lernbeispiel mit dem Label *true*, das als *true* vorausgesagt wurde
- *true negative*: Ein Lernbeispiel mit dem Label *false*, das als *false* vorausgesagt wurde
- *false positive*: Ein Lernbeispiel mit dem Label *false*, das als *true* vorausgesagt wurde
- *false negative*: Ein Lernbeispiel mit dem Label *true*, das als *false* vorausgesagt wurde

# 5. Bäume und Baumstrukturen

Daten, die in Form eines Baumes abgelegt sind, enthalten in ihrer Struktur Informationen, die von besonderem Interesse für maschinelle Lernverfahren sein können. Für die Erforschung von Sprachen sind z. B. Parsebäume von großer Wichtigkeit, da sie die Möglichkeit bieten, Einblicke in den grammatischen Aufbau eines Satzes zu erhalten. Strukturen von Bäumen können zerlegt, beschnitten oder erweitert werden. Im folgenden Kapitel sollen einige Bäume und Baumformen vorgestellt werden.

## 5.1. Baumstrukturen und Darstellung

Ein syntaktischer Parsebaum (Abb. 5.1) bildet die grammatische Struktur eines Satzes ab. Blätter enthalten die Terminalsymbole der Grammatik, also Satzzeichen und Wörter. In den inneren Knoten finden sich die Nichtterminalsymbole. Die Kinder eines inneren Knotens müssen durch eine gültige Ableitungsregel der Grammatik erzeugt werden können. Die Beschriftung (Label) eines Knotens ist der Name des Symbols, das er repräsentiert. Aufgrund der Position eines Knotens im Baum lassen sich unter Umständen bestimmte Eigenschaften ableiten. So enthält in einem Parsebaum die Ebene von Knoten vor dem Terminalsymbol (Präterminal) ein Symbol, das gleichbedeutend ist mit dem *Part-Of-Speech*-Tag.

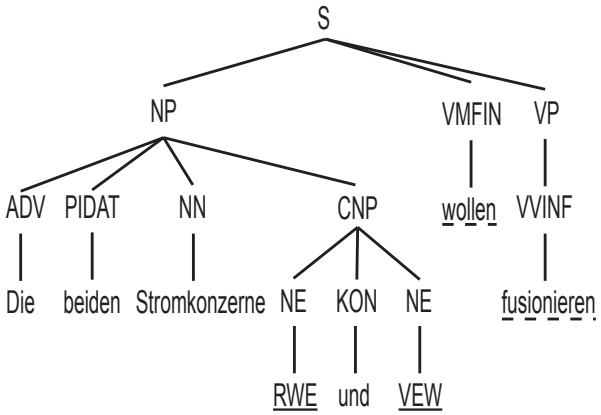



Abbildung 5.1.: Ein vollständiger Parsebaum

### 5.1.1. Teilstrukturen von Bäumen

Um mit Hilfe von Faltungskernen Bäume untersuchen zu können, ist es notwendig, einen Baum in kleinere Strukturen zerlegen zu können. In der in der Diplomarbeit verwendeten Kernfunktion für Bäume können zwei verschiedene Arten der Zerlegung genutzt werden: *Teilbäume* und *Unterbäume*. Die Verwendung der Teilbäume stellt einem Kern mehr Informationen über Satzstrukturen zur Verfügung als die Unterbäume. Dennoch ist es von Interesse zu untersuchen, für welche Lernaufgabe welche Baumform am besten geeignet ist.

#### Unterbäume

Als Unterbaum *(Subtree, ST)* wird eine Substrukur bezeichnet, die aus einem Knoten des Ursprungsbaumes sowie allen seinen Nachfahren, bis hinab in die Blätter, gebildet wird. In Abb. 5.2 werden alle Unterbäume, die aus einem kleinen Baum gebildet werden können, gezeigt.

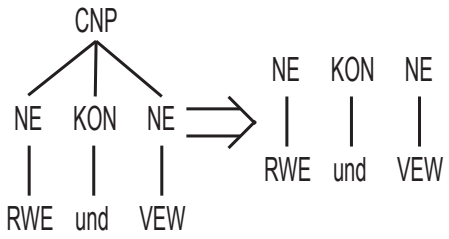



Abbildung 5.2.: Alle Unterbäume einer Baumstruktur

#### Teilbäume

Ein Teilbaum (*Subset Tree, SST*) dagegen erhält die Struktur des Originalbaums, beginnend in einem beliebigen Knoten. Es werden jedoch nicht zwangsläufig alle Nachfahren eingeschlossen. Dabei ist wichtig zu beachten, dass Produktionsregeln nicht getrennt werden dürfen. Die durch sie erzeugten Strukturen müssen immer komplett in dem Teilbaum enthalten sein. Abb. 5.3 zeigt eine Auswahl an Teilbäumen, die aus einem Baumfragment gebildet werden können. Im Vergleich zu 5.2 sieht man, dass deutlich mehr Strukturen enstehen. Die Zahl der Teilbäume eines Baumes wächst exponentiell mit der Anzahl seiner Blätter. Die Repräsentation eines Baumes durch Aufzählung aller seiner Teilbäume folgt dem *DOP (Data-Oriented Parsing)*- Modell, das auf Bod (Bod98) zurückgeht.

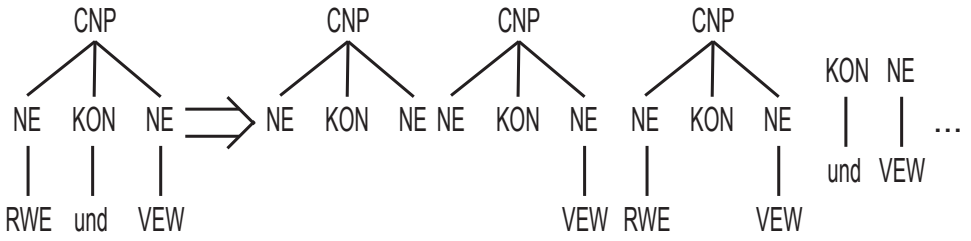



Abbildung 5.3.: Einige Teilbäume einer Baumstruktur

### 5.1.2. Produktionsregeln

Knoten in einem Parsebaum entsprechen Anwendungen von Regeln der zugrunde liegenden Grammatik. Um die Produktionsregel eines Knotens zu erhalten, betrachtet man zunächst sein eigenes Label. Dies wird auf die linke Seite der Produktionsregel geschrieben. Die rechte Seite entspricht den Labels seiner Kinder. Abb. 5.4 zeigt die Bildung der Produktionsregeln von Knoten anhand zweier Beispiele.

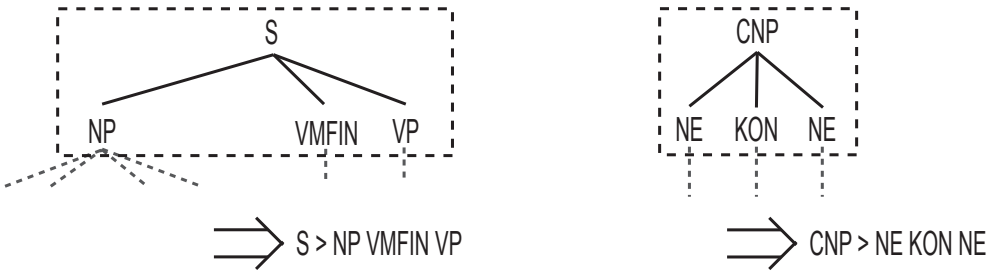



Abbildung 5.4.: Das Bilden von Produktionsregeln aus Knoten des Baumes

In die Produktionsregeln kann auch ein Rückblick auf die Vorgängerknoten mit aufgenommen werden. Die linke Seite der Produktionsregel wird erweitert. Vom Knoten ausgehend werden die Elternknoten in Richtung Wurzel durchlaufen. Somit geht die Umgebung, also der Kontext eines Knotens, in die Produktionsregel mit ein. Die Tiefe des Rückblicks kann dabei unterschiedlich ausfallen und über einen Parameter $m$ festgelegt werden. Abb. 5.5 gibt dafür ein Beispiel. (ZZJZ07) benutzen diese kontextsensitiven Produktionsregeln zur Berechnung ihrer kontextsensitiven Kernfunktion über Parsebäume, die in Abschnitt 6.4.4 vorgestellt wird.

### 5.1.3. Darstellung als String

Um den flexiblen Umgang mit Bäumen über die Grenzen mehrerer Programme hinweg zu ermöglichen, ist es notwendig, eine Datenrepräsentation zu finden, die allgemein verständlich und leicht zu verarbeiten ist. Die strukturellen Informationen, die in einem

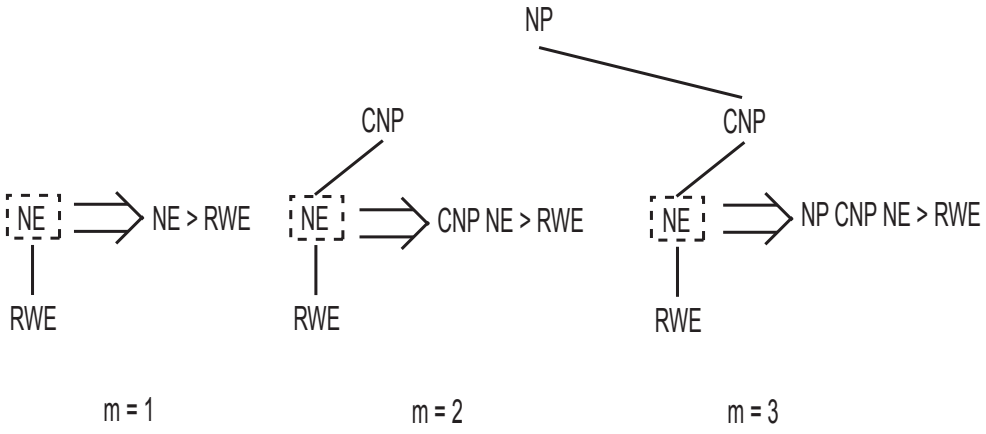



Abbildung 5.5.: Produktionsregeln eines Knotens für unterschiedliche m - Werte

Baum enthalten sind, dürfen dabei nicht verloren gehen. Die Repräsentation als String, wie in Abb. 5.6 gezeigt, erlaubt diesen Austausch. Dabei ist zu beachten, dass die Labels eines Knotens keine Klammern enthalten dürfen, da das Parsen des Strings, also die Zurückverwandlung in ein Baumobjekt, sonst fehlschlagen wird.

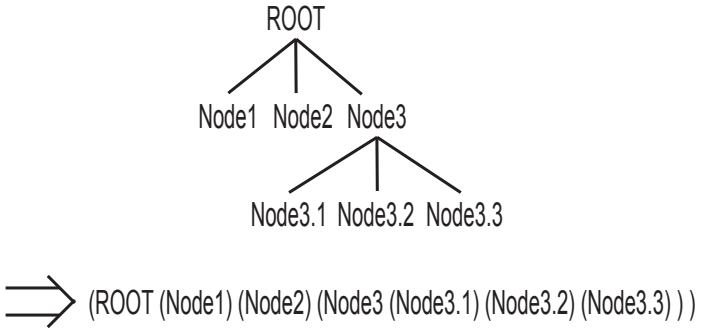



Abbildung 5.6.: Repräsentation eines Baumes als TreeString

## 5.2. Beschneiden der Baumformen

Von Zhang et. al. (ZZS06) wird der Einfluss der Form des Baumes, der an eine Kernfunktion übergeben werden soll, untersucht. Jeweils zwei Entitäten eines Satzes bilden die Grundlage für einen Relationskandidaten. Aus einem vorliegenden kompletten Parsebaum werden sieben verschiedene Typen von Bäumen erzeugt und deren Resultate bei der Relationserkennung verglichen. Die unterschiedlichen Typen der Bäume entstehen, indem die Blattknoten der beiden Relationspartner die Grundlage bilden und der übrige Baum auf unterschiedliche Weise mehr oder weniger stark beschnitten wird. Dieses erfolgt sowohl durch das Entfernen von Blattknoten als auch durch das Abflachen innerer

Strukturen. Einige der Baumformen sind in Abbildung 5.7 vorgestellt und werden im Folgenden beschrieben:

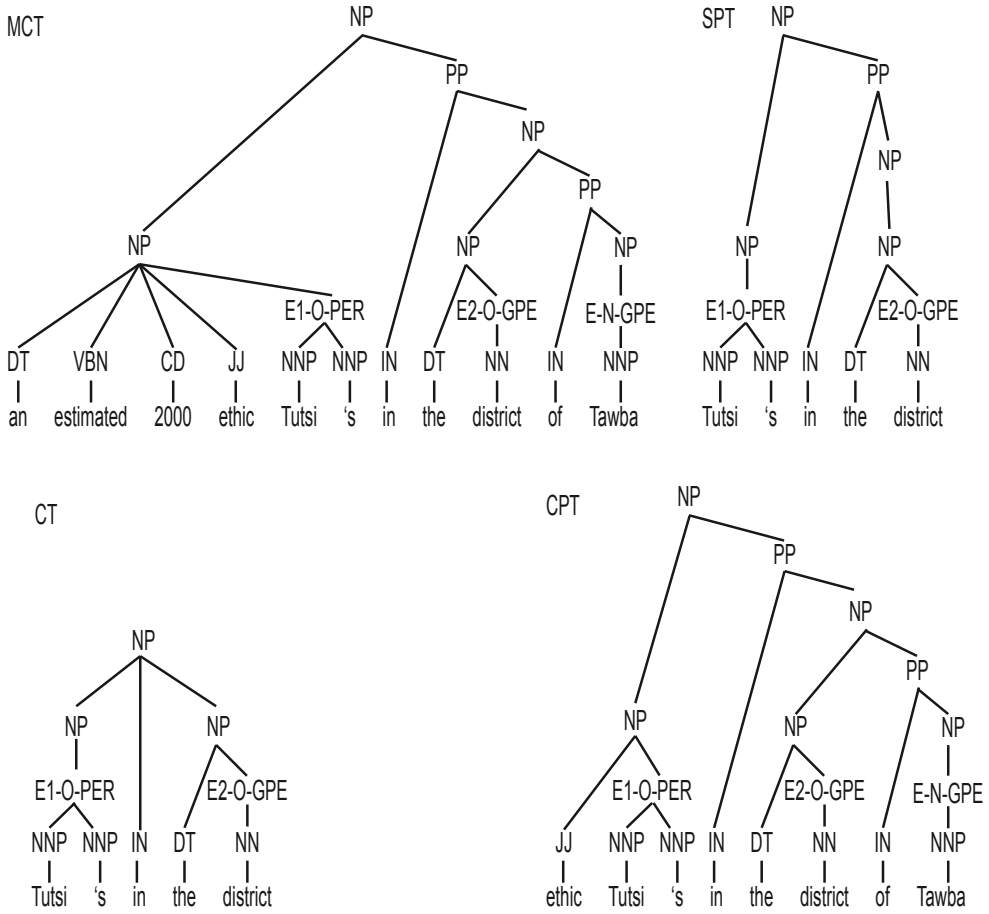



Abbildung 5.7.: Beschnittene Baumformen nach (ZZS06)

- Der *Minimum Complete Tree* (MCT) wird gebildet, indem der kleinste gemeinsame Elternknoten der beiden Entitäten der Relation als Wurzelknoten für den neuen Baum genommen wird. Alle Strukturen unterhalb dieses Wurzelknotens bleiben erhalten.
- Der *Shortest Path Tree* (SPT) wird deutlich stärker beschnitten. Er enthält nur die Knoten, die auf dem kürzesten Pfad im Baum zwischen den Entitäten liegen. Auch Blattknoten zwischen den Entitäten bleiben erhalten.
- Beim *Chunking Tree* (CT) werden die inneren Strukturen des SPT entfernt. Es bleiben nur der Wurzelknoten, die Konstituentenknoten sowie die unterste Ebene des Baumes erhalten.
- Der *Context-Sensitive Path Tree* (CPT) erhält einen Teil der Umgebung der Entitäten. Dazu werden der linke benachbarte Knoten von Entität1 sowie der rechte

benachbarte Knoten von Entität2 zum SPT hinzugefügt.

Als Testdatensätze wurden die englischen Korpora der *ACE* 2003- sowie der *ACE* 2004-Konferenz genutzt. Für diese Testdaten hat es sich gezeigt, dass derjenige Baum, der nur die beiden Entitäten sowie die dazwischen liegenden Knoten berücksichtigt (*Shortest Path Tree*, SPT), den besten f-measure Wert (61,87) erzielte. Durch die Integration von Entity Informationen und semantischen Features in den Blattknoten konnte das f-measure auf 69,02 gesteigert werden.

### 5.2.1. Dynamische Auswahl der Baumform

Von Zhou et al. (ZZJZ07) wurde dieses Konzept abermals verbessert und erweitert. Die vorhergehenden Versuche hatten gezeigt, dass die Verwendung der Baumform *SPT* die besten Ergebnisse lieferte. Entgegen der Erwartungen war der kontext-sensitive Baum, der zusätzliche Informationen über die Umgebung der Entitäten enthielt, weniger erfolgreich. Zhou et al. zeigen, dass die Ursache darin liegt, dass sich die *ACE*-Relationen in fünf Kategorien aufteilen lassen(LDC04b).
Bei vier dieser Kategorien ist die komplette Relation innerhalb einer Nominalphrase aufzufinden wie im folgenden Beispiel:
America's Department of Defense.
[EMP-ORG.Subsidiary("America's Department of Defense", "America")]

Die fünfte Kategorie enthält diejenigen Relationen, deren Partner über ein Prädikat miteinander in Verbindung stehen:
the crowd trapped inside the compartment...
[PHYS.Located("the crowd trapped inside the compartment", "the compartment")]

Nutzt man bei den ersten vier Kategorien den kontextsensitiven Baum, so verschlechtert sich das Gesamtergebnis, da durch überflüssige Informationen zusätzliche Störungen eingebracht werden. Bei der fünften Kategorie allerdings ist für über 70% der Fälle die Angabe des Prädikats entscheidend für den Erfolg der Erkennung.
Zhou veränderte daher in seinem Experiment die Auswahl des Baumtyps. Je nach Art des vorliegenden vollständigen Baums wählte er entweder den *SPT* oder eine Form, die um das zugehörige Prädikat erweitert wurde.

# 6. Kernmethoden zur Relationserkennung

Kernmethoden bieten vielversprechende Möglichkeiten, die Strukturen, die in sprachlichen Konstrukten enthalten sind, auszuwerten und für das Auffinden von Relationen zu nutzen. Die Relationserkennung mit Hilfe von Kern-Methoden macht es allerdings erforderlich, die Aufgabe so zu formulieren, dass sie als Klassifikationsaufgabe betrachtet werden kann. Im folgenden Kapitel sollen die Grundlagen für auf Kernmethoden basierende Lernverfahren sowie verschiedene Kernfunktionen vorgestellt werden.

## 6.1. Relationserkennung als Klassifikationsaufgabe

Häufige Aufgabe des maschinellen Lernens ist die Klassifikation. Das Ziel ist dabei, Objekte in Klassen einzuordnen. Im einfachsten Fall existieren genau zwei Klassen; dies wird als binäres Klassifikationsproblem bezeichnet. Auch das Finden von Relationen lässt sich als binäre Klassifikationsaufgabe auffassen. Dazu werden aus einem Text Kandidaten gebildet, die theoretisch eine Relation enthalten können. Relationskandidaten können genau zwei Zustände haben. Sie können wahr sein, d. h. tatsächlich eine Relation beschreiben, oder falsch sein, d. h. zwischen den im Relationskandidaten gespeicherten Entitäten besteht keine Verbindung. Aufgabe des Klassifikators ist es, diese beiden Klassen von Relationskandidaten zu trennen.
Allgemein kann eine Klassifikationsaufgabe formal auf folgende Weise beschrieben werden:

$$\{(x_1, y_1), \ldots, (x_m, y_m) \in X \times \{+1, -1\}\}$$

Dabei ist $X$ eine nichtleere Menge (Instanzenraum), aus der die Instanzen $x_i$ stammen. Instanzen sind die Eingaben, die der Beschreibung eines einzelnen Falles bzw. einer Beobachtung entsprechen. Die $y_i$-Werte sind das zugehörige Label, also die Klasse, der das Objekt zugehörig ist.
Um Vorhersagen über die Kategoriezugehörigkeit neuer Objekte zu treffen, gilt es, eine Entscheidungsfunktion $h : X \to Y$ zu finden, die einem bisher unbekannten Objekt $x_i \in X$ ein passendes Label $y_i \in Y$ zuordnet.
Die Art der Instanzen ist nicht beschränkt, die Objekte der Menge $X$ können beliebiger Natur sein.

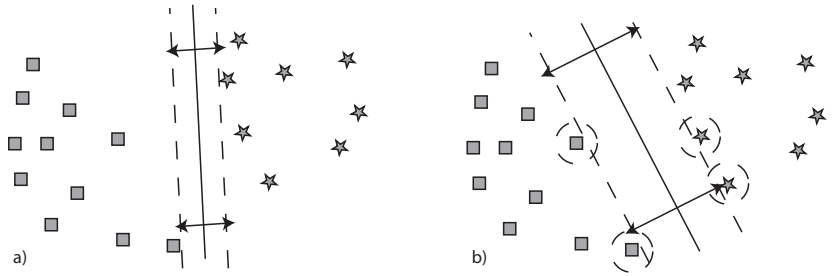



Abbildung 6.1.: Konstruktion einer trennenden Hyperebene zwischen zwei Klassen von Punkten. In a) wurde eine beliebige Trennline zwischen den Mengen gezogen, während bei b) versucht wird, einen maximalen Abstand zwischen den Klassen zu schaffen. Die Stützvektoren sind eingekreist.

Zur Lösung von Klassifikationsproblemen wurden zahlreiche unterschiedliche Verfahren entwickelt. Entscheidungsbäume können schnelle Lösungen liefern, indem jedes Blatt der Zuordnung zu einer Kategorie entspricht und jede Verzweigung einer Aussage über Werte von Features. Liegen die Daten in Form dünn besetzter Featurevektoren mit hoher Dimensionalität vor, werden häufig lineare Klassifikatoren zur effizienten Problemlösung eingesetzt. Des Weiteren gibt es quadratische oder statistische Klassifikatoren, k-nearest neighbour-Verfahren, Hidden Markov Models etc. Die Effizienz der einzelnen Verfahren hängt stark von dem Charakter der zu untersuchenden Daten ab. Jeder Algorithmus hat Vor- und Nachteile, die es abzuwägen gilt; ein für alle Probleme gleichermassen optimales Verfahren gibt es nicht.

## 6.2. Support Vector Machines

Im Rahmen dieser Arbeit wird eine *Support Vector Machine* (SVM) genutzt, die zur Familie der linearen Klassifikatoren gehört.
Eine Support Vector Machine ist ein Algorithmus, der versucht, eine optimale Hyperebene zwischen den zu separierenden Klassen von Trainingsbeispielen zu berechnen. Eine optimale Hyperebene ist hierbei eine lineare Entscheidungsfunktion, die einen maximalen Abstand zwischen den Beispielen beider Klassen im Vektorraum einhält. Das Konzept wurde von Cortes und Vapnik (CV95) veröffentlicht.
Das Lernverfahren erfordert eine Repräsentation der Eingabedaten als Aufzählung ihrer Features. Jedes Objekt wird in eine Menge von Features $f_1 \dots f_N$ umgewandelt, wodurch ein $N$-dimensionaler Vektor entsteht. Dies kann für Texte z. B. eine Vektordarstellung sein, wie sie in 3.3.1 beschrieben wird. Diese Vektoren spannen einen Vektorraum auf. In diesem bildet die SVM zunächst ein Modell, in dem versucht wird, die Datenpunkte durch eine Hyperebene möglichst gut voneinander zu trennen (Abb. 6.1). Dies erfolgt un-

ter Zuhilfenahme der Berechnung der Ähnlichkeiten der einzelnen Vektoren. Können die Datenpunkte einer Datenmenge erfolgreich voneinander getrennt werden, ist es möglich, im Anschluss gleichartige, bisher unbekannte Daten zu untersuchen und in das gelernte Modell einzuordnen. Dadurch werden sie automatisch einer Klasse zugeteilt. Zur Konstruktion der Hyperebene werden sogenannte Stützvektoren herangezogen. Dies erfolgt in der Annahme, dass nur sich im „Grenzgebiet“ befindende Punkte einen Einfluss auf die Lage der Trennlinie haben. Weiter entfernte Punkte können aus der Betrachtung entfernt werden. Durch Einhaltung des maximalen Abstandes zu diesen Grenzpunkten wird erreicht, dass spätere Datenpunkte bei Anwendung des Modells zuverlässiger der richtigen Kategorie zugeordnet werden können.
Um den Abstand der Datenpunkte im Merkmalsraum zu berechnen, wird das Skalarprodukt verwendet. Dieses wird in der Regel jedoch nicht im klassischen Sinne, sondern mit Hilfe der in Abschnitt 6.3 beschriebenen Kernfunktion berechnet. Die Dimensionalität des Raumes, in dem diese Trennung stattfindet, hat keinen negativen Einfluss auf die Leistungsfähigkeit des Verfahrens. Auch das Arbeiten in sehr hochdimensionalen Räumen, wie sie von den Kernfunktionen erzeugt werden, ist daher möglich.

## 6.3. Kernfunktionen

Wenn die Klassen der Beispiele einer Datenmenge durch eine Hyperebene im Beispielraum voneinander trennbar sind, spricht man von linear separierbaren Daten. Dies ist in der Regel jedoch nicht der Fall. Eine Möglichkeit, dieses Problem zu überwinden, ist die Transformation der Daten in einen höherdimensionalen Raum. Diesen nennt man *Merkmalsraum* (Abb. 6.2). Die Dimension des Merkmalsraumes muss groß genug sein, dass die Beispiele in ihm durch eine Hyperebene separiert werden können. Der Nachteil, den diese Transformation mit sich bringt, ist, dass die Berechnung aufgrund der unter Umständen deutlich erhöhten Dimensionalität wesentlich aufwändiger oder sogar undurchführbar wird. Basiert ein Algorithmus auf der Auswertung des Skalarprodukts zwischen den Feature-Vektoren, schafft der sogenannte *Kernel Trick* Abhilfe. Mercers Theorem (Mer09) sagt aus, dass jede kontinuierliche positiv semidefinite Kernfunktion $K(x, z)$ als Skalarprodukt ausgedrückt werden kann. Durch Formulierung einer geeigneten Kernfunktion kann also das Skalarprodukt ersetzt werden. Dadurch wird es unnötig, die Transformation der Daten für die Berechnung tatsächlich durchzuführen. Diese geschieht nur noch innerhalb der Kernfunktion. Innerhalb der Kernfunktion können außerdem die Eigenschaften komplexer strukturierter Objekte direkt für die Berechnung genutzt werden. Daraus ergibt sich ein weiterer Vorteil. Die Merkmale werden durch die Kernfunktion implizit berücksichtigt und die explizite Erzeugung eines Merkmalsvektors kann entfallen. Anstatt also z. B. für einen Baumkern einen hochdimensionalen Vektor

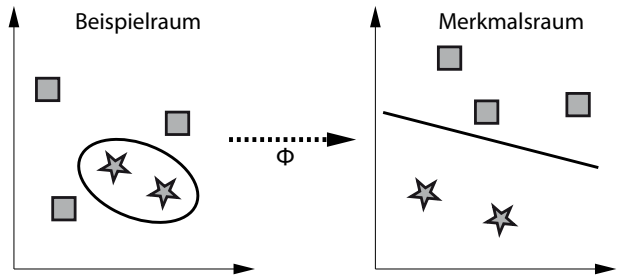



Abbildung 6.2.: Die Beispiele im Beispielraum (links) sind nicht linear separierbar. Deshalb werden mittels der Funktion $\phi$ in einen höherdimensionalen Merkmalsraum (rechts) transformiert, in dem sie durch eine Hyperebene getrennt werden können. (Abb. nach (SS01))

aufzustellen, in dem jedes theoretisch mögliche Baumfragment eine Dimension darstellt, kann die Struktur des Baumes direkt an den Kern übergeben werden.

### 6.3.1. Beschreibung von Kernfunktionen

Eine Funktion, die das innere Produkt zwischen den in den Merkmalsraum abgebildeten Beispielen berechnet, nennt man Kernfunktion.

$$\phi : X \rightarrow H$$

$$K(x, z) = \langle \phi(x) \cdot \phi(z) \rangle$$

- $K(x, z)$ ist kontinuierlich
- $K(x, z)$ ist symmetrisch
- $K(x, z)$ ist positiv definit

Der Merkmalsraum $H$ ist ein *Hilbertraum*.

### 6.3.2. Kernfunktionen für beliebige Objekte

In der Statistik oder bei der Mustererkennung werden oft diskrete Strukturen wie Graphen, Bäume oder Zeichenketten untersucht. Dazu ist eine Repräsentation solch einer Struktur $x \in X$ durch Merkmalsvektoren erforderlich. $\phi_1(x), \phi_2(x), \ldots, \phi_n(x)$. Enthält ein Objekt endlich viele Features, so entspricht dieser Prozess der Abbildung $X \rightarrow \Re^d$ in den d-dimensionalen Euklidischen Raum. Bei unendlich vielen Features enspricht er einer Abbildung in den Hilbertraum. Man erhält eine Darstellung von $x$ als unendliche

Reihe, ähnlich wie bei einer Fouriertransformation. Diese Transformation soll mit Hilfe einer bestimmten Art von Kernfunktionen erfolgen, die *Faltungskerne* (Convolution Kernel) genannt werden. Das Konzept wurde von David Haussler (Hau99) beschrieben und funktioniert wie folgt:
Gegeben ist ein Objekt $x \in X$. Dieses Objekt $x$ ist zerlegbar in eine Menge kleinerer Strukturen. $x$ lässt sich dann als Vektor darstellen, durch eine Aufzählung seiner Komponenten: $\vec{x} = x_1, \ldots, x_D$. Diese Komponenten stammen aus einer Menge $X_D$, die alle möglichen Teilstrukturen von Objekten der Menge $X$ enthält. Weiterhin lässt sich eine Relation $R(\vec{x}, x)$ definieren, die genau dann wahr ist, wenn $x_1, \ldots, x_D$ die Teile von $x$ sind. Zusätzlich sei $R^{-1}(x) = \{\vec{x} : R(\vec{x}, x)\}$.
Im Falle von Bäumen können diese Teilstrukturen die Teil- oder Unterbäume sein. Die Menge $X_D$ entspricht dann allen möglichen Baumstrukturen, die theoretisch existieren können. $R(\vec{x}, x)$ ist genau dann wahr, wenn der Baum $x$ die Unterbäume $x_1, \ldots, x_D \in X_D$ enthält.
Es soll nun die Ähnlichkeit zweier Objekte $x, z \in X$ bestimmt werden. Diese sind jeweils zerlegbar in kleinere Strukturen $\vec{x} = x_1, ..., x_D$ und $\vec{x} = z_1, ..., z_D$. Es gibt eine Kernfunktion $K_i(x_d, z_d)$, die die Ähnlichkeit zwischen den kleineren Strukturen $x_d$ und $z_d$ berechnen kann. Um die Gesamtähnlichkeit $K(x, z)$ zwischen $x$ und $z$ bestimmen zu können, kann man nun eine Funktion bilden, die nur auf Berechnung der Ähnlichkeit der Teilstrukturen basiert:

$$K(x, z) = \sum_{\{\vec{x}|R(\vec{x},x)\}} \sum_{\{\vec{z}|R(\vec{z},z)\}} \prod_{i=1}^{D} K_i(x_i, z_i) \tag{6.1}$$

Die Klasse von Kernen, die dabei entsteht, generalisiert die Familie von Radialbasis-Kernen und einfachen exponentiellen Kernen. Sie erfüllt eine Vielzahl nützlicher mathematischer Eigenschaften, die aus der Verwandtschaft zur Klasse der positiv definiten Funktionen begründet sind.

### 6.3.3. Kombination von Kernfunktionen

Über der Menge $X \times X$ gilt für die Klasse der Faltungskerne unter anderem Abgeschlossenheit gegenüber Multiplikation und Addition. Dies ermöglicht es, beliebige Faltungskerne zu kombinieren und als Ergebniss einen neuen Faltungskern zu erhalten. Der entstehende Kern aus der Kombination einzelner Kerne wird *Composite Kernel* genannt.

$$K(x, z) = K_1(x, z) \circ K_2(x, z) \tag{6.2}$$

Von Zhang et. al. (ZZSZ06) wurde z. B. ein Composite Kernel aus einem linearen Kern und einem Kern für Parsebäume zur Relationsextraktion angewendet. Auf diese Weise

können sowohl flache als auch strukturierte Features des Datensatzes auf effiziente Weise berücksichtigt werden. Je nach Kombinationsmethode und Gewichtung kann Einfluss genommen werden, welche Eigenschaften der Daten besonders stark beachtet werden sollen. Der erste Kern ist in diesem Fall ein einfacher linearer Kern $K_L(R_1, R_2)$. Er berechnet die Anzahl der übereinstimmenden Features zweier Vektoren. Als zweiter Kern wird ein Baumkern $K(T_1, T_2)$ verwendet.

Die Kombination erfolgt auf zwei verschiedene Arten:
1.) Lineare Kombination

$$K_1(R_x, R_z) = \alpha \cdot \hat{K}_L(R_x, R_z) + (1 - \alpha) \cdot \hat{K}(T_x, T_z) \tag{6.3}$$

2.) Polynomielle Erweiterung

$$K_1(R_x, R_z) = \alpha \cdot \hat{K}_L^P(R_x, R_z) + (1 - \alpha) \cdot \hat{K}(T_x, T_z) \tag{6.4}$$

$\hat{K}(\cdot, \cdot)$ ist der normalisierte Kern $K(\cdot, \cdot)$.
$K^P(\cdot, \cdot)$ ist die polinomielle Erweiterung 2. Grades, also $K^P(\cdot, \cdot) = (K(\cdot, \cdot) + 1)^2$.

### 6.3.4. Normalisierung von Kernfunktionen

Vor der Kombination von Kernfunktionen ist es sinnvoll, diese zu normalisieren. Aufgrund unterschiedlicher Wertebereiche kann es sonst passieren, dass ein Kern den anderen vollständig überlagern würde. Das Ergebnis einer Kernfunktion kann folgendermaßen im Kernraum normalisiert werden :

$$K'(T_x, T_z) = \frac{K(T_x, T_z)}{\sqrt{K(T_x, T_x) \times K(T_z, T_z)}} \tag{6.5}$$

Normalisierung ist ebenfalls notwenig bei Kernen, die die Anzahl gemeinsamer Unterstrukturen zählen, wie es z. B. bei den Baumkernen der Fall ist. Der Einfluss unterschiedlicher Baumgrößen muss ausgeglichen werden. Würde man keine Normalisierung durchführen, ergäbe sich folgender Effekt: Die Berechnung der Ähnlichkeit eines kleinen Baumes mit nur fünf Unterstrukturen zu sich selbst hätte als Ergebnis den Wert 5. Die Berechnung für einen großen Baum, mit 50 Unterstrukturen, ergäbe dementsprechend den Wert 50. Diese beiden Werte sind nicht sinnvoll miteinander vergleichbar. Durch Normalisierung wird das Ergebnis in beiden Fällen zu 1.

## 6.4. Kerne für Baumstrukturen

Je ähnlicher die Strukturen zweier Sätze sich sind, desto wahrscheinlicher ist, dass sie auch inhaltliche Übereinstimmungen haben. Dieses kann für die Relationserkennung genutzt werden. Um festzustellen, ob ein neuer, bisher unbekannter Satz eine der gesuchten Relationen enthält, vergleicht man ihn mit bekannten Beispielen aus der Trainingsmenge und bestimmt deren Ähnlichkeit. Das Verfahren zur Bestimmung des Ähnlichkeitsmaßes ist hierbei ausschlaggebend für den Erfolg dieses Unterfangens. Der klassische Ansatz ist, für jeden Relationskandidaten einen Merkmalsvektor aufzustellen. Die Merkmale, die in diesem enthalten sind, werden im Vorfeld festgelegt und aus den Sätzen generiert. Es wird oft eine Vielzahl verschiedener linguistischer Merkmale lexikalischer, syntaktischer oder semantischer Natur herangezogen. Auf diese Weise werden allerdings nur diejenigen Informationen beachtet, die vorher explizit erzeugt und in den Vektor eingebracht wurden.
Baumkerne sind ein weiterer vielversprechender Ansatz, um syntaktische Informationen in natürlichsprachlichen Texten zu finden. Sie nutzen die von Haussler (Hau99) vorgestellten Eigenschaften der Convolution Kernels, um strukturierte Informationen, die in Sätzen enthalten sind, direkt zu verarbeiten und deren Ähnlichkeit zu bestimmen. Bäume von Sätzen können auf verschiedene Weise gebildet werden; einige Baumformen sind in Abschnitt 5.1 vorgestellt. Durch Beschneiden (Pruning) des Baumes können vor der Berechnung Zweige entfernt werden, sodass nur Ausschnitte von Bäumen betrachtet werden, die zur Lösung einer bestimmten Aufgabe als notwendig erscheinen. Im Gegensatz zum Pruning, das unwichtige Daten entfernt, können an anderer Stelle des Baums auch zusätzliche wichtige Informationen eingefügt werden. Durch die Anwendung der Kernfunktion wird implizit eine große Anzahl an Merkmalen der Strukturen berücksichtigt, deren explizite Aufzählung und Darstellung als Vektor aufgrund von Speicherplatzbedarf und Rechenaufwand nicht praktikabel wäre.

### 6.4.1. Verschiedene Kernfunktionen zur Relation Extraction

Zahlreiche Kerne über verschiedene Baumstrukturen oder Graphen, die aus Sätzen von natürlicher Sprache gebildet wurden, sind untersucht worden, um Aufgabenstellungen im Bereich NLP zu lösen. Die Zielsetzungen sind dabei unterschiedlich. Häufig geht es darum, Relationen in einem Text zu finden. Aber auch andere Aufgaben, wie die Vorhersage der Prädikat-Argument-Struktur, lassen sich verwirklichen. Die verwendeten Kernfunktionen sind meist zwar eng verwandt mit den von Haussler vorgestellten Convolution Kernels, erfüllen deren Definition jedoch nicht immer vollständig.
Von Vishwanathan und Smola (VS02) wird eine allgemeine Kernfunktion für Bäume über deren Unterbäume (siehe 5.1.1) vorgestellt. Zelenko et al. (ZAR03) haben Shal-

low Parse Trees und eine eigene Kernfunktion zur Relation Extraction genutzt. Culotta und Sorensen (CS04) verwenden eine Kernfunktion über Abhängigkeitsbäume zur Relationserkennung. Bunesco und Mooney nutzen eine Kernfunktion über den kürzesten Pfad zwischen zwei Entities im Abhängigkeitsgraphen in (BM05) und einen Kern über die umgebenden Wörter von Relationspartnern in (BM06). Zhao und Grishman (ZG05) verwenden einen Composite Kernel, bestehend aus mehreren Kernfunktionen, die in unterschiedlichen Kombinationen für die *ACE*-Relation-Detection Aufgabe getestet werden.

### 6.4.2. Eine Kernfunktion für Parsebäume

Eine Kernfunktion für Parsebäume (siehe 5.1) stammt von Collins und Duffy (CD01). Der Kern betrachtet die in Abschnitt 5.1.2 beschriebenen grammatischen Produktionsregeln der Knoten. Verwendet wurde die Kernfunktion zum Parsing von Nachrichtentexten sowie zur Named Entity-Erkennung mit vielversprechenden Ergebnissen.
Zunächst ist es erforderlich, die in einer Trainingsmenge enthalteten Bäume $T_i$ als Vektoren $\boldsymbol{h}(T)$ darstellen zu können. Die Komponenten des Vektors werden aus der gesamten Menge der verschiedenen Teilbäume $(1 \dots n)$ der Trainingsbeispiele gebildet (siehe 5.1.1). Die $i$-te Komponente des Vektors ist die Häufigkeit des $i$-ten Baumfragments in $T$. Aufgrund des exponentiellen Wachstums der Anzahl der Baumfragmente ist es allerdings nicht sinnvoll, diese explizit aufzuzählen. Mithilfe einer Kernfunktion lässt sich die Berechnung dennoch durchführen.
Es wird eine Funktion definiert, die als Ergebnis die Anzahl des $i$-ten Baumfragments des Baumes zurückgibt: $h_i(T)$. Dadurch ergibt sich die folgende Darstellung:

$$\boldsymbol{h}(T) = h_1(T), h_2(T), \dots, h_n(T)) \tag{6.6}$$

Für die Kernfunktion zur Berechnung der Ähnlichkeit zweier Bäume $T_i$ und $T_j$ folgt daraus:

$$K(T_i, T_j) = \boldsymbol{h}(T_i) \cdot \boldsymbol{h}(T_j) \tag{6.7}$$

Sie basiert auf der zuvor definierten Funktion zur impliziten Darstellung der Vektoren $\boldsymbol{h}(T_i)$ und $\boldsymbol{h}(T_j)$.

Zur effizienten Berechnung von $\boldsymbol{h}(T)$ wird eine Indikatorfunktion $I_i(n)$ benötigt. Diese ist folgendermaßen definiert:

$$I_i(n) = \begin{cases} 1 & \text{falls Unterbaum } i \text{ in } n \text{ verwurzelt ist} \\ 0 & \text{sonst} \end{cases} \tag{6.8}$$

$N_1$ bzw. $N_2$ enspricht der Menge der Knoten in $T_i$ und $T_j$. Somit gilt:

$$h_i(T_1) = \sum_{n_1 \in N_1} I_i(n_1) \tag{6.9}$$

$$\begin{aligned} K(T_1, T_2) &= \boldsymbol{h}(T_1) \cdot \boldsymbol{h}(T_2) \\ &= \sum_i h_i(T_1) h_i(T_2) \\ &= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \sum_i I_i(n_1) I_i(n_2) \\ &= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} C(n_1, n_2) \end{aligned} \tag{6.10}$$

Definition $C(n_1, n_2)$:

$$C(n_1, n_2) = \sum_i I_i(n_1) I_i(n_2) \tag{6.11}$$

Mit Hilfe einer Fallunterscheidung kann $C(n_1, n_2)$ in polynomieller Zeit rekursiv berechnet werden:

- Falls Produktionsregeln in $n_1$ und $n_2$ unterschiedlich:

$$C(n_1, n_2) = 0 \tag{6.12}$$

- Falls Produktionsregeln in $n_1$ und $n_2$ gleich und $n_1$ und $n_2$ präterminal (Knoten, deren Kinder Blattknoten sind):

$$C(n_1, n_2) = 1 \tag{6.13}$$

- Falls Produktionsregeln in $n_1$ und $n_2$ gleich und $n_1$ und $n_2$ nicht-preterminal (innere Knoten, die keinen Blattknoten als Kind haben):

$$C(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (1 + C(ch(n_1, j), ch(n_2, j))) \tag{6.14}$$

$nc(n_1)$ = Anzahl der Kinder von Knoten $n_1$ (number of children)
$ch(n_1, i) = i$. Kind von Knoten $n_1$

Die Komplexität dieser Berechung ist somit nur noch abhängig von den in $T_i$ und $T_j$ enthaltenen Knoten und enspricht $O(|N_1| \cdot |N_2|)$

Da der Wert von $K(T_i, T_j)$ stark abhängig von der Größe der Bäume $T_i$ und $T_j$ ist, ist eine Normalisierung im Kernelraum wie zuvor beschrieben erforderlich.

Ein weiterer Faktor mit großem Einfluss auf die Berechnung sind große Baumfragmente, die in beiden Bäumen übereinstimmen. Aufgrund der rekursiven Berechnung erhöhen sie den Wert eines Kerns sehr stark und können dadurch andere Merkmale überdecken und die Vergleichbarkeit erschweren. Am deutlichsten wird dies, wenn ein Kern auf zwei Kopien desselben Baums angewendet wird und das Ergebnis der Berechnung im Vergleich zur Anwendung auf unterschiedlichen Bäumen betrachtet wird. Für dieses Problem existieren zwei Lösungsansätze. Eine Möglichkeit ist, die Tiefe von Baumfragmenten, die berücksichtig werden, zu begrenzen. Dieses wird durch einen Filterschritt bei der Bestimmung der übereinstimmenden Baumfragmente erreicht. Die Alternative dazu ist die Anwendung eines Dämpfungsfaktors. Dazu wird eine Variable $0 < \lambda < 1$ eingeführt. Im Vergleich zu der zuvor vorgestellten Berechnung ergeben sich folgende Unterschiede:

- Falls Produktionsregeln in $n_1$ und $n_2$ gleich und $n_1$ und $n_2$ preterminal:

$$C(n_1, n_2) = \lambda \tag{6.15}$$

- Falls Produktionsregeln in $n_1$ und $n_2$ gleich und $n_1$ und $n_2$ nicht-preterminal:

$$C(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + C(ch(n_1, j), ch(n_2, j))) \tag{6.16}$$

Dadurch wird eine exponentielle Dämpfung der großen Baumfragmente erreicht. Die entsprechende Kernfunktion sieht wie folgt aus:

$$\boldsymbol{h}(T_1) \cdot \boldsymbol{h}(T_2) = \sum_i \lambda^{size_i} h_i(T_1) h_i(T_2) \tag{6.17}$$

wobei $size_i$ der Anzahl der Produktionsregeln, die in dem $i$-ten Baumfragment enthalten sind, entspricht.

### 6.4.3. Fast Tree Kernel

Von Alessandro Moschitti (Mos06) wurde eine Erweiterung der Kernfunktion von Collins und Duffy vorgestellt. Anwendungsbereich ist die Vorhersage der Prädikat-Argument-Struktur. Die Berechnung der Kernfunktion wurde von Moschitti durch geschickte Vorverarbeitung effizienter gestaltet. Es wurde außerdem eine Variable $\sigma$ eingeführt, die es erlaubt, eine einfache Auswahl zu treffen, ob Teil- oder Unterbäume (siehe Abschnitt 5.1.1) zur Berechnung genutzt werden sollen.

#### Auswahl der Baumstruktur

Zur Auswahl an Baumstrukturen stehen die von Vishwanathan und Smola genutzen ST (siehe 5.1.1) und die SST (siehe 5.1.1), wie sie im Kern von Collins und Duffy genutzt werden. Die Strukturinformationen, die von Teilbäumen (ST) bereitgestellt werden, enthalten weniger Details als die von Unterbäumen (SST). Die Struktur eines Baumes wird gröber abgebildet, da bei einem Vergleich zweier Bäume komplette Teilbäume bis hinab in die Blätter identisch sein müssen und nicht bereits eine gemeinsame Abfolge innerer Knoten als Übereinstimmung gezählt wird. Mithilfe des Parameters $\sigma$ können Untersuchungen darüber angestellt werden, welche der beiden Varianten zu einer höheren Genauigkeit in verschiedenen Aufgabenbereichen führt. Gleichung (6.14) bzw. (6.16) wird dazu wie folgt verändert:

$$C(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (\sigma + C(ch(n_1, j), ch(n_2, j))) \tag{6.18}$$

#### Beschleunigung der Berechnung

Eine deutliche Beschleunigung der Berechnung wird von Moschitti durch Vorbereitung der Daten erreicht. Es wird ausgenutzt, dass nur Knoten beider Bäume, deren Produktionsregeln identisch sind, bei der Berechnung Bedeutung haben. Zunächst wird für die beiden Bäume jeweils eine Liste der Produktionsregeln aller Knoten aufgestellt. Diese werden anschließend sortiert. Im nächsten Schritt werden die beiden sortierten Listen parallel durchlaufen. Stimmen die Produktionsregeln zweier Knoten der Bäume überein, so bilden sie ein Paar und werden abgespeichert. Knoten ohne eine Entsprechung werden verworfen. Aufgrund der vorsortierten Listen gelingt diese Paarung sehr effizient. Die nachfolgende Kernberechnung muss nun ausschließlich für die zuvor gefundenen Knotenpaare durchgeführt werden. Das Endergebnis wird durch diese Vorgehensweise nicht beeinflusst, da der Wert aller übrigen Berechnungen ohnehin 0 wäre. Im Gegensatz zum unmodifizierten Kern, der im schlechtesten Falle eine quadratische Laufzeit hat, kann durch diese vorhergehende Auswahl eine durchschnittlich lineare Laufzeit erreicht werden.

Diese Verbesserung der Laufzeiteigenschaften wird in der für diese Diplomarbeit entwickelten eigenen Java-Implementierung der Convolution Tree Kernels ebenfalls verwendet.

### 6.4.4. Context Sensitive Convolution Tree Kernel

Eine weitere Veränderung an dem Baumkern wurde von Zhou et al. (ZZJZ07) entwickelt. Die bisher vorgestellten Implementierung waren kontextfrei; das bedeutet, dass

die Stelle, an der sich ein Teilbaum innerhalb des Gesamtbaumes befunden hat, nicht für die Berechnung berücksichtigt wurde. Es wurden nur die Strukturen innerhalb der Teilbäume verglichen, aber keinerlei Informationen über deren Umgebung mit einbezogen. In einem früheren Kern von Culotta and Sorensen (CS04) floss der Pfad vom Wurzelknoten des Baumes zum Wurzelknoten des kontextfreien Teilbaumes mit in die Berechnung ein. Diese Idee wurde hier wieder aufgegriffen. Ausgehend von den Wurzelknoten der Teilbäume findet bei der Erzeugung der Produktionsregeln ein Rückblick auf die Vorgängerknoten statt. Dieser Pfad fließt dabei in die linke Seite der Produktionsregel ein. Über den Parameter $m$ kann die maximale Tiefe angegeben werden, die der Kern berücksichtigen soll. Eine Abbildung zur Bildung dieser kontextsensitiven Produktionsregeln findet sich in Abschnitt 5.1.2. Die Ergebnisse der Berechnung der Übereinstimmung auf den verschiedenen Tiefen $1 \dots m$ werden vom Kern aufsummiert. Im Fall $m = 1$ entspricht das Ergebnis genau der Berechnung der ursprünglichen kontextfreien Funktion. Die entstehende kontextsensitive Kernfunktion sieht wie folgt aus:

$$K_C(T[1], T[2]) = \sum_{i=1}^{m} \sum_{n_1^i[1] \in N_i^1[1], n_1^i[2] \in N_i^1[2]} C(n_1^i[1], n_1^i[2]) \tag{6.19}$$

- $m$ ist die maximale zu berücksichtigende Länge des Wurzelknotenpfades.
- $n_1^i[j]$ ist ein Knoten des Baums $j$, dessen Produktionsregel unter Beachtung des Pfades über $i$ Vorgänger gebildet wurde. $n_1[j]$ enspricht dabei dem Wurzelknoten des kontextfreien Teilbaumes. Der Vorfahre von $n_k[j]$ ist $n_{k+1}[j]$.
- $N_1^i[j]$ ist die Menge aller Knoten mit Produktionsregeln, gebildet unter Berücksichtigung von $i$ Vorfahren.

## 6.5. Composite Kernel

Neben dem Einfluss der Baumform wurden von Zhang et. al. (ZZSZ06) auch Features der Entities, also Eigenschaften der beiden Relationspartner, zur Verbesserung der Erkennungsrate mit einbezogen. Dazu wurde ein *Composite Kernel* (siehe Abschnitt 6.5) entwickelt. Dieser wird aus einem linearen Featurekern und dem unter 6.4.2 vorgestellten Baumkern zusammengesetzt.
Der lineare Kern zählt die übereinstimmenden Features der Entities beider Relationskandidaten und ist folgendermassen definiert:

$$K_L(R_1, R_2) = \sum_{i=1,2} K_E(R_1.E_i, R_2.E_i) \tag{6.20}$$

$$K_E(E_1, E_2) = \sum_j C(E_1.f_j, E_2.f_j) \tag{6.21}$$

$R_x.E_1$ und $R_x.E_2$ sind die Entities des ersten Relationskandidaten.
$E_i.f_j$ ist das $j$-te Feature der Entity $E_i$.
Die Funktion $C(\cdot, \cdot)$ gibt 1 zurück, wenn beide Features identisch sind, sonst 0.

Die Features, die von Zhang et. al. mit dem linearen Kern verglichen werden, entsprechen den *Entity Features*, die in den *ACE 2004*-Daten angegeben sind: *entity headword*, *entity type and subtype*, *mention type* und *LDC mention type*. Der beste Wert, der mit dem Composite Kernel für die *ACE 2004*-Daten erreicht werden konnte, war ein f-measure von $72,1$.

Von (ZZJZ07) wurde ebenfalls ein Composite Kernel eingesetzt. Er kombiniert den *CSCTK* mit einem linearen Kern, dessen Features in (ZSZZ05) vorgestellt wurden.
Versuche ergaben ein f-measure von $75,8$ auf dem *ACE 2004*-Korpus. Interessant ist, dass die Einzelausführung des lineare Kerns ein f-measure von $70,1$ erreichte; der Baumkern alleine erzielte ein f-measure von $73,2$. Das bedeutet, dass beide Kerne unterschiedliche Eigenschaften des Relationskandidaten berücksichtigen, und durch die Kombination ein echter Zugewinn erreicht werden konnte.

# 7. Information Extraction mit RapidMiner

Die Wissensentdeckung und Mustererkennung in Datenmengen erfordert in der Regel eine Verarbeitungskette für die Daten, die zahlreiche Schritte umfasst. Weiterhin sollen zur Erfüllung spezifischer Aufgabenstellungen oft mehrere Lernverfahren auf ihre Eignung hin untersucht und bestmöglich auf deren besondere Anforderungen angepasst werden können. Um ein universelles, wiederverwendbares Framework für solche Arbeitsabläufe zu schaffen, wurde am Lehrstuhl für künstliche Intelligenz der Technischen Universität Dortmund die Software *RapidMiner* (ehemals *YALE*) entwickelt (MWK+06). Es handelt sich um ein universell einsetzbares Tool, in dem eine Vielzahl maschineller Lernverfahren implementiert sind. Der modulare Aufbau sowie die freie Verfügbarkeit des Quellcodes in dem Projektverzeichniss *SourceForge* [1] erlauben eine stetige Erweiterung und Verbesserung der Funktionen durch Programmierer aus aller Welt.

## 7.1. Grundkonzepte

Grundlegende Bestandteile, die bei der Durchführung eines Experiments mit *RapidMiner* betrachtet werden müssen, sind zum einen die Repräsentation der Daten, die untersucht werden sollen, zum anderen die Operatoren, die diese Daten verarbeiten und visualisieren können.

### 7.1.1. Daten

Die Daten werden in *RapidMiner* in der Regel in Form einer Tabelle repräsentiert. Diese wird als *ExampleSet* bezeichnet. Jede Zeile enthält genau ein Beispiel; die zugehörigen Attribute sind in den Spalten des Datensatzes abgelegt. Zahlreiche Dateiformate für den Import werden unterstützt, die direkte Nutzung von verschiedenen Datenbanksystemen ist ebenfalls möglich.
Es wird zwischen normalen und speziellen Attributen unterschieden. Normale Attribute sind die Eigenschaften, die ein Eintrag eines Datensatzes haben kann und die als Merkmale für die Lernaufgabe genutzt werden sollen. Ein spezielles Attribut ist z. B. das Label, das für eine Klassifikationsaufgabe angibt, welcher Klasse das Objekt zuzuordnen ist.

[1] `http://sourceforge.net/`

In einer Zelle der Tabelle kann ein Wert nur in Form einer Zahl oder einer Folge von Zeichen abgelegt werden. Strukturierte Objekte wie z. B. Parsebäume können nicht direkt gespeichert werden und müssen zuvor in eine serialisierte Form gebracht werden. Außerdem werden Strings nicht direkt gespeichert; die Zeichenfolgen werden auf einen *double*-Wert abgebildet und müssen beim Zugriff durch Operatoren zuerst konvertiert werden.
In Abb. 7.1 ist ein Ausschnitt aus einem ExampleSet zu sehen, das für die Relationserkennung erstellt wurde.

ExampleSet (3602 examples, 1 special attribute, 24 regular attributes)

| row no. | Label | w | w2 | parse | newFirmsB... | newFirmsB... | b1 | newFirm |
|---|---|---|---|---|---|---|---|---|
| 79 | false | MAN | Scania | (ROOT (MPN (NE VW) (NE MAN | true | MAN | MAN | jetzt |
| 80 | false | Scania | VW | (ROOT (NUR (S (NP (PDAT Die | false | durch | ? | jetzt |
| 81 | false | MAN | Salzgitter | (ROOT (NUR (MPN (NE Dieter) | false | Standort | Standort | jetzt |
| 82 | false | MAN | VW | (ROOT (NUR (S (NP (KOKOM a | false | und | ? | jetzt |
| 83 | true | BASF | Inmont | (ROOT (NUR (S (NP (ART Die) | false | und | ? | die |
| 84 | false | BASF | United Technologie | (ROOT (NUR (S (NP (ART Die) | false | und | ? | die |
| 85 | false | Inmont | United Technologie | (ROOT (NUR (S (NP (ART Die) | false | und | ? | von |
| 86 | false | BASF | Akzo Nobel | (ROOT (NUR (CS (S (NP (ART I | true | und | ? | von |
| 87 | false | Bayer | BASF | (ROOT (NUR (S (PP (APPRART | false | und | ? | Carl |
| 88 | false | Bayer | Hoechst | (ROOT (NUR (S (PP (APPRART | false | und | ? | Carl |
| 89 | false | Bayer | Agfa | (ROOT (NUR (S (PP (APPRART | false | und | ? | Carl |
| 90 | false | BASF | Bayer | (ROOT (NUR (S (PP (APPRART | false | , | ? | Carl |
| 91 | false | BASF | Hoechst | (ROOT (NUR (S (PP (APPRART | false | , | ? | Bayer |
| 92 | false | BASF | Agfa | (ROOT (NUR (S (PP (APPRART | false | , | ? | Bayer |
| 93 | false | Bayer | Hoechst | (ROOT (NUR (S (PP (APPRART | false | , | ? | Farbwer |

Abbildung 7.1.: Eine Tabelle mit Daten in *RapidMiner*. Die Spalte *Label* enthält das spezielle Attribut, das angibt, ob die zugehörige Zeile ein positives oder ein negatives Lernbeispiel enthält.

### 7.1.2. Operatoren

Die Zusammenstellung einer Lernaufgabe in *RapidMiner* erfolgt über eine Kombination von Operatoren. Jeder Operator erfüllt eine bestimmte Teilaufgabe; aus einer Menge von Eingaben wird eine Menge von Ausgaben erzeugt. Prozessabläufe können auf modulare Weise zusammengestellt werden, indem die Ausgabe eines Operators als Eingabe an einen anderen Operator weitergegeben wird. Da lineare Abfolgen von Operatoren oftmals nicht ausreichend sind, um komplexe Prozesse abzubilden, greifen viele Systeme auf eine Modellierung der Abläufe als gerichtete Graphen zurück. Dies hat den Vorteil der freien Gestaltung, bringt aber gleichzeitig auch Nachteile mit sich . In *RapidMiner* sind die Abläufe deshalb in einer Baumstruktur organisiert (MWK$^+$06). Dies erlaubt die Benutzung von bedingten Verzweigungen und Schleifen, die Ablaufstrukturen bleiben jedoch trotzdem einfach und übersichtlich.
Es gibt verschiedene Typen von Operatoren. Diese umfassen z. B. die Ein- und Ausgabe, die Vorverarbeitung von Daten, maschinelle Lernverfahren und Datenvisualisierung.

Die Auswahl ist sehr umfangreich; in der Basisversion sind bereits zahlreiche häufig genutzte Lernverfahren wie Support Vector Machines, Entscheidungsbäume etc. integriert. Spezielle Plugins, die auf besondere Aufgabenbereiche zugeschnitten sind, sind ebenfalls erhältlich.
Die Zusammenstellung des Operatorbaumes kann über eine komfortable GUI erfolgen. Abgespeichert werden *RapidMiner*-Projekte in einem unkomplizierten XML-Format, das es ermöglicht – auch direkt und ohne Verwendung der GUI – Experimente zusammenzustellen oder zu verändern. Abb. 7.2 zeigt den Ablauf eines einfachen Versuchs zur Klassifikation mit Hilfe einer SVM, für eine Trainings und eine Testdatenmenge.
Jeder Operator verfügt über eine Liste von Parametern, mit denen individuelle Einstellungen festgelegt werden können. Die Anzahl und Art der Parameter ist abhängig von der Implementierung des Operators.

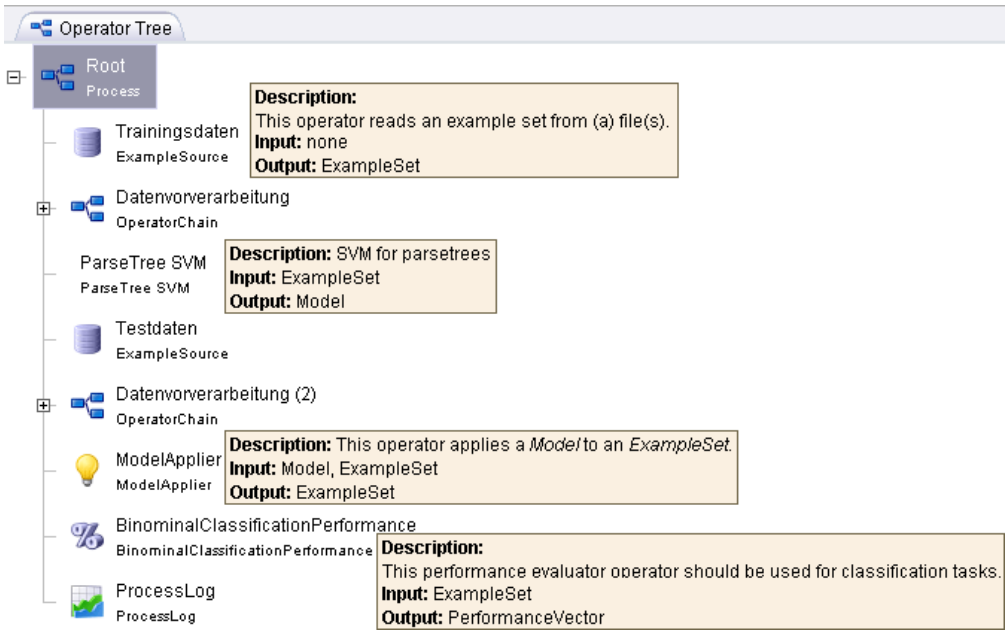



Abbildung 7.2.: Eine Prozesskette in *RapidMiner*. Ein Datensatz wird eingelesen und vorverarbeitet. Im nächsten Schritt wird darauf eine SVM trainiert. Anschließend wird ein Testdatensatz geladen und ebenfalls zuerst vorverarbeitet. Auf diesen wird das zuvor gelernte Modell der SVM angewendet. Die Ergebnisse werden mit dem *PerformanceEvaluator*-Operator ausgewertet und abschließend in eine Logdatei geschrieben.

## 7.2. Information Extraction Plugin

Mit dem *Information Extraction Plugin* soll eine umfangreiche Sammlung von Operatoren zur Verarbeitung und Analyse natürlichsprachlicher Texte in *RapidMiner* zusammen-

gestellt werden. Das Projekt wurde von Felix Jungermann ins Leben gerufen (Jun09). Als erstes Lernverfahren wurde die *Named Entity Recognition* mit Hilfe von *Conditional Random Fields* implementiert. Im Rahmen dieser Diplomarbeit wurde die Möglichkeit zur Verarbeitung und Erkennung von Relationen mit Hilfe von Kernmethoden eingefügt.

### 7.2.1. Arten von Operatoren

Das Plugin enthält eine Reihe von Operatoren, die sich verschiedenen Aufgabenbereichen zuordnen lassen.

#### Eingabe und Tokenisierung

Textdokumente, die in unterschiedlichen Formaten wie html oder pdf vorliegen, können eingelesen und verarbeitet werden. Tokenizer zerlegen die Texte in ihre Elemente. Die auf diese Weise gewonnenen Daten werden in einem *RapidMiner*-ExampleSet in Tabellenform abgelegt. Dabei wird über Referenznummern die Struktur des ursprünglichen Textes im Datensatz mit abgespeichert.

#### Annotierung und Darstellung

Mit Hilfe der Visualisierungsoperatoren können Texte und Strukturen, die bei der Verarbeitung natürlicher Sprache von Bedeutung sind, angezeigt und bei Bedarf mit zusätzlichen Anmerkungen wie *Part-of-Speech*-Tags versehen werden. Dabei wird die interne Tabellenstruktur wieder in einen fortlaufenden Text umgewandelt, um eine einfache Lesbarkeit für den Benutzer zu gewährleisten. Die Annotierungen werden farblich hervorgehoben. Für die Darstellung von Parsebäumen wurde ebenfalls ein Operator integriert.

#### Vorverarbeitung

Für die natürliche Sprache ist die Vorverarbeitung ein besonders wichtiger Schritt, da die Daten zunächst in einer Form vorliegen, die mit klassischen Lernverfahren nicht sinnvoll zu bearbeiten sind. Durch die Generierung neuer Features, das Filtern oder Aufsplitten von Beispielen, die Erzeugung von neuen Strukturen etc. kann ein Datensatz passend zusammengestellt und mit wichtigen Zusatzinformationen angereichert werden, sodass er für ein Lernverfahren optimal aufbereitet ist. Das Information Extraction Plugin enthält zahlreiche Vorverarbeitungsschritte, die auf verschiedenen Ebenen eines Textes agieren können. Die Erzeugung von *n-grammen*, die wichtige Features z. B. für die NER darstellen, betrachtet die einzelnen Wörter sowie deren unmittelbare Umgebung. Andere Operatoren wie der *RelationCandidateGenerator* bearbeiten ganze Sätze und entnehmen diesen alle Paare von *Named Entities*, zwischen denen eine Relation bestehen kann.

#### Lernverfahren

Als Lernverfahren stehen die Erkennung von *Named Entities* mittels *Conditional Random Fields* sowie die Relation Extraction mittels Kernmethoden und *SVM* zur Verfügung. Die Lernverfahren erzeugen Modelle. Mit Hilfe der Modelle können später neue, unbekannte Texte verarbeitetet werden. Außerdem können die Lernverfahren mit verschiedenen Parametern getestet und auf ihre Eignung für verschiedene Aufgabenstellungen untersucht werden.

## 7.3. Experimente in RapidMiner

Experimente mit *RapidMiner* können auf verschiedene Weise durchgeführt werden. Der zunächst einfachste und intuitivste Weg führt über die GUI. Hier kann jede Einstellung per Mausklick und über beschriftete Eingabefelder getätigt werden. Eine andere Möglichkeit ist, *RapidMiner* über die Kommandozeile zu starten. Die XML-Datei, die das Experiment beschreibt, wird als Parameter angegeben. Wichtige Ausgaben des Programms erfolgen ebenfalls über die Kommandozeile, sodass der Benutzer bei Bedarf die Kontrolle über das Programm behalten und seinen Fortschritt überwachen kann. Auf diese Weise können jedoch auch Berechnungen unüberwacht auf entfernten Rechnern stattfinden, bei denen kein graphisches Interface zur Verfügung steht. Die dritte Möglichkeit ist, *RapidMiner* in eigene Programme einzubinden und die bereitgestellten Operatoren direkt aus diesen zu nutzen. Zu diesem Zweck steht eine umfangreiche, gut dokumentierte API zur Verfügung, die die Integration in Java-Applikationen erlaubt.

### 7.3.1. Ablauf eines Experiments

Experimente lassen sich in *RapidMiner* komfortabel über die GUI zusammenstellen. Der Ablauf eines typischen Experiments zur Relation Extraction soll im Folgenden beschrieben werden. In Abb. 7.2 entspricht die Abfolge von Operatoren einem ähnlichen Vorgang wie dem hier beschriebenen.

Am Anfang eines Experiments in *RapidMiner* steht das Einlesen der Daten. Falls diese bereits in dem *RapidMiner*-eigenen Format vorliegen, können sie direkt verwendet werden. Für cvs - Dateien und verwandte Formate, bei denen jede Zeile einer Datei eine Tabellenzeile beschreibt und ein definiertes Trennzeichen die Spalten festlegt, ist ein Assistent beim Import behilflich. Im folgenden Schritt müssen dann noch die Datentypen der Spalte festgelegt und spezielle Attribute wie das Label markiert werden. Für das Einlesen anderer Dokumentformate wie html und pdf stellt das *Information Extraction*-Plugin einen Import-Operator zur Verfügung.

Nachdem die Daten in *RapidMiner* in Tabellenform vorliegen, werden die Vorverarbeitungsoperatoren angewendet. Zahlreiche Operatoren dieser Art sind verfügbar. Falls die Daten in Form ganzer Sätze eingelesen wurden, ist es notwendig, zuerst den *RelationCandidateGenerator* anzuwenden. Dieser erzeugt aus den ganzen Sätzen alle möglichen Relationskandidaten und speichert diese als neue RapidMiner-Tabelle ab. Ausgehend von den Relationskandidaten können jetzt durch die Vorverarbeitung weitere Features generiert und der Datensatz damit angereichert werden.

Um die Ergebnisse eines Lernverfahrens zu untersuchen, bietet sich eine *Kreuzvalidierung* an. Dabei wird die gesamte Beispielmenge in gleich große Teile aufgesplittet, von denen alle bis auf jeweils eines genutzt werden, um das Lernverfahren zu trainieren. Auf dem übrigen Teil wird das gelernte Modell dann angewendet und die Ergebnisse werden abgespeichert. Dieser Schritt wird für jedes einzelne der erzeugten Teile des Datensatzes durchgeführt, sodass bei einer 10fach Kreuzvalidierung insgesamt 10 verschiedene Testmengen untersucht werden. Das abschließende Ergebnis wird aus dem Mittelwert aller Durchläufe gebildet; die maximale Abweichung nach oben und unten wird ebenfalls angezeigt.

Alternativ wird der Lerner auf den kompletten Daten trainiert und das Modell anschließend auf eine bis dahin unbekannte Testmenge von Daten angewendet.

Am Ende der Untersuchung steht der Operator, der Auskunft über die Performance des verwendeten Lernverfahrens gibt. Der Nutzer erhält über die GUI eine Darstellung der Ergebnisse. Diese können zusätzlich auch in Form einer Log-Datei abgespeichert werden.

Für umfangreichere Untersuchungen stellt *RapidMiner* praktische Operatoren bereit, die es erlauben, z. B. eine automatisierte Parameteroptimierung durchzuführen. Dabei wird eine Versuchsanordnung mehrfach durchlaufen und jedesmal ein Parameter in einem festgelegten Wertebereich verändert. Die Ergebnisse jeder Veränderung werden protokolliert und können später ausgewertet werden. Auf diese Weise lassen sich ohne manuellen Aufwand zahlreiche Kombinationen erproben.

# 8. Eigene Implementierungen

Die Aufgabenstellung, Relationen aus Texten zu extrahieren und die gefundenen Relationen zusammen mit allgemeinen Informationen über Unternehmen darzustellen, machte die Implementierung mehrerer Programme notwendig. Die im Rahmen der Diplomarbeit entwickelte Software umfasst die Datensammlung aus dem Internet, die Datenvorverarbeitung, die Umsetzung des Lernverfahrens sowie die Visualisierung der Egebnisse. Um Plattformunabhängigkeit zu gewährleisten, wird als Programmiersprache *Java*[1] in der Version 1.6 genutzt, Entwicklungsumgebung ist *Eclipse*[2].

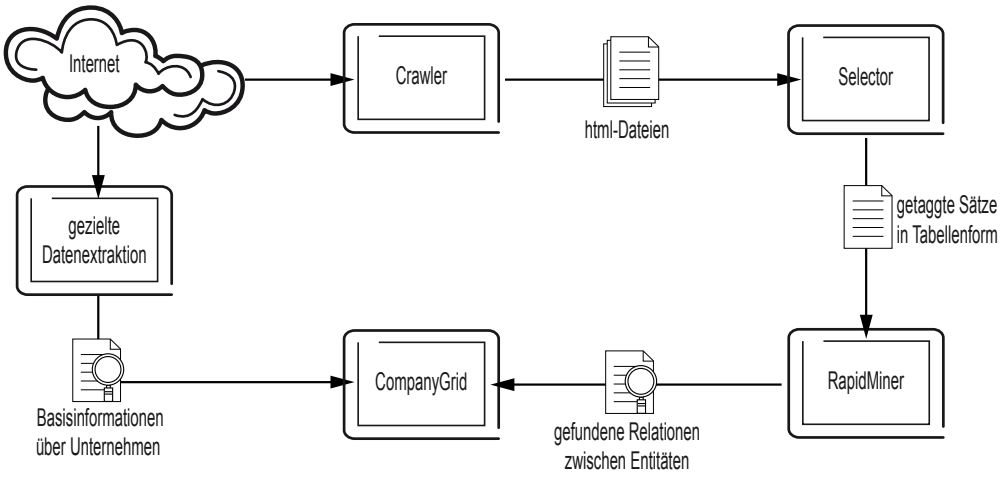



Abbildung 8.1.: Der Ablauf der Datenverarbeitungsschritte zum Finden und Darstellen von Firmendossiers und Relationen zwischen Unternehmen

In Abb. 8.1 ist der grundsätzliche Ablauf der Prozesse dargestellt. Zunächst werden Informationen über Unternehmen, die bereits in strukturierter Form im Internet vorliegen, in einer lokalen Datenbank abgespeichert. Diese bilden die grundlegende Datenbasis, auf der die *CompanyGrid*-Software operieren kann. Für das Auffinden zusätzlicher Relationen werden mit der *Crawler*-Komponente themenspezifische html-Dateien aus dem Internet heruntergeladen. Diese werden in der *Selector*-Anwendung vorverarbeitet und mit Zusatzinformationen angereichert. Die Daten werden in einer Tabellenform an die Lernumgebung *RapidMiner* weitergegeben. Dort sorgt das *Information Extraction*-Plugin

[1] `http://java.sun.com/`
[2] `http://www.eclipse.org/`

für die Erkennung von Relationen. Gefundene Relationen werden in die Datenbank der *CompanyGrid*-Software eingefügt. Die *CompanyGrid*-Software dient der Visualisierung des Graphen, der durch die Relationen aufgespannt werden kann.

## 8.1. Crawler

Zur Erstellung der Korpora, auf deren Basis die Relationserkennung durchgeführt werden soll, wurde das *Crawler*-Programm implementiert. Dieses soll Dokumente zu festgelegten Themenbereichen aus dem Internet herunterladen und im lokalen Dateisystem in einem einheitlichen Format zur Weiterverarbeitung ablegen.

### 8.1.1. Allgemeine Vorgehensweise

Die Vielzahl der im Internet vorhandenen Dokumente macht es erforderlich, dass die Suche nach geeigneten Webseiten bereits von vornherein auf passende Weise eingeschränkt wird. Suchmaschinen haben einen großen Teil des World Wide Web indiziert und ermöglichen die gezielte Anzeige von Dokumenten, die ein oder mehrere vom Nutzer vorgegebene Suchwörter enthalten. Die Internetsuchmaschine *Google* wird daher verwendet, um in gezielter Weise passende Dokumente auffinden zu können, bei denen es wahrscheinlich ist, dass sie eine Relation des gesuchten Typs enthalten.
Als Hilfsmittel für Programmierer und Webseitenersteller bieten einige Suchmaschinenbetreiber wie Google oder Yahoo ein sog. *„application programming interface“ (API)* an. Mittels dieser Schnittstelle ist es möglich, die Dienste des Anbieters in eigene Applikationen einzubinden und die Suchanfragen sowie die Ergebnisse darin weiterzuverarbeiten.

### 8.1.2. Google API

Im Verlauf dieser Arbeit wird die *Google AJAX Search API*[3] verwendet. Diese ist ursprünglich für die Integration in Webseiten mittels JavaScript konzipiert. Um die Nutzung auch aus anderen Umgebungen wie beispielsweise Flash oder Java zu ermöglichen, implementiert die API ein Interface basierend auf der *REST*-Architektur (Fie00), das eine Rückgabe der Suchergebnisse im *JSON*-Format[4] realisiert. Die API ist kostenlos und ohne vorherige Registrierung verwendbar, jede Anfrage muss jedoch einen gültigen *„http referer header“* übertragen.
Pro Anfrage werden die ersten acht Ergebnisse zurückgeliefert, diese können jeweils in maximal Achterschritten auf bis zu 32 Ergebnisse erweitert werden. Die Geschwindigkeit des Zugriffs wird automatisch von Google reguliert, häufige Anfragen resultieren

[3] `http://code.google.com/intl/de/apis/ajaxsearch/`
[4] `http://json.org/json-de.html`

in einer längeren Wartezeit. Diese liegt im Extremfall im Bereich von einigen Sekunden pro Request, sodass massenhafte Anfragen in kurzer Zeit verhindert werden. Die gewünschte Sprache der Ergebnisseiten ist wählbar; sie wird entweder per http-flag oder per Parameter eingestellt.

### 8.1.3. Abspeichern der Suchergebnisse

Die durch die Suche gefundene Liste von URLs wird aus der JSON-Antwort entnommen und nacheinander aufgerufen. Google-Suchergebnisse können auf verschiedene Dateiformate verweisen. Der Großteil der Webseiten liegt im html-Format vor, vertreten sind aber auch z. B. direkte Verweise auf PDF oder ppt-Dateien. Nur Seiten im html-Format werden zur Weiterverarbeitung genutzt. Dazu gilt es zunächst, die vorliegende Zeichencodierung zu ermitteln. Diese ist normalerweise bereits durch die Antwort des Webservers im http-response festgelegt. Alternativ gilt es, innerhalb der html-Datei das *charset*-Tag ausfindig zu machen und auszuwerten. Danach erfolgt falls nötig eine Umwandlung in das *UTF-8*-Format und eine Anpassung der entsprechenden Tags im Quelltext. Abschließend wird die Datei zur späteren Verwendung im Dateisystem der Festplatte gespeichert.

## 8.2. Selector

*Selector* ist ein Tool, das zur Bearbeitung und Erzeugung von Trainings- und Testdatensätzen für maschinelle Lernverfahren zur Relationsextraktion entwickelt wurde. Es erlaubt die textuellen Inhalte von html-Dateien auszulesen, diese zu verarbeiten und daraus Sätze zu erzeugen, in denen Relationen markiert werden können. Die auf diese Weise getaggten Sätze können abgespeichert und anschließend zu Relationskandidaten für die Relation Extraction weiterverarbeitet werden.

### 8.2.1. Benutzeroberfläche

#### Hauptfenster

Im Zentrum des Hauptfensters von *Selector* (Abb. 8.2) wird der aktuelle Satz dargestellt. NER und Relationstags werden farbig hervorgehoben. Über Buttons in der linken unteren Ecke lassen sich die Labels für die Relationen setzen. Zur Auswahl stehen *„POS“*, *„NEG“* und *„IGN“*. *„IGN“* steht dabei für Ignore mit diesem Label versehene Sätze werden nicht zur Kandidatengenerierung genutzt. Das aktuell gesetzte Label wird farbig in der rechten obigen Ecke angezeigt. Über die Pfeilbuttons in der unteren Reihe kann zwischen den Sätzen gewechselt werden. Die grünen Buttons berücksichtigen nur als positive Relationskandidaten gelabelte Sätze. Die grauen Buttons springen ohne Rücksicht auf das Label zum nächsten Satz. Um Tags zu Wörtern hinzuzufügen, werden ein ein-

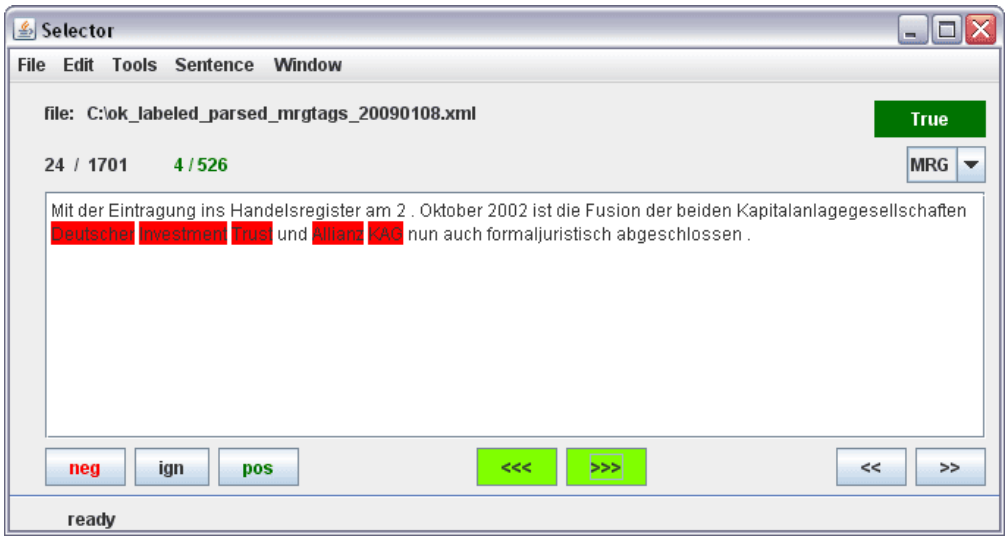



Abbildung 8.2.: Über die GUI hat der Benutzer Zugriff auf alle wichtigen Funktionen von *Selector*.

zelnes Wort oder mehrere aufeinanderfolgende Wörter mit der Maus markiert. Durch Druck auf die rechte Maustaste kann dann ein Tag ausgewählt werden. Die Wahl der Art des Labels (MRG oder NER) erfolgt über die Drop-Down-Box in der rechten oberen Ecke. Um den Vorgang komfortabler zu gestalten, gibt es ausserdem einen *Quick Tag*-Modus, bei dem die Auswahl für das Tag direkt erscheint, sobald nach dem Markieren eines Textes die Maustaste losgelassen wird.

### Menü

Über das Menü erhält der Benutzer Zugriff auf alle wichtigen Funktionen von Selector. Die einzelnen Menüpunkte sind nach Aufgabengebieten zusammengefasst.

**File** Zum Einlesen von Dateien stehen im Menü *„File“* die Befehle *„Open“* bzw *„Open Folder“* zur Verfügung. Diese ermöglichen es, einzelne Dateien oder alle in einem Ordner enthaltenen Dateien einzulesen. Um zu einem späteren Zeitpunkt weitere Daten hinzuzufügen, kann der Punkt *„add“* aufgerufen werden. Als Eingabeformate stehen html-Dateien, sowie das *Selector*-eigene XML-Format zur Verfügung. Die Funktion zum Abspeichern bearbeiteter Dokumente findet sich ebenfalls hier.

**Edit** Das *„Edit“*-Menü enthält diejenigen Funktionen, die Auswirkungen auf das gesamte Dokument haben. Es bietet die Möglichkeit, alle Sätze parsen zu lassen, die NER-Erkennung der Firmennamen durchzuführen, allen Sätzen das gleiche Relationslabel zuzuordnen, Relationskandidaten zu generieren und die Filterfunktionen anzuwenden.

**Tools** Unter dem Menüpunkt *„Tools“* gibt es die Möglichkeit, den Parser zu laden. Dieser wird nicht automatisch beim Programmstart aktiviert, da das Laden des Modells einige Zeit in Anspruch nimmt und der Parser nicht immer benötigt wird. Die QuickTag-Funktion kann hier ebenfalls aktiviert werden.

**Sentence** Der Menüpunkt *„Sentence“* betrifft die Eigenschaften des aktuellen Satzes. Die Menüpunkte enthalten die gleichen Funktionen wie die Buttons im unteren Bereich des Hauptfensters. Es kann ein Relationslabel gesetzt und zwischen den einzelnen Sätzen navigiert werden.

**Window** Im Menü *„Window“* gibt es die Möglichkeit, eine Listenübersicht aller im aktuellen Dokument enthaltenen Sätze anzuzeigen und diese direkt anzuwählen. Außerdem kann der Parsebaum des aktuellen Satzes ausgegeben werden.

### 8.2.2. Verarbeitungsstruktur

Die Verarbeitungsstruktur von *Selector* ist in Abb. 8.3 dargestellt. Zunächst werden die zu analysierenden html-Dateien in das Programm eingelesen. Es erfolgt eine Trennung in einzelne Sätze und Tokens. Named Entities werden markiert. Verschiedene Filterschritte entfernen Sätze, die als Relationskandidaten nicht in Frage kommen. Ein grammatischer Parser bildet Parsebäume. Basierend auf diesen Daten werden durch einen menschlichen Annotator die Relationen markiert. Das Ergebnis wird in einem flexiblen Format zur Weiterverarbeitung abgespeichert.

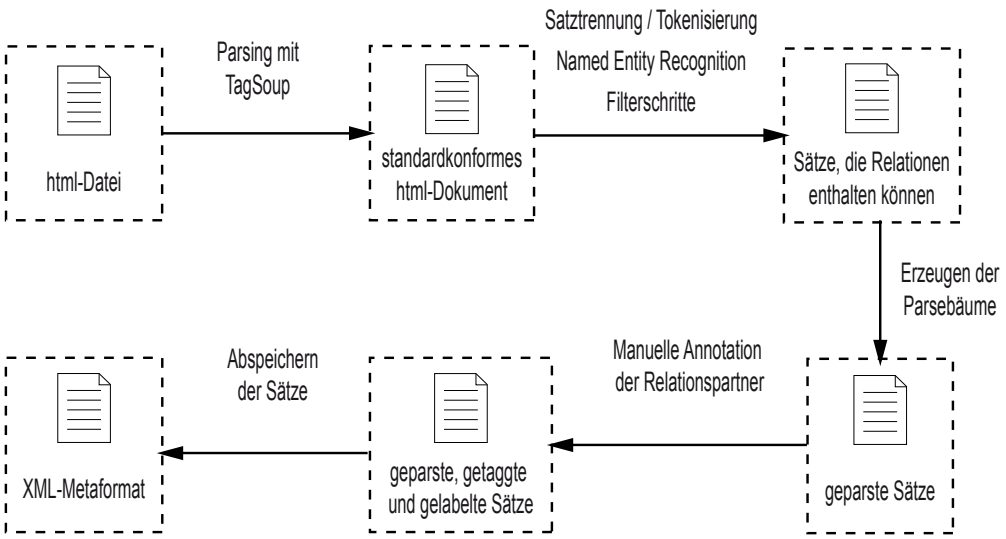



Abbildung 8.3.: Die Verarbeitungskette in Selector

### Einlesen von html-Dateien

Im ersten Schritt müssen die html-Dateien eingelesen werden, die als Grundlage für die Relationsextraktion dienen sollen. Obwohl eindeutige Standards für das html-Format festgelegt sind, ist es eher die Ausnahme, dass sich die Gestalter von Webseiten auch tatsächlich vollständig an diese halten. Im Gegensatz zur Verarbeitung von XML-Dateien kann man also bei html nicht von Wohlgeformtheit ausgehen. Der Versuch, einen gewöhnlichen XML-Parser für Internet-Dokumente zu verwenden, schlägt daher häufig aufgrund von Fehlern in der Struktur fehl. Beispiele für solche Fehler sind fehlende oder falsche Attribute, falsche Verschachtelung von Elementen oder fehlende abschließende Tags. Spezielle Parser sind an diese Probleme angepasst und versuchen, das Dokument bestmöglich zu interpretieren, ohne vollständig korrekte Strukturen vorauszusetzen. TagSoup ist ein kostenloser, schneller und zuverlässiger Parser, der in der Lage ist, selbst stark fehlerhafte Dokumente in eine korrekte Form zu überführen. Er wird von *Selector* verwendet, um aus den html-Dateien ein gültiges XML-Dokument-Objekt zu erstellen, das zur Weiterverarbeitung genutzt werden kann. Der Parser ist zwar sehr robust gegenüber Fehlern innerhalb eines Dokuments, scheitert jedoch, wenn eine html-Datei nicht mit dem ⟨!DOCTYPE ..⟩ Element beginnt. Es ist deshalb wichtig sicherzustellen, dass sämtliche Zeichen vor diesem Tag entfernt werden, um die korrekte Funktion des Parsers zu gewährleisten.

### Extraktion der Texte aus html

Der Zugriff auf die Elemente des erzeugten XML-Dokument-Objekts kann jetzt komfortabel mit XPath erfolgen. XPath ist eine Sprache zum Auffinden von Informationen in XML-Dokumenten. Sie ermöglicht den gezielten Zugriff auf Elemente und Attribute durch Formulierung von Abfragen. Ergebnis einer Abfrage ist eine Menge von Knoten oder Information über Elemente und Attribute des Dokuments. Viele Elemente wie ⟨script⟩, ⟨style⟩ oder ⟨comment⟩ enthalten keinen Textinhalt, in dem Relationen auftauchen können, und können direkt entfernt werden. Die übrigen Elemente können Texte enthalten und müssen untersucht werden. Um die Satzgrenzenerkennung (siehe 3.3.1) zu verbessern, ist es hilfreich, Informationen, die die Struktur des Dokumentes vorgibt, zu berücksichtigen. Die html-Elemente ⟨p⟩ und ⟨div⟩ deuten z. B. Absätze bzw. zusammengehörige Blöcke an, in ⟨h*⟩ eingeschlossene Texte bilden Überschriften. Leider wird jedoch auch eine solche Unterteilung nicht immer vollständig umgesetzt. Dazu kommt, dass html-Elemente auf verschiedene Weisen ineinander verschachtelt werden können. Ein Versuch, dieses Problem zu umgehen und trotzdem die vorhandenen Informationen bestmöglich auszunutzen, wurde auf folgende Weise gemacht: Zunächst werden sämtliche Zeilenumbrüche des Quelltextes entfernt. Dann werden am Ende jedes Elements, das in der Regel einen zusammengehörigen Text enthält, neue Zeilenumbrüche eingefügt. Ab-

schließend werden sämtliche Inhaltselemente ausgelesen. Bevor diese an den Satztrenner übergeben werden, wird zuvor noch eine Trennung an allen Zeilenumbrüchen durchgeführt. Auf diese Weise werden die Informationen der html-Block-Elemente genutzt, um innerhalb verschachtelter Elemente dem Satztrenner die Arbeit zu erleichtern, da an dieser Stelle von ihm keine Entscheidung mehr getroffen werden muss, sondern ein eindeutiges Satzende vorliegt.
Leider enthalten html-Dokumente eine Vielzahl von Informationen und Textelementen, die für die Relationsextraktion nicht von Bedeutung sind. Dazu zählen Menü- und Navigationselemente der Webseite, Werbeinhalte, Seitenleisten mit Vorschautexten etc. Der Satztrenner versucht, bestmöglich mit diesen Inhalten umzugehen, stößt aber häufig an seine Grenzen. Im Ergebnis erhält man daher neben korrekten Sätzen häufig auch einfache Aneinanderreihungen von Wörtern, Tabelleninhalte, die aus dem Zusammenhang gerissen sind, etc. Weiterhin ist die Trennung auch unter optimalen Bedingungen nicht immer zuverlässig. Dadurch kommt es vor, dass im Ergebnis mehrere Sätze ungetrennt hintereinander folgen und folglich im weiteren Verlauf der Verarbeitung wie ein Satz behandelt werden. Ebenso kommt es vor, dass Sätze zu früh getrennt werden und danach nur noch unvollständig vorliegen. Die Ursache dafür sind Zeichen im Text, die potentiell einen Satz trennen können, dies aber nicht in jedem Fall tun. Der Trenner muss jedes Mal eine Entscheidung treffen, ob er einen Satz beendet oder nicht, was nicht immer korrekt gelingt.

#### Filtern der Sätze

Um die Menge der gefundenen Sätze auf diejenigen zu begrenzen, die sinnvoll nutzbar sind, gibt es unterschiedliche Filterfunktionen. Gefiltert werden können Duplikate, Sätze, die zu kurz sind, um Relationen zu enthalten, Sätze mit weniger als zwei unterschiedlichen potentiellen Relationspartnern sowie Sätze, die nicht erfolgreich geparsed werden können. Durch diese Filterungmöglichkeiten reduziert sich die Zahl der durch den menschlichen Annotator zu betrachtenden Daten erheblich.

#### Parsen der Sätze

Um die Parsebäume der Sätze zu erhalten, wird der *Stanford Parser* (KM02) genutzt. Es handelt sich um eine Java-Implementierung, die sich leicht in eigene Programme integrieren lässt. Für die deutsche Sprache verwendet der Parser ein Modell, das auf dem *NEGRA*-Corpus trainiert wurde. Die ausgegebenen Parsebäume sind teilweise nicht sehr zufriedenstellend, was einerseits am Modell des Parsers liegt (siehe auch: 3.3.5), andererseits aber auch an der zuvor beschriebenen teilweise sinnlosen Struktur der erzeugten Internet-Sätze. Die Daten werden in Form eines Array vom Typ String übergeben, die Elemente des Arrays sind die Tokens des Satzes. Das Format der Tokens muss dem

Format des Trainingscorpus des Parsers entsprechen (in diesem Fall das *Penn Treebank*-Format), wodurch z. B. runde Klammern durch ihre Entsprechung *LRB* ersetzt werden müssen. Es besteht die Möglichkeit, den Tokens ein POS-Tag mitzugeben, anderenfalls versucht der Parser dieses automatisch zu bestimmen. *Selector* übergibt für alle gefundenen Named Entities das Tag NE. Ausserdem werden mehrteilige, aber nur eine Entity beschreibende Tokens wie *Deutsche Bank* zusammengefasst. Der Parser benötigt viel Platz im Hauptspeicher. Bei den Versuchen wurde mindestens 1 GB zur Verfügung gestellt. Lange Sätze mit vielen Wörtern sorgen für unverhältnismäßig lange Wartezeiten und enden in den meisten Fällen ergebnislos. Darum werden nur Sätze mit weniger als 80 Tokens bearbeitet. Das Laden des als serialisiertes Objekt vorliegenden Parser-Modells dauert ca. 20 Sekunden. Die Parsezeit pro Satz beträgt auf einem 2 Ghz Core2Duo-Prozessor ca. 1-5 Sekunden. Der erzeugte Baum liegt zunächst als Stanford-Tree-Objekt vor und wird später in eine String-Form serialisiert.

#### Erkennen von Named Entities

Die Named-Entity-Erkennung ist auf das Entdecken von Firmennamen beschränkt. Zu diesem Zweck steht eine Liste mit 2259 Firmennamen zur Verfügung. Der Text wird automatisch nach Vorkommen dieser Firmennamen untersucht und die entsprechenden Tokens mit dem NER-Tag *„ORG“* versehen. Weiterhin besteht die Möglichkeit, Tokens manuell mit den Tags *„ORG“*, *„PER“*, *„LOC“* und *„MISC“* zu versehen. Die Tags werden nach dem *IOB*-Schema, das in Abschnitt 3.3.4 beschrieben wurde, gesetzt. Standardwert ist *„O“*.

#### Markieren der Relationskandidaten und Setzen der Labels

Die Markierung von Firmenfusionsrelationen erfolgt auf ähnliche Weise wie die manuelle Markierung der Named Entities. Tokens im Text können mit einem Tag versehen werden, das sie als Relationskandidaten kennzeichnet. Ein solches Tag kann z. B. den Wert *„MRG“* haben, dieses kennzeichnet das Token als Teilnehmer an einer (symmetrischen) Firmenfusion. Weiterhin stehen die Tags *„ACT“* und *„PAS“* zur Verfügung; dies sind der aktive (also der übernehmende) und der passive (der übernommene) Partner in einer Relation. Um ein aus einer Fusion hervorgegangenes neues Unternehmen zu markieren kann das Tag *„NEW“* verwendet werden.

#### Interne Repräsentation der Daten

Die interne Verwaltung der Texte in *Selector* basiert auf einer hierarchischen Struktur von Listen und Objekten (Abb. 8.4). Auf der obersten Ebene steht das *Document*-Objekt. Dieses enthält eine Liste mit *Sentence*-Objekten. Ein *Sentence*-Objekt wiederum besteht aus einer Liste von *Tokens*, jedes Token ist eine Liste von Attributen. Au-

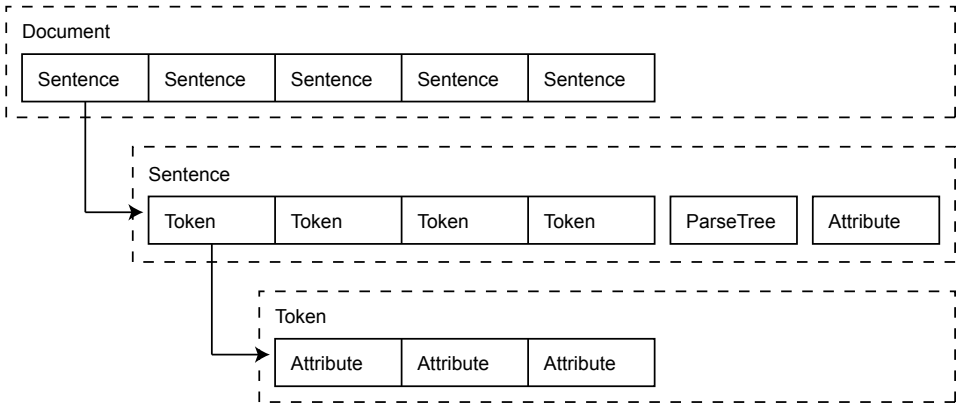



Abbildung 8.4.: Die Datenstruktur in Selector

ßerdem besteht bei jedem der Objekte die Möglichkeit, zusätzliche Informationen wie Parsebäume, Labels etc. zu speichern. Durch diese verkettete Datenstruktur wird Redundanz vermieden und ein schneller Zugriff auf die gewünschte Abstraktionsebene erlaubt. Jedes Objekt stellt eine Reihe passender Methoden zur Verarbeitung zur Verfügung. Außerdem enthält es eine *toXML()*-Methode die ein XML-Element zurückliefert, sowie einen Konstruktor, der als Parameter ein XML-Element entgegennimmt.

#### Abspeichern der Daten

Zur Weiterverarbeitung und Sicherung der Relationsdaten speichert Selector die Daten in einem eigenen XML-Format ab. Die internen Datenstrukturen werden dabei komplett abgebildet, sodass alle Daten nach erneutem Laden wieder vollständig zur Verfügung stehen. Besonders wichtig ist, dass auch die Parsebäume komplett erhalten bleiben, da das Parsen großer Textmengen ein sehr zeitaufwändiger Prozess ist. Daher werden die Stanford-Tree-Objekte durch die Java-Objektserialisierung in einen String umgewandelt. Dieser wird in ein XML-Element eingebettet und kann später durch Deserialisierung wieder hergestellt werden.

## 8.3. Implementierte Kernfunktionen

Es wurden die in (ZZJZ07) vorgestellten Kernfunktionen implementiert. Die Umsetzung erfolgte in Form einzelner JAVA-Klassen. Daher können die Kerne sowohl mittels des *ParseTree SVM*-Operators des *RapidMiner*-Information-Extraction-Plugins verwendet werden als auch durch andere *SVM*-Implementierungen, die Java-Klassen als Kernfunk-

tion einbinden können. So kann beispielsweise die *JNI Kernel Extension for SVM$^{light}$* [5] genutzt werden. Einzig die Funktion, die direkt zur Berechnung der Ähnlichkeit zweier Beispiele aufgerufen wird, muss individuell angepasst werden. Sie muss den richtigen Namen haben, die von der SVM übergebenen Daten entgegennehmen und diese in die im Kern verwendete Datenrepräsention umwandeln können. Die Parameter, die bei den einzelnen Kernen gesetzt werden können, sind unter 8.4.2 beschrieben.

### 8.3.1. Context Sensitive Convolution Tree Kernel

Der *Context Sensitive Convolution Tree Kernel* berechnet die Ähnlichkeit zweier Baumstrukturen. Im Detail ist er beschrieben unter 6.4.4. Als Eingabeparameter kann er entweder zwei Baumobjekte vom Typ *Tree* oder zwei Strings, die einen serialisierten Baum enthalten, entgegennehmen. Ausgabe ist ein normalisierter Wert für die Ähnlichkeit der beiden Bäume. Die Implementierung folgt den von Moschitti in (Mos06) vorgeschlagenen Verfahren zur Verbesserung der Laufzeit.
Vor Beginn der Berechnung der einzelnen Kernfunktionen werden als Vorverarbeitungsschritt – für alle Bäume des Datensatzes – die Produktionsregeln ihrer Knoten aufgestellt. Diese werden in Form einer sortierten Liste in dem Baumobjekt abgespeichert. Für die Verwendung im kontextsensitiven Kern wird für jeden Wert $m$ eine eigene Liste angelegt, die genau diejenigen Produktionsregeln enthält, die unter Berücksichtigung von $m$ gebildet wurden. Im eigentlichen Berechnungsschritt geht es darum, für zwei Bäume deren Ähnlichkeit zu bestimmen. Hier kann jetzt auf die zuvor erstellten Listen direkt zugegriffen werden. Als erstes werden parallel die Listen beider Bäume schrittweise, von oben nach unten durchlaufen. Immer wenn übereinstimmende Paare gefunden werden, werden diese zur späteren Bearbeitung abgespeichert. Durch die vorherige Sortierung können auf diese Weise sehr effizient alle Paare von Knoten beider Bäume ermittelt werden, die die gleichen Produktionsregeln haben. Nur für diese Paare müssen die nachfolgenden Berechnungen durchgeführt werden. Der nächste Schritt umfasst die algorithmische Umsetzung von Formel 6.19 und von den Bedingungen, die in Abschnitt 6.4.2 bzw. 6.4.4 beschrieben sind.
Abschließend wird das Ergebnis unter Anwendung von Formel 6.5 normalisiert und an die SVM zurückgegeben.
Die Bäume, die innerhalb der Kernberechnung betrachtet werden, bestehen aus Objekten vom Typ *Tree*. Ein Tree-Objekt ist eigentlich ein Knoten eines Baumes. Der Konstruktor der Klasse ist in der Lage, einen serialisierten Baum als String entgegenzunehmen und daraus rekursiv einen Baum aus zusammenhängenden Baumobjekten aufzubauen. Das Tree-Objekt, das den String des kompletten Baumes entgegengenommen hat, also dessen Konstruktor zuerst aufgerufen wurde, ist der Wurzelknoten des gesamten Baumes.

[5] urlhttp://www.aifb.uni-karlsruhe.de/WBS/sbl/software/jnikernel/

Eines der wichtigen Attribute eines Baumknotens ist sein Name. Bei Parsebäumen ist dies bei inneren Knoten ein Nichtterminalsysmbol und bei Blättern ein Terminalsymbol der Grammatik. Außerdem enthält er eine Liste mit Verweisen auf die Knoten, die seine Kinder sind, und eine Referenz auf seinen Elternknoten.
Das Erstellen der Produktionsregeln sowie deren Auflistung und Sortierung wird ebenfalls von dieser Klasse übernommen. Des Weiteren werden zahlreiche zusätzliche Methoden bereitgestellt, die den einfachen und effizienten Umgang mit den Baumobjekten ermöglichen.

### 8.3.2. Linear Entity Kernel

Dieser Kern ist sehr einfach aufgebaut. Die genaue mathematische Beschreibung findet sich in Abschnitt 6.5. Als Eingabe werden zwei Vektoren übergeben, der Kern addiert alle übereinstimmenden Wertepaare. Das Ergebnis wird normalisiert und zurückgegeben.

### 8.3.3. Composite Kernel

Der *Composite Kernel* dient dazu, zwei Kernfunktionen miteinander zu verknüpfen. Die in Formel 6.3.3 dargestellten Kombinationsmöglichkeiten werden umgesetzt. Die zu kombinierenden Kerne können als Parameter übergeben werden. Auf diese Weise können neue Kernfunktionen später auf einfache Weise integriert werden.

## 8.4. RE - Operatoren für RapidMiner

Zwei Operatoren zur Relation Extraction wurden im Rahmen dieser Diplomarbeit für das *RapidMiner* Information Extraction Plugin implementiert. Der Vorverarbeitungsoperator *TreeShapePreprocessing* dient dazu, die Form eines Parsebaumes zu beschneiden oder den Baum mit neuen Features anzureichern. Der Operator *ParseTreeSVM* ist eine modifizierte Form des MySVM-Operators, der in der Lage ist, mit einem *Context Sensitive Convolution Tree Kernel*, einem *Linear Entity Kernel* sowie einer Kombination aus beiden in einem *Composite Kernel* binäre Klassifikationsaufgaben durchzuführen.

### 8.4.1. TreeShapePreprocessing

Die Anpassung der Bäume, die an den Baumkern übergeben werden sollen, ist ein wichtiger Schritt zur Vorverarbeitung. Für die Generierung der Relationskandidaten soll die Möglichkeit gegeben werden, unterschiedliche Baumformen integrieren zu können. Die Erzeugung dieser Formen kann mit Hilfe des *TreeShapePreprocessing* Operators geschehen.

#### operatorName

Der Name des Operators. Dieser ist gleichzeitig auch der Name des neuen Attributs, das durch seine Anwendung erzeugt wird. In das neue Attribut wird der veränderte Baum abgespeichert.

#### wordAttributeName

In diesem Parameter wird festgelegt, wie der Name des Attributs im ExampleSet ist, das die Namen der Entities enthält, die die Relationspartner des aktuellen Relationskandidaten sind.

#### treeStringAttributeName

Hier wird das Attribut angegeben, in dem der String abgelegt ist, der den kompletten Parsebaum beschreibt. Dies ist die Baumform, auf deren Grundlage der neue, umgeformte Baum erzeugt wird.

#### featureAttributeList

Zur Anreicherung des Baums mit neuen Features kann hier eine Liste aller Attributnamen angegeben werden, die eingefügt werden sollen. Diese Features werden oberhalb der Blattknoten der Relationspartner eingefügt. Es muss jeweils nur der Grundname eines Attributs angegeben werden. Das System geht davon aus, dass das zum zweiten Relationspartner gehörende Attribut unter dem gleichen Namen mit angehängter 2 zu finden ist *(AttributName, AttributName2)*.

#### featureDepth

Über diesen Parameter wird die Höhe angegeben, in der die mittels *featureAttributeList* angegebenen Attribute eingefügt werden sollen. Die Höhe 0 entspricht der Position unmittelbar über den Blattknoten. Je größer der Wert eingestellt wird, desto höher in Richtung Wurzelknoten werden die neuen Knoten verschoben.

#### PruningMode

Der Modus zum Beschneiden des Baumes kann an dieser Stelle ausgewählt werden. Es stehen mehrere Optionen zur Verfügung deren Auswirkungen in Abb. 8.5 dargestellt werden:

- *None*
  Der Baum wird nicht beschnitten.

- *SPT*
  *(Shortest Path Tree)* Es wird ein Ausschnitt aus dem Baum gebildet, der alle Blätter, die rechts und links von den beiden Relationskandidaten gelegen sind, entfernt. Auch Knoten, die keine Vorfahren eines der beiden Relationskandidaten sind, werden gelöscht.

- *Incl. Verb*
  Diese Einstellung entspricht dem SPT, erweitert um das zu den Relationspartnern am nächsten gelegene Verb.

- *TopNode*
  TopNode ist der erste gemeinsame Vorgängerknoten der beiden Relationspartner. Dieser Knoten dient als Wurzel des beschnittenen Baumes, der zurückgegeben wird. Diese Baumform entspricht dem *MCT* von Zhang (vgl. Abschnitt 5.2).

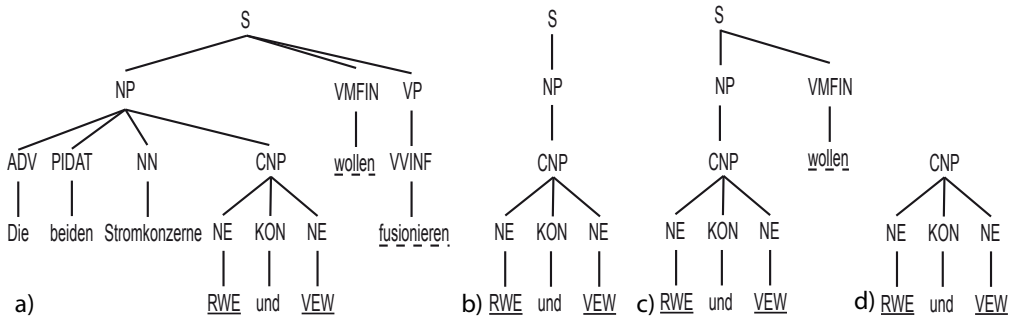



Abbildung 8.5.: Die Verschiedenen Baumformen die per *PruningMode*-Option eingestellt werden können. a) *None*, b) *SPT*, c) *Incl. Verb*, d) *TopNode*

### 8.4.2. ParseTree SVM

Der *ParseTree SVM*-Operator ist ein Lernoperator, der die in *RapidMiner* integrierte *MySVM* dahingehend erweitert, dass Kernfunktionen für komplexe Objekttypen zur Klassifikation genutzt werden können.
Die Kernfunktionen, die implementiert wurden, entsprechen den in (ZZJZ07) vorgestellten. Neben dem *Context-Sensitive Convolution Tree Kernel* können auch der *Linear Entity Kernel* sowie deren Kombination als *Composite Kernel* genutzt werden.
Daten liegen in *RapidMiner* in der Form von Tabellen vor. Die Tabellenzellen enthalten Zahlenwerte im *double*-Format. Zur Erzeugung der Featurevektoren für die *MySVM* werden die *double*-Werte in eine spärliche Vektordarstellung überführt. Ein Beispiel wird somit intern in zwei Arrays umgewandelt: *index[i]* und *att[i]*. *index[i]* gibt an, welche Attributnummer an der Stelle $i$ im Array *att[]* gespeichert ist. *att[i]* enthält dementsprechend den Wert des Attributes mit der Nummer, die in *index[i]* abgelegt ist. Die

Kernfunktionen der MySVM bestimmen die Ähnlichkeit dieser spärlich besetzen Arrays. Der Aufruf der Kernfunktion erfolgt auf folgende Weise:

```
public double calculate_K(
              int[] x_index, double[] x_att, int[] y_index, double[] y_att
              )
```

Um mit strukturierten Objekten wie Parsebäumen zu arbeiten, ist diese Vektordarstellung jedoch ungeeignet. Die Zahl der in einem Baum enthaltenen Unterbäume wächst exponentiell mit der Anzahl der Blätter und um deren Aufzählung zu vermeiden, wurde das Verfahren der *Convolution Kernels* entwickelt, das in 6.3.2 vorgestellt wurde.
Ein weiteres Problem ist, dass Objekte nicht direkt in der ExampleSet-Tabelle abgelegt werden können. Die einzige Möglichkeit ist, sie in Form eines Strings zu codieren. Das macht es erforderlich, dass zur Berechnung der Ähnlichkeit die Strings zuerst in Baumobjekte konvertiert werden müssen.
Bei der Implementierung der *ParseTree SVM* wurde viel Wert darauf gelegt, die Datenstrukturen möglichst universell zu gestalten. Dies erlaubt eine flexible Gestaltung und Einbindung verschiedener Kernfunktionen. Dazu werden keine weiteren Änderungen an der SVM nötig. Einzig die Kernfunktion bestimmt, in welcher Form die im *RapidMiner* ExampleSet vorliegenden Daten in Objekte umgewandelt und wie sie später zur Berechnung übergeben werden. Alle Daten werden in ein *SVMExample*-Objekt gespeichert, dessen genauer Typ nur vom Kernel ausgewertet wird und der von *RapidMiner* transparent gehandhabt wird. Die Umwandlung der Strings in Objekte findet vor der eigentlichen Berechnung statt, sodass sie pro Example nur genau ein Mal durchgeführt werden muss und nicht bei jedem der zahlreichen Aufrufe der Kernfunktion. Die Aufruf der Distanzfunktion sieht durch diese Änderungen folgendermaßen aus:

```
public double calculate_K(
              SVMExample x, SVMExample y
              )
```

Eine Vielzahl von Parametern erlaubt es, das Verhalten der Kernfunktion zu beeinflussen:

#### kernel_type

Zur Auswahl stehen drei verschiedenen Arten von Kernfunktionen:

- *Context-Sensitive Convolution Tree Kernel (CSCTK)*
- *Linear Entity Kernel (LEK)*
- *Composite Kernel (CK)*

#### kernel_tree_attribute

*Betrifft: CSCTK*
Dies ist der Attributname, in dem der String gespeichert ist, aus dem das Baumobjekt erzeugt werden soll.

#### kernel_combination

*Betrifft: Composite*
Die Art der Kombination der beiden Kerne im *Composite Kernel*. Zur Auswahl stehen *Addition* und *Polynomielle Erweiterung* wie in 6.3.3 beschrieben.

#### kernel_alpha

*Betrifft: Composite*
Der Wert *kernel_alpha* bestimmt die Gewichtung der Kerne im *Composite Kernel*. Der zulässige Wertebereich ist eine Zahl zwischen 0 und 1.

#### kernel_binary

*Betrifft: LEK*
Dieser Parameter wurde hauptsächlich zu Testzwecken eingeführt. Wird er ausgewählt, so gibt der *LEK* genau dann 1 zurück, wenn beide Featurevektoren übereinstimmen, in jedem anderen Fall, also auch bei Unterscheidung an nur einer einzigen Stelle, den Wert 0.

#### kernel_countAll

*Betrifft: LEK*
Dieser Parameter wurde hauptsächlich zu Testzwecken eingeführt. Wird er ausgewählt, werden vom *LEK* auch „leere“ Features, die in beiden Vektoren nicht vorkommen bzw. nicht besetzt sind, als Übereinstimmung gewertet. Der Ähnlichkeitswert wird entsprechend berechnet.

#### kernel_lambda

*Betrifft: CSCTK*
Dieser Parameter entspricht dem Wert $\lambda$ aus Formel 6.18. Mit ihm wird der Beitrag großer Baumfragmente bei der Ähnlichkeitsberechnung abgeschwächt.

#### kernel_lambda

*Betrifft: CSCTK*
Dieser Parameter entspricht dem Wert $\sigma$ aus Formel 6.18. Durch diesen Parameter wird ausgewählt, ob Unterbäume oder nur komplette Teilbäume (siehe: 5.1.1) zur Berechnung betrachtet werden.

#### kernel_m

*Betrifft: CSCTK*
Dieser Parameter entspricht dem Wert $m$ aus Formel 6.19. $m$ legt fest, wie weit der Rückblick auf Elternknoten bei der Kernberechnung stattfinden soll. Der Wertebereich geht von 1 bis 10, wobei 1 keinem Rückblick entspricht. Je größer der Wert, desto länger der Pfad über Vorgängerknoten, der in den Produktionsregeln auf der linken Seite beachtet wird.

## 8.5. CompanyGrid

Soziale Netzwerke sind für die Wissenschaft zu einem Forschungsgebiet von stetig wachsender Bedeutung geworden. Von der Erweiterung automatischer Empfehlungssysteme (DGM08), (PMLR04), (DR01) bis zum Gewinn wissenschaftlicher Erkenntnisse über Eigenschaften des Kommunikations- und Sozialverhalten der Mitglieder sozialer Gemeinschaften wie Facebook (GWH07) ist der Themenbereich weit gefasst. Die Eigenschaften wirtschaftlicher Netzwerke, deren Struktur bestimmt wird durch die zahlreichen Verbindungen zwischen Firmen und Unternehmen, ist ein Bereich, der bislang noch wenig Aufmerksamkeit erfahren hat. Um die Untersuchung dieser Strukturen zu ermöglichen, wurde mit der *CompanyGrid*-Software (HJM09) eine Grundlage geschaffen, die Verwaltung und Visualisierung solcher Wirtschaftsnetzwerke zu realisieren. Über ein graphisches Interface wird dem Benutzer die Möglichkeit gegeben, sich interaktiv durch das Netzwerk zu bewegen und den für ihn interessanten Pfaden zu folgen.

### 8.5.1. Erstellung der Firmendatenbank

Grundlegende Basis des Programms bildet eine Datenbank, in der Informationen über Unternehmen hinterlegt werden können. Aus dieser werden die Untenehmensnetzwerke generiert.
Die Befüllung mit einem grundlegenden Datensatz stellt keine großen Schwierigkeiten dar. Im Internet existieren zahlreiche Datenquellen in halbstrukturierter Form. Dies ermöglicht eine Datenextraktion unter Zuhilfenahme einfacher regulärer Ausdrücke. Dabei wurden ca 13 000 verschiedene Einträge heruntergeladen. Anschließend wurde eine

Einteilung in Qualitätsstufen vorgenommen, abhängig von der Anzahl der enthaltenen Details.

- Die niedrigste Stufe enthält keinerlei nützliche Informationen außer Firmenname sowie Wertpapierkennzahl und Branche. Dies ist die größte Gruppe mit 6700 Mitgliedern.
- 5343 Einträge enthalten außerdem noch mindestens eine Adresse.
- Die übrigen 2000 Datensätze sind deutlich vollständiger und geben eine kurze textuelle Beschreibung der Firma, Auskünfte über Mitglieder des Vorstands und Aufsichtsrates, Aktienanteile etc. Außerdem sind betriebswirtschafliche Kennzahlen sowie Unternehmenskennzahlen in unterschiedlichem Umfang enthalten. Nur diese vollständigen Datensätze bilden den Grundstock des Systems.

Die Daten werden in einer SQL-Datenbank in verschiedenen Tabellen abgelegt, um einen schnellen und flexiblen Zugriff zu gewährleisten. Durch die flexible Gestaltung von Abfragen in SQL-Syntax können verschiedene Fragestellungen auf einfache Weise beantwortet werden. Auch die komfortable Möglichkeit zur Erweiterung und Pflege der Daten ist dadurch gegeben. Die Aktualisierung vorhandener Datensätze sowie das Einbringen neuer Tabellen oder Felder erfordert keinen großen Arbeitsaufwand, sodass der Funktionsaufwand des Gesamtsystems auch zu einem späteren Zeitpunkt noch problemlos erweitert werden kann.

### Analyse der Datenbasis

Die in dieser strukturierten Form vorliegenden Daten ermöglichen es bereits, erste Querverbindungen zwischen Firmen anzeigen zu lassen. Ein Beispiel dafür sind die Besitzverhältnisse in Form von Aktienanteilen. Eine wichtige Verbindung ist auch gegeben, wenn eine Person, die als Mitglied im Aufsichtsrat oder dem Vorstand einer Firma vertreten ist, ebenfalls Mitglied im Vorstand einer anderen Firma ist. Eine große Anzahl (1354) der im initialen Datensatz gespeicherten Firmen haben mindestens eine solche Verbindung. In Tabelle 8.5.1 werden die zehn Unternehmen mit den meisten Verbindungen aufgeführt. Der höchste Wert wird von einem Unternehmen mit 37 ausgehenden Verbindungen ereicht. Diese Zahlen sprechen dafür, dass der Graph, der durch diese Relationen aufgespannt wird, wichtige Hinweise auf Strukturen in der Geschäftswelt geben kann. Insgesamt werden 13 983 verschiedene Leute in dem Datensatz aufgeführt. Nur 1781 von ihnen werden mehrfach erwähnt. Tabelle 8.5.1 zeigt die Spitzenwerte. Hier ist der höchste Wert von unterschiedlichen Posten, die von der gleichen Person bekleidet werden, 9. In Tabelle 8.5.1 wird die Zahl der Verbindungen pro Person in einem Unternehmen berechnet. Es werden somit dort die Firmen mit den durchschnittlich „einflussreichsten“ Personen abgebildet.

| Verbindungen | Firma |
|---|---|
| 37 | Pirelli & C. S.p.A. |
| 32 | Assicurazioni Generali S.p.A. |
| 29 | Allianz SE |
| 27 | Oberbank AG |
| 27 | RCS MEDIAGROUP S.p.A. |
| 26 | $n^2$ Nanotech AG |
| 25 | Telecom Italia S.p.A. |
| 25 | FONDIARIA-SAI S.p.A. |
| 25 | UniCredito Italiano S.p.A. |
| 23 | Rheinmetall AG |

Tabelle 8.1.: Die 10 am besten verbundenen Firmen

| Firmen | Person |
|---|---|
| 9 | Olaf Neugebauer |
| 8 | Bernd Günther |
| 7 | Lukas Lenz |
| 7 | Marcus Deetz |
| 7 | Reiner Ehlerding |
| 7 | Robert Zeiss |
| 7 | Sergio Erede |
| 6 | Andreas Lange |
| 6 | Diego Della Valle |
| 6 | Gabriele Galateri di Genola e Suniglia |

Tabelle 8.2.: Die 10 Personen mit den meisten Posten

| Personen | Verbindungen | Firma | Verbindungen/Person |
|---|---|---|---|
| 2 | 14 | Mandarin Capital AG | 7 |
| 2 | 14 | Mereo AG | 7 |
| 4 | 26 | $n^2$ Nanotech AG | 6,5 |
| 2 | 12 | ARTEMIS Global Capital SE | 6 |
| 2 | 12 | Equipotential SE | 6 |
| 4 | 19 | OCTAGON Energy AG | 4,75 |
| 2 | 9 | MINDFIRE Solutions AG | 4,5 |
| 2 | 9 | Schraad Metallbau AG | 4,5 |
| 2 | 8 | NanoStrategy AG | 4 |
| 4 | 15 | Valara Capital AG | 3,75 |

Tabelle 8.3.: Firmen mit den meisten Verbindungen pro Person

### 8.5.2. Verwendete Software-Bibliotheken

Für Softwareprojekte ist es oft von großem Vorteil, auf bereits fertige Programmteile zurückgreifen zu können, in der Teilprobleme bereits auf effiziente Weise gelöst wurden. Diese werden meist in Form von Bibliotheken oder als Quellcode unter verschiedenen Lizenzen zur Verfügung gestellt. Projekte, die unter einer Open Source Lizenz bereitgestellt werden, dürfen – unter Beachtung der Richtlinien der entsprechenden Lizenz – kostenfrei in eigene Projekte eingebunden werden. *CompanyGrid* greift auf mehrere solcher Bibliotheken zurück, die im Folgenden kurz vorgestellt werden sollen.

#### JUNG Framework

*JUNG* („Java Universal Network/Graph“) [6] ist eine javabasierte quelloffene Software-Bibliothek zur Erzeugung, Analyse und Darstellung von Graphen. Die Methoden, die bereitgestellt werden, sind dabei auf mathematische und algorithmische Anwendungen zugeschnitten. Die besondere Eignung zur Visualisierung von komplexen sozialen Netzwerken sowie die Datenanalyse und Wissensentdeckung werden ausdrücklich als Schwerpunkte der Implementierung genannt. Die Bibliothek wurde von drei Doktoranden der „University of California, Irvine“ entwickelt (JO03). Sie wird als Sourceforge-Projekt weitergeführt und erweitert. Eine der Stärken von JUNG ist, beliebige JAVA-Objekte als Elemente des Graphen nutzen zu können. In der *CompanyGrid*-Anwendung werden eigene Objekte für die Knoten des Graphen erzeugt, die als Container für die Entitäten dienen und je nach Typ der Entität über unterschiedliche Eigenschaften verfügen. Zahlreiche Möglichkeiten zur Anpassung der Darstellung erlauben die flexible Gestaltung der

[6] `http://jung.sourceforge.net/`

Oberfläche in Abhängigkeit von den genutzten Objekte.

#### Apache Derby

*Apache Derby*[7] ist eine Relationale Datenbank, die als .jar-Datei in eigene Programme eingebettet werden kann und diesen eine vollständige Datenbank zur Verfügung stellt. Derby basiert auf Java, JDBC und SQL. Auf diese Weise kann die Software von allen Vorteilen einer Datenbank profitieren, ohne dass eine separate Datenbank-Server-Anwendung zur Verfügung stehen muss. Der Platzbedarf der Bibliotheken, die eingebunden werden, ist sehr gering; insgesamt werden nur knapp 2,4 MB benötigt. Derby kann sowohl im eingebetten Modus exklusiv für die eigene Anwendung genutzt werden, als auch im Servermodus mehrfache, gleichzeitige Verbindungen auch von externen Programmen erlauben. Von Sun wird Apache Derby unter dem Namen *JavaDB* offiziell als Teil des JDK 6 unterstützt.

### 8.5.3. Das Wirtschaftsnetzwerk

Das Wirtschaftsnetzwerk $G = (V, E)$, das basierend auf diesen Daten aufgebaut werden soll, ist wie folgt definiert: Entities bilden die Knoten des Graphen ($v \in V$). Knoten können vom Typ *Firma* oder *Person* sein. Die Kanten des Graphen ($e \in E$) sind die Relationen, die zwischen Entities bestehen können. Besteht eine Kante zwischen zwei Knoten vom Typ *Firma*, so kann sie z. B. vom Typ *„Merger“* sein und somit eine Fusion anzeigen. Zwischen einer Person und einer Firma steht der Kantentyp *„Employee“*, der angibt, dass die Person ein Mitarbeiter des Unternehmens ist.
Erweitert werden soll dieses Beziehungsnetzwerk durch das Finden zusätzlicher Relationen aus Textmeldungen im Internet. Der Zusammenschluss von Firmen ist eine Relation, die für einen Nutzer interessante Informationen bereitstellen kann. Deshalb wurde dieser Relationstyp als erstes implementiert. Ein Firmenzusammenschluss ist definiert als symmetrische Relation. Die Relationspartner sind zwei Firmen, die zu einem gemeinsamen Unternehmen verschmelzen. Die Menge der auffindbaren Relationen wird dadurch eingeschränkt, dass sie vollständig in einem Satz beschrieben werden müssen.

#### Benutzeroberfläche

Die Benutzeroberfläche von *CompanyGrid* ist in Abb. 8.6 dargestellt. Sie erlaubt es, den Graphen, der durch Firmen, Personen und deren Verknüpfungen gebildet wird, interaktiv zu erforschen. Dieser Graph wird als zentrales Element der *CompanyGrid*-Benutzeroberfläche dargestellt. Das Netzwerk wird, ausgehend von einem zentralen Knoten, Schritt für Schritt durch den Benutzer aufgebaut. Zu Beginn wird eine Firma im

[7] `http://db.apache.org/derby/`

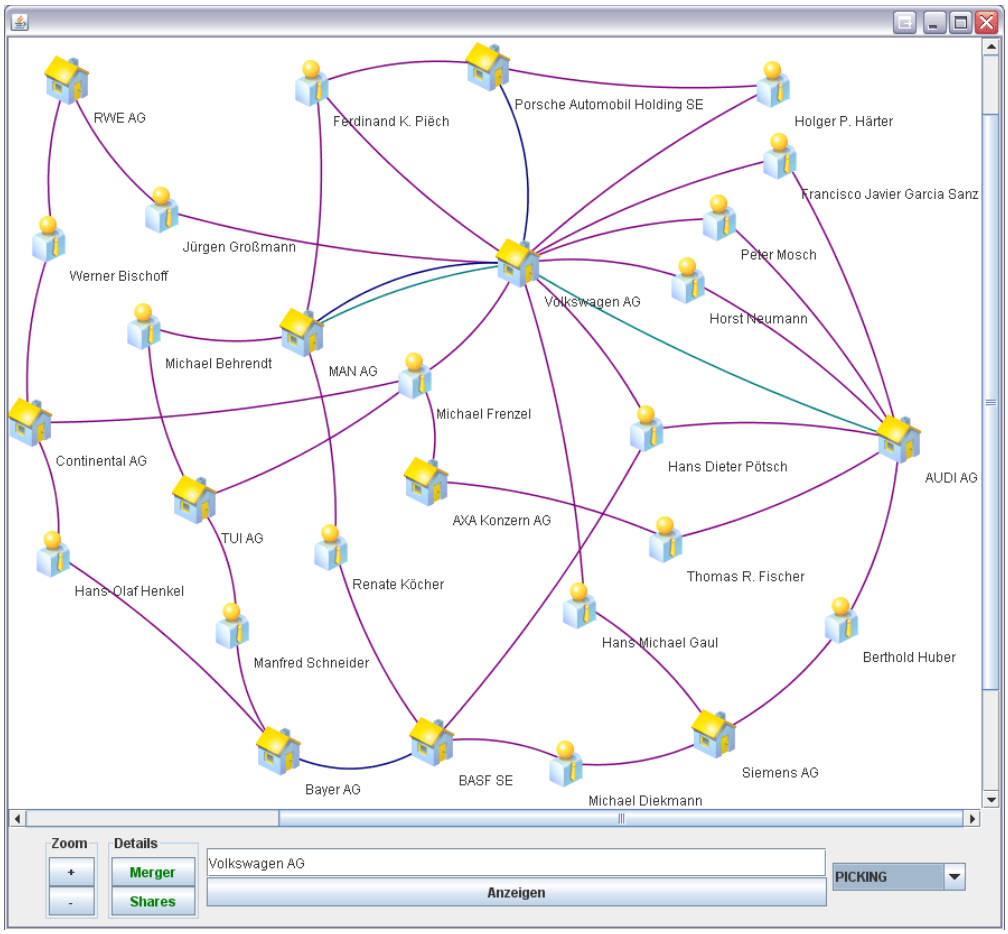



Abbildung 8.6.: Die Benutzeroberfläche von CompanyGrid.

Zentrum, umgeben von Personen, die in deren Vorstand und Aufsichtsrat vertreten sind, angezeigt. Durch Anklicken eines am Rande gelegenen Knotens wird der Graph in diese Richtung erweitert. Handelt es sich um eine Person, so werden alle mit ihr verbundenen Firmen als neue Knoten in den Graphen integriert. Handelt es sich um eine Firma, so werden die zugehörigen Personen eingefügt. Knoten können auch wieder ausgeblendet werden: Klickt man auf einen bereits aktivierten Knoten ein zweites Mal, so werden alle Knoten, die nur mit dem aktuellen Knoten verbunden sind und keine andere Verbindung innerhalb des Graphen haben, ausgeblendet. Dadurch kann die Übersicht in manchen Fällen stark verbessert werden.
Ein Suchfeld unterhalb der Zeichenfläche bietet die Möglichkeit, nach einer Firma zu suchen. Wird diese innerhalb der Datenbank gefunden, so wird der aktuelle Graph gelöscht und die gefundene Firma als zentraler Startpunkt eines neuen Graphen genutzt.
Mit Hilfe der beiden Buttons *Merger* und *Shares* können dem Graphen ebenfalls neue Knoten und Kanten hinzugefügt werden. Die Aktivierung des *Merger*-Buttons bewirkt, dass Firmenfusionsrelationen in Form blauer Kanten zwischen Firmenknoten abgebildet

werden. Durch Auswahl des *Shares*-Button werden Aktienbesitz-Beziehungen zwischen Firmenknoten als türkise Kanten eingeblendet.
Der Graph kann durch Drehen des Mausrades oder mit Hilfe der Buttons gezoomt werden. Knoten können frei auf der Zeichenfläche bewegt werden. Dazu werden sie mit der linken Maustaste angeklickt und bei gedrückter Taste auf der Zeichenfläche an die gewünschte Position verschoben.
Durch die gleichzeitige Anzeige mehrerer Relationen, z. B. von Vorstandsmitgliedern und Fusionsrelationen, lassen sich interessante Zusammenhänge entdecken. Abb. 8.7 zeigt die Mitglieder des Vorstandes und Aufsichtsrates von Volkswagen. Erweitert man nun den Graphen um die Anzeige von Fusionsrelationen (Abb. 8.8), stellt man fest, dass zwei Personen (Holger P. Härter und Ferdinand K. Piëch) in beiden Unternehmen tätig sind, wodurch eine enge Verbindung gegeben ist.

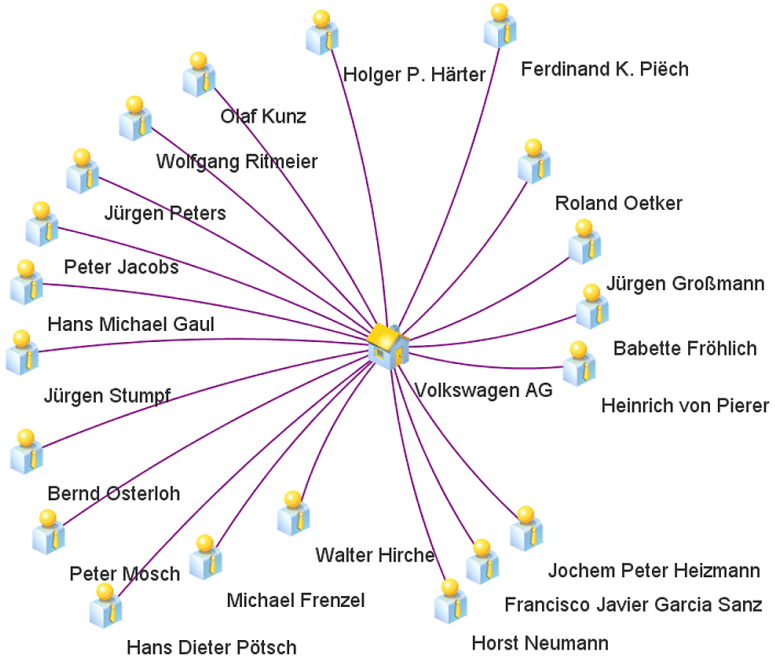



Abbildung 8.7.: Nach Auswahl von Volkswagen werden alle Personen des Vorstands und Aufsichtsrates angezeigt.

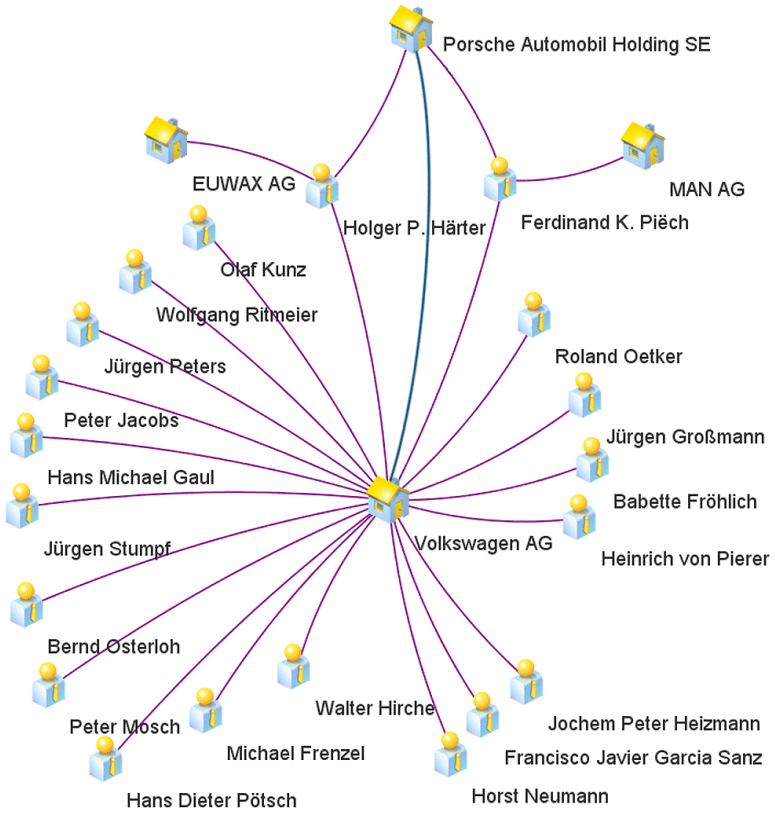



Abbildung 8.8.: Zwei Vorstandsmitglieder von Volkswagen sitzen auch im Vorstand von Porsche. Die Fusion zwischen Volkswagen und Porsche wird durch die direkte Kante zwischen den Firmen dargestellt.

# 9. Experimente

## 9.1. Korpuserstellung

Um ein automatisches Lernverfahren für die Aufgabe vorzubereiten, Relationen zu finden, die in aus dem Internet gewonnenen Texten enthalten sind, muss zunächst eine genügend große Menge an Trainingsbeispielen zur Verfügung gestellt werden. Aus diesen lernt das Verfahren die Merkmale, die notwendig sind, um später auch in unbekannten Texten Entscheidungen treffen zu können, ob eine Relation vorliegt. Zu diesem Zweck wird eine repräsentative Menge von Texten aus dem Internet gesammelt. Die Weiterverarbeitung erfolgt mit Hilfe der selbst implementierten Software *Selector*, die in Abschnitt 8.2 vorgestellt wurde. Zunächst werden die notwendigen Vorverarbeitungsschritte wie Tokenisierung, NER und Parsing durchgeführt. Dann kommen die verschiedenen Filtermöglichkeiten von *Selector* zum Einsatz. In den verbleibenden Sätzen werden Relationen und Relationspartner von Hand markiert. Aus den auf diese Weise vorbereiteten Sätzen werden alle gültigen Relationskandidaten extrahiert. Weiterhin erfolgt eine Anreicherung mit zusätzlichen Features. Daraus lassen sich dann abschließend die benötigten Trainings- und Testdatensätze erzeugen.

Ursprünglich war geplant, verschiedene Arten von Relationen und Events zu untersuchen. Diese umfassten *Firmenfusionen*, *Quartalszahlen* und *Bestechungsvorfälle*. Im Laufe der Erstellung der Korpora und der Entwicklung der Extraktionsmethoden wurde der Fokus jedoch eindeutig auf die *Firmenfusions* (Company-Merger) - Relation gerichtet. Der wichtigste Grund dafür ist, dass die Merger-Relation für die Integration in die Darstellung der Firmennetzwerke am besten geeignet ist. Weitere Gründe sind bei den Unterabschnitten zu den anderen Korpora erwähnt.

### 9.1.1. Erzeugung des Korpus für „Firmenfusion/Kooperation“

Als Vorbereitung für die Aufgabe, Relationen des Typs „Firmenfusionen und Kooperationen“ zu erkennen, wurden für jedes der deutschen DAX-30 Unternehmen passende Suchmaschinenanfragen gestellt. Diese bestehen aus dem Firmennamen sowie einem relevanten Suchwort, das mit hoher Wahrscheinlichkeit Treffer für die gesuchte Relation ergibt. Die für diese Aufgabe gewählten Wörter sind *Zusammenschluss* und *Fusion*. Die

Gesamtmenge der heruntergeladenen html-Dokumente betrug 758. Davon stellten sich 433 Dokumente als relevant und 325 als von vornherein ungeeignet für die gesuchte Relation heraus. Im nächsten Schritt wurden alle Sätze isoliert, die mindestens zwei Firmennamen enthalten. Dies ergab eine Menge von 1698 Sätzen, die zur Erzeugung von Relationskandidaten herangezogen wurden. Die Gesamtzahl der Relationskandidaten, die aus der Kombination von jeweils zwei Firmen-Entities aus einem Satz gebildet wurden, beträgt 3602. Durch manuelle Auswahl wurden darin 672 positive und 2930 negative Lernbeispiele markiert.

#### Anmerkungen zur Fusionsrelation

Eine Durchsicht der Ergebnisseiten zur Fusionsrelation macht deutlich, dass Fusionen in verschiedenen Zusammenhängen erwähnt werden. Die häufigsten sind „geplante Fusionen“, „durchgeführte Fusionen“, „gescheiterte Fusionen“ und „Gerüchte um Fusionen“. Weiterhin gibt es strategische Partnerschaften und Kooperationen, die nur bestimmte Geschäftsbereiche betreffen. Außerdem versuchen Firmen, durch Aktienzukäufe Anteile an anderen Unternehmen zu gewinnen, was ab einem Besitzanteil von über 50% einer Übernahme entspricht. Die Unterscheidung zwischen diesen verschiedenen Auftretensformen ist oft schwierig. Informationen, die eindeutig sicherstellen, ob eine Fusion komplett, noch nicht, nur teilweise oder gar nur potentiell durchgeführt worden ist, sind häufig nicht eindeutig bzw. unsicher formuliert und werden nur unter Berücksichtigung der umgebenden Sätze klar. Ein gutes Beispiel dafür bietet der folgende Satz: *„Ohne Stellenabbau wird die Fusion zwischen Bayer und Schering nicht über die Bühne gehen“, sagt Bayer-Chef Wenning.* Dieser kann sowohl unter der hypothetischen Annahme, dass Bayer und Schering vielleicht fusionieren, gesehen werden als auch einen direkten Blick auf die Zukunft darstellen, mit dem Hintergrund, dass die Fusion bereits sicher beschlossen ist.
Diese Entscheidung fällt durch den Annotator teilsweise sehr subjektiv aus. Ab einem gewissen Zeitpunkt ist die Information „Schering und Bayer haben fusioniert“ so fest eingeprägt, dass auch nicht eindeutige Sätze als gültige Fusionsinformation wahrgenommen werden können. Eine Grenze, ab wann eine Relation markiert wird, ist somit nicht klar definierbar.
Die folgenden Beispiele geben einen Überblick über Sätze, in denen die Firmen **Schering** und **Bayer** erwähnt werden. Diese Unternehmen haben tatsächlich fusioniert. Die Sätze beschreiben jedoch nach Ansicht des Annotators keine eindeutige Fusionsrelation. Aus diesem Grund wurden sie als negativ markiert, da ohne Kontext oder Hintergrundwissen nicht eindeutig entscheidbar ist, ob eine Fusion tatsächlich stattgefunden hat.

- *In jedem Fall ist **Bayer** fest gewillt , die Akquisition von **Schering** durchzuführen.*

- ***Bayer*** *und* ***Schering*** *wollten sich am Wochenende zu diesem Thema nicht äußern.*
- *Tatsächlich ist die* ***Bayer****-Offerte nicht nur finanziell attraktiver für* ***Schering****.*
- ***Schering*** *begrüßte das* ***Bayer****-Angebot.*
- ***Bayer*** *werde 86 Euro in bar je* ***Schering****-Aktie bieten, teilte der Leverkusener Konzern am späten Donnerstagabend mit.*
- *Nun fehlt nicht mehr viel, um die* ***Schering****-Übernahme durch* ***Bayer*** *zu vereiteln.*
- *Nach Angaben von* ***Bayer*** *liegt das Angebot um 39 Prozent über dem Kurs der* ***Schering****-Aktie vor Bekanntwerden des Übernahmeangebotes von Merck und um 12 Prozent über dem konkurrierenden Angebot.*
- ***Bayer*** *hat bei seinem Angebot die volle Unterstützung des* ***Schering****-Vorstandes und tritt damit als „weißer Ritter“ auf – gegen das mittlerweile zurückgezogene, feindliche Übernahmeangebot von Merck.*

Sätze, die Fusionen enthalten, sind unterschiedlich komplex und können auf zahlreiche verschiedene Arten formuliert werden. Daher sollen hier auch einige Beispiele für gültige Sätze, die eine Fusion zwischen **Schering** und **Bayer** beschreiben, vorgestellt werden.

- ***Schering*** *und* ***Bayer*** *einigen sich auf Zusammenschluss.*
- ***Bayer*** *hatte* ***Schering*** *im September übernommen.*
- *Damit ist* ***Schering*** *die bisher größte Akquisition in der* ***Bayer****-Geschichte.*
- ***Bayer****:* ***Schering****-Kauf stärkt Standort Berlin.*
- *Der Kauf des Pharmakonzerns* ***Schering*** *durch* ***Bayer*** *bedroht weltweit 6000 Arbeitsplätze.*
- *Der Standort Berlin wird nach Ansicht von* ***Bayer****-Chef Werner Wenning langfristig von der Fusion der* ***Bayer****-Pharmasparte mit dem* ***Schering****-Konzern profitieren.*
- *Der Berliner* ***Schering****-Konzern und der Chemieriese* ***Bayer*** *in Leverkusen haben sich im Übernahmepoker auf dem deutschen Pharma-Markt auf einen Zusammenschluss geeinigt.*

### 9.1.2. Erzeugung des Korpus für „Quartalszahlen“

Im Zusammenhang mit Quartalszahlen war es Aufgabe zu erkennen, ob die Quartalszahlen eines Unternehmens steigen oder sinken. Dafür verwendete Suchworte waren *„Quartalszahlen“* und *„Vierteljahreszahlen“*. Die Ergebnismenge bestand aus 795 Dokumenten. Davon waren 213 sinnvoll verwendbar und 582 ungeeignet. Es sind in diesem Zusammenhang zahlreiche Probleme aufgetreten. Relationen sind üblicherweise als Beziehung zwischen Entitäten definiert. Im vorliegenden Fall der Quartalszahlen handelt es sich eher um ein Event, da es verschiedene Slots wie (Firma, Gewinn/Umsatz, Währung, Datum) zu füllen gibt. Mit Hilfe von Relation Extraction dieser Aufgabe gerecht zu werden, stößt auf folgende Schwierigkeiten: Nachrichten, die Quartalszahlen behandeln, tun dies häufig, ohne im gleichen Satz das zugehörige Datum oder eine Zeitangabe zu erwähnen. Maximal wird das Quartal des aktuellen Jahres erwähnt. Da die Relationserkennung jedoch nur innerhalb eines Satzes erfolgt, kann hier keine Aussage getroffen werden. Auch die konkreten Informationen über die Quartalszahl ist meistens über mehrere Sätze verteilt. Im Falle von Zahlenangaben liegen diese meist in Tabellenform oder als PDF-Datei vor. Artikel, die Quartalszahlen erwähnen, tun dies häufig in dem Kontext, dass eine Firma „demnächst Zahlen präsentieren wird“. Auch Hinweise auf die Zahlen eines Mitbewerbers als Entscheidungshilfe für Aktionäre sind üblich. Weiterhin finden sich oft Sätze wie *Trotz Gewinnsteigerung konnten die Erwartungen in die Quartalszahlen nicht erfüllt werden*, wodurch sich das Problem ergibt, ob dies nun als positiv oder negativ gewertet werden soll. Sinnvolle Aussagen über den Verlauf der Quartalszahlen eines Unternehmens nur mit Methoden der Relation Extraction zu treffen, ist daher nicht möglich. Erforderlich wäre eine umfangreiche Zusatzverarbeitung, die den Kontext der gesamten Webseite mit einbezieht. Dazu sind auf jeden Fall folgende Schritte notwendig:

- Erkennen des Datums, an dem ein Text verfasst wurde. Dies ist nicht nur mit simplen regulären Ausdrücken erreichbar, da im Allgemeinen mehrere Datumsangaben auf einer Webseite aufgefunden werden können.
- Auch ist es wichtig, zusammengehörige Textabschnitte als solche zu erkennen, da häufig mehrere Artikel, Kurznachrichten, Randspalten etc. gleichzeitig vorliegen. Es ist also nötig, über Kenntnisse der Struktur der Seite zu verfügen.
- Weiterhin erforderlich ist eine gute Koreferenz-Erkennung, um Zusammenhänge, die mehrere Sätze umfassen, auswerten zu können.

Aus den oben genannten Gründen wurde auf eine Erstellung von Trainingsbeispielen verzichtet und die Untersuchung an dieser Stelle abgebrochen. Als bessere Quelle für Quartalsberichte zeigen sich die Webseiten der Unternehmen, die die Berichte in der Regel als PDF zum Download zur Verfügung stellen.

### 9.1.3. Erzeugung des Korpus für „Bestechungsvorwurf“

Suchworte für Bestechungsvorwurf waren *Bestechung* und *Schmiergeld*. 795 Dateien wurden heruntergeladen. 213 davon sind verwendbar, 582 eignen sich nicht zur Weiterverarbeitung. Bei Bestechungsvorwürfen traten folgende Schwierigkeiten auf:

- Im Regelfall werden keine konkreten Namen erwähnt.
- Häufig sind mehrere nicht genauer festgelegte Personen beteiligt.
- In den aufgefundenen Berichten wird von unterschiedlichen Quellen oft über die gleichen Vorfälle berichtet. Über 50% der aufgefundenen Seiten beziehen sich auf die gleiche Bestechungsaffäre bei der Firma Siemens, unabhängig von dem eingegebenen Firmennamen.
- Insgesamt konnte nur eine überschaubare Anzahl unterschiedlicher Ereignisse gefunden werden.
- Aufgrund der Tatsache, dass oft nur über Bestechung bei einer Firma, nicht aber über die beteiligten Personen gesprochen wird, liegt auch hier keine Relation im eigentlichen Sinne vor.

Auch auf die Erkennung von Bestechungsvorwürfen wurde daher verzichtet.

## 9.2. Experimente

Zur Untersuchung der vorgestellten Methoden für ihre Eignung zum Auffinden der Company-Merger-Relation in deutschsprachigen Internet-Seiten wurden Versuche mit den Kern-Methoden von (ZZJZ07) durchgeführt, die auf dem *ACE 2004*-Korpus die besten Ergebnisse für die Relationsextraktion erzielen konnten. Außerdem wurden eigene Veränderungen entwickelt und mit den State-of-the-Art-Methoden verglichen. Alle Ergebnisse beziehen sich auf den eigenen, selbst zusammengestellten Korpus und sind daher nicht direkt mit den Ergebnissen anderer Veröffentlichungen vergleichbar. Dennoch ist aufgrund der relativen Änderungen in der Performance eine Abschätzung der unterschiedlichen Eignung der einzelnen Verfahren möglich.

### 9.2.1. Versuchsdurchführung

Die Experimente wurden mit dem *Information Extraction Plugin* in *RapidMiner* durchgeführt. Als erster Schritt wurden Versuche mit den Kernen alleine durchgeführt, um festzustellen, welche Ergebnisse jeder Kern für sich erzielen kann. Anschließend wurden dann verschiedene Kombinationen im Composite Kernel getestet.

**Aufbau und Zusammensetzung der Testdaten** Die Daten, die für die folgenden Experimente verwendet wurden, basieren auf dem *Firmenfusion*-Korpus, der in 9.1.1 vorgestellt wurde. Die Verteilung der Sätze sowie der daraus erzeugten Relationsbeispiele sind in Tabelle 9.1 dargestellt. Der Anteil der Sätze, die gleichzeitig sowohl positive als auch negative Relationskandidaten enthalten, ist sehr gering.

| | Gesamt | Positiv | Negativ | Gemischt |
|---|---|---|---|---|
| *Sätze* | 1698 | 426 | 1174 | 98 |
| *Relationskandidaten* | 3602 | 672 | 2930 | – |

Tabelle 9.1.: Verteilung der positiven und negativen Beispiele

**Laufzeiten** Aufgrund der unterschiedlichen Ausstattung der genutzten Rechner sowie der Tatsache, dass die Systeme parallel für andere Aufgaben genutzt wurden, lassen sich keine verbindlichen Aussagen über Laufzeiten und Laufzeitunterschiede der einzelnen Kerne treffen. Weiterhin haben auch zahlreiche andere Faktoren großen Einfluss. Die Berechnungszeiten der Baumkerne sind abhängig von der Größe der Bäume und der Anzahl der übereinstimmenden Produktionsregeln. Das bedeutet, dass durch Veränderung der Parameter zur Baumform (*Pruning*) oder zur Aufstellung der Produktionsregeln ($m$) ebenfalls die Laufzeit beeinflusst wird. Daher sollen diese hier nicht weiter untersucht werden. Als grober Richtwert lässt sich sagen, dass ein Lernlauf des Baumkerns in der Regel ca. 5-10 Minuten, im schlechtesten Fall aber auch über 30 Minuten in Anspruch nimmt. Das gleiche gilt dementsprechend auch für den kombinierten Kern. Der lineare Kern alleine ist deutlich schneller und viel geringeren Schwankungen unterworfen. Durch die in 8.4.2 vorgestellte Repräsentation der Featurevektoren wird auch durch eine Erhöhung der Dimension, durch Hinzufügen von spärlich besetzten Features, die Laufzeit nicht maßgeblich beeinflusst. Der Kern benötigt im Schnitt nur ca. 1-2 Minuten.

### 9.2.2. Versuche mit dem Baumkern

Es wurden verschiedene Baumformen mit unterschiedlichen Werten für $m$ getestet. Die Ergebnisse sind in den Tabellen 9.2, 9.3 und 9.4 aufgeführt. Aufgrund der Eigenschaft deutscher Sätze, sehr weitläufig sein zu können, ist der von Zhang (ZZS06) beschriebene SPT-Baum (siehe Abschnitt 8.4.1) für diese Sprache nicht optimal. Bessere Ergebnisse könnte der von Zhou (ZZJZ07) beschriebene Teilbaum erzielen. Bei diesem bleiben zusätzliche Kontextinformationen auf Wortebene im Baum erhalten. Das gelingt, indem das zu einer Relation gehörige Prädikat ermittelt und der SPT um das entsprechende Baumfragment erweitert wird. Die zuverlässige Erzeugung dieser Baumform wird im vorliegenden Fall stark erschwert durch die teilweise mangelhaften Ergebnisse des Parsers.

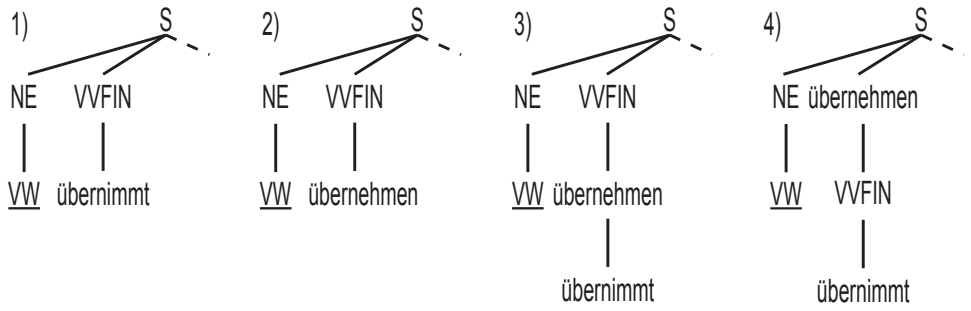



Abbildung 9.1.: Wortstämme auf verschiedenen Ebenen des Parsebaums

Versuche mit andereren beschnittenen Baumformen haben ebenfalls keine vielversprechenden Ergebnisse gebracht. Es wurden die Baumformen aus Abschnitt 8.4.1 getestet. In dem Fall, dass ein Satz mehrere Relationskandidaten enthält, ist es notwendig, die zu dem aktuellen Relationskandidaten gehörenden Entities im Parsebaum zu markieren. Ohne diese Markierung würde sonst das Lernverfahren bei Anwendung des reinen Baumkerns für jeden Relationskandidaten einen identischen Baum sehen und hätte keine Möglichkeit, diese voneinander unterscheiden zu können. Die Merger-Kandidaten wurden daher durch einen zusätzlichen Knoten *„MRG“* im Baum oberhalb der Blätter markiert. Zusätzlich sollten weitere Merkmale in den Baum integriert werden. Die Stammformen der Wörter wurden in verschiedenen Höhen als neue Knoten in den Baum integriert. Abbildung 9.1 zeigt die vier verschiedenen Baumtypen, die im Experiment verwendet wurden. (1)) ist der unveränderte Teilbaum. Bei (2)) wurden alle Terminalsymbole durch ihre Stammform ersetzt. (3)) zeigt den Baum nach Einfügen der Stammform auf Ebene 0. In Baum (4)) wurde die Stammform auf Ebene 1 integriert.
Die Parsebäume wurden mit dem Stanford-Parser erzeugt. Die Ergebnisse wurden durch 10fache Kreuzvalidierung überprüft. Die Parameterwerte sind folgendermaßen gesetzt: $C = 2.4$, $m = 1 \ldots 3$ und $\lambda = 0.4$.

| Featureset | Precision | Recall | F-meas. |
|---|---|---|---|
| *fulltree* | 20,58% | 74,63% | 32,26% |
| *Stammform-ersetzen* | 21,43% | 58,21% | 31,33% |
| *Stammform-0* | 19,37% | 55,22% | 28,68% |
| ***Stammform-1*** | **22,67%** | **76,12%** | **34,93%** |
| *Stammform-2* | 21,39% | 64,18% | 32,09% |

Tabelle 9.2.: Ergebnisse für Bäume mit Stammformen, $m = 1$

| Featureset | Precision | Recall | F-meas. |
|---|---|---|---|
| *fulltree* | 20,08% | 76,12% | 31,78% |
| *Stammform-ersetzen* | 18,92% | 31,34% | 23,60% |
| *Stammform-0* | 19,05% | 53,73% | 28,13% |
| ***Stammform-1*** | **25,36%** | **79,10%** | **38,41%** |
| *Stammform-2* | 24,07% | 58,21% | 34,06% |

Tabelle 9.3.: Ergebnisse für Bäume mit Stammformen, $m = 2$

| Featureset | Precision | Recall | F-meas. |
|---|---|---|---|
| *fulltree* | 22,50% | 40,30% | 28,88% |
| ***Stammform-ersetzen*** | **26,32%** | **74,63%** | **38,91%** |
| *Stammform-0* | 22,83% | 31,34% | 26,42% |
| *Stammform-1* | 21,49% | 73,13% | 33,22% |
| *Stammform-2* | 19,46% | 43,28% | 26,85% |

Tabelle 9.4.: Ergebnisse für Bäume mit Stammformen, $m = 3$

### 9.2.3. Versuche mit dem linearen Featurekern

Der Einfluss der flachen Features für den linearen Kern sollte ebenfalls untersucht werden. Die Ergebnisse werden in Tabelle 9.5 dargestellt. In beiden Vektoren nicht gesetzte Features wurden nicht als Übereinstimmung gezählt. Sowohl die Features von (ZSZZ05) als auch die eigenen Wortvektor-Features werden in den Experimenten miteinander verglichen.

**Zhou-Wort-Features** Der erste Datensatz enthält die von Zhou et. al. (ZSZZ05) zur Relation Extraction vorgeschlagenen Wort-Features. Diese Features betrachten die Wörter innerhalb eines Satzes, die neben oder zwischen den Relationspartnern stehen. Es werden von Zhou 14 verschiedene Arten von Wort-Features beschrieben, von denen hier nur einige genannt werden:

- **WBNULL**: TRUE, falls keine Wörter zwischen Entity1 und Entity2
- **WBFL**: Das Wort, falls nur genau ein Wort zwischen E1 und E2 liegt
- **BM1F**: Das erste Wort vor Entity1
- **AM2F**: Das erste Wort nach Entity2
- ...

Features dieser Art, die sich direkt auf die Entität beziehen, haben sich für andere Lernaufgaben wie z. B. NER als sehr nützlich herausgestellt. Sie beschreiben unter anderem Eigenschaften des Wortes in Form von N-Grammen, Verallgemeinerungen, Worte an bestimmter Position relativ zu den Entities. Dadurch wird die sequentielle Struktur abgebildet, die für einige Lernaufgaben von großer Bedeutung ist. Für die Relation Extraction ist es jedoch dringend notwendig, ein weiteres Umfeld innerhalb des Satzes mit einzubeziehen. Die notwendigen Informationen, um eine Relation zu bestimmen, können in Sätzen an sehr unterschiedlichen Stellen platziert sein. Die in Abschnitt 3.3.5 genannten Besonderheiten der deutschen Sprache erschweren die Suche nach zusammengehörigen Satzelementen. Aufgrund dieser besonderen Eigenschaft sind die zuvor genannten Features für die Relation Extraction nicht optimal geeeignet. Stattdessen soll, insbesondere zur Unterscheidung positiver und negativer Relationskandidaten, der weitere Kontext mit einbezogen werden.

**Wortvektor-Features** Aufgrund der Vermutung, dass das Auftreten bestimmter Schlüsselwörter ein sehr guter Indikator für das Vorliegen einer Relation sein kann, wurden Word-Vector-Features integriert. Das bedeutet, der Featurevektor wird aus den im Korpus vorkommenden Wörtern gebildet. Zur Verallgemeinerung werden jedoch stets nur die Stammformen der Wörter betrachtet. Ist ein Wort in einem Satz enthalten, so hat der zugehörige Wortvektor an dieser Stelle den Wert 1, ansonsten ist er 0. Außerdem werden nur Wörter mit einer Mindestlänge von drei Zeichen beachtet. Dies erfolgt aufgrund der Annahme, dass zum einen zu kurze Wörter nur eine geringe Wichtigkeit haben und in der Mehrzahl Stoppwörter wie „ein“, „und“, „der“, „die“, „das“ etc. sind, zum anderen, dass Sonder- und Satzzeichen somit ebenfalls unberücksichtig bleiben.

| Featureset | Precision | Recall | F-meas. |
|---|---|---|---|
| *Zhou-Features* | 46,32% | **78,57%** | 32,83% |
| *WV-Features* | 78,95% | 24,54% | 37,46% |
| ***Zhou & WV-Features*** | **80,26%** | 51,64% | **61,60%** |

Tabelle 9.5.: Ergebnisse des linearen Featurekerns

### 9.2.4. Versuche mit dem kombinierten Kern

Alle nachfolgenden Versuche wurden mit dem Composite Kernel durchgeführt, der eine Kombination aus dem flachen Featurekern und dem kontextsensitiven Baumkern ist. Als Kombination wurde die *Linearkombination* mit dem Wert $\alpha = 0,6$ eingestellt. Die Ergebnisse sind in Tabelle 9.6 abgebildet.

**Zhou-Features/fulltree - Datensatz** Der Datensatz für den flachen Featurekern besteht aus den Zhou-Features.
Der Datensatz für den Baumkern besteht aus dem kompletten Parsebaum des Satzes, aus dem der Relationskandidat generiert wurde.

**WV/fulltree - Datensatz** Der Datensatz für den flachen Featurekern besteht aus den Wortvektor-Features.
Der Baumkern erhält den kompletten Parsebaum des Satzes.

**Zhou & WV-Features/fulltree - Datensatz** Der Featurekern erhält eine Kombination aus den Zhou-Features und den Wortvektoren.
Der Baumkern erhält den kompletten Parsebaum des Satzes.

**Stammform-x - Datensätze** Der Featurekern erhält die Daten des Wortvektor-Featuresets.
Der Baumkern bekommt den kompletten Parsebaum des Satzes in einer modifizierten Form. In der Variante *Stammform-ersetzen* werden die in den Blattknoten enthaltenen Wörter durch ihre Stammform ersetzt. In den Versionen *Stammform-(*$0 \ldots 2$*)* wird die Stammform der Wörter in den Höhen $0 \ldots 2$ in den Baum integriert, wie in Abb. 9.1 dargestellt.

| Featureset | Precision | Recall | F-meas. |
|---|---|---|---|
| *Zhou-Features/fulltree* | 33,47% | 52,27% | 38,64% |
| *WV/fulltree* | 36,41% | 69,93% | 45,45% |
| ***Zhou & WV-Features*** | 36,83% | 74,86% | **48,73%** |
| *Stammform-ersetzen* | 31,46% | **76,03%** | 44,08% |
| *Stammform-0* | 37,94% | 47,90% | 41,79% |
| *Stammform-1* | **44,33%** | 53,42% | 47,51% |
| *Stammform-2* | 36,28% | 62,91% | 45,64% |

Tabelle 9.6.: Ergebnisse der Experimente mit dem Composite Kernel

### 9.2.5. Auswertung der Experimente

**Baumkern** Die Integration der Stammformen in den Baum erbrachte eine leichte Verbesserung der Performance. Die Position, an der sie eingefügt wird, ist ein wichtiger Faktor. *Stammform-1* lieferte in der Regel die besten Ergebnisse. Die Wahl des Parameters $m$ hat ebenfalls deutlichen Einfluss auf die Berechnung. Er ist allerdings stark abhängig von der gewählten Form des Baumes.

| Featureset | Precision | Recall | F-meas. |
|---|---|---|---|
| *Zhou-Features/fulltree* | **3,88%** | 21,99% | 8,88% |
| *WV/fulltree* | 12,16% | 11,64% | 5,12% |
| ***Zhou & WV-Features*** | 5,15% | 9,55% | **3,12%** |
| *Stammform-ersetzen* | 4,63% | **8,99%** | 4,32% |
| *Stammform-0* | 6,29% | 14,55% | 9,07% |
| *Stammform-1* | 7,58% | 9,89% | 4,40% |
| *Stammform-2* | 3,95% | 10,89% | 4,62% |

Tabelle 9.7.: Standardabweichung der Experimente mit dem Composite Kernel

**Linearer Kern** Die Auswertung der Versuche mit dem linearen Kern zeigt, dass die Stärke der Wortvektor-Features die Präzision ist. Das unterstützt die Theorie, dass bestimmte Schlüsselwörter als wichtiger Indikator für Fusionen dienen können. Die Zhou-Features hingegen haben einen hohen Recall, aber nur geringe Präzision. Die von ihnen abgebildeten Features sind also zu allgemein, um präzise unterscheiden zu können, ob eine Fusion vorliegt oder nicht. Kombiniert man beide Featuremengen, so verbessern sich die Ergebnisse signifikant: ein *f-measure* von $61,90$ bei einer Standardabweichung von +/- 7.28% ist der beste Wert insgesamt, der bei den Versuchen erreicht werden konnte.

**Kombinierter Kern** Betrachtet man die Ergebnisse des zusammengesetzten Kerns, so sind diese in der Regel besser, als die der einzelnen Kerne für sich genommen. Auch hier konnte der Kern mit den erweiterten Wortfeatures das beste Ergebnis erzielen. Dieses fiel jedoch erstaunlicherweise schlechter aus, als bei alleiniger Benutzung des linearen Kerns. Durch die Kombination mit den strukturierten Features des Baums konnte also kein zusätzlicher Performancegewinn erreicht werden. Das zweitbeste Ergebnis der zusammengesetzten Kerne erreichte der *Stammform-1*-Kern, der nur das Wortvektor-Featureset benutzt. Dieser hat zwar eine höhere Präzision, aber einen geringeren Recall zu verzeichnen.

### 9.2.6. Fazit

Abschließend lässt sich sagen, dass für die Aufgabe, Firmenfusions-Relationen zu erkennen, der einfache lineare Kern mit der Kombination aus den Zhou- mit den Wortvektor-Features sich als am besten geeignet herausgestellt hat. Sowohl bei den Ergebnisse als auch bei den Laufzeiten ist er deutlich im Vorteil. Zusätzliche zeitaufwändige Vorverarbeitungsschritte wie das vollständige Parsen entfallen bei seiner Nutzung ebenfalls.
Die Ergebnisse, die mit dem Composite Kernel von (ZZJZ07) für die *ACE - RDC*-Aufgabe erzielt wurden, konnten für das vorliegende Korpus nicht erreicht werden. Dafür

liegen mehrere Gründe vor:

- Die Eigenschaften der deutschen Sprache stellen andere Anforderungen an die Erzeugung von Baumformen und Features als die englische Sprache. Die weitläufigen Satzkonstruktionen machen es erforderlich, die Struktur eines Satzes noch genauer zu betrachten. Ansätze wie der SPT-Baum sind nicht in der Lage, alle Informationen abzubilden, die notwendig sind, eine Relation zu erkennen.

- Die Qualität des auf dem *NEGRA*-Korpus trainierten Stanford-Parsers war bei einigen Sätzen mangelhaft. Die erzeugten Parsebäume sind oft nicht korrekt, wodurch wichtige übereinstimmende Strukturmerkmale, die theoretisch in Sätzen enthalten wären, durch falsches Parsen jedes Mal unterschiedlich dargestellt werden und somit nicht erkannt werden können.

- Aufgrund der zahlreichen Parameter ist die Optimierung des Systems eine schwierige Angelegenheit. Bereits kleine Änderungen der Werte haben oft einen großen Einfluss auf die Berechnungsqualität. Die zahlreichen Kombinationsmöglichkeiten sowie die Dauer der Durchläufe inkl. der notwendigen Kreuzvalidierungen machen eine systematische Erforschung zu einer langwierigen Aufgabe, an deren Ende man möglicherweise für diese spezielle Trainingsmenge bessere Performancewerte erzielen könnte.

- Der Aufbau des Korpus selber ist ebenfalls ein Faktor, der die Relationserkennung erschwert. Die Sätze entstammen aus den unterschiedlichsten Quellen im Internet, sodass sie von ihrer Struktur und Formulierung her völlig inhomogen sind. Dies unterscheidet die Daten vom *ACE*-Korpus insofern sehr deutlich, als dass dort alle Texte aus Pressemeldungen (Broadcast News und Newswire) und aus nur zwei unterschiedlichen Quellen stammen. Daraus folgt, dass diese in der Regel relativ ähnlich formuliert sind.

- In Abschnitt 9.1 wurde das Problem genannt, dass es bereits für einen Menschen oftmals relativ schwer ist zu entscheiden, ob ein Satz eine Relation enthält oder nicht. Das bedeutet, es gibt keine klar definierte, saubere Trennung zwischen positiven und negativen Beispielen. Dies spiegelt sich auch in der Trainingsmenge wider. Dadurch erhält das Lernverfahren von Anfang an keine Grundlage, vollkommen sicher entscheiden zu können.

- Die Vorverarbeitung des *ACE*-Korpus durch professionelle Linguisten und Annotatoren trägt ein Übriges dazu bei, das dieses wesentlich effizienter verarbeitet werden kann. Fehler, die durch falsche Satztrennung, fehlerhaftes Tagging von Entities etc. entstehen, sind dadurch ausgeschlossen. Diese Arten von Fehler haben starken

Einfluss auf die Strukturen, wodurch die vorliegenden Ergebnisse beeinträchtigt werden.

- Die positiven Ergebnisse des linearen Kerns unter Berücksichtigung der Wortvektor-Features lässt sich dadurch erklären, dass nur wenige Sätze sowohl positive als auch negative Relationskandidaten enthalten. Dadurch wird die Aufgabe, Relationen zu erkennen, im Grunde reduziert auf die Aufgabe, eine Textklassifikation auf Basis einzelner Sätze durchzuführen. Für diese Aufgabe hat sich bereits früher z. B. bei (Joa98) die Wortvektor-Repräsentation in Kombination mit SVM-Lernverfahren als erfolgreich herausgestellt.

Grundsätzlich ist die Idee, strukturierte Informationen aus Parsebäumen für die Relationserkennung zu nutzen, jedoch nicht zu verwerfen. Auf anderen Korpora konnten damit gute Ergebnisse erzielt werden. Die Resultate des linearen Kerns alleine waren bei weitem noch nicht perfekt und lassen viel Raum für Verbesserungen zu. Daher ist nicht ausgeschlossen, dass durch Optimierung der Verfahren auch für die vorliegende Aufgabe bessere Ergebnisse durch Anwendung von Baumkernen erzielt werden könnten. Die oben genannten Punkte bieten dabei zahlreiche Ansatzpunkte, an welchen Stellen Verbesserungen durchgeführt werden könnten.

# 10. Zusammenfassung und Ausblick

Diese Diplomarbeit hat sich mit Relationserkennung zur Extraktion von Firmenfusions-Relationen aus deutschsprachigen Internetseiten befasst. Die Entwicklung von zahlreichen Tools zur Erfüllung dieser Aufgabe stellte dabei einen großen Teil der Arbeit dar. Kernfunktionen für Parsebäume von Sätzen wurden auf ihre Eignung hin untersucht. Durch die Integration neuer Operatoren in die Lernumgebung *RapidMiner* wurde die Grundlage für spätere Forschungen in diesem Bereich geschaffen.

Der eigentlichen Implementierung ging eine umfassende Einarbeitung in den Bereich der Information Extraction voraus. Die zahlreichen vorhandenen Techniken und Verfahren machten die Auswahl eines geeigneten Verfahrens nicht leicht. Durch die Notwendigkeit umfassender Vorverarbeitungsschritte sowie die Tatsache, dass viele Softwarelösungen zwar für das Englische, nicht aber für die deutsche Sprache verfügbar waren, wurden intensive Recherchen nach Lösungen für grundlegende – der eigentlichen Aufgabe vorgelagerte – Probleme erforderlich.

Die Auswahl der Lernmethode erfolgte in mehreren Schritten. Zu Anfang wurden einige Versuche mit dem *Frequent Termset Clustering* (BEX02) durchgeführt. Dafür wurde ein entsprechender Operator in *RapidMiner* integriert und mit Hilfe des *Word Vector*-Plugins getestet. Dieser Ansatz wurde jedoch bereits nach kurzer Zeit aufgegeben. Statt dessen sollten Kernmethoden untersucht werden. Dafür stellte sich die Implementierung des *Fast Tree Kernel* von Moschitti (Mos06) als vielversprechend heraus. Dieser operiert auf Parsebäumen von Sätzen. Zur Erzeugung von diesen – auch für die deutsche Sprache – konnte der Stanford-Parser gefunden und eingesetzt werden. Erste Tests mit diesem Kern in der SVM$^{light}$ stellten sich als vielversprechend heraus, sodass die Arbeiten in dieser Richtung weitergeführt wurden. Später wurde die Implementierung erweitert, sodass der vorliegende Kern dem *Context Sensitive Convolution Tree Kernel* (ZZJZ07) entspricht. Auch die Betrachtung und Umsetzung von Experimenten mit einfachen linearen Kernen, sowie eine Kombination beider Kernarten als *Convolution Kernel* wurde durchgeführt.

Die eigentliche Bearbeitung der Aufgabe, Firmenfusionen zu erkennen, erfolgte in mehreren Schritten. Zunächst wurden geeignete Dokumente im Internet gesucht und abge-

speichert. Dies erfolgte mit Hilfe der dafür entwickelten *Crawler*-Anwendung, die auf die Google-API zurückgreift und anhand bestimmter Schlagworte große Mengen von Dokumenten herunterlädt.

Um die Dokumente verarbeiten zu können, wurde ein weiteres Softwaretool vom Diplomanden entwickelt. Diese *Selector* genannte Anwendung ist dafür zuständig, aus den html-Dateien alle Relationskandidaten zu erzeugen. Die html-Dateien werden eingelesen und automatisch in einzelne Sätze und Wörter zerteilt. Basierend auf einer Liste von bekannten Firmennamen werden alle Firmen-Entitäten markiert. Filterschritte ermöglichen, die Menge der Sätze so einzuschränken, dass diejenigen, die mit Sicherheit die gewünschte Relation nicht enthalten, automatisch entfernt werden. Die übrigen Sätze wurden vom Diplomanden einzeln mit Tags versehen. Ziel dabei war es, Firmen, die als Fusionspartner in einem Satz auftreten, kenntlich zu machen. Die *Selector*-Anwendung soll diese Aufgabe dabei so komfortabel wie möglich gestalten. Auch die Erzeugung von Parsebäumen und deren Visualisierung kann innerhalb dieser Anwendung erfolgen. Das XML-Format, in dem die bearbeiteten Daten durch *Selector* abgespeichert werden, erlaubt eine flexible Weiterverarbeitung.

Die auf diese Weise in *Selector* vorbereiteten Sätze werden als nächstes in eine Tabellenform konvertiert, die durch die Lernumgebung *RapidMiner* genutzt werden kann. Hier sorgt das *Information Extraction*-Plugin für eine Weiterverarbeitung der Daten. In Zusammenarbeit mit Felix Jungermann entstand der *RelationCandidateGenerator*-Operator, der die Sätze in einzelne Relationskandidaten zerlegt. Ebenfalls zusammen mit Felix Jungermann wurde der *ParseTreeVisualizer*-Operator zur Darstellung der Bäume in *RapidMiner* umgesetzt. Zwei weitere Operatoren wurden vom Diplomanden zur Erfüllung der eigentlichen Lernaufgabe implementiert. Um den Einfluss der Form von Bäumen untersuchen zu können, wurde der *TreeShape*-Operator entwickelt. Dieser kann einen Parsebaum auf unterschiedliche Weisen beschneiden. Weiterhin können beliebige Features oberhalb der Relationskandidaten neu in den Baum integriert werden. Der *ParseTreeSVM*-Operator ist eine auf der *MySVM* basierende Implementierung einer *Support Vector Machine*. Diese wurde um die Möglichkeit erweitert, beliebige Kernfunktionen für Objekte jeder Art nutzen zu können. Daten, die z. B. in serialisierter Form als String in den Attributen des *RapidMiner*-Examplesets gespeichert sind, werden vor der Verarbeitung durch den Kern in ein entsprechendes Java-Objekt umgewandelt. Die Methode dafür wird nur durch die Kernfunktion selbst festgelegt und die Umwandlung erfolgt für *RapidMiner* völlig transparent.

In der vorliegenden Version wurden außerdem die in (ZZJZ07) vorgestellten Kerne imple-

mentiert. Dieses umfasst den *Context Sensitive Convolution Tree Kernel*, einen linearen Kern, der übereinstimmende Features zählt, sowie einen kombinierten Kern (Composite Kernel). Zahlreiche Parameter ermöglichen, das Verhalten der Kerne auf verschiedene Weisen zu beeinflussen.

Mit Hilfe dieser Werkzeuge wurden Untersuchungen zur Eignung verschiedener Kerne und Kombinationen von Kernen zur Relationserkennung durchgeführt. Die Ergebnisse dieser Untersuchungen ergaben für den vorliegenden Datensatz keinen Vorteil für die Nutzung eines Baumkerns. Es wurden jedoch zahlreiche Gründe ermittelt, die die vorliegenden Ergebnisse erklären und Ansätze für zukünftige Verbesserungen und Erweiterungen geben.

Die implementierte Kernfunktion kann in *RapidMiner* auch in andere Operatoren integriert werden, die den Abstand zwischen Objekten berechnen müssen. Am Lehrstuhl für Künstliche Intelligenz der Universtität Dortmund konnte sie z. B. erfolgreich zum Clustern von SQL-Parsebäumen eingesetzt werden.

Zur Erzeugung und Visualisierung von Firmendossiers in Form eines Netzwerkes wurde vom Diplomanden eine weitere Software entworfen. Diese Anwendung wurde *CompanyGrid* genannt. Beziehungen, die in der Geschäftswelt zwischen Unternehmen und deren wichtigen Managern bestehen, sollten auf eine strukturierte, für den Benutzer leicht zu erfassende Art sichtbar gemacht werden.
Die Grundlage dieser Anwendung bildet eine Datenbank mit Firmeninformationen. Diese wurde zu Beginn aus Datenquellen aus dem Internet befüllt. Dort liegen die Daten in strukturierter Form als html-Dateien vor und konnten mit Hilfe eines einfachen Parsers extrahiert und lokal gespeichert werden. Erste Versuche, die Daten in Form von XML-Dateien abzuspeichern, ergaben, dass die gezielte Abfrage von Informationen aus diesen Dateien zu lange dauerte. Daher wurde die Umsetzung mit Hilfe einer SQL-Datenbank realisiert. Die Benutzeroberfläche konnte durch Einsatz des *JUNG*-Frameworks zur Graph-Visualisierung flexibel und vielseitig gestaltet werden. Das Ziel, einem Benutzer die Erforschung der Firmengraphen zu erlauben, konnte damit erfüllt werden. Die Einbindung der Firmenfusionsrelationen ist ebenfalls ermöglicht worden. Dafür wurde eine entsprechende Methode in der Datenbank-Klasse implementiert, die neue Relationen jederzeit und unkompliziert in das System einfügen kann. In Kombination mit einem System zur Relationsextraktion kann damit eine automatische Ergänzung der Datenbank von *CompanyGrid* durchgeführt werden.

## 10.1. Ausblick

Die getesteten Kernfunktionen verfügen über zahlreiche Parameter, deren unterschiedlicher Einfluss auf die Genauigkeit der Erkennung systematisch untersucht werden kann. Zu diesem Zweck sollte ein einheitlicher, homogener und gut vorverarbeitetes Korpus genutzt werden, um andere Einflussfaktoren und Fehlerquellen weitestgehend ausschließen zu können. Auch der Einfluss unterschiedlicher Baumfunktionen sowie der zusätzlichen Features, die in den Baum integriert werden können, bieten Raum für weitere Forschungen.

Die entwickelten Programme und Operatoren bieten ebenfalls zahlreiche Möglichkeiten zur Erweiterung und Verbesserung. Die für *RapidMiner* entwickelten Operatoren können für neue Aufgabenbereiche und Objektstrukturen angepasst werden. Langfristig soll mit dem *Information Extraction*-Plugin eine umfassende Sammlung von Werkzeugen entstehen, die für verschiedene Aufgaben im Bereich der Verarbeitung natürlicher Sprache verwendet werden kann. Aber auch in anderen Bereichen kann der neue SVM-Operator mit Kernfunktionen für strukturierte Objekte erfolgversprechend eingesetzt werden.

Auch die *CompanyGrid*-Software bietet viel Freiraum für Erweiterungen. Sinnvoll wäre eine Erweiterung der GUI, um dem Benutzer weitere Interaktionsmöglichkeiten zur Verfügung zu stellen. Die Anzeige von Detailinformationen zu den Firmen kann erweitert werden. Es können weitere Abfragemöglichkeiten hinzugefügt werden, sodass gezielt nach verschiedenen Arten von Informationen gesucht werden kann. Außerdem können mathematische Verfahren zur Auswertung der Graphstruktur genutzt werden, durch die z. B. automatisch interessante Strukturen innerhalb der Graphen aufgefunden werden. Es können außerdem neue Relationstypen integriert werden. Auch eine Oberfläche, die es Benutzern ermöglicht, die Datenbank direkt zu bearbeiten und zu erweitern, wäre eine sinnvolle Ergänzung.

# Abkürzungsverzeichnis

| | |
|---|---|
| API | Application Programming Interface |
| CFG | Context Free Grammar |
| CK | Composite Kernel |
| CSCTK | Context-Sensitive Convolution Tree Kernel |
| DOP | Data-Oriented Parsing |
| HMM | Hidden Markov Model |
| IE | Information Extraction |
| IR | Information Retrieval |
| JDBC | Java Database Connectivity |
| JSON | JavaScript Object Notation |
| JUNG | Java Universal Network/Graph |
| LEK | Linear Entity Kernel |
| MEMM | Maximum Entropy Markov Model |
| NER | Named Entity Recognition |
| NLP | Natural Language Processing |
| PCFG | Probabilistic Context-Free Grammar |
| POS | Part-Of-Speech |
| REST | Representational State Transfer |
| SPT | Shortest Path Tree |
| SQL | Structured Query Language |
| UTF | Unicode Transformation Format |

# Literaturverzeichnis

[AG00] Agichtein, Eugene und Luis Gravano: *Snowball: extracting relations from large plain-text collections.* In: *DL '00: Proceedings of the fifth ACM conference on Digital libraries*, Seiten 85–94, New York, NY, USA, 2000. ACM.

[BaSBE07] Banko, Michele, Michael J Cafarella andStephen Soderland, Matt Broadhead und Oren Etzioni: *Open Information Extraction from the Web.* In: *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, Seiten 2670–2676, 2007.

[BE08] Banko, Michele und Oren Etzioni: *The Tradeoffs Between Open and Traditional Relation Extraction.* In: *Proceedings of ACL-08: HLT*, Seiten 28–36, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[BEX02] Beil, Florian, Martin Ester und Xiaowei Xu: *Frequent term-based text clustering.* In: *KDD*, Seiten 436–442, 2002.

[Blo33] Bloomfield, Leonard: *Language.* University of Chicago Press, Chicago, 1933.

[BM05] Bunescu, Razvan und Raymond Mooney: *A Shortest Path Dependency Kernel for Relation Extraction.* In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, Seiten 724–731, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics.

[BM06] Bunescu, Razvan und Raymond Mooney: *Subsequence Kernels for Relation Extraction.* In: Weiss, Y., B. Schölkopf und J. Platt (Herausgeber): *Advances in Neural Information Processing Systems 18*, Seiten 171–178. MIT Press, Cambridge, MA, 2006.

[Bod98] Bod, Rens: *Beyond Grammar: An Experience-Based Theory of Language.* CSLI Publications, 1998.

[Bri92] Brill, Eric: *A simple rule-based part of speech tagger.* In: *HLT '91: Proceedings of the workshop on Speech and Natural Language*, Seiten 112–116, Morristown, NJ, USA, 1992. Association for Computational Linguistics.

[Bri98] Brin, Sergey: *Extracting Patterns and Relations from the World Wide Web.* In: *WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT'98*, 1998.

[CD01] Collins, Michael und Nigel Duffy: *Convolution Kernels for Natural Language.* In: *Advances in Neural Information Processing Systems 14*, Seiten 625–632. MIT Press, 2001.

[Cha93] Charniak, Eugene: *Statistical Language Learning.* MIT Press, 1993.

[Cho56] Chomsky, N.: *Three models for the description of language.* Information Theory, IEEE Transactions on, 2(3):113–124, 1956.

[CS04] Culotta, Aron und Jeffrey Sorensen: *Dependency tree kernels for relation extraction.* In: *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, Seite 423, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

[CV95] Cortes, Corinna und Vladimir Vapnik: *Support-Vector Networks.* Mach. Learn., 20(3):273–297, 1995.

[DBL+] Dipper, Stefanie, Thorsten Brants, Wolfgang Lezius, Oliver Plaehn und George Smith: *The TIGER Treebank.*

[DGM08] Debnath, Souvik, Niloy Ganguly und Pabitra Mitra: *Feature Weighting in Content Based Recommendation System Using Social Network Analysis.* In: *WWW 2008.* ACM Press, 2008.

[DR01] Domingos, Pedro und Matt Richardson: *Mining the network value of customers.* In: *Procs. KDD*, Seiten 57–66. ACM Press, 2001.

[Dub06] Dubey, Amit: *Statistical Parsing for German : modeling syntactic properties and annotation differences.* Doktorarbeit, Professur für Psycholinguistik Universität des Saarlandes, 2006.

[ECD+04] Etzioni, Oren, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld und Alexander Yates: *Web-scale information extraction in knowitall: (preliminary results).* In: *WWW '04: Proceedings of the 13th international conference on World Wide Web*, Seiten 100–110, New York, NY, USA, 2004. ACM.

[Fie00] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures.* Doktorarbeit, University of California, Irvine, 2000.

[Gee96] Gee, F. Ruth: *The TIPSTER text program overview.* In: *Proceedings of a workshop on held at Baltimore, Maryland*, Seiten 3–5, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

[GH06] Geyken, Alexander und Thomas Hanneforth: *TAGH: A Complete Morphology for German based on Weighted Finite State Automata.* In: *Finite State Methods and Natural Language Processing. 5th International Workshop, FSMNLP 2005, Helsinki, Finland, September 1-2, 2005. Revised Papers*, Band 4002, Seiten 55–66. Springer, 2006.

[GK01] Geyken, Alexander und Wolfgang Klein: *Projekt Digitales Woerterbuch der deutschen Sprache des 20. Jh.* In: *Jahrbuch der BBAW 2000*, Seiten 263–270. Akademie Verlag, 2001.

[Gri97] Grishman, Ralph: *Information Extraction: Techniques and Challenges.* In: *SCIE*, Seiten 10–27, 1997.

[GS96] Grishman, Ralph und Beth Sundheim: *Message Understanding Conference-6: a brief history.* In: *Proceedings of the 16th conference on Computational linguistics*, Seiten 466–471, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

[GT94] Grefenstette, G. und P. Tapanainen: *What is a word, what is a sentence? problems of tokenization*, 1994.

[GWH07] Golder, Scott A., Dennis M. Wilkinson und Bernardo A. Huberman: *Rhythms of social interaction: Messaging within a massive online network.* In: *Procs. 3rd Intl. Conf. on Communities and Technologies*, 2007.

[Hau99] Haussler, David: *Convolution Kernels on Discrete Structures.* Technischer Bericht UCSC-CRL-99-10, UC Santa Cruz, 1999.

[HJM09] Had, Martin, Felix Jungermann und Katharina Morik: *Relation Extraction for Monitoring Economic Networks.* In: *Proceedings of the NLDB 2009*, 2009. Not yet published.

[JMNR99] Jones, Rosie, Andrew McCallum, Kamal Nigam und Ellen Riloff: *Bootstrapping for Text Learning Tasks.* In: *IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications*, 1999.

[JO03] Joshua O'Madadhain, Danyel Fisher, Scott White Yan-Biao Boey: *The JUNG (Java Universal Network/Graph) Framework.* Technischer Bericht Technical Report UCI-ICS 03-17, School of Information and Computer Science University of California, Irvine, CA 92697-3425, 2003.

[Joa98] Joachims, Thorsten: *Text Categorization with Support Vector Machines: Learning with Many Relevant Features.* In: *Procs. Of European Conference on Machine Learning*, Seiten 137 – 142. Springer, 1998.

[Jun09] Jungermann, Felix: *Information Extraction with RapidMiner.* In: *Proceedings of the GSCL Symposium 'Sprachtechnologie und eHumanities'.* Universität Duisburg-Essen, Abteilung für Informatik und Angewandte Kognitionswissenschaft Fakultät für Ingenieurwissenschaften, 2009.

[KHW06] Kuebler, Sandra, Erhard W. Hinrichs und Maier Wolfgang: *Is it Really that Difficult to Parse German.* In: *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP 2006)*, Seiten 111–119, 2006.

[KM02] Klein, Dan und Christopher D. Manning: *Fast Exact Inference with a Factored Model for Natural Language Parsing.* In: *NIPS*, Seiten 3–10, 2002.

[KP08] Kübler, Sandra und Gerald Penn (Herausgeber): *Proceedings of the Workshop on Parsing German.* Association for Computational Linguistics, Columbus, Ohio, June 2008.

[LDC04a] LDC: *Annotation Guidelines for Entity Detection and Tracking (EDT).* LDC, 2004.

[LDC04b] LDC: *Annotation Guidelines for Relation Detection and Characterization (RDC).* LDC, 2004.

[Lez96] Lezius, Wolfgang: *Morphologiesystem Morphy.* In: Hausser, R. (Herausgeber): *Linguistische Verifikation. Dokumentation zur ersten Morpholympics 1994*, Seiten 25–35. Niemeyer, 1996.

[Lov68] Lovins, Julie B.: *Development of a Stemming Algorithm.* Technischer Bericht, MASSACHUSETTS INST OF TECH CAMBRIDGE ELECTRONIC SYSTEMS LAB, 1968.

[Mer09] Mercer, J.: *Functions of positive and negative type, and their connection with the theory of integral equations.* Philosophical Transactions of the Royal Society, London, 209:415–446, 1909.

[Mos06] Moschitti, Alessandro: *Making Tree Kernels Practical for Natural Language Learning.* In: *EACL*, 2006.

[MWK+06] Mierswa, Ingo, Michael Wurst, Ralf Klinkenberg, Martin Scholz und Timm Euler: *YALE: Rapid Prototyping for Complex Data Mining Tasks.* In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006)*, 2006.

[PMLR04] Palau, Jordi, Miquel Montaner, Beatriz López und Josep Lluís De La Rosa: *Collaboration Analysis in Recommender Systems Using Social Networks.* In: *Procs. Cooperative Information Agents VIII*, Seiten 137–151. Springer, 2004.

[Por80] Porter, M.F.: *An algorithm for suffix stripping.* Program: electronic library and information systems, 14(3):130 – 137, 1980.

[RF07] Rosenfeld, Benjamin und Ronen Feldman: *Using Corpus Statistics on Entities to Improve Semi-supervised Relation Extraction from the Web.* In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, Seiten 600–607, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[Ril96] Riloff, Ellen: *Automatically Generating Extraction Patterns from Untagged Text.* In: *AAAI/IAAI, Vol. 2*, Seiten 1044–1049, 1996.

[RM95] Ramshaw, Lance A. und Mitchell P. Marcus: *Text Chunking using Transformation-Based Learning.* CoRR, cmp-lg/9505040, 1995.

[SB88] Salton, Gerard und Christopher Buckley: *Term-weighting approaches in automatic text retrieval.* Inf. Process. Manage., 24(5):513–523, 1988.

[Sch95] Schmid, H.: *Improvements in part-of-speech tagging with an application to German*, Kapitel Lexikon und Text, Seiten 47–50. Feldweg and Hinrichs, eds, 1995.

[SFL97] Soderland, Stephen, David Fisher und Wendy Lehnert: *Automatically Learned vs. Hand-crafted Text Analysis Rules*, 1997.

[Sha51] Shannon, C. E.: *Prediction and entropy of printed English.* Bell Sys. Tech. Journal, 30:50–64, 1951.

[SS01] Scholkopf, Bernhard und Alexander J. Smola: *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA, 2001.

[SSIW00] Schmid, Helmut und Sabine Schulte Im Walde: *Robust german noun chunking with a probabilistic context-free grammar.* In: *In Proceedings of the 18th International Conference on Computational Linguistics (COLING-00)*, Seiten 726–732, 2000.

[TKMS03] Toutanova, Kristina, Dan Klein, Christopher D. Manning und Yoram Singer: *Feature-rich part-of-speech tagging with a cyclic dependency network.* In: *NAACL '03: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, Seiten 173–180, Morristown, NJ, USA, 2003. Association for Computational Linguistics.

[TKSDM03] Tjong Kim Sang, Erik F. und Fien De Meulder: *Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition.* In: Daelemans, Walter und Miles Osborne (Herausgeber): *Proceedings of CoNLL-2003*, Seiten 142–147. Edmonton, Canada, 2003.

[VS02] Vishwanathan, S. V. N. und Alexander J. Smola: *Fast Kernels for String and Tree Matching.* In: *NIPS*, Seiten 569–576, 2002.

[Wyn05] Wynne, M: *Developing Linguistic Corpora: a Guide to Good Practice.* Oxbow Books, 2005.

[YGTH00] Yangarber, Roman, Ralph Grishman, Pasi Tapanainen und Silja Huttunen: *Automatic acquisition of domain knowledge for Information Extraction.* In: *Proceedings of the 18th conference on Computational linguistics*, Seiten 940–946, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[ZAR03] Zelenko, Dmitry, Chinatsu Aone und Anthony Richardella: *Kernel methods for relation extraction.* J. Mach. Learn. Res., 3:1083–1106, 2003.

[ZG05] Zhao, Shubin und Ralph Grishman: *Extracting relations with integrated information using kernel methods.* In: *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, Seiten 419–426, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[ZSZZ05] Zhou, GuoDong, Jian Su, Jie Zhang und Min Zhang: *Exploring Various Knowledge in Relation Extraction.* In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, Seiten

427–434, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[ZZJZ07] Zhou, GuoDong, Min Zhang, DongHong Ji und QiaoMing Zhu: *Tree Kernel-Based Relation Extraction with Context-Sensitive Structured Parse Tree Information.* In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, Seiten 728–736, 2007.

[ZZS06] Zhang, Min, Jie Zhang und Jian Su: *Exploring Syntactic Features for Relation Extraction using a Convolution Tree Kernel.* In: *HLT-NAACL*, 2006.

[ZZSZ06] Zhang, Min, Jie Zhang, Jian Su und Guodong Zhou: *A Composite Kernel to Extract Relations between Entities with Both Flat and Structured Features.* In: *ACL*, 2006.

# Erklärung

Hiermit erkläre ich, Martin Had, die vorliegende Diplomarbeit mit dem Titel

**Relation Extraction zur Ergänzung deutschsprachiger Firmendossiers**

selbstständig verfasst und keine anderen als die hier angegebenen Hilfsmittel verwendet zu haben.

Dortmund, 2. März 2009