Universität zu Köln
Institut für Linguistik
Sprachliche Informationsverarbeitung

Masterarbeit
im Fach Informationsverarbeitung

Analyse von Anforderungsprofilen.

Eine Studie zur Informationsextraktion aus Stellenanzeigen

Vorgelegt von:
Mandy Neumann
Waldecker Str. 42
51065 Köln
Matrikelnummer: 4839250

Betreut durch: Prof. Dr. Rolshoven

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1. Einleitung

Natürlichsprachliche Texte gezielt nach Informationen bestimmter Kategorien zu durchsuchen ist eine Aufgabe, die den meisten Menschen kaum Probleme bereitet. Allerdings ist es aus Zeit- und Kostengründen unpraktikabel, einen oder auch mehrere Personen damit zu beauftragen, eine Sammlung von mehreren Tausend bis Millionen Dokumenten zu sichten und die enthaltene Information manuell zu strukturieren. Deswegen ist es erstrebenswert, dem Computer diese Aufgabe zu überlassen, welcher sie in einem Bruchteil der Zeit effizienter erledigen kann.

Ein mögliches und interessantes Anwendungsgebiet dieses Falles ist die automatische Verarbeitung von Stellenanzeigen. Wenn alle veröffentlichten Ausschreibungen an zentraler Stelle gesammelt und archiviert werden, wie dies beispielsweise bei der Bundesagentur für Arbeit (BA) der Fall ist, eröffnet sich eine breite Palette an Möglichkeiten der Weiterverarbeitung. Denkbar sind beispielsweise weiterführende Analysen auf den archivierten Daten, etwa Trendanalysen über Anforderungsprofile, wie sie im Laufe der Zeit an Bewerber gestellt werden.

Zur Durchführung dieser Analyseaufgaben bietet das Gebiet des *Data Mining* zahlreiche Tools. Allerdings muss dafür die Voraussetzung gegeben sein, dass die zu analysierenden Daten in strukturierter Form, z. B. in einer Datenbankstruktur, vorliegen. Bei Stellenanzeigen handelt es sich dagegen zunächst um (mehr oder weniger) unstrukturierte, natürlichsprachliche Texte. Entsprechend ist eine Vorverarbeitung nötig, welche die Überführung des unstrukturierten in das strukturierte Format ermöglicht. Werkzeuge hierfür finden wir im Bereich des *Text Mining* – konkret im Teilgebiet der *Informationsextraktion*.

Die folgenden Abschnitte geben eine kurze Einführung in das Gebiet der Informationsextraktion, seine Entwicklung und die wichtigsten Aufgaben, bevor anschließend konkret auf die Domäne der Stellenanzeigen eingegangen wird.

1.1. Informationsextraktion

Informationsextraktion, kurz IE, ist ein Teilgebiet der maschinellen Sprachverarbeitung (*Natural Language Processing*, NLP), dessen Ziel es ist, die in unstrukturierten Texten enthaltene Information zu finden und zu extrahieren. Die extrahierte Information kann anschließend etwa als Modul eines Systems zur Fragebeantwortung (*Question Answering*),

zur Erstellung besserer Indizes für *Information Retrieval* (IR) oder auch zur Hilfestellung bei der Textzusammenfassung dienen. Nicht zuletzt ist es das Ziel vieler IE-Systeme, die gesuchte Information in ein strukturiertes Format – meist in Form von Datenbanken – zu überführen, um auf diesem mittels anschließendem Data Mining weitere Analysen durchführen zu können.

Als Input der IE dient stets eine Wissensquelle. Dabei handelt es sich für gewöhnlich um eine Dokumentensammlung. Weiterhin muss die gefragte Information von vornherein klar definiert sein. Der Output der IE ist die strukturierte, semantisch explizit gemachte Information, meist in Form von Datenbankeinträgen, bei denen das Feld jeweils den Typ der Information angibt.

Im Gegensatz zum Information Retrieval, bei welchem ein Informationsbedürfnis als Anfrage an einen Suchdienst formuliert wird, der aus einer großen Dokumentensammlung eine Teilmenge relevanter Dokumente zurückliefert, ist die der IE als Input dienende Dokumentensammlung oft bereits auf eine bestimmte Domäne beschränkt. In diesem Fall dienen häufig IR-Verfahren zur Eingrenzung der Dokumentensammlung auf die Domäne. Immer häufiger werden allerdings auch domänenoffene IE-Systeme entwickelt (vgl. Etzioni et al., 2005; Banko et al., 2007; Neumann, 2010).

Die Festlegung der zu extrahierenden Information erfolgt in Form von *Templates* („Schablonen“). Dabei handelt es sich um Attribut-Wert-Paare. Es wird meist in tabellarischer Form vorgegeben, welche Felder (*Slots*) zu füllen sind und von welchem Typ der jeweilige Attributwert ist. Nach Feldman & Sanger (2007: 96) gibt es vier Typen von Information, die extrahiert werden können: Entitäten, Attribute, Fakten und Ereignisse. Entitäten sind die zentralen Einheiten in einem Text (z. B. Orte, Personen, Firmen), Attribute stellen deren Eigenschaften dar. Als Fakten werden Beziehungen zwischen Entitäten (z. B. Arbeitgeber – Arbeitnehmer) bezeichnet, während Ereignisse Aktivitäten oder Begebenheiten sind, in denen die Entitäten partizipieren. Die Slots werden entweder direkt mit den extrahierten Textsegmenten gefüllt, oder mit normalisierter Information, die von diesen durch Weiterverarbeitung abgeleitet wurde (vgl. Jurafsky & Martin, 2009: 786).

Es gibt zwei grundlegende Herangehensweisen, IE-Systeme zu entwerfen. Beim Ansatz mittels *Knowledge Engineering* werden die Regeln zur Extraktion von Hand entworfen. Demgegenüber geht es beim *Automatischen Training* darum, dem System anhand von vorausgezeichneten Trainingskorpora beizubringen, wie es die Extraktion vorzunehmen hat. Welches Verfahren zu bevorzugen ist, hängt vor allem von der Verfügbarkeit entspre-

chender Ressourcen wie Zeit, Geld, Expertenwissen oder Trainingskorpora ab (vgl. Eikvil, 1999; Appelt & Israel, 1999; Mooney & Bunescu, 2005). Beide Ansätze, ihre jeweiligen Vor- und Nachteile sowie ihre Anwendung auf das vorliegende Problem werden in Kapitel 3 respektive 4 dargelegt.

Entwicklung der IE

Die Entwicklung domänenspezifischer IE-Systeme begann etwa Ende der 1960er Jahre mit LSP (Linguistic String Project, Sager (1981)), FRUMP (Fast Reading Understanding and Memory Program, DeJong (1979)) und den Systemen von Cowie (1983) und Zarri (1983). Die recht frühe Entwicklung auch kommerzieller Systeme wie ATRANS (Lytinen & Gershman, 1986), JASPER (Andersen et al., 1992) oder SCISOR (Jacobs & Rau, 1990) zeigte, dass IE ein fruchtbares Anwendungsgebiet darstellt. Besonders vorangetrieben wurde das Forschungsfeld schließlich ab Ende der 1980er aufgrund der staatlichen Förderung durch die US-Regierung in Form der *Message Understanding Conferences* (MUCs), einer etwa jährlich stattfindende Konferenzreihe zur Evaluation von IE-Systemen. Auch nach Ende der MUCs 1998 fanden ähnliche Programme statt, wie etwa die *Automatic Content Extraction* (ACE) Evaluationen von 2000 bis 2007, sodass sich nicht zuletzt aufgrund staatlicher Förderungen die IE als wesentlicher Bestandteil des Text Mining etablieren konnte.

Die Domänen, auf die IE angewandt wurde und wird sind sehr vielschichtig und umfassen beispielsweise Arztberichte (LSP), Geldtransfermeldungen (ATRANS) sowie Nachrichten über Marineoperationen, terroristische Vorfälle, Unternehmenskooperationen oder Halbleiterfertigung (MUCs). Dennoch sind sich die jeweiligen Extraktionsaufgaben sehr ähnlich. Zur Veranschaulichung stellt Abbildung 1 den Zusammenhang zwischen einem Textausschnitt und dem zugehörigen, gefüllten Template dar. Im Text sind jeweils die extrahierten Einheiten hervorgehoben. Wie zu erkennen ist, sind die Einträge im Template teilweise direkt aus dem Text übernommen, teilweise weiterverarbeitet worden, etwa zur Normalisierung von Datums- und Zeitangaben.

Teilaufgaben

So vielfältig die einzelnen Extraktionsaufgaben in den genannten Systemen auch sind, ist ihnen allen gemein, dass das letztliche Ziel die Extraktion semantischer Relationen ist. Der Prozess lässt sich in mehrere Teilaufgaben gliedern, typischerweise sind das (nach

4 Apr Dallas - Early last evening, a **tornado** swept through an area **northwest of Dallas**, causing extensive damage. Witnesses confirm that the twister occurred without warning at approximately **7:15 p.m.** and destroyed **two mobile homes**. The **Texaco station**, at 102 Main Street, **Farmers Branch, TX**, was also severely damaged, but **no injuries** were reported. Total property damages are estimated to be **$350,000**.

$\longrightarrow$

Event	tornado
Date	4/3/97
Time	19:15
Location	Farmers Branch : "northwest of Dallas" : TX : USA
Damage	"mobile homes" (2) "Texaco station" (1)
Estimated Losses	$350,000
Injuries	none

Abbildung 1: Beispiel für einen Nachrichtentext und das zugehörige Template mit den extrahierten Informationen, nach Cardie (1997: 66).

Jurafsky & Martin, 2009):

- Eigennamenerkennung (Named Entity Recognition, NER)
- Referenzauflösung (Reference Resolution)
- Relationsaufdeckung (Relation Detection)
- Ereignisaufdeckung (Event Detection)
- Templatefüllung (Template Filling)

Zur Erfüllung dieser Aufgaben werden bekannte NLP-Techniken wie Pattern Matching oder probabilistische Methoden genutzt, die eine bestimmte Vorprozessierung der Texte erfordern. In einem ersten Schritt werden die zu verarbeitenden Texte in Einheiten verschiedener Granularität segmentiert, und zwar zum einen in thematische Sinneinheiten (*Zoning*) wie Paragraphen, und zum anderen in linguistische Einheiten wie Sätze und Tokens[1]. Im nächsten Schritt wird eine morphologische und lexikalische Analyse vorgenommen, was das Auszeichnen von Wortarten (*Part of Speech Tagging*) einschließt. Da die semantischen Relationen zwischen Entitäten in der Syntax codiert sind, wird schließlich auch meist eine syntaktische Analyse (*Parsing*) vorgenommen. Auf die durch diese Vorverarbeitungsschritte aufgebaute Struktur können nun domänenspezifischere Komponenten aufsetzen, welche sich die morphologisch-syntaktischen Informationen zunutze machen, um die eigentliche Extraktion der gewünschten Einheiten vorzunehmen und die Templates auszufüllen (vgl. Feldman & Sanger, 2007: 104f.).

[1] Tokens sind am ehesten mit dem gleichzusetzen, was gemeinhin als „Wörter“ bezeichnet wird. Je nach Kontext werden als Tokens auch alle zusammenhängenden alphanumerischen Zeichensequenzen, auch etwa Interpunktionszeichen, bezeichnet.

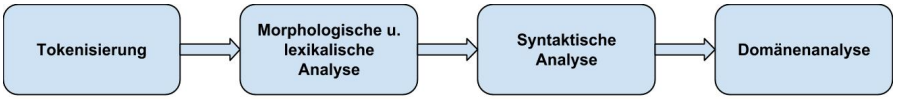

Abbildung 2: Grundlegende Arbeitsschritte eines IE-Systems (nach Feldman & Sanger (2007: 105).

1.2. Stellenanzeigen

Die Domäne der Stellenanzeigen bietet ein typisches Anwendungsgebiet für Informationsextraktion. Die Anzeigen sind in der Regel in Fließtextform verfasst, man spricht von unstrukturierten Daten. Im Text verbergen sich allerdings stereotypische Informationen, die sich für eine Extraktion in strukturierter Form, zum Zwecke der kompakten Archivierung und Weiterverarbeitung, anbieten. Es handelt sich dabei beispielsweise um Informationen zu Charakteristika des ausschreibenden Unternehmens, der ausgeschriebenen Stelle und des Anforderungsprofils des potentiellen Bewerbers.

Besonders nützlich ist die Tatsache, dass Stellenanzeigen häufig eine kanonische Struktur aufweisen, wodurch die verschiedenen Themen innerhalb der Anzeige oft klar voneinander getrennt sind. Die Vorstellung des ausschreibenden Unternehmens, der Titel der ausgeschriebenen Stelle, ihre Beschreibung, das Anforderungsprofil an den Bewerber sowie Formalia finden sich zumeist in je eigenen Abschnitten. Je nach Format (z. B. PDF, HTML) besteht auch die Möglichkeit, Formatinformationen zur IE zu nutzen, etwa die Tatsache, dass die Stellenbezeichnung meist abgesetzt und durch visuelle Mittel wie Fettdruck hervorgehoben ist. Zudem kann man eine grammatikalisch korrekte Sprache und fehlerfreie Orthographie voraussetzen.

Informationsextraktion aus dieser Textform ist eine recht einfache Form der IE, da die zu extrahierenden Einheiten lediglich Entitäten und Attribute sind – Relationen und Ereignisse kommen im Prinzip nicht vor. Zudem wird ein bestimmtes lexikalisches Register verwendet, welches die Erkennung der Entitäten und Attribute erleichtert.

Die oben genannten Teilaufgaben der IE kommen daher in dem vorliegenden Fall nur bedingt zum Einsatz. Im Prinzip kommt nicht einmal die Eigennamenerkennung im klassischen Sinne in Frage. Allerdings können die hierfür verwendeten Techniken auch zur Extraktion von Entitäten wie Stellenbezeichnungen, Kompetenzanforderungen oder Formalia angewandt werden. Gegenstand der vorliegenden Arbeit ist eine Evaluation verschiedener Ansätze zur Lösung dieser IE-Aufgabe. Im Rahmen der Studie liegt dabei der Fokus auf der Extraktion des Anforderungsprofils, also der vom potentiellen Bewerber

geforderten Kompetenzen wie Bildungsabschlüsse, fachliche Kenntnisse, Berufserfahrung oder technische Fähigkeiten. Sowohl manuelle wie auch automatische Verfahren werden hierfür getestet und die Resultate miteinander verglichen.

Verwandte Arbeiten

Califf (1998) entwickelte mit RAPIER (Robust Automated Production of IE Rules) ein System, das in der Lage ist, einfache Regeln zum Füllen eines vorgegebenen Templates automatisch zu erlernen, indem die Wörter und deren Wortarten und semantische Klassen im Kontext eines Fillers betrachtet und verallgemeinert werden. Das System ist prinzipiell domänenunabhängig und wurde unter anderem auf Stellenanzeigen aus dem USENET angewandt. Auf dem gleichen Korpus wurden später von Freitag & McCallum (2000) Tests mit einem stochastischem Modell (Hidden Markov Modell, HMM) durchgeführt, welche nochmals eine deutliche Verbesserung des statistischen gegenüber dem regelbasierten Ansatz bei der Extraktion von Firmennamen demonstrierten.

Ein recht frühes, kommerzielles Projekt, welches IE aus Stellenanzeigen nutzte, war FlipDog[2]. Dabei handelte es sich um eine Jobsuchmaschine, welche statt ihr Korpus aus Stellenanzeigen von Arbeitgebern gegen Bezahlung zu beziehen einfach selbst automatisch alle Ausschreibungen von Unternehmenswebseiten extrahierte und dabei IE betrieb, um die Inhalte zu strukturieren und in eine Ontologie zu überführen. Darüber hinaus bot der Dienst monatliche Berichte über sich verändernde Muster und Trends auf dem Arbeitsmarkt an. Die Präzision der Extraktion war relativ hoch oder wurde ggf. mithilfe menschlicher Prüfer verbessert. FlipDog wurde später von Monster[3] übernommen. (vgl. McCallum, 2005: 52)

Bsiri & Geierhos (2007) entwickelten ein System, das sein Korpus ebenfalls aus dem Web bezieht und die (französischsprachigen) Stellenanzeigen in sogenannte Repräsentationsvektoren transformiert, mit dem Ziel der Steigerung der Effizienz von Jobsuchmaschinen. Mithilfe lokaler Grammatiken und elektronischer Lexika werden ca. 20 verschiedene Typen an Information aus den Anzeigen extrahiert und in eine Datenbank eingepflegt, wie etwa Stellenbezeichnung, Arbeitsort, gewünschte Ausbildung oder Gehaltsangaben.

Das SIRE-Projekt (Sémantique, Internet, Recrutement et Emploi, Loth et al. (2010)) verfolgt auf lange Sicht die Entwicklung eines vollständigen Systems zum Crawlen von Stellenanzeigen aus dem Internet, der Extraktion von detaillierter Information und dem

[2] http://www.flipdog.com (zuletzt besucht am 21.10.2014).
[3] http://www.monster.com (zuletzt besucht am 21.10.2014).

Aufbau einer Ontologie sowie der Zugänglichmachung der strukturierten Information über einen Suchindex. Die in der ersten Projektphase durchgeführte manuelle Inspektion des Korpus führte zu der Erkenntnis, dass musterbasierte Extraktion genüge, um die verschiedenen Typen gewünschter Information anhand ihrer Kontexte hinreichend auseinanderzuhalten.

Eine der neuesten Arbeiten in diesem Bereich stellt die These von Krönke (2013) dar. Auch hier ist die IE eingebettet in ein ganzheitliches System zum Sammeln und Verarbeiten von Stellenanzeigen mit dem Ziel, qualitativ hochwertige Jobvorschläge generieren zu können. Die Extraktion selbst erfolgt mithilfe eines stochastischen Modells (Conditional Random Field, CRF), welches in der Evaluation gute Resultate erzielt, die hauptsächlich auf den Formatabweichungen einzelner Tokens (Stellenanzeigen im HTML-Format) beruhen.

1.3. Gliederung der Arbeit

Die vorliegende Arbeit ist wie folgt aufgebaut. In Kapitel 2 werden die Vorverarbeitungsschritte erläutert, die zur Durchführung der Informationsextraktion nötig waren. Anschließend werden in den Kapiteln 3 und 4 die verschiedenen erprobten Verfahren beschrieben – Kapitel 3 erläutert die Eigenschaften und die Erprobung des Ansatzes mittels Knowledge Engineering, während Kapitel 4 die Verfahren des automatischen Lernens und die Anwendung eines Klassifikationsverfahrens erläutert. Kapitel 5 behandelt die Evaluation der Ergebnisse der einzelnen Verfahren, bevor in Kapitel 6 schließlich ein Fazit aus der Arbeit sowie ein Ausblick auf zukünftige Projekte gegeben wird.

2. Vorverarbeitung

Je nach Datenquelle liegen Stellenausschreibungen in einem spezifischen Format vor, z. B. als PDF-Dateien, HTML-Seiten oder im reinen Textformat. Im vorliegenden Fall stammen die Anzeigen aus einer Datenbank[4], sodass sie als Zeichenketten ohne weitere Formatinformationen (abgesehen von Textabsätzen) vorliegen. Es ist daher nicht nötig, eine Vorverarbeitung zu leisten, die Text von Format trennt – ebensowenig ist es allerdings auch möglich, Formatinformationen gezielt für die IE einzusetzen.

Um die Extraktion effektiv zu gestalten, ist es hilfreich, das Suchfenster für das System von vornherein einzugrenzen. Einige Systeme nutzen dafür *Triggering*, d. h. die Suche nach speziellen Wörtern (Triggern), welche ein Hinweis auf das Vorhandensein der gewünschten Information in der unmittelbaren Nähe sind (vgl. Jackson & Moulinier, 2002: 75). Eine damit verwandte alternative Möglichkeit ist die Anwendung von Klassifikationsverfahren auf die einzelnen Abschnitte[5] der Stellenanzeige, die durch teils komplexe Merkmalskombinationen automatisch die Klassenzugehörigkeit und damit das Thema des jeweiligen Abschnitts bestimmen. Die Durchführung der Klassifikation wurde in einem vorgeschalteten Teilprojekt (Hermes, 2014) vorgenommen, sodass für diese Arbeit bereits ein Korpus mit klassifizierten Paragraphen von Stellenanzeigen zur Verfügung stand. Da für jeden Paragraphen angegeben ist, welcher Klasse sein Inhalt thematisch zugeordnet ist, kann ein einfacher Filter angewandt werden, der etwa für die Extraktion von Kompetenzen ausschließlich Paragraphen der Klasse „Kompetenz“ in Betracht zieht.

Abhängig vom spezifischen IE-Ansatz sind noch weitere Präprozessierungen nötig. Für das Matchen regulärer Ausdrücke (vgl. Abschnitt 3.1) ist es ausreichend, den gesamten Paragraphentext zu übergeben. Möchte man allerdings die syntaktische Struktur nutzen (vgl. Abschnitt 3.2), muss der Paragraph zunächst in Sätze unterteilt werden, welche anschließend jeweils tokenisiert und morphologisch und syntaktisch analysiert werden. Dieser Vorgang ist schematisch in Abbildung 3 dargestellt.

[4] Die Daten für diese Studie wurden vom Bundesinstitut für Berufsbildung (BIBB) in anonymisierter Form bereitgestellt.

[5] Das Trennen der Abschnitte erfolgte mit Hilfe eines selbst entworfenen Splitters, welcher neben der Suche nach Leerzeilen auch die Tatsache berücksichtigt, dass einige Textabschnitte trotz räumlicher Trennung inhaltlich zusammengehörig sind, wie etwa Listenpunkte oder die ursprünglich visuell abgesetzte Stellenbezeichnung.

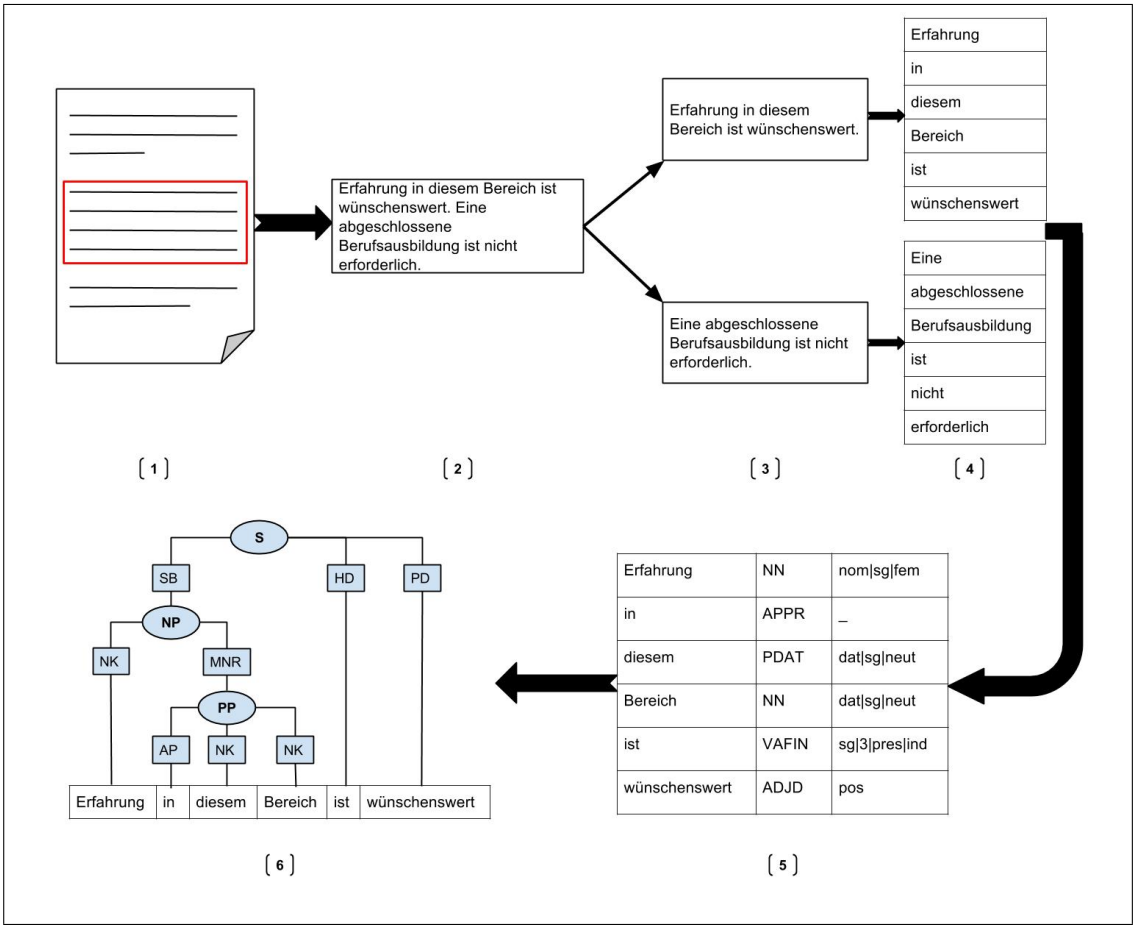

Abbildung 3: Schematische Darstellung des Workflows zur Vorverarbeitung eines Paragraphen. Aus der Stellenanzeige (1) wird der zu verarbeitende Paragraph zunächst extrahiert (2), anschließend in Sätze (3) und diese wiederum in Tokens (4) gesplittet. Diese können dann etwa mit POS-Tags und morphologischen Informationen ausgezeichnet (5) und ggf. syntaktisch analysiert (6) werden, hier mittels eines Dependenzparsers.

2.1. Manuelle Inspektion des Korpus

Nachdem das Korpus bereits auf Paragraphen des gewünschten Themas – vom Bewerber geforderte Kompetenzen – eingegrenzt wurde, können diese genauer hinsichtlich ihrer Struktur untersucht werden. Es stellt sich heraus, dass die Kompetenzen generell in zwei verschiedenen Formen ausgedrückt werden: entweder im Fließtext oder in einem Listenformat. Die Tabellen 1 und 2 geben Beispiele für beide Formate.

Im Fließtext (vgl. Tabelle 1) werden vielfältige Formulierungen gebraucht, um auszudrücken, welche Eigenschaften, Kenntnisse und Fähigkeiten der Adressat mitbringen sollte. Bestimmte Verben lassen sich dabei häufig als Signalwörter ausmachen, etwa *erwarten*, *voraussetzen*, *mitbringen* oder *verfügen*. Die Kompetenz tritt als Argument des Verbs auf, je nach Semantik des Verbs als Subjekt wie in (2) oder als Objekt, wobei in letztem Fall das Subjekt entweder „Sie“ (1) oder „Wir“ (3) ist.

Im Falle von Listen (vgl. Tabelle 2) stellt jeder Aufzählungspunkt häufig ein einzelnes Stichwort oder eine Phrase (1), etwas seltener auch eine komplexere syntaktische Konstruktion bis hin zu ganzen Sätzen (2), dar. Es kommt auch vor, dass eine solche Liste in Teillisten gegliedert wird, wobei jeweils nach der Stärke der geforderten Kompetenz unterteilt wird wie in Beispiel (3). Die Listen werden mit dem vorangegangenen Satz (etwa „Von Bewerbern wird erwartet, dass“) oder häufig auch nur mit einem Titel wie „Ihr Profil“, „Anforderungen“, „Kenntnisse“, „Anforderungsprofil“, „Voraussetzungen“ oder „Unsere Erwartungen“ eingeleitet. Die Modifizierer, die angeben, wie stark die jeweilige Kompetenz gewünscht wird („zwingend“, „wünschenswert“ etc.) können ebenfalls der Liste vorangestellt sein, sodass sie für alle nachfolgenden Punkte gelten, oder werden pro Listenpunkt separat aufgeführt.

(1)	*Idealerweise* verfügen Sie bereits über **Erfahrungen im Pflegebereich** und besitzen eine **adäquate Berufsausbildung** bzw. **fühlen sich den Aufgaben gewachsen**.
(2)	Hierzu werden **umfangreiche Kenntnisse der Elektrotechnik**, ein **sicherer Umgang mit dem PC** als auch **sehr gute Datenverarbeitungskenntnisse** *benötigt*. Eine **sichere Kommunikation in englischer Sprache** ist ebenfalls *Grundvoraussetzung*.
(3)	Wir setzen den **Besitz des Führerscheins der Klasse CE** *zwingend* voraus.

Tabelle 1: Beispielparagraphen, die geforderte Kompetenzen in Fließtextform beschreiben. Die Kompetenzen sind fett gedruckt, die Modifikatoren kursiv hervorgehoben.

(1)	Voraussetzung: - **abgeschlossene Ausbildung zum/zur Zimmerer/Zimmerin** - **zuverlässig**, **flexibel**, **motiviert** - **Führerschein und Fahrzeug** zum Erreichen des Arbeitsortes *erforderlich*
(2)	Sie sollten insbesondere : - über **handwerkliche Kenntnisse** verfügen - **belastbar**, **zuverlässig** und **flexibel einsetzbar** sein - **Berufserfahrung** wäre *wünschenswert* - **PKW-FS** ist *zwingend erforderlich*
(3)	*ZWINGEND*: - **Führerschein Klasse B** *ERWÜNSCHT*: - **Teamfähigkeit** - **Flexibilität** - **Verantwortungsbewusstsein**

Tabelle 2: Beispielparagraphen, die geforderte Kompetenzen in Listenform beschreiben. Die Kompetenzen sind fett gedruckt, die Modifikatoren kursiv hervorgehoben.

Insgesamt lässt sich feststellen, dass die zu extrahierenden Kompetenzen in einer mehr oder weniger regelhaften Form im Text auftreten, die sich durchaus mit Mustern – insbesondere im Hinblick auf den jeweiligen Kontext – beschreiben lassen sollte. Auch die Variabilität der Syntax scheint einigermaßen begrenzt zu sein. Im Rahmen dieser Studie wurde daher entschieden, die folgenden IE-Verfahren zu evaluieren: Den Einsatz regulärer Ausdrücke sowie eines Dependenzparsers zur Mustererkennung im Text und in der syntaktischen Struktur der Stellenanzeige, sowie ein einfacher Klassifikationsalgorithmus als maschinelles Lernverfahren.

2.2. Entwurf der Templates

Da die vorliegende Arbeit die Evaluation verschiedener Ansätze zur IE aus Stellenanzeigen behandelt, beschränken sich die erprobten Verfahren jeweils auf die Extraktion nur eines Themas, nämlich der Bewerberkompetenzen. Entsprechend war es nicht nötig, eine komplexe Datenstruktur für das Template zu entwerfen, da es zu diesem Zeitpunkt nur einen Slot „Kompetenz“ enthalten würde, dessen Füller eine Liste der extrahierten Daten wäre. Daher wurde entschieden, eine lose Kopplung zwischen Paragraphen und extrahierten Textsegmenten zu implementieren, indem zu jedem Segment die eindeutige

Template für Paragraph ID=787dcf1d-d222-41a7-9387-d8b696f8c99f	
Kompetenz	umfangreiche Kenntnisse der Elektrotechnik
	sicherer Umgang mit dem PC
	sehr gute Datenverarbeitungskenntnisse
	sichere Kommunikation in englischer Sprache

Abbildung 4: Beispiel für das simple Template, das konzeptuell dieser Arbeit zugrundeliegt.

ID des zugehörigen Paragraphen gespeichert wird (vgl. Abbildung 4). Im Hinblick auf Folgeprojekte wäre natürlich der Entwurf eines umfangreichen Templates mit getrennten Feldern für verschiedenartige Information notwendig. Auch wird derzeit der Einfachheit halber darauf verzichtet, die Attributwerte feiner nach der Stärke der geforderten Kompetenz zu unterteilen, sodass Modifizierer ggf. einfach Teil des textuellen Contents sind.

2.3. Manuelles Tagging des Trainingskorpus

Um die Resultate der erprobten Ansätze evaluieren zu können, ist es nötig, sämtliche Textsegmente im Vorhinein zu bestimmen, die aus dem Korpus als Ergebnisse erwartet werden. Es wurde entschieden, diese Auszeichnung zunächst recht grob vorzunehmen, um Raum für Verfeinerung im Nachfolgeprojekt zu lassen. Entsprechend sollte pro Paragraph diejenige Zeichensequenz (Wort oder Phrase) markiert werden, welche eine vom Bewerber geforderte Kompetenz darstellt.

Zur Durchführung der Auszeichnung wurde ein Programm entwickelt, welches eine Interaktion mit einem Benutzer ermöglicht, der die Auszeichnung vornimmt. Diesem wird hierfür auf der Konsole jeweils ein Paragraph präsentiert, wobei die einzelnen Tokens durchnummeriert sind. Anschließend erwartet es die Eingabe der Tokennummer(n) derjeniger Tokens, die als Kompetenz zu werten sind. Anhand der Tokennummern wird das betreffende Textsegment identifiziert und in einem Objekt zusammen mit der eindeutigen ID des Paragraphen abgespeichert. Zusätzlich wird die Information zur besseren Übersicht in ein lesbares Format (*Tab Separated Values*, TSV) überführt.

3. Musterbasierte Extraktion

Wie eingangs erwähnt gibt es grundsätzlich zwei verschiedene Methoden, ein IE-System zu entwerfen. Eine davon ist der *Knowledge Engineering* Ansatz. Es handelt sich dabei um ein manuelles Verfahren, das sich wie folgt beschreiben lässt:

> The Knowledge Engineering Approach is characterized by the development of the grammars used by a component of the IE system by a "knowledge engineer," i.e. a person who is familiar with the IE system, and the formalism for expressing rules for that system, who then, either on his own, or in consultation with an expert in the domain of application, writes rules for the IE system component that mark or extract the sought-after information.
>
> (Appelt & Israel, 1999: 7)

Es wird deutlich, dass die Fähigkeiten des Knowledge Engineerers einen großen Einfluss auf die Leistung des Systems haben. Dieser benötigt Zugang zu einem ausreichend großen Korpus domänenrelevanter Texte, um die gesuchten Muster durch dessen Inspektion (sowie zusätzlich durch Introspektion) exakt bestimmen zu können. Der Vorteil daran ist, dass sehr genaue Regeln entworfen werden können, die auch insbesondere die relevanten linguistischen Merkmale mit einschließen (vgl. Neumann, 2010: 21). Allerdings ist dieser Vorgang auch sehr arbeits- und kostenintensiv. Insbesondere ist er nicht praktikabel, wenn die benötigten Ressourcen, vor allem die Expertise eines Knowledge Engineers, nicht zur Verfügung stehen (vgl. Eikvil, 1999: 8).

Es waren hauptsächlich die frühen IE-Systeme, die den Knowledge Engineering Ansatz mit der Erstellung manueller Regeln für die Extraktion verfolgten. Bis Ende der 1990er herrschte der Tenor vor, „the best performing systems are often hand crafted" (ebd.: 8), und auch viele der in den MUCs partizipierenden Systeme waren von dieser Art. Bei den manuell erstellten Regeln handelt es sich meist um reguläre Ausdrücke, d. h. das IE-System arbeitet mit Endlichen Automaten (en. *Finite State Machines*, FSMs). Ein recht bekanntes Beispiel für ein solches System ist FASTUS (permutiertes Akronym für *Finite State Automaton Text Understanding System*) von Appelt, Hobbs et al. (1993), welches aus den MUCs hervorging. Aber es existieren auch deutlich neuere Systeme, deren IE-Verfahren auf regulären Ausdrücken basieren, wie etwa das von Rosier et al. (2008).

In Abschnitt 2.1 wurde dargelegt, dass die aus einer Stellenanzeige zu extrahierenden Anforderungen in Kontexten auftreten, die sich vermutlich in Form von Mustern beschreiben lassen. Im Folgenden wird dargelegt, wie diese Muster zum einen mittels regulärer

Ausdrücke, zum anderen mithilfe eines Dependenzparsers zur Extraktion genutzt werden können.

3.1. Extraktion mithilfe regulärer Ausdrücke

Da es nicht das Ziel der IE ist, künstlich intelligente Systeme zu schaffen, die eine tatsächliche Verstehensleistung zu einem Text erbringen, ist es oft ausreichend, *Pattern Matching* mithilfe regulärer Ausdrücke zu betreiben, um die gewünschte Information aufzufinden.

Reguläre Ausdrücke und Pattern Matching

Reguläre Ausdrücke (en. *regular expressions*, kurz RegEx) sind nicht nur ein zentraler Gegenstand der theoretischen Informatik (vgl. etwa Hopcroft et al., 2011), sondern auch beliebtes Werkzeug zur Textverarbeitung, und finden daher häufig Anwendung in der maschinellen Sprachverarbeitung.

Formal handelt es sich bei einem RegEx um eine kompakte algebraische Notation um eine Menge von Strings[6] zu charakterisieren. „Thus, they can specify search strings as well as define a language in a formal way“ (Jurafsky & Martin, 2009: 52). Daher eignen sich RegExes besonders gut für die Spezifizierung von Textmustern. Die einfachsten RegExes sind einfache Sequenzen von Literalen, man definiert etwa einen RegEx `Bewerber`, um alle Vorkommen der Zeichensequenz[7] 'Bewerber' in einem Text zu suchen. RegExes können allerdings auch deutlich mächtiger sein und erlauben die Definition einer potentiell unendlich großen Menge an Zeichenketten durch die Konkatenation einfacherer Ausdrücke und mithilfe der Angabe von Alternativen, Wiederholungen und Variablen. Zudem bietet die Unix-Notation für reguläre Ausdrücke, die auch von den meisten Programmiersprachen verwendet wird, diverse Kürzel an, welche die Arbeit mit bestimmten Zeichenklassen vereinfachen.

Das Pattern Matching Problem lässt sich nun folgendermaßen definieren:

[6] Als Strings bezeichnet man im Allgemeinen eine Sequenz von Symbolen. Im Kontext der Sprachverarbeitung bezieht man sich damit auf Sequenzen von Zeichen eines bestimmten Zeichensatzes, was neben Buchstaben und Ziffern auch Sonder- und Steuerzeichen einschließt.

[7] Man darf dabei nicht von Wörtern sprechen – sofern im RegEx nicht explizit angegeben, wird auch innerhalb von Wörtern gematcht, solange die Zeichensequenz übereinstimmt, so etwa auch das Muster `opfer` in „Blumentopferde“.

> Given a String $\mathcal{P}$ called the *pattern* and a longer string $\mathcal{T}$ called the *text*, the **exact matching** problem is to find all occurences, if any, of pattern $\mathcal{P}$ in text $\mathcal{T}$
>
> (Gusfield, 1997: 2)

Das Pattern wird nun in Form eines RegEx $\mathcal{E}$ definiert, welcher die Sprache $\mathcal{L}(\mathcal{E})$ beschreibt. Es gilt (nach Navarro, 2004: 928), alle Textpositionen zu finden, an denen ein Vorkommen von $\mathcal{E}$ beginnt, d. h. die Menge:

$$\{i, \exists j, \mathscr{T}_{i...j} \in \mathscr{L}(\mathscr{E})\}$$

Hierfür wird der RegEx in einen Endlichen Automaten[8] kompiliert, der in der Lage ist, in von Text- und RegEx-Länge linear abhängiger Zeit zu entscheiden, ob ein gegebener Input-Text das definierte Muster enthält (vgl. Gusfield, 1997: 66).

Pattern Matching in Java

Die Programmiersprache Java bietet mehrere Möglichkeiten, mit RegExes zu arbeiten. In der simpelsten Form wird der RegEx an Methoden eines `String`-Objektes übergeben, welches die zu untersuchende Zeichensequenz enthält. Mithilfe der Methode `String.matches(String regex)` lässt sich dann beispielsweise prüfen, ob die Zeichensequenz auf einen regulären Ausdruck passt. `String.split(String regex)` erlaubt das Auftrennen der Zeichenkette an bestimmten, durch den RegEx definierten Mustern. Diese Methoden genügen für simple Anwendungen, ermöglichen allerdings z. B. keine Suche und Extraktion von Mustern innerhalb des Strings.

Für umfangreichere Matching-Aufgaben bieten sich die Klassen `Pattern` und `Matcher` aus dem Paket `java.util.regex` an. Ein `Pattern`-Objekt verwaltet einen kompilierten regulären Ausdruck und bietet damit u.a. die Möglichkeit, Fehler in der Definition des RegEx bereits vor der Laufzeit zu erkennen. Die Klasse `Pattern` stellt ebenfalls eine Methode `matches(String regex, CharSequence input)` zur Verfügung, mit der unmittelbar geprüft werden kann, ob ein Input auf einen gegebenen RegEx passt. Allerdings ist der Weg über das Kompilieren des RegExes effizienter, wenn ein und dasselbe Pattern mehrfach verwendet wird. In diesem Fall liefert die Methode `Pattern.matcher(CharSequence input)` das `Matcher`-Objekt, welches zudem weitere Methoden anbietet um etwa mehrere Matches

[8] Ein endlicher Automat ist eine abstrakte Maschine, die eine Symbolkette als Input erhält und zeichenweise einliest. Nach jedem Zeichen wechselt er gemäß seiner Zustandsüberführungsfunktion seinen Zustand. Stoppt der Automat den Prozess am Ende der Zeichenkette in einem definierten Endzustand, akzeptiert er die Eingabe.

innerhalb des Inputs zu finden, deren Positionen zu ermitteln oder sogar gematchte Teile im Inputstring zu ersetzen.

Das folgende Snippet demonstriert die Benutzung der `Pattern`- und `Matcher`-Objekte in Java:

```
String input = "Bitte senden Sie Ihre Bewerbung bis zum 31.03.2014 ein.";
/* regulärer Ausdruck für Datumsangaben: */
Pattern p = Pattern.compile("(0[1-9]|[12][0-9]|3[01])([- /.])(0?[1-9]|1[012])
    \\2(19|20)\\d\\d");
Matcher m = p.matcher(input);
while(m.find()){
       System.out.println(m.group()); // der gematchte Ausdruck = 31.03.2014
       System.out.println(m.start()); // Startposition im Input = 40
       System.out.println(m.group(1)); // erste Gruppe des Matches = 31
}
```

Listing 1: Beispiel für Pattern-Matching in Java

Extraktion von Zeichensequenzen aus Stellenanzeigen

Reguläre Ausdrücke sind ein sehr mächtiges Werkzeug. Ihre volle Kraft entfalten sie naturgemäß bei der Anwendung auf reguläre Sprachen, etwa zur Validierung von Passwörtern, E-Mail-Adressen oder Datumsangaben wie im Beispiel-Listing 1.

Natürliche Sprachen werden in der Computerlinguistik häufig als schwach-kontextsensitiv eingestuft, sind jedenfalls mindestens kontextfrei und damit deutlich mächtiger und variabler als reguläre Sprachen (vgl. Klabunde, 2010). Lässt man allerdings eingebettete Konstruktionen und Rekursion außer Acht, lassen sich auch natürlichsprachliche Texte mithilfe endlicher Automaten verarbeiten, wie Church (1980) bereits zeigte. Entsprechend ist es auch möglich, Muster in Texten mithilfe regulärer Ausdrücke zu finden. Der Balanceakt besteht allerdings darin, den RegEx weder zu spezifisch, noch zu mächtig zu machen. Der Entwurf von Mustern für die Informationsextraktion ist deshalb ein iterativer Prozess:

> [A] set of rules is written, the system is run over a training corpus of texts, and the output is examined to see where the rules under- and overgenerate. The knowledge engineer then makes appropriate modifications to the rules, and iterates the process.
>
> (Appelt & Israel, 1999: 7)

Für die vorliegende Aufgabe wurde zunächst durch Inspektion des Korpus und Überlegungen zur Vorhersage möglicher, im Trainingskorpus nicht vorliegender Muster ein anfängliches Set regulärer Ausdrücke entworfen. Wie in Abschnitt 2.1 dargelegt lässt sich ein gewisser Kontext definieren, in welchem die gesuchten Inhalte auftreten. Reguläre Ausdrücke bieten die Konstruktionen *look-behind* und *look-ahead* an, mit deren Hilfe das Prüfen des Kontextes bewerkstelligt werden kann. Diese Konstrukte als Teil eines RegEx dienen nicht dazu, Zeichenketten zu matchen, sondern Positionen innerhalb der Zeichenkette zu bestimmen. Ein *look-behind* matcht demnach die Position nach dem spezifizierten Ausdruck, ein *look-ahead* entsprechend davor. Kombiniert mit dem Ausdruck, der tatsächlich matchen soll, ist es möglich die Bedingung vorzugeben, dass nur nach bzw. vor bestimmten Mustern gematcht wird. Der RegEx

```
(?<=wir setzen) [^.\n]+? (?=voraus\.)
```

beispielweise enthält das look-behind Konstrukt `(?<=wir setzen)` sowie das look-ahead Konstrukt `(?=voraus\.)`. Das matchende Muster `[^.\n]+?` ist dagegen nicht näher spezifiziert und enthält alle Zeichen außer Punkt oder Zeilenumbruch, sodass beliebig viele alphanumerische Zeichen bis zum (Ab-)Satzende konsumiert werden. Somit können mit diesem RegEx alle Teilstrings erfasst werden, die innerhalb eines Satzes zwischen „wir setzen“ und „voraus“ auftreten.

Ein weiteres recht reguläres Muster sind die Listen. Hier bietet es sich an, jeden Listenpunkt zu matchen und den Inhalt zu extrahieren. Der hierfür verwendete RegEx lautet:

```
(?<=^\[\*\] )(?=(\P{M}\p{M}*)+$).+
```

Die look-behind Struktur erfasst das Symbol, das den Listenpunkt einleitet. Zur Vereinfachung wurden in einem Vorverarbeitungsschritt bereits alle derartige Symbole wie `-,*,-*,·,¿` zu einem eindeutigen Symbol `[*]` vereinheitlicht. Der RegEx fordert, dass dieses Symbol unmittelbar am Zeilenanfang auftritt. Der matchende Ausdruck selbst erfasst eine beliebig lange Kette von Graphemen bis zum Zeilenende, das sind entweder einzelne oder mit einem kombinierenden Zeichen (etwa für Diakritika) gepaarte Unicode-Symbole. Hierfür muss der Ausdruck `\P{M}\p{M}*` in Java verwendet werden, da das Kürzel `\X` – anders als beispielsweise in Perl oder PHP – nicht zur Verfügung steht.

In dieser Form wurde nun eine Menge zum Korpus passender RegExes entworfen. Durch wiederholtes Matchen gegen das Trainingskorpus und Untersuchen der Ergebnisse wurde anschließend beurteilt, welche Muster über- bzw. untergenerieren. Schließlich wurden die

entsprechenden RegExes modifiziert und der Vorgang wiederholt, bis ein akzeptables Set an Ausdrücken gefunden wurde. Listing 2 listet alle entworfenen RegExes auf. Diese kombinieren Lookbehind- und Lookahead-Konstrukte mit Literalen, Alternativen, Wiederholungen und insbesondere Wildcards für das Matchen beliebiger Zeichensequenzen zwischen den spezifizierten Kontexten. Das Kürzel `(?i)` vor einem Ausdruck ermöglicht das Matchen unabhängig von Groß- und Kleinschreibung, `\b` markiert Wortgrenzen und `\n` Zeilenumbrüche.

```
(?i)(?<=((Sie␣verfügen|verfügen␣Sie)␣über)|(Sie␣(sind|haben)))[^\.\n]+

(Kenntnis(se)?|Erfahrung(en)?)␣(von|in|im|der|des)␣.+?(?=\.|\n)

(?i)(?=abgeschlossene␣(berufs)?ausbildung)[^.\n]+

(?<=^\[\*\]␣)(?=(\P{M}\p{M}*)+$).+

(?i)(?<=((wir␣erwarten␣)|(wünschen␣(?:wir␣)?uns␣))\b)(.+?)(?=\.)

(?i)(?<=(wünschenswert|erforderlich|vorausgesetzt|voraussetzung|gewünscht|
    erwartet)␣(ist|sind|wird|wäre(n)?+)\b)([^.\n]+)

(?i)(?<=(wir␣setzen|setzen␣wir)\b)([^.\n]+?)(?=voraus\.)

\b\p{javaUpperCase}\p{javaLowerCase}+(heit|keit)\b

([^.\n]+?)(?=\b(ist|sind|wird|werden|wäre(n)?)␣(wünschenswert|erforderlich|
    vorausgesetzt|gewünscht|erwartet))

(?i)(?=(berufs)?ausbildung)[^.\n]+(?=abgeschlossen\.)

(Führerschein|FS)␣(Klasse␣|Kl.)?+.+?(?=\.|\n)

(?i)(?<=\b((der(/die)?)|die)␣Bewerber(/?in)?␣sollten?)␣([^.\n]+?)(?=\bsein|
    mitbringen|haben)

(?<=\bist|sind|wird|werden|wäre(n)?+)([^.\n]+?)(wünschenswert|erforderlich|
    vorausgesetzt|gewünscht|erwartet)
```

Listing 2: Verwendete Reguläre Ausdrücke zum Matchen von Kompetenzen.

Zur Extraktion der gewünschten Zeichenketten aus den gegebenen Paragraphen der Stellenanzeigen wurden alle RegExes in eine Menge von `Pattern`-Objekten kompiliert. Der Matcher jedes Patterns wird nun auf einen einzelnen Paragraphen angesetzt und sucht nach einem Match. Die dabei gefundenen Teilstrings werden jeweils zusammen mit der Information, aus welchem Paragraphen sie stammen, gespeichert. Der Quellcode-Ausschnitt in Listing 3 veranschaulicht das Vorgehen.

```
for (Pattern pattern : regExes) {
        matcher = pattern.matcher(paragraphText);
        while (matcher.find()) {
                token = matcher.group();
                results.add(new SlotFiller(token.trim(), paragraph.getID()));
        }
}
```

Listing 3: Umsetzung der Musterextraktion aus einem Paragraphen.

3.2. Extraktion mithilfe eines Dependenz-Parsers

In Abschnitt 2.1 wurde festgestellt, dass im Fließtext von Stellenanzeigen oft bestimmte Verben als Signalwörter auftreten, die die geforderte Kompetenz zum Argument haben. Daher scheint es angemessen, neben dem Textmatching auch die Extraktion über die Argumentstruktur des Satzes zu erproben. Als Analysewerkzeug für die syntaktische Struktur bietet sich ein Dependenzparser an.

Dependenzgrammatik und Parsing

In der Dependenzgrammatik nach Tesnière (1980) besteht zwischen den Wörtern eines Satzes eine Beziehung, die als Konnexion bezeichnet wird. Konnexionen definieren *Dependenzen* (Abhängigkeiten) zwischen Satzteilen, verbinden also übergeordnete und untergeordnete Wörter. Das jeweils übergeordnete Element wird Regens (*Head*), das untergeordnete Dependens (*Dependent*) genannt. Dadurch, dass ein Regens selbst wiederum Dependens eines anderen Wortes sein kann, bildet die Gesamtheit der Wörter eines Satzes eine Hierarchie, deren zentrales Element das (finite) Verb des Satzes ist.

Parsing ist der Vorgang der syntaktischen Analyse eines Satzes, der Erstellung einer Strukturbeschreibung nach einer gegebenen Grammatik. Dependenzparser erzeugen demnach Strukturbeschreibungen für Sätze anhand von Dependenzgrammatiken. Im resultierenden Baumgraphen werden die Dependenzrelationen durch die Kanten wiedergegeben (vgl. Langer, 2010: 282). Für dieses Projekt hat das den Vorteil, dass das finite Verb als Kopf des Satzes markiert wird und Subjekt sowie Objekte direkt von diesem abhängig sind. Im Gegensatz zur Phrasenstruktur gibt es in der Dependenzstruktur keine nicht-terminalen Knoten, denn jedes Wort steht in Bijektion zu den Knoten des Dependenzbaums. Abbildung 5 stellt exemplarisch die vereinfachte Dependenzstruktur des Satzes „Eine abgeschlossene Berufsausbildung ist erforderlich.“ dar.

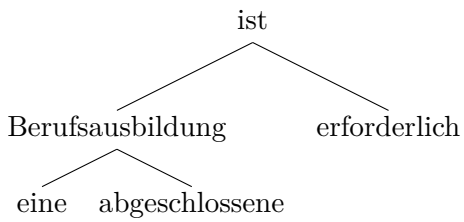

Abbildung 5: Ein einfacher Dependenzbaum.

Dependenzparsing zur Informationsextraktion

Da eine zentrale Aufgabe der IE die Extraktion von Relationen ist, bietet sich der Einsatz syntaktischer Analysewerkzeuge an. Allerdings ist der maschinelle Parsingvorgang bislang sehr zeit- und ressourcenintensiv, sodass erst in jüngster Zeit auf dieses Mittel zurückgegriffen wird (vgl. Neumann, 2010: 605).

Sekine (2006) entwickelte mit ODIE ein System, das er als „On-Demand Information Extraction“ bezeichnet. Dependenzbasiertes Parsing wird hier als zentrales Werkzeug eingesetzt, um die syntaktische Struktur relevanter Relationen zu bestimmen (vgl. auch Neumann, 2010: 604).

In dieser Arbeit wurde ebenfalls erprobt, inwieweit dependenzbasiertes Parsing geeignet ist, um die Anforderungen einer Stellenanzeige über die syntaktische Struktur zu extrahieren.

Extraktion verbaler Argumente

Um Anforderungen als verbale Argumente extrahieren zu können, müssen zunächst die einzelnen Sätze eines jeden Paragraphen geparst werden. Für das Deutsche stehen einige wenige gebrauchsfertige Dependenzparser zur Verfügung. Einer davon findet sich im *Mate Toolkit*[9]. Dabei handelt es sich um eine Sammlung verschiedener NLP-Werkzeuge, die einen Lemmatisierer, POS-Tagger, morphologischen Tagger und Dependenzparser enthält. Die einzelnen Komponenten können in einer Pipeline zusammengeschaltet werden (vgl. Listing 4).

Das vorgegebene Input-Format für die Mate-Pipeline fordert, dass der Text bereits in Sätze gesplittet und tokenisiert wurde. Da man für ideale Ergebnisse den gleichen Tokenizer benutzen sollte, mit dem das Modell trainiert wurde, wurde in dieser Arbeit die *Apache OpenNLP* Bibliothek[10] verwendet. Diese bietet sowohl einen *Sentence Splitter* als auch einen *Tokenizer*, sodass beide Vorverarbeitungsschritte mit dem gleichen Werkzeug erledigt werden können[11]. Als Input dienen nun die bereits gefilterten Paragraphen, die zunächst in Sätze unterteilt und diese wiederum tokenisiert werden. Die Tokenliste kann

[9] https://code.google.com/p/mate-tools/ (zuletzt besucht am 23.10.2014).

[10] Die Bibliothek steht zum Download unter http://opennlp.apache.org/ zur Verfügung. Modelle für Tokenizer und Sentence Splitter finden sich unter http://opennlp.sourceforge.net/models-1.5/ (beide zuletzt besucht am 23.10.2014).

[11] Apache OpenNLP ist eine Werkzeugsammlung für NLP, die ebenfalls auch eine Komponente zum Parsing enthält. Allerdings basiert der Parser auf Konstituenten- statt Dependenzstruktur.

dann in ein `SentenceData09`-Objekt überführt werden, welches von den Komponenten der Pipeline anschließend um morphologische und syntaktische Information angereichert wird.

```
String[] sentences = sentenceDetector.sentDetect(paragraphText);

for (String sentence : sentences) {
        sentenceData = new SentenceData09();

        ArrayList<String> forms = new ArrayList<String>();
        String[] tokens = tokenizer.tokenize(sentence);

        forms.add(CONLLReader09.ROOT); //add imaginary root node

        for (String token : tokens)
                forms.add(token);

        sentenceData.init(forms.toArray(new String[0]));

        /* apply mate pipeline: */
        sentenceData = depParser.apply(posTagger.apply(morphTagger.apply(
            lemmatizer.apply(sentenceData))));
}
```

Listing 4: Anwendung der Mate-Pipeline

Das Ergebnis des Parsing-Vorgangs ist eine tabellenartige Struktur (intern als korrespondierende Arrays repräsentiert), in der zu jedem Token dessen ID, Lemma, POS-Tag, morphologische Merkmale und Dependenzrelationen notiert sind (vgl. Tabelle 3).[12] Die Dependenzstruktur ist dabei so codiert, dass zu jedem Token die ID desjenigen Tokens angegeben ist, das dessen Kopf darstellt, sowie die Art der Beziehung. Das Verb als Kopf des Satzes verweist mit der ID 0 auf den imaginären Wurzelknoten des Satzes mit der Dependenzrelation `"--"`. Die gesamte Struktur lässt sich konzeptionell auf eine Graphenstruktur abbilden, wie sie in Abbildung 6 dargestellt ist[13].

[12] Die von Mate zur Auszeichnung verwendeten Labels entsprechen den Richtlinien des TIGER Annotationsschemas, vgl. http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/TIGERCorpus/annotation (zuletzt besucht am 01.11.2014).

[13] Unter http://de.sempar.ims.uni-stuttgart.de/ (zuletzt besucht am 27.10.2014) steht eine Online-Demonstration bereit, die das interaktive Ausprobieren der Mate Tools ermöglicht. Dort wird auch eine grafische Ausgabe generiert.

ID	Form	Lemma	POS	Head	DepRel
1	Eine	ein	ART	3	NK
2	abgeschlossene	abgeschlossen	ADJA	3	NK
3	Berufsausbildung	berufsausbildung	NN	4	OA
4	setzen	setzen	VVFIN	0	--
5	wir	wir	PPER	4	SB
6	voraus	voraus	PTKVZ	4	SVP
7	.	--	$.	6	--

Tabelle 3: Ergebnis der Analyse des Beispielsatzes „Eine abgeschlossene Berufsausbildung setzen wir voraus“, wie es von den Mate Tools präsentiert wird.

Abbildung 6: Grafische Darstellung der Dependenzstruktur des Beispielsatzes.

Im nächsten Schritt müssen zunächst die betreffenden Verben spezifiziert werden, die als Schlüsselworte dienen. Während der manuellen Inspektion des Korpus zeigte sich, dass das lexikalische Register der Stellenanzeige nur wenige spezifische Verben zum Ausdruck des Anforderungsprofils enthält. Diese sind beispielsweise semantisch sehr enge Verben wie „voraussetzen“ oder „erwarten“, aber auch das Kopulaverb „sein“, was häufig in Konstruktionen wie „... ist Voraussetzung“ auftritt. Zu jedem Verb wird zusätzlich definiert, ob es weitere bestimmte Dependentien erfordert. Im dargestellten Beispiel etwa ist erkennbar, dass im Falle des trennbaren Verbs „voraussetzen“ der finite Verbanteil den Kopf des Satzes bildet, während das Präfix abgetrennt am Satzende auftritt und vom Dependenzparser als *Separable Verb Prefix* (SVP) als Dependens des Verbs ausgezeichnet wurde. Entsprechend wird in der Verbliste das Verb „setzen“ mit der Zusatzbedingung, dass es ein SVP-Dependens mit dem POS-Tag *PTKVZ* (abgetrennter Verbzusatz) erfordert, aufgenommen. Im Falle der Modalverben „müssen“ oder „sollen“ sowie dem Kopulaverb „werden“ ist ebenfalls eine Zusatzbedingung nötig, da diese für sich allein keine guten Indikatoren für Anforderungsbeschreibungen sind, es sei denn, sie treten in Konstruktionen wie „wird vorausgesetzt“, „müssen nachgewiesen werden“ oder „sollte vorhanden sein“ auf. Es wird daher definiert, dass diese Verben ein zweites Verb als Dependens erfordern. Tabelle 4 listet alle Verben zusammen mit ihren Bedingungen auf, die zur Extraktion von Kompetenzen herangezogen wurden.

Verb	Bedingung	Beispiel
abschließen		„Sie haben ... abgeschlossen“
beherrschen		„Sie beherrschen ...„
benötigen		„Hierfür benötigen Sie ...“
besitzen		„Sie besitzen ...“
bringen	PTKVZ	„Sie bringen ... mit“
erwarten		„Wir erwarten ...“
gehören	AP	„... gehören zu Ihren Stärken“
haben		„Sie haben ...“
hoffen	AP	„Wir hoffen auf ...“
mitbringen		„Wenn Sie ... mitbringen“
müssen	V	„Hierfür müssen Sie ... haben“
rund	PTKVZ	„... runden Ihr Profil ab“
sammeln		„Sie haben ... gesammelt“
sein		„Sie sind ...“
setzen	PTKVZ	„Wir setzen .. voraus“
sollen	V	„Sie sollten ... mitbringen“
suchen		„Wir suchen ...“
verfügen	AP	„Sie verfügen über ...“
voraussetzen		„Da wir ... voraussetzen“
werden	V	„... wird erwartet“
wünschen		„Wir wünschen uns ...“

Tabelle 4: Liste der Verben, die vom Dependenzparser als Köpfe zu extrahierender Phrasen in Betracht gezogen werden. Definiert wird das Verb selbst sowie ggf. eine Zusatzbedingung wie das Vorhandensein von abgetrennten Verbzusätzen oder eines zweiten Verbs im Satz.

Anhand dieser Informationen kann nun die Extraktion vorgenommen werden. Zunächst wird bestimmt, ob der Satz eines der vordefinierten Verben enthält, indem alle Lemmata durchlaufen und jeweils mit der Verbliste abgeglichen werden. Bei einem erfolgreichen Abgleich wird der Index des Lemmas zwischengespeichert, da die Dependenzstruktur anhand der IDs codiert ist, die jeweils `arrayIndex+1` entsprechen (vgl. Spalten „ID“ und „Head“ in Tabelle 3). Nun wird überprüft, ob eine ggf. angegebene Zusatzbedingung für das Verb erfüllt ist. Hierfür müssen die Dependentien des Verbs durchlaufen und je nach Bedingung deren POS Tag oder Dependenzrelation auf Übereinstimmung geprüft werden. Nur sofern die Bedingung erfüllt ist, wird der Satz zur Extraktion weiterverarbeitet.

Um schließlich die Extraktion vorzunehmen, werden die direkten Dependentien des Verbs genauer untersucht, d.h. das Subjekt sowie ggf. vorhandene Objekte. Sofern es sich bei dem Subjekt um eine Nominalphrase (NP) handelt, also nicht um eines der Personalpronomen

„wir“ oder „Sie“, wird angenommen, dass diese NP die zu extrahierende Kompetenz darstellt, wie das beispielsweise in Beispiel (2) in Tabelle 1 der Fall ist[14]. Im anderen Fall werden alle Objekte als je eine Phrase extrahiert.

Die Extraktion einer ganzen Phrase erfolgt mittels eines rekursiven Durchlaufs durch die Dependentien des Phrasenkopfes. Listing 5 veranschaulicht das Vorgehen im Pseudocode. Übergeben wird jeweils die ID des Phrasenkopfes.

```
getPhrase(headID, sentence, tokenIDs):
    head = sentence.lemmas[headID-1]
    heads = sentence.heads

    for i:=0 to heads.length do
        if heads[i] == headID do
            getPhrase(i+1, sentence, tokenIDs)
        end if
    end

    tokenIDs.add(headID)

    foreach i in tokenIDs do
        phrase += sentence.tokens[i-1] + " "
    end

    return phrase
end
```

Listing 5: Pseudocode zur Extraktion der gewünschten Phrasen aus der Dependenzstruktur durch Rekursion über Phrasenköpfe.

Schließlich werde analog zum Vorgehen mittels Pattern Matching die extrahierten Phrasen gespeichert.

[14] Diese Verallgemeinerung kann in diesem Fall relativ sicher getroffen werden, da die definierten Verben semantisch sehr eng sind.

4. Maschinelle Lernverfahren

Dem manuellen Entwurf des Regelapparats für das IE-System steht der Ansatz des automatischen Trainings gegenüber, bei dem das System anhand eines Trainingskorpus selbstständig die passenden Merkmale zur Auszeichnung der gesuchten Information lernt und mithin seinen Regelapparat selbst erzeugt. Entsprechend ist dieses Verfahren dem Paradigma des Maschinellen Lernens (ML) zuzuordnen, das auch in vielen anderen Anwendungsgebieten des NLP populär ist.

Bereits in den 1990ern partizipierten neben den manuellen Systemen auch solche in den Message Understanding Conferences, die mit maschinellem Lernen arbeiteten. Ihre Performanz war allerdings zunächst den manuellen Systemen nicht überlegen, was sich erst innerhalb der letzten Dekade geändert hat. Inzwischen sind automatische Systeme State of the Art (vgl. Mooney & Bunescu, 2005: 4). Die Vorgehensweise dieses IE-Verfahrens lässt sich wie folgt beschreiben:

> [S]omeone with sufficient knowledge of the domain and the task at hand annotates a set of training documents. Once a training corpus has been annotated, a training algorithm is run, training the system for analysing novel texts.
>
> (Eikvil, 1999: 8)

Im Gegensatz zum manuellen Verfahren wird entsprechend für den ML-Ansatz kein Knowledge Engineerer benötigt. Stattdessen wird das System in die Lage versetzt, aus einem vorausgezeichneten Korpus selbstständig Muster und Regularitäten zu erkennen und das gewonnene Wissen zur Verarbeitung unbekannter Texte zu verwenden.

4.1. Extraktion als Klassifikation

Grundsätzlich gibt es zwei verschiedene Typen von ML-Aufgaben: „With *classification* the learned model has to choose from a pre-defined set of classes, while *regression* forecasts a real value within a certain interval for a given test instance“ (Farkas, 2009: 8)[15]. Bei der Klassifikation gilt es demnach zu einem gegebenen Objekt zu entscheiden, welcher der vordefinierten Klassen es zuzuordnen ist. Die Abbildung der Objektmenge $\mathbb{X}$ auf die Menge der Klassen $\mathbb{C}$ erfolgt mittels der Klassifizierungsfunktion γ, die sich (nach

[15] Ein dritter Typ ist das *Clustering*, welches sich von der Klassifikation insofern unterscheidet, als dass die Objekte nicht in vordefinierte Klassen, sondern anhand ihrer intrinsischen Eigenschaften gruppiert werden.

Manning et al., 2008: 237) formal wie folgt definieren lässt:

$$\gamma\colon \mathbb{X} \rightarrow \mathbb{C} \tag{4.1}$$

Die Parameter, anhand deren der Klassifizierer die Zuordnung vornimmt, werden in überwachten (*supervised*) Lernverfahren aus einer Menge von Trainingsdokumenten, die bereits die korrekten Klassenzuweisungen für Objekte enthalten, bestimmt. Informationsextraktion kann demnach als Klassifikationsaufgabe betrachtet werden: Die zu klassifizierenden Objekte sind in diesem Fall die Tokens eines Textes. Die Aufgabe lässt sich hierbei auch als Sequence Labeling Task betrachten – der Text wird als Tokensequenz aufgefasst und für jedes Token ist zu bestimmen, ob es Teil der gesuchten Information ist (vgl. Jurafsky & Martin, 2009: 763f.).

Die Gestalt der Klassifizierungsfunktion ist vom angewandten Verfahren abhängig. Einige Systeme arbeiten mit numerischen Klassifikatoren wie *Support Vector Machines* (SVMs), andere mit probabilistischen Modellen wie dem *Hidden Markov Model* (HMM) (Freitag & McCallum, 2000) oder den mit den HMMs verwandten *Conditional Random Fields* (CRFs) (vgl. Ireson et al., 2005). Aufgrund des vorgegebenen Zeitrahmens für diese explorative Studie und der begrenzten Menge an zur Verfügung stehenden Trainingsdaten wurde ein recht simpler Klassifikator auf das vorliegende Problem angewandt. Dabei handelt es sich um den *Naive Bayes* Algorithmus.

4.2. Extraktion mithilfe eines Naive Bayes Klassifikators

Ein Vertreter der überwachten Klassifikationsverfahren ist *Naive Bayes* (vgl. Manning et al., 2008: 238ff.). Dabei handelt es sich um eine simple („naive“) Klassifikationsmethode, die auf der Bayes'schen Regel beruht. Nach dieser Regel lässt sich die Wahrscheinlichkeit (*Probability*), dass $c_i \in \mathbb{C}$ die korrekte Klasse für ein Objekt t[16] ist, folgendermaßen berechnen:

$$P(c_i|t) = \frac{P(t|c_i) \times P(c)}{P(t)} \tag{4.2}$$

Ziel der Klassifikation ist es, zu einem gegebenen Objekt t diejenige Klasse $c_i \in \mathbb{C}$ zu bestimmen, welche die höchste Wahrscheinlichkeit hat. Man nennt sie „maximum a

[16] Da es im folgenden um die Klassifikation einzelner Tokens gehen wird, werden die Objekte an der Stelle bereits mit t bezeichnet.

posteriori class“ c_{map}:

$$c_{map} = \arg\max_{c_i \in \mathbb{C}} P(c|t) = \frac{P(t|c_i) \times P(c)}{P(t)} \tag{4.3}$$

Da der Nenner für alle $c_i \in \mathbb{C}$ stets den gleichen Wert hat und damit eine Konstante ist, kann er vernachlässigt werden. Übrig bleiben die bedingte Wahrscheinlichkeit $P(t|c)$ (*likelihood*) sowie die a priori Wahrscheinlichkeit der Klasse $P(c)$ (*prior*). Beide werden aus der Trainingsmenge geschätzt. Die einfachste Form der Schätzung ist die *Maximum Likelihood Estimation* (MLE), bei welcher die Parameter aus den relativen Häufigkeiten bestimmter Merkmale der Trainingsmenge geschätzt werden. Für die a priori Wahrscheinlichkeit $P(c)$ wird daher angenommen, dass diese umso höher ist, je häufiger die Klasse in der Trainingsmenge auftritt. Sie entspricht damit der Anzahl derjeniger Objekte, die Klasse c zugehörig sind im Verhältnis zur Gesamtzahl der Objekte[17]:

$$\hat{P}(c) = \frac{N_c}{N} \tag{4.4}$$

Um den likelihood zu bestimmen, wird ein zu klassifizierendes Objekt als eine Menge von Merkmalen (*Features*) $x_1, x_2, ..., x_n$ repräsentiert – der Parameter wird dann interpretiert als die Wahrscheinlichkeit eines Merkmalsvektors bei gegebener Klasse:

$$\hat{P}(t|c) = P(x_1, x_2, ..., x_n|c) \tag{4.5}$$

Was Naive Bayes zu einem „naiven“ Verfahren macht, ist unter anderem die vereinfachte Annahme der „conditional independence“ – es wird angenommen, dass die Merkmals-Wahrscheinlichkeiten $P(x_i|c_j)$ jeweils unabhängig voneinander sind. Das Resultat dieser Annahme ist, dass die Berechnung von $\hat{P}(t|c)$ weiter vereinfacht werden kann, indem die Merkmals-Wahrscheinlichkeiten faktorisiert werden:

$$\begin{aligned} \hat{P}(t|c) &= P(x_1|c) \times P(x_2|c) \times ... \times P(x_n|c) \\ &= \prod_{1 \leq k \leq n} P(x_k|c) \end{aligned} \tag{4.6}$$

[17] Man schreibt nun $\hat{P}$ statt P, da die tatsächlichen Werte nicht bekannt, sondern lediglich aus der Trainingsmenge geschätzt worden sind.

Mittels MLE werden auch die bedingten Merkmals-Wahrscheinlichkeiten aus relativen Häufigkeiten geschätzt. Um zu verhindern, dass das Nichtauftreten eines Merkmals unmittelbar dazu führt, dass die aktuelle Klasse nicht mehr (unabhängig von der sonstigen Evidenz) in Betracht gezogen wird, wird zusätzlich *Laplace-Smoothing* angewandt, indem jeweils 1 addiert wird.

$$\hat{P}(x_i|c_j) = \frac{count(x_i, c_j) + 1}{\sum\limits_{x \in X} (count(x, c_j) + 1)} \tag{4.7}$$

Es ergibt sich für die Berechnung der besten Klasse nach Naive Bayes folgende Formel:

$$c_{NB} = \arg\max_{c \in \mathbb{C}} \hat{P}(c) \times \prod_{x \in X} \hat{P}(x|c) \tag{4.8}$$

Um nun dieses Verfahren auf das vorliegende Problem anzuwenden, werden die einzelnen Tokens eines Paragraphen als zu klassifizierende Objekte definiert. Im nächsten Schritt müssen die Merkmale bestimmt werden, die zur Klassifikation der Tokens benutzt werden können.

Auswahl der Merkmale

Bei der Auswahl der passenden Merkmale, die das Klassifikationsverfahren zur Bestimmung der Klassifizierungsfunktion verwendet, können bei der Tokenklassifikation sowohl Eigenschaften des Tokens selbst, als auch des umgebenden Kontext herangezogen werden. Diese Eigenschaften umfassen sowohl Form als auch morphologische und syntaktische Merkmale. Das zu klassifizierende Token zeichnet sich etwa durch eine spezifische Wortart, die im POS-Tag notiert ist, sowie durch Groß- oder Kleinschreibung oder das Vorhandensein spezifischer Suffixe wie „-heit“, „-nis“, „-schaft“ oder „-lich“ aus. Weiterhin wurde bereits gezeigt, dass Kompetenzen in mehr oder weniger regelhaften Kontexten auftreten, sodass zusätzlich auch Text und Wortart des jeweils vorangehenden und folgenden Tokens[18] einbezogen werden sollten.

Während der Vorverarbeitungsphase wurden die in Tabelle 5 gelisteten Merkmale im Anschluss an das Parsing mit erfasst und gespeichert.

[18] Da der Kontext durch die Begrenzung auf je einen Paragraphen bereits stark beschränkt ist, erschien es nicht sinnvoll, mehr als ein Token auf jeder Seite im Kontext des Zieltokens zu betrachten.

Merkmal	Erklärung
POS	POS Tag des zu klassifizierenden Tokens (String)
token	das zu klassifizierende Token selbst (String)
precedingPOS	POS Tag des vorangehenden Tokens (String)
precedingToken	vorangehendes Token (String)
followingPOS	POS Tag des folgenden Tokens (String)
followingToken	folgendes Token (String)
upperCaseStart	Token beginnt mit Großbuchstaben (Boolean)
hasSuffixOfInterest	Token endet auf einem bestimmten Suffix (Boolean)
punctuationFollowing	folgendes Token ist Punktuationszeichen (Boolean)

Tabelle 5: Merkmale, die für die Tokenklassifikation mittels Naive Bayes in Betracht gezogen wurden.

Um zu verhindern, dass die Parameter des Klassifikators zu stark an das Trainingskorpus angepasst werden und damit bei der Anwendung auf unbekannte Daten nicht gut skalieren (*Overfitting*, vgl. Farkas (2009: 9)), wurde das Verhalten des Modells bereits während der Entwicklung mithilfe eines Kreuzvalidierungsverfahrens[19] evaluiert. Dadurch konnte auch festgestellt werden, welche der Merkmale die bestmöglichen Klassifikationsentscheidungen hervorrufen. Beispielsweise schien das zu klassifizierende Token selbst kein guter Indikator zu sein, was auch plausibel erscheint, da die zu extrahierende Kompetenz im vorliegenden Problem eine Unbekannte darstellt. Demgegenüber scheinen die POS-Tags der Tokens ein verlässlicheres Merkmal zu sein (vgl. auch Abschnitt 5.3).

Klassifikation von Tokens als Kompetenz-Anker

Die vorliegende Klassifikationsaufgabe besteht nun darin, jedem Token eines Paragraphen eine von zwei möglichen Klassen zuzuordnen. Es handelt sich also um eine binäre Klassifikation, bei der es zu entscheiden gilt, ob das Token einen sogenannten *Kompetenz-Anker* darstellt oder nicht. Ein Kompetenz-Anker wird definiert als das zentrale Token in einer Phrase, die eine Kompetenz bezeichnet, wie etwa das Token „Kenntnisse“ in der Phrase „umfangreiche Kenntnisse in Java“. Diese Vereinbarung wurde getroffen, um der geringen Komplexität des Klassifikationsverfahrens entgegen zu kommen und da davon ausgegangen wird, dass die komplette Kompetenzphrase vom Anker ausgehend durch syntaktische Analyse zurückgewonnen werden kann.

Während der Trainingsphase wir dem Klassifikator eine Menge vorausgezeichneter

[19] Das Vorgehen bei diesem Verfahren wird in Abschnitt 5.3 erläutert.

Token	**ist Anker**
Es	false
werden	false
umfangreiche	false
Kenntnisse	true
der	false
Elektrotechnik	false
und	false
sehr	false
gute	false
Englischkenntnisse	true
benötigt	false
.	false

Tabelle 6: Ergebnis der Token-Klassifikation (konzeptionell).

Kompetenz-Anker übergeben. Dieser speichert in seinem Modell jeweils die Häufigkeiten der Ausprägungen der zu berücksichtigenden Merkmale sowie die Gesamthäufigkeit der Anker pro Klasse.

Um nun innerhalb eines Paragraphen diejenigen Tokens zu bestimmen, die Kompetenz-Anker sind, wird in der Testphase jedes einzelne Token an den Klassifikator übergeben. Dieser berechnet entsprechend den oben definierten Formeln diejenige Klasse, der das Token am wahrscheinlichsten zugehörig ist. Effektiv wird damit bestimmt, ob das Token einen Anker darstellt oder nicht. Da das Aufmultiplizieren der vielen bedingten Wahrscheinlichkeiten zu einem Floating Point Underflow führen kann, wird in der Implementation des Naive Bayes Verfahrens häufig stattdessen mit der Aufsummierung von Logarithmen gearbeitet (siehe Formel (4.9)), was auch in dieser Arbeit so umgesetzt wurde.

$$c_{map} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(e_k|c)] \tag{4.9}$$

Das Ergebnis der Klassifikation ist die Zuordnung eines Booleschen Wertes als Antwort auf die Frage, ob es sich bei dem klassifizierten Token um einen Kompetenz-Anker handelt oder nicht, zu jedem Token eines Paragraphen. Tabelle 6 veranschaulicht diese Zuordnung anhand eines fiktiven Beispielsatzes.

5. Evaluation

Ein zentraler Aspekt in der IE-Forschung ist die Evaluation der entworfenen Systeme. Mithilfe standardisierter Metriken wird ermöglicht, Vergleiche zwischen verschiedenen Systemen anzustellen, was vor allem Gegenstand der MUCs und darauffolgender Evaluationsprogramme war. Aber auch bereits während der Entwicklung eines IE-Systems ist es von zentraler Bedeutung, dessen Performanz objektiv zu beurteilen. In der vorliegenden Studie wurden mehrere Verfahren zur Extraktion von Anforderungsprofilen aus Stellenanzeigen erprobt, die nun jeweils und gegeneinander evaluiert werden sollen um feststellen zu können, welches Verfahren für das vorliegende Problem am geeignetsten scheint. Das während der Vorverarbeitung manuell annotierte Korpus war dabei nicht nur für die Inspektion zum Aufdecken relevanter Muster sowie als Trainingskorpus für das maschinelle Lernverfahren nützlich, sondern dient gleichzeitig als Referenz für die Evaluation der jeweiligen Resultate der Verfahren.

5.1. Evaluationsmetriken für IE

Im Rahmen der MUCs (vgl. Abschnitt 1.1) wurden die entsprechenden Metriken zur Evaluation von IE-Systemen entwickelt. Ausgangspunkt waren die im IR bereits etablierten Maße *Precision* (Genauigkeit) und *Recall* (Vollständigkeit). Im Kontext einer Suchanfrage etwa bezeichnet Precision den Anteil der relevanten Dokumente unter allen vom System gelieferten Dokumenten. Recall demgegenüber gibt das Verhältnis zwischen den zurückgelieferten relevanten und allen Dokumenten, die tatsächlich relevant sind, wieder. Formel 5.1 (nach Eikvil, 1999) veranschaulicht die Zusammenhänge:

$$\begin{aligned} Precision &= \frac{\#\text{correct answers}}{\#\text{answers produced}} \\ Recall &= \frac{\#\text{correct answers}}{\#\text{total possible corrects}} \end{aligned} \tag{5.1}$$

Der Wertebereich der beiden Maße liegt im Intervall $[0, 1]$ mit dem Optimum bei $1,0$ (100%). Die beiden Maße stehen außerdem in einem Verhältnis in der Form, dass man etwa einen höheren Recall auf Kosten der Precision und umgekehrt erreichen kann. Eines der Maße allein ist zudem nicht aussagekräftig genug, da ein System beispielsweise einen Recall von 100% erreichen kann, wenn es einfach alle Dokumente zurückliefert. Dies ist

der Grund, weshalb sich zusätzlich das *F-Maß* etabliert hat. Dabei handelt es sich um das gewichtete harmonische Mittel aus Precision (P) und Recall (R), welches folgendermaßen berechnet wird:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{5.2}$$

Der Parameter β erlaubt dabei eine Gewichtung: Werte für $\beta > 1$ legen den Schwerpunkt auf Recall, für $\beta < 1$ auf Precision. Mit $F1$ wird das Maß bezeichnet, wenn es Precision und Recall als gleichwertig behandelt ($\beta = 1$).

Während der MUCs wurde die Entscheidung getroffen, diese IR-Metriken zur Evaluation der IE-Systeme zu adaptieren (vgl. Chinchor et al., 1993: 418): „Recall then refers to how much of the information that[sic!] was correctly extracted, while precision refers to the reliability of the information extracted“ (Eikvil, 1999: 7). Sie mussten allerdings für IE-Aufgaben teilweise neu definiert werden, da hier die Dichotomie zwischen korrekt und inkorrekt nicht unmittelbar gegeben ist. Beispielsweise kann es in der IE im Gegensatz zum IR vorkommen, dass Ergebnisse als teilweise korrekt gewertet werden müssen, wenn eine extrahierte Phrase zu Teilen mit der Vorgabe überlappt oder sie enthält. Lavelli et al. (2004) weisen auf diese und weitere problematische Aspekte hin, die sich bei der Evaluation von IE-Systemen ergeben können. Vor der Evaluation sollte man sich demnach Gedanken machen, wie man etwa mit Tokenisierungsproblemen umgehen möchte. Wie relevant sind beispielsweise die Grenzen der extrahierten Textfragmente? Wie ist ein Fragment zu bewerten, dem ein Komma fehlt, das in der Referenz vorhanden ist? Die Antworten auf diese Fragen sowie alle weiteren Umstände der Evaluation (verwendete Software, Gestalt der Templates, Größe und Aufteilung der Trainings- und Testmenge etc.) sollten daher stets gemeinsam mit der Evaluation dokumentiert werden.

In der hier dargelegten Studie wurden insgesamt 110 Paragraphen untersucht, die das Ergebnis der Filterung aus einer ursprünglichen Menge von 376 Paragraphen nach der Klasse *Kompetenz* sind. Die manuelle Auszeichnung kam zu dem Ergebnis, dass sich in den Paragraphen insgesamt 483 Slot-Filler (*Goldstandard*) befinden. In der Evaluation muss nun bestimmt werden, wie viele von diesen von den einzelnen Verfahren tatsächlich geliefert werden, sowie weiterhin, wie viele Tokens oder Phrasen fälschlicherweise als Kompetenzen ausgezeichnet wurden. In den folgenden Abschnitten werden die Resultate der musterbasierten Extraktion und des Naive Bayes Verfahrens separat erläutert.

5.2. Ergebnisse der musterbasierten Extraktion

Die musterbasierte Extraktion liefert in jedem Fall, ob sie nun mittels regulärer Ausdrücke oder mithilfe des Dependenzparsers durchgeführt wurde, ganze Phrasen zurück, von denen das Verfahren annimmt, dass diese jeweils eine Kompetenz bezeichnen. Diese Phrasen können nun mit den Phrasen aus dem Goldstandard verglichen werden, um die Korrektheit der Ergebnisse zu bestimmen.

Neben den korrekten Ergebnissen, d. h. den extrahierten Phrasen, die mit denen aus dem Goldstandard übereinstimmen (*true positives*, TP), können zwei Fehlertypen auftreten: Entweder wurde eine Phrase nicht gefunden (*false negatives*, FN), oder aber es wurde fälschlicherweise eine Phrase extrahiert, die nicht Teil des Goldstandards ist (*false positives*, FP). Die Anzahl der falsch positiven Resultate wird bestimmt, indem jede extrahierte Phrase mit dem Goldstandard abgeglichen wird. Umgekehrt entspricht die Anzahl der falsch negativen Resultate der Größe der Menge aller Phrasen im Goldstandard, die nicht in der Ergebnismenge enthalten sind. Diejenigen Phrasen, die keine Kompetenzen darstellen und auch vom System nicht extrahiert wurden (*true negatives*, TN) spielen natürlich keine Rolle.

Um die korrekten Ergebnisse zu bestimmen, werden großzügige Kriterien zur Übereinstimmung von Phrasen angelegt. Sowohl Ergebnis- als auch Goldstandard-Phrasen werden jeweils mit einem eigenen Tokenisierer bearbeitet, der lediglich alphanumerische Zeichensequenzen als Tokens zurückliefert, sodass etwa Interpunktionszeichen beim Vergleich nicht berücksichtigt werden. Desweiteren wird ein Stopwort-Filter angewandt, sodass in den zu vergleichenden Phrasen nur die Inhaltswörter übrig bleiben. Schließlich wird per Mengenoperation bestimmt, ob eine Schnittmenge zwischen den Wörtern beider Phrasen besteht. Sofern die Schnittmenge nicht der leeren Menge entspricht, wird die Ergebnisphrase als korrekt bewertet.[20]

[20] Die Alternative zu diesem Verfahren wäre die Vergabe von partiellen Scores abhängig von der Größe der Schnittmenge beider Phrasen, wie dies beispielsweise in den MUC-Evaluationen vorgenommen wurde (vgl. Chinchor et al., 1993). Dieses Prozedere wurde im Rahmen dieser Arbeit als zu aufwendig betrachtet, sollte aber in der Projektfortsetzung in Betracht gezogen werden.

Die Werte für Precision und Recall lassen sich nun anhand der Häufigkeiten von FP, FN und TP folgendermaßen berechnen:

$$P = \frac{TP}{TP + FP}$$
$$R = \frac{TP}{TP + FN} \tag{5.3}$$

Folgende Ergebnisse wurden von den manuellen Verfahren erzielt. Das Verfahren mittels Pattern Matching war in der Lage, 389 der 483 im Goldstandard enthaltenen Phrasen zu extrahieren. Zusätzlich wurden 63 Phrasen fälschlicherweise extrahiert (FP) sowie 68 Phrasen nicht gefunden (FN). Das Pattern Matching Verfahren erreicht damit einen Recall von 85,1% sowie eine Precision von 86%, was einen F1-Wert von 85,6% entspricht.

Demgegenüber konnten mithilfe des Dependenzparsers lediglich 137 korrekte Phrasen extrahiert werden. Insbesondere erreichte er eine hohe FN-Rate von 143 nicht erkannten Phrasen. Aufgrund der großen Zahl von Fehlentscheidungen erreicht dieses Verfahren einen Recall von nur 48,9% mit einer Precision von 71%. Dies entspricht einem F1-Wert von 57,9%.

5.3. Ergebnisse des maschinellen Lernverfahrens

Bei der Anwendung des Naive Bayes Klassifikators wurden jeweils nur einzelne Tokens betrachtet und zu jedem Token separat entschieden, ob er einen Kompetenz-Anker darstellt oder nicht. Es sind daher nicht dieselben Vergleichskriterien wie in Abschnitt 5.2 nötig, um die Korrektheit der Klassifikationsentscheidungen zu evaluieren, da in diesem Fall unmittelbar mit dem entsprechenden Token aus dem Goldstandard verglichen werden kann.

Zunächst wird der Goldstandard bestimmt, indem die Information darüber, ob ein Token einen Kompetenz-Anker darstellt, aus dem Trainingskorpus entnommen wird. Anschließend wird das *Kreuzvalidierungs-Verfahren* angewandt: Das gesamte vorliegende Trainingskorpus T besteht aus 4920 Tokens (potentielle Kompetenz-Anker). Dieses wird in 10 etwa gleich große Teilmengen $T_1 ... T_{10}$ zerlegt. Iterativ wird nun eine dieser Teilmengen T_i als Testmenge ausgewählt, die übrigen $\{T_1 ... T_{10}\} \setminus \{T_i\}$ bilden die Trainingsmenge, mit der der Naive Bayes Klassifikator trainiert wird. Anschließend wird jeweils das Testset T_i klassifiziert und das Ergebnis der Klassifikation gegen den Goldstandard evaluiert.

Verfahren		**Precision**	**Recall**	**F1**
Pattern Matching		**0,86**	0,851	**0,856**
Dependenzparsing		0,71	0,489	0,579
NaiveBayes	alle Merkmale	0,331	**0,956**	0,491
	nur POS Tags	0,485	0,673	0,561
	POS Tags und Form	0,445	0,804	0,571

Tabelle 7: Gesamtergebnisse der Evaluation. Der jeweils beste Wert ist fett hervorgehoben.

Dabei gilt: Ein Token ist genau dann korrekt als Kompetenz-Anker ausgezeichnet worden (TP), wenn das gleiche Token auch nach dem Goldstandard einen Kompetenz-Anker darstellt. Fehlerhafte Entscheidungen sind die fälschlich vorgenommene Auszeichnung als Kompetenz-Anker (FP) sowie die nicht vorgenommene Auszeichnung (FN). Die Ergebnisse der 10-fachen Kreuzvalidierung wurden schließlich gemittelt.

Wie in Abschnitt 4.2 erläutert hat die Auswahl der Merkmale als Parameter für das Klassifikationsmodell einen Einfluss auf dessen Performanz, darum sind auch die Werte für Precision, Recall und F1 abhängig von den für die Klassifikation genutzten Parametern. Beispielsweise kann durch die Einbeziehung aller Merkmale ein sehr hoher Recall von 95,6% erreicht werden, was allerdings mit einer geringen Precision von nur 33,1% einhergeht und entsprechend in einem F1-Wert von 49% resultiert. Demgegenüber konnte die höchste Precision von 48,4% erreicht werden, indem ausschließlich die POS-Informationen herangezogen wurden – in diesem Fall sank allerdings der Recall auf 67,3%. Der höchste F1-Wert (57,1% bei 44,6% Precision und 80,3% Recall) wurde erreicht, indem neben den POS-Informationen auch die Formmerkmale des zu klassifizierenden Tokens herangezogen wurden.

5.4. Diskussion

Die hier erprobten Verfahren zur Extraktion von Anforderungsprofilen aus Stellenanzeigen unterscheiden sich stark in ihrer Performanz (vgl. Tabelle 7). Die Extraktion mithilfe regulärer Ausdrücke erreicht gleichermaßen hohe Werte für Precision und Recall, d. h. dass sowohl ein Großteil der vorhandenen Information geliefert wird, als auch der größte Teil der gelieferten Information tatsächlich eine Kompetenzbeschreibung darstellt. Die Precision erreicht zudem den höchsten Wert unter allen getesteten Verfahren. Es muss hierbei allerdings bedacht werden, dass zur Evaluation recht großzügige Kriterien angelegt wurden, was die Übereinstimmung der extrahierten Phrasen mit den Vorgaben betrifft.

Daher ist davon auszugehen, dass die zurückgelieferten Phrasen nur teilweise korrekt sind und eine umfangreiche Nachbearbeitung notwendig ist, in der insbesondere die korrekten Begrenzungen der Kompetenzbeschreibungen ermittelt werden müssen.

Die Extraktion mithilfe des Dependenzparsers erreicht unter denselben losen Kriterien eine immer noch recht hohe Präzision, liefert allerdings insgesamt deutlich weniger der im Korpus vorhandenen Daten zurück. Dies ist vermutlich insbesondere darauf zurückzuführen, dass der Dependenzparser nur auf grammatikalisch korrektem Input am besten funktioniert. Wie allerdings dargelegt wurde, wird ein Großteil der Anforderungsprofile in einem Listenformat verfasst, welches für das Parsing eher ungeeignet ist.

Diese Ergebnisse legen nahe, den Knowledge Engineering Ansatz für den vorliegenden Anwendungsfall weiter zu verfolgen. Die meisten Systeme, die mit manuell erstellten Regeln arbeiten, erreichen ähnliche und bessere Werte für die IE in dieser und in anderen Domänen (vgl. etwa Soderland, 1999; Califf, 1998; Bsiri & Geierhos, 2007). Dabei verwenden sie Techniken, bei denen die spezifizierten Muster Kombinationen aus exakt zu matchendem Text und Variablen sind, die Informationen aus Vorverarbeitungsschritten einbeziehen, wie etwa POS-Tags oder semantische Labels.

> Because of the complexity of natural language (except in the most restricted of sublanguages), it is not practical to decribe these patterns directly as word sequences. So, as in most natural language processing systems, we begin by structuring the input, identifying various levels of constituents and relations, and then state our patterns in terms of these constituents and relations.
>
> (Grishman, 1997: 12)

Entsprechend erscheint es sinnvoll, die Stärken des Dependenzparsers mit der Mächtigkeit der regulären Ausdrücke zu kombinieren, um bestmögliche Resultate bei der Extraktion zu erzielen. Denkbar wäre zum einen eine Trennung zwischen dem Listen- und dem Fließtextformat, sodass mittels RegExes die Bestandteile von Listen zuverlässig extrahiert werden können, während vollständige Sätze mit dem Dependenzparser behandelt werden können. Ebenfalls wäre es möglich, morphologische und syntaktische Informationen bei der Spezifizierung der zu suchenden Muster durch reguläre Ausdrücke mit einzubeziehen, indem beispielweise der Kontext einer gesuchten Information mit entsprechenden Variablen belegt wird. Systeme wie FASTUS liefern ein Modell, wie derartige Muster aussehen können.

Die Klassifikation von Kompetenz-Ankern als zentrale Tokens einer zu extrahierenden Phrase mithilfe des Naive Bayes Klassifikators hat in dieser Studie die schlechtesten

Ergebnisse erzielt. Diese sind zudem stark abhängig von der Auswahl der verwendeten Merkmale zur Bestimmung des Klassifikationsmodells. Alle getesteten Merkmalskombinationen resultieren in einer Precision unter 50%, sodass meist weniger als die Hälfte der als Kompetenz-Anker ausgezeichneten Tokens auch tatsächlich solche darstellen. Insbesondere in diesem Fall ist es von entscheidender Bedeutung festzulegen, ob der Schwerpunkt der Klassifikation auf dem Recall oder der Precision liegen sollte. Der Spitzenwert von ca. 96% Recall etwa zeigt, dass es möglich ist, die relevanten Kompetenz-Anker fast vollständig aus dem Korpus zu bestimmen, wobei gleichzeitig ebensoviel ungewünschte Information geliefert wird, die in der Nachbearbeitungsphase bereinigt werden muss.

Die meisten der aktuellen IE-Systeme basieren auf mächtigen statistischen Verfahren und erzielen damit inzwischen mindestens ebenso gute Ergebnisse wie die Systeme, die auf manuelle Regelerstellung setzen (vgl. Ireson et al., 2005). Daher ist anzunehmen, dass der hier erprobte Klassifikationsalgorithmus schlicht zu simpel ist, um im vorliegenden Fall zufriedenstellend anwendbar zu sein. Entsprechend wird an dieser Stelle vorgeschlagen, noch weitere Klassifizierungsverfahren wie etwa HMMs oder CRFs zu erproben, insbesondere da Krönke (2013) ebenfalls in der Domäne der Stellenanzeigen mithilfe eines CRF-Klassifizierers zufriedenstellende Resultate erzielen konnte. Alternativ könnte auch versucht werden, das Naive Bayes verfahren selbst weiter zu verbessern, da etwa Freitag (1998) zeigt, wie Naive Bayes zur Extraktion von Textfragmenten genutzt werden kann. Voraussetzung ist allerdings in beiden Fällen die Verfügbarkeit größerer, sorgfältig annotierter Korpora.

Schließlich ist auch eine Kombination von regelbasierten und statistischen Verfahren denkbar, indem die jeweiligen Stärken geschickt gemeinsam eingesetzt werden. Beispielsweise würde das Verfahren mit dem höchsten Recall – die Klassifikation einzelner Tokens als relevant oder nicht relevant – in einem ersten Schritt eingesetzt werden, um eine große Abdeckung zu erreichen. In einem zweiten Schritt könnten anschließend die musterbasierten Methoden angewandt werden, um die gelieferten Ergebnisse zu bereinigen und unerwünschte Information zu entfernen.

6. Schlussbemerkungen und Ausblick

In der vorliegenden Arbeit wurden verschiedene Verfahren erprobt, die es ermöglichen sollen, automatisch Textfragmente aus dem Text von Stellenanzeigen zu extrahieren, die in ihrer Gesamtheit das Anforderungsprofil der Stelle bilden. Der langfristige Zweck dieser Extraktion ist die Speicherung dieser und weiterer Informationen in einem kompakten, strukturierten Format, das die Weiterverarbeitung mit Data Mining Werkzeugen erleichtert. Die Aufgabe dieser Studie bestand in der Evaluation möglicher Lösungsansätze für dieses Problem, um einen Wegweiser für nachfolgende Projekte bieten zu können.

Die erprobten Verfahren fallen in zwei Kategorien. Auf der einen Seite steht der Versuch, die zu extrahierenden Textsegmente über möglichst spezifische Muster zu beschreiben, sodass das System diese Muster im Text suchen und die Zeichenkette unmittelbar extrahieren kann. Die Spezifizierung der Muster erfolgte dabei einerseits mithilfe regulärer Ausdrücke, andererseits auf syntaktischer Ebene über die Dependenzstruktur des Satzes. Alternativ zu dieser manuellen Methode wurde ebenfalls erprobt, inwieweit ein Klassifikationsalgorithmus in der Lage ist, aus bestimmten vorannotierten Merkmalen des Textes ein Modell zu bilden, mit dessen Hilfe er in der Lage ist, jeweils den Kern der gesuchten Information zu ermitteln.

Die Evaluation der Verfahren kommt zu dem Ergebnis, dass von den hier erprobten Verfahren die manuelle Regelerstellung dem maschinellen Lernverfahren überlegen ist. Das manuelle Verfahren bietet zudem noch Raum für Verbesserungen, indem beispielsweise die jeweiligen Stärken von regulären Ausdrücken und Dependenzparser gezielter eingesetzt sowie ggf. miteinander kombiniert werden. Hierfür ist es nötig, dass entsprechende Experten zur Verfügung stehen, die Muster und Grammatiken mit angemessener Genauigkeit erstellen.

Aufgrund des beschränkten zeitlichen Rahmens konnte lediglich ein recht einfaches Verfahren des maschinellen Lernens, der Naive Bayes Klassifikationsalgorithmus, eingesetzt werden. Insbesondere im Hinblick auf Präzision konnte dieses Verfahren mit den musterbasierten Ansätzen nicht mithalten. Es wird davon ausgegangen, dass mächtigere statistische Verfahren deutlich bessere Resultate in der vorliegenden Domäne erzielen können, wie dies in verwandten Arbeiten bereits demonstriert wurde. Hierfür muss insbesondere der Annotationsprozess ausgebaut werden, um verlässlichere und eindeutigere Merkmale definieren zu können, die ein automatisches Verfahren zur Entscheidungsfindung nutzen kann.

Im Hinblick auf die Implementierung eines lauffähigen Systems zur Extraktion von strukturierter Information aus Stellenanzeigen sollten in einem nächsten Schritt zunächst folgende Fragen geklärt werden: Stehen die entsprechenden Ressourcen zur Verfügung, die zur ausführlichen manuellen Annotation großer Korpora oder zum Erstellen domänenspezifischer Muster benötigt werden? Liegt der Schwerpunkt bei der Extraktion der Information auf Genauigkeit oder auf Vollständigkeit, und in welchem Umfang ist eine Nachbearbeitung der Resultate akzeptabel? Erst die Antworten auf diese Fragen ermöglichen eine Entscheidung, welches Verfahren schließlich implementiert werden sollte. Schließlich sollte insbesondere im Hinblick auf die Extraktion von Bewerberkompetenzen in Betracht gezogen werden, Modifizierer derselben, d. h. Angaben über die relative Stärke der geforderten Kompetenz, separat zu behandeln, da das Vorhandensein dieser Modifizierer insbesondere für statistische Extraktionsverfahren ein wertvolles Hilfsmittel sein könnte.

Erklärung

Hiermit erkläre ich an Eides statt, dass ich diese Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken und Quellen, einschließlich der Quellen aus dem Internet, entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Dasselbe gilt sinngemäß für Tabellen, Karten und Abbildungen. Diese Arbeit habe ich in gleicher oder ähnlicher Form oder auszugsweise nicht im Rahmen einer anderen Prüfung eingereicht.

Ich versichere zudem, dass der Text der elektronischen Fassung mit dem Text der vorgelegten Druckfassung identisch ist.

Köln, den 6. November 2014 Unterschrift: _____

A. Hinweise zur beiliegenden CD

Auf der dieser Arbeit beiliegenden CD befindet sich neben der schreibgeschützten elektronischen Fassung dieses Dokuments auch der Quelltext aller zur Durchführung der Evaluationen implementierten Software-Komponenten. Diese liegen in Form eines Projektarchivs vor, welches wahlweise entpackt oder unmittelbar in eine IDE wie eclipse importiert werden kann.

Innerhalb des Projektes sind die einzelnen Klassen und weitere Dateien in einer Paketstruktur angeordnet, welche die jeweilige Funktionalität wiederspiegeln soll:

```
/
├── src
│   └── spinfo/tm
│       ├── data ... zentrale Datenstrukturen
│       ├── evaluation ... Evaluierung der Extraktionsergebnisse
│       ├── extraction ... Informationsextraktion
│       │   ├── pattern ... Extraktion mittels RegExes
│       │   ├── parsing ... Extraktion mithilfe des Dependenzparsers
│       │   └── learning ... Extraktion durch Klassifikation
│       ├── preprocessing ... Vorverarbeitung der Daten
│       └── util ... Hilfsklassen
├── test ... Enthält JUnit-Testklassen
├── data ... zugrundeliegende Daten
├── doc ... Javadoc
├── lib ... verwendete Drittanbieter-Bibliotheken
├── models ... Modelle
└── output ... Ausgabeverzeichnis
```

Es werden in dem Projekt mehrere ausführbare Klassen bereitgestellt. Diese stellen komplette *Workflows* dar, d. h. sie führen die Informationsextraktion mit dem jeweiligen Verfahren durch und evaluieren das Resultat im Anschluss. Es handelt sich um die Klassen `PatternMatcherWorkflow`, `DepCompetenceFinderWorkflow` sowie `ClassifierWorkflow`. Die Ergebnisse werden jeweils in entsprechend dem Klassennamen benannten Dateien im *data* Ordner des Projektes abgelegt, wobei die extrahierten bzw. klassifizierten Daten in einem TSV-Format (`*.csv`) sowie die Evaluationsergebnisse in einer Textdatei (`*.txt`) hinterlegt werden.

Zentrale Datengrundlage für die Informationsextraktion ist die Datei *SingleClassTrai-*

ningDataFiltered.csv, die die vorklassifizierten Paragraphen enthält. Die Dateien *trainingsSet_ML.csv* und *trainingIE_140816.csv* enthalten die durch manuelle Auszeichnung gewonnenen Goldstandards für die Evaluation. Da insbesondere der Parsingvorgang sehr zeit- und ressourcenintensiv ist, werden weiterhin bereits einige zentrale Datenstrukturen als Binärdateien zur Verfügung gestellt. Die ausführbare Klasse `Preparation` ist für die Generierung dieser Dateien zuständig, sollten sie nicht vorliegen oder erneut generiert werden müssen.

B. Abkürzungsverzeichnis

IE Informationsextraktion

IR Information Retrieval

HTML Hypertext Markup Language

PDF Portable Document Format

NLP Natural Language Processing

MUC Message Understanding Conference

FRUMP Fast Reading Understanding and Memory Program

FASTUS Finite State Automaton Text Understanding System

LSP Linguistic String Project

RAPIER Robust Automated Production of IE Rules

SIRE Sémantique, Internet, Recrutement et Emploi

RegEx Regular Expression

FSM Finite State Machine

ML Maschinelles Lernen

HMM Hidden Markov Model

CRF Conditional Random Field

SVM Support Vector Machine

POS Part of Speech

MLE Maximum Likelihood Estimation

TP True Positive

TN True Negative

FP False Positive

FN False Negative

Literatur

Andersen, Peggy M. et al. (1992). „Automatic Extraction of Facts from Press Releases to Generate News Stories“. In: *ANLC '92: Proceedings of the Third Conference on Applied Natural Language Processing*. Stroudsburg, PA, USA: Association for Computational Linguistics, S. 170–177. URL: http://dx.doi.org/10.3115/974499.974531 (besucht am 02. 11. 2014).

Appelt, Douglas E., Jerry R. Hobbs et al. (1993). „FASTUS: A Finite-state Processor for Information Extraction from Real-world Text“. In: *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, S. 1172–1178. URL: http://www.isi.edu/~hobbs/ijcai93.pdf (besucht am 14. 10. 2014).

Appelt, Douglas E. & David J. Israel (1999). *Introduction to Information Extraction Technology*. A Tutorial Prepared for IJCAI-99. Menlo Park, CA, USA: SRI International. URL: http://www.ai.sri.com/~appelt/ie-tutorial/IJCAI99.pdf (besucht am 03. 11. 2014).

Banko, Michele et al. (2007). „Open Information Extraction from the Web“. In: *IJCAI 2007: Proceedings of the 20th International Joint Conference on Artificial Intelligence*. Hrsg. von Manuela M. Veloso, S. 2670–2676. URL: http://ijcai.org/Past%20Proceedings/IJCAI-2007/PDF/IJCAI07-429.pdf (besucht am 15. 10. 2014).

Bsiri, Sandra & Michaela Geierhos (2007). „Informationsextraktion aus Stellenanzeigen im Internet“. In: *LWA 2007: Lernen - Wissen - Adaption, Workshop Proceedings*. Hrsg. von Alexander Hinneburg. Halle: Martin-Luther-Universität Halle-Wittenberg, S. 229–236. URL: http://www.cis.uni-muenchen.de/download/publikationen/lwa07_bsiri_geierhos.pdf (besucht am 13. 10. 2014).

Califf, Mary Elaine (1998). *Relational Learning Techniques for Natural Language Information Extraction*. Dissertation.

Cardie, Claire (1997). „Empirical Methods in Information Extraction“. In: *AI Magazine* 18.4, S. 65–70. URL: http://www.aaai.org/ojs/index.php/aimagazine/article/view/1322/1223 (besucht am 16. 10. 2014).

Carstensen, Kai-Uwe et al., Hrsg. (2010). *Computerlinguistik und Sprachtechnologie. Eine Einführung*. 3. Aufl. Heidelberg: Spektrum Akademischer Verlag.

Chinchor, Nancy, David D. Lewis & Lynette Hirschman (1993). „Evaluating Message Understanding Systems: An Analysis of the Third Message Understanding Conference

(MUC-3)“. In: *Computational Linguistics* 19.3, S. 409–449. URL: http://dl.acm.org/citation.cfm?id=972487.972488 (besucht am 30. 10. 2014).

Church, Kenneth Ward (1980). „On Memory Limitations In Natural Language Processing“. Master's Thesis. Cambridge, MA, USA: Massachusetts Institute of Technology. URL: http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.18.5231&rep=rep1&type=pdf (besucht am 15. 10. 2014).

Cowie, James R. (1983). „Automatic Analysis of Descriptive Texts“. In: *ANLC '83: Proceedings of the First Conference on Applied Natural Language Processing*. Stroudsburg, PA, USA: Association for Computational Linguistics, S. 117–123. URL: http://dx.doi.org/10.3115/974194.974218.

DeJong, Gerald (1979). „Prediction and Substantiation: A New Approach to Natural Language Processing“. In: *Cognitive Science* 3 (3), S. 251–273. URL: http://onlinelibrary.wiley.com/doi/10.1207/s15516709cog0303_4/pdf (besucht am 14. 10. 2014).

Eikvil, Line (1999). *Information Extraction from World Wide Web. A Survey*. Technical Report 945. Oslo, Norwegen: Norwegian Computing Center. URL: http://www.nr.no/~eikvil/webIE_rep945.pdf (besucht am 12. 10. 2014).

Etzioni, Oren et al. (2005). „Unsupervised Named-Entity Extraction from the Web: An Experimental Study“. In: *Artificial Intelligence* 165 (1), S. 91–134.

Farkas, Richárd (2009). „Machine Learning techniques for applied Information Extraction“. Dissertation. University of Szeged. URL: http://www.inf.u-szeged.hu/~rfarkas/rfarkas_PhD_thesis.pdf (besucht am 18. 08. 2014).

Feldman, Ronen & James Sanger (2007). *The Text Mining Handbook - Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press.

Freitag, Dayne (1998). „Multistrategy Learning for Information Extraction“. In: *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., S. 161–169. URL: http://dl.acm.org/citation.cfm?id=645527.657302.

Freitag, Dayne & Andrew McCallum (2000). „Information Extraction with HMM Structures Learned by Stochastic Optimization“. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative*

Applications of Artificial Intelligence. AAAI Press, S. 584–589. URL: http://dl.acm.org/citation.cfm?id=647288.723414 (besucht am 13. 09. 2014).

Grishman, Ralph (1997). „Information Extraction: Techniques and Challenges“. In: *SCIE '97: International Summer School on Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*. London, UK: Springer-Verlag, S. 10–27. URL: http://dl.acm.org/citation.cfm?id=645856.669801 (besucht am 05. 09. 2014).

Gusfield, Dan (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge u.a.: Cambridge University Press.

Hermes, Jürgen (2014). *JASC - Job Advertisement Section Classifier. Framework zur Evaluation von Klassifikatoren für die Zuordnung von Abschnitten*. Ms. Institut für Sprachliche Informationsverarbeitung. Universität zu Köln.

Hopcroft, John E., Rajeev Motwani & Jeffrey D. Ullman (2011). *Einführung in Automatentheorie, Formale Sprachen und Berechenbarkeit*. 3. Aufl. München: Pearson Studium.

Ireson, Neil et al. (2005). „Evaluating Machine Learning for Information Extraction“. In: *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*. New York, NY, USA: ACM, S. 345–352. URL: http://doi.acm.org/10.1145/1102351.1102395 (besucht am 29. 10. 2014).

Jackson, Peter & Isabelle Moulinier (2002). „Information Extraction“. In: *Natural Language Processing for Online Applications*. Hrsg. von Ruslan Mitkov. Bd. 5. Natural Language Processing. Amsterdam / Philadelphia: John Benjamins Publishing Company, S. 75–118.

Jacobs, P. S. & Lisa F. Rau (1990). „SCISOR: Extracting Information from On-line News“. In: *Communications of the ACM* 33.11, S. 88–97. URL: http://doi.acm.org/10.1145/92755.92769 (besucht am 03. 11. 2014).

Jurafsky, Daniel & James H. Martin (2009). „Information Extraction“. In: *Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2. Aufl. Prentice Hall Series in Artificial Intelligence. New Jersey, USA: Pearson Prentice Hall, S. 759–798. URL: http://www.pearsonhighered.com/educator/product/Speech-and-Language-Processing/9780131873216.page (besucht am 12. 10. 2014).

Klabunde, Ralf (2010). „Automatentheorie und Formale Sprachen“. In: *Computerlinguistik und Sprachtechnologie. Eine Einführung*. Hrsg. von Kai-Uwe Carstensen et al. 3. Aufl. Heidelberg: Spektrum Akademischer Verlag, S. 66–93.

Krönke, Tobias (2013). „Parsing von Stellenanzeigen zur Vorhersage von Bewerberqualifikation“. Master's Thesis. Technische Universität Darmstadt. URL: http://www.lt.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_LangTech/student_theses/FINAL_Kroenke-JAPAQP-2013.pdf (besucht am 17. 06. 2014).

Langer, Hagen (2010). „Syntax und Parsing“. In: *Computerlinguistik und Sprachtechnologie. Eine Einführung*. Hrsg. von Kai-Uwe Carstensen et al. 3. Aufl. Heidelberg: Spektrum Akademischer Verlag, S. 280–329.

Lavelli, A. et al. (2004). „A Critical Survey of the Methodology for IE Evaluation“. In: *Proceedings of the 4th International Conference on Language Resources and Evaluation*. ELRA, S. 1655–1658. URL: http://staffwww.dcs.shef.ac.uk/people/F.Ciravegna/paperi/lrec2004.pdf (besucht am 11. 10. 2014).

Loth, Romain et al. (2010). „Linguistic Information Extraction for Job Ads (SIRE Project)“. In: *RIAO '10: Adaptivity, Personalization and Fusion of Heterogeneous Information*. Paris: C.I.D., S. 222–224. URL: http://dl.acm.org/citation.cfm?id=1937055.1937114 (besucht am 25. 06. 2014).

Lytinen, Steven L. & Anatole Gershman (1986). „ATRANS: Automatic Processing of Money Transfer Messages“. In: *AAAI '86:Proceedings of the 5th National Conference on Artificial Intelligence*. Bd. 2, S. 1089–1095. URL: http://www.aaai.org/Papers/AAAI/1986/AAAI86-180.pdf (besucht am 14. 10. 2014).

Manning, Christopher D., Prabhakar Raghavan & Hinrich Schütze (2008). *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press.

McCallum, Andrew (2005). „Information Extraction: Distilling Structured Data from Unstructured Text“. In: *Queue* 3.9, S. 48–57. URL: http://delivery.acm.org/10.1145/1110000/1105679/p48-mccallum.pdf (besucht am 02. 09. 2014).

Mooney, Raymond J. & Razvan Bunescu (2005). „Mining Knowledge from Text Using Information Extraction“. In: *SIGKDD Explorations* 7 (1), S. 3–10. URL: http://www.kdd.org/sites/default/files/issues/7-1-2005-06/2-Mooney.pdf (besucht am 12. 10. 2014).

Navarro, Gonzalo (2004). „Pattern Matching“. In: *Journal of Applied Statistics* 31.8, S. 925–949. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.4.649&rep=rep1&type=pdf (besucht am 02. 11. 2014).

Neumann, Günter (2010). „Text-basiertes Informationsmanagement“. In: *Computerlinguistik und Sprachtechnologie. Eine Einführung*. Hrsg. von Kai-Uwe Carstensen et al. 3. Aufl. Heidelberg: Spektrum Akademischer Verlag GmbH, S. 576–615.

Rosier, Arnaud, Anita Burgun & Philippe Mabo (2008). „Using regular expressions to extract information on pacemaker implantation procedures from clinical reports“. In: *AMIA 2008 Symposium Proceedings*. American Medical Informatics Association. AMIA, S. 81–85. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2656039 (besucht am 21. 10. 2014).

Sager, Naomi (1981). *Natural Language Information Processing: A Computer Grammmar of English and Its Applications*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Sekine, Satoshi (2006). „On-demand Information Extraction“. In: *COLING-ACL '06: Proceedings of the COLING/ACL on Main Conference Poster Sessions*. Stroudsburg, PA, USA: Association for Computational Linguistics, S. 731–738. URL: http://dl.acm.org/citation.cfm?id=1273073.1273167 (besucht am 22. 10. 2014).

Soderland, Stephen (1999). „Learning information extraction rules for semi-structured and free text“. In: *Machine Learning* 34. Hrsg. von Claire Cardie & Raymond Mooney, S. 233–272. URL: http://link.springer.com/article/10.1023/A:1007562322031 (besucht am 12. 09. 2014).

Tesnière, Lucien (1980). *Grundzüge der strukturalen Syntax*. Hrsg. und übers. von Ulrich Engel. Stuttgart: Klett-Cotta.

Zarri, Gian Piero (1983). „Automatic Representation of the Semantic Relationships Corresponding to a French Surface Expression“. In: *Proceedings of the First Conference on Applied Natural Language Processing*. Santa Monica, California, USA: Association for Computational Linguistics, S. 143–147. URL: http://www.aclweb.org/anthology/A83-1024 (besucht am 14. 10. 2014).