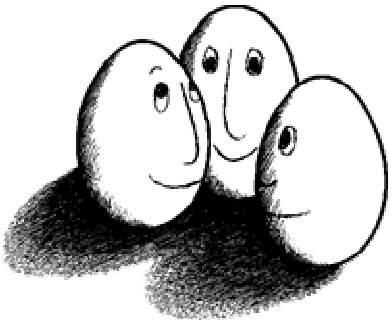
Diplomarbeit
zum Thema

Named Entity Recognition mit Conditional Random Fields

Felix Jungermann



Fachbereich Informatik
Lehrstuhl für künstliche Intelligenz (LS 8)
Prof. Dr. Katharina Morik

1. Betreuerin: **Prof. Dr. Katharina Morik**
2. Betreuer: **Dipl.-Inform. Timm Euler**

Dortmund, 27.07.2006

Abstract

Diese Diplomarbeit behandelt das Thema *Named entity recognition* (NER), das ausführlich in [29] behandelt wird. Da das Thema an sich für eine Diplomarbeit zu weitläufig ist, soll in dieser Arbeit nur ein bestimmtes Verfahren für die NER dargestellt werden: *Conditional random fields* (CRF).

CRF werden als das aktuelle Verfahren zur NER vorgestellt. Parallel zu dieser Arbeit wurde vom Diplomanden eine Implementierung der CRF in Java erweitert. Diese Implementierung wird vorgestellt und in YALE integriert. YALE ist ein Programm für *Data Mining*-Aufgaben, und wird kurz erläutert.

Abschließend werden Versuche auf zwei verschiedenen Datensätzen vorgestellt, und die Ergebnisse werden dargestellt.

Danksagung

Ich danke allen Mitarbeitern vom Lehrstuhl 8 für Künstliche Intelligenz in Dortmund, die mich bei dieser Arbeit unterstützt haben.

Ganz besonders danke ich meiner Betreuerin und Professorin Katharina Morik, meinem Betreuer Timm Euler, sowie Ingo Mierswa, der mir bei Implementierungsfragen immer wieder zur Seite stand.

Weiterhin danke ich meiner Freundin Simone, meiner Familie und allen, die diese Arbeit Korrektur gelesen haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

BNC	British national corpus
CFG	Context free grammar
CONLL	Conference on computational natural language learning
CRF	Conditional random field
DBN	Dynamic Bayesian network
DCRF	Dynamic CRF
GIS	Generalized iterative scaling
HMM	Hidden markov model
IE	Informationsextraktion
IIS	Improved iterative scaling
JNLPBA	Joint workshop on natural language processing in biomedicine and its applications
KCRF	Kernel conditional random fields
KDD	Knowledge discovery in databases
L-BFGS	Limited memory – BFGS (Broyden-Fletcher-Goldfarb-Shanno)
MEMM	Maximum entropy markov model
MeSH	Medical subject headings
MET	Multilingual entity task
MP	Markov Prozess
MRF	Markov random field
MUC	Message understanding conference
NE	Named entity
NER	Named entity recognition
NLP	Natural language processing
POS	Part of speech
SVM	Support vector machine
WE	Wissensentdeckung
YALE	Yet another learning environment

1 Einführung

1.1 Motivation

Das Thema der Diplomarbeit lautet „Named Entity Recognition mit Conditional Random Fields“. Einerseits behandelt die Arbeit also *Named entity recognition* (NER). Auf der anderen Seite werden *Conditional random fields* (CRF) vorgestellt.

NER entspricht der Aufgabe, in unbekannten Texten die so genannten *Named entities* (NE) zu finden und zu markieren. NE werden im Deutschen oft als Eigennamen bezeichnet, allerdings soll in dieser Arbeit die Bezeichnung NE benutzt werden. Welche Worte eines Textes als NE bezeichnet werden, wird durch die jeweilige Aufgabenstellung definiert. Ursprünglich waren NE triviale Typen wie PERSON, ORGANISATION oder LOCATION. Im Laufe der Zeit sind aber auch andere Bereiche durch NER untersucht worden, so dass NE kaum Beschränkungen unterliegen. NE sind jedoch größtenteils Substantive. Die Motivation für NER ist vor allem durch das Gebiet der *Informationsextraktion* (IE) gegeben. IE umfasst den Bereich der automatischen Erfassung von interessanten Daten (Informationen) aus Texten. Anwendungsgebiete für die IE gibt es viele: zum Beispiel sollten gute Suchmaschinen nicht nur die Texte von Webseiten erfassen, sondern am besten auch als relevant definierte Daten extrahieren. Auch in Wissensmanagement-Systemen ist IE sinnvoll, da aus umfangreichen Dokumenten automatisiert zum Beispiel eine Zusammenfassung erstellt werden kann. Die NER kann einerseits eine Vorverarbeitung sein, um bestimmte Daten im Text zu kennzeichnen. Andererseits kann jedoch auch die NER selbst als IE bezeichnet werden, wenn man ausschließlich an den NE eines Textes interessiert ist.

Diese Arbeit ist durch drei konkrete Punkte motiviert:

1. Es gibt mehrere Verfahren, NER zu betreiben, wobei die bekanntesten dieser Verfahren in Kapitel 2.5 vorgestellt werden. In dieser Diplomarbeit sollen CRF als konkurrenzfähiges Verfahren zur NER vorgestellt werden.
2. CRF sollen natürlich einerseits als nutzbares Instrument implementiert und andererseits in YALE (siehe Kapitel 10) eingebunden werden.
3. Viele (erfolgreiche) NER-Verfahren benutzen auf den Datensatz abgestimmte externe Quellen (vergleiche 14.2.1), um gute Ergebnisse zu erzielen. Jedoch muss zur Erstellung oder Nutzung dieser externen Quellen der Datensatz auf seine Besonderheiten untersucht werden. Um möglichst unabhängig von bestimmten Datensätzen zu sein und um den zu bearbeitenden Datensatz nicht mühsam analysieren zu müssen, soll ein allgemein nutzbares System entwickelt werden. Somit soll auch die Frage geklärt werden, wie gut die Ergebnisse eines Systems sind, das auf externe Quellen verzichtet.

1.2 Überblick

Kapitel 2 gibt einen Einblick in die NER, wobei die Entstehung erläutert und daraufhin NER in den Bereich der Wissensentdeckung in Datenbanken (*Knowledge discovery*

in databases (KDD)) eingeordnet wird. In einem Unterkapitel folgen die – vor allem linguistisch definierten – Probleme und Besonderheiten der NER sowie deren mögliche Vermeidung. Bevor im letzten Teil von Kapitel 2 vorhandene konkrete Verfahren zur NER vorgestellt werden, wird noch erwähnt, inwiefern man die Güte von NER-Verfahren erfassen kann.

Der größte Teil der Diplomarbeit befasst sich jedoch mit den CRF, deren Entstehung in Kapitel 3 beschrieben ist. Beginnend mit Markov-Prozessen (Kapitel 3.1) wird die Entwicklung der CRF vorgestellt, die viele Techniken der *Hidden markov models* (HMM) (Kapitel 3.2) aufgreifen. HMM sind aufgrund einiger Nachteile im Vergleich zu CRF jedoch eher schlecht für die NER geeignet, so dass die *Maximum entropy markov models* (MEMM) (Kapitel 3.3) vorgestellt werden. Allerdings weisen auch die MEMM Nachteile auf, die durch CRF überwunden werden können. Bevor allerdings auf CRF im Kapitel 4 näher eingegangen wird, werden noch die *Markov random fields* (MRF) (Kapitel 3.4) beschrieben, die eine Generalisierung von CRF darstellen.

CRF (Kapitel 4) werden als das momentan populärste Verfahren zur Markierung von Texten oder schlicht Sequenzen vorgestellt. Äquivalent zu den bereits bekannten Verfahren HMM und MEMM wird die Berechnung der bedingten Wahrscheinlichkeit mit CRF dargestellt. Weiterhin wird erläutert, wie die CRF in der Trainingsphase möglichst gut eingestellt werden. Die Einstellung der CRF funktioniert mittels des *maximum log-likelihood*-Verfahrens: mittels einer Menge von Potentialfunktionen, die über der Sequenz definiert sind, wird ein Wert berechnet. Da jede Potentialfunktion noch ein Gewicht hat, wird im Training dieses Gewicht so eingestellt, dass der aus den Potentialfunktionen resultierende Wert über den Trainingsdaten maximal wird. Bei der Anwendung auf nicht markierte Sequenzen wird die Labelsequenz zurückgegeben, die den maximalen Wert über die Potentialfunktionen liefert.

Kapitel 5 definiert noch einmal die Ähnlichkeit von CRF zu den HMM, indem die schon in Kapitel 2 definierten Aufgaben, die von HMM zu lösen sind, auf CRF übertragen werden. Diese Aufgaben umfassen die Berechnung der bedingten Wahrscheinlichkeit, die Erzeugung der Markierungen für eine Sequenz sowie die Einstellung der Parameter für das CRF. Ein wichtiger Punkt ist jeweils die effiziente Durchführung dieser Aufgaben.

Kapitel 6 befasst sich mit der Optimierung der Gewichte für die Potentialfunktionen in der CRF-Trainingsphase. Nachdem die Gewichte initialisiert sind, wird der Wert für den Trainingsdatensatz über allen Potentialfunktionen bestimmt. Danach werden mittels eines Optimierungsverfahrens die Gewichte verändert, so dass daraufhin ein (sofern möglich) höherer Wert über den Potentialfunktionen entsteht. Verschiedene Optimierungsmethoden werden in diesem Kapitel mitsamt ihrer Vor- und Nachteile vorgestellt.

Kapitel 7 erläutert andere Anwendungsgebiete von CRF neben der NER und führt in die aktuellen Weiterentwicklungen der CRF ein.

Da die CRF später noch experimentell untersucht werden sollen, wird eine Implementierung benötigt. In Kapitel 8 wird die grundlegende Implementierung vorgestellt, die vom Diplomanden im Laufe der Diplomarbeit verändert sowie erweitert

wird. Es wird auf die Implementierung der Trainings- und Anwendungsphase sowie der bereits vorhandenen Potentialfunktionen eingegangen.

Kapitel 9 umfasst die vom Diplomanden vorgenommenen Veränderungen an der grundlegenden Implementierung. Das Ziel ist es, durch Änderungen und Erweiterungen der Implementierung bessere Ergebnisse zu erzielen. Bessere Ergebnisse sind entweder durch eine höhere Güte bei der Evaluierung oder durch eine geringere Laufzeit der Versuche definiert. Um dieses Ziel zu erreichen werden neue Potentialfunktionen auf Basis der in Kapitel 2 vorgestellten NER-Prinzipien entwickelt.

Als Experimentierumgebung für KDD-Versuche im Allgemeinen und zum Beispiel CRF-Versuche im Speziellen wird YALE in Kapitel 10 vorgestellt. Hierbei handelt es sich um ein Programm, das am *Lehrstuhl für Künstliche Intelligenz in Dortmund* entwickelt wurde. Das Programm ist eine intuitive Umgebung zur Einbindung sowie Verarbeitung von Datensätzen. Data-Mining-Verfahren können anhand dieser Datensätze in YALE trainiert und schließlich auf weitere Datensätze angewandt werden. Da YALE bis dato nicht für NER-Aufgaben benutzt wurde, wird vom Diplomanden zusätzlich zu den CRF-Operatoren noch ein NERpreprocessing-Operator implementiert, der Texte um NER-spezifische Merkmale erweitert und als Datensatz in YALE einliest. Anhand der erwähnten CRF-Operatoren können diese dann untersucht werden.

Die Datensätze, die untersucht werden sollen, werden in Kapitel 11 mit ihren Besonderheiten vorgestellt. Einer der Datensätze (JNLPBA) wird ganz besonders gründlich untersucht. Um Vergleichsmöglichkeiten zu dem vom Diplomanden entwickelten Verfahren zu haben, werden in Kapitel 11.2 Verfahren vorgestellt, die bereits erfolgreich auf den JNLPBA-Datensatz angewandt wurden. Besonderes Augenmerk wird auf die benutzten Merkmale gelegt, die später in Kapitel 13.7 mit denen vom Diplomanden verglichen werden.

Bevor die Versuche des Diplomanden beschrieben werden, wird in Kapitel 12 Grundsätzliches zu den Versuchen erläutert.

Kapitel 13 umfasst alle Versuche, die mit dem vom Diplomanden entwickelten CRF-System durchgeführt wurden. Wie bereits erwähnt, wird zu Beginn der JNLPBA-Datensatz untersucht, bevor anschließend ein anderer Datensatz (CoNLL) untersucht wird. Auf den JNLPBA-Daten werden die in der Implementierung vorhandenen Potentialfunktionen untersucht. Danach werden die vom Diplomanden entwickelten Potentialfunktionen untersucht, bevor beide Arten miteinander kombiniert werden. Es besteht die Erwartung, dass Kombinationen aus vorhandenen und eigenen Potentialfunktionen bessere Ergebnisse liefern, als Versuche, die ausschließlich die vorhandenen Potentialfunktionen enthalten. Um die Versuche schnell abzuschließen, wurden die Versuche vorerst auf einem Teil (*Sample*) des Trainingsdatensatzes durchgeführt. Abschließend wird der Versuch, der die besten Ergebnisse liefert, auf dem kompletten Trainingsdatensatz angewandt.

Im folgenden Kapitel werden die Ergebnisse des vom Diplomanden entwickelten Systems mit den in Kapitel 11.2 vorgestellten Verfahren verglichen und mögliche Gründe für unterschiedliche Ergebnisse gegeben.

Einige der Potentialfunktionskombinationen werden dann in Versuchen über den CoNLL-Daten benutzt. Auch hier werden die Ergebnisse mit denen anderer Systeme, die auf den Datensatz angesetzt wurden, verglichen.

Das letzte Kapitel (14.1) bietet einen Rückblick auf die erzielten Ergebnisse. Zudem werden mögliche zukünftige Entwicklungsmöglichkeiten für das entwickelte CRF-System in einem Ausblick dargestellt (14.2).

2 Named entity recognition

2.1 Einleitung

Einleitend soll nun beschrieben werden, worum es sich bei der NER handelt. NER ist, wie in Kapitel 1.1 bereits erwähnt wurde, in den Bereich der IE einzuordnen. Auch wurde bereits erwähnt, dass das Ziel der NER ist, vorher definierte NE in Texten zu erfassen. Jedoch ist eine NE oft mehr als ein einfacher Name oder ein Ort, den das NER-System anhand eines Lexikon-Nachschlags finden könnte. Man sollte eine NE vielmehr als eine Art semantisch zu erfassende Einheit ansehen.

Als Gegenbeispiel zu einem NER-System kann eine primitive Suchmaschine angesehen werden, die ausschließlich versucht, die zu suchenden Worte in Texten zu finden. Die sogenannte zeichenbasierte Suche von Suchmaschinen nach in einem Text vorkommenden Zeichenketten ist sehr rudimentär und vernachlässigt wichtige semantische Zusammenhänge. Wo der zeichenbasierte Ansatz noch versagt, da eine bestimmte Zeichenkette einfach nicht existent ist, können Verfahren, die zur semantischen Erschließung von Texten dienen, vielleicht trotzdem Informationen gewinnen. Sucht man zum Beispiel in einer Suchmaschine nach „Mercedes Benz“, so werden einem nur Seiten angezeigt, die wirklich auch diesen Suchbegriff enthalten. Seiten, die zum Beispiel den Begriff „die Silberpfeile“ enthalten und auf die Oldtimer-Rennwagen von Mercedes hindeuten, werden nur durch semantische Verfahren als Hinweis auf „Mercedes Benz“ erkannt. Ein gutes NER-Verfahren sollte solch ein semantisches Erschließungsverfahren beinhalten, um durch verschiedenste Merkmale darauf zu schließen, dass beispielsweise ein „Silberpfeil“ auch ein Auto sein kann.

Der Begriff der NER tauchte erstmals in der Aufgabenstellung zur *Message Understanding Conference 6* (MUC-6) auf (vergleiche [14]). Diese Konferenzen wurden ins Leben gerufen, um neue und bessere Verfahren zur IE zu entwickeln.

Das Interessante an den MUC ist jedoch nicht die Konferenz selbst sondern der Wettbewerb, der im Vorfeld jeder der MUC stattfindet. Bei diesem Wettbewerb müssen die Teilnehmer Systeme entwickeln, die aus Texten bestimmte Informationen extrahieren. Die Art der Informationen, die zu erfassen sind, sind zwar unterschiedliche, jedoch gibt es immer einen umfassenden Kontext, sozusagen ein Thema, das übergeordnet für die Informationen ist.

So gab es bei der zweiten MUC zehn Informationen, die zu einem Kontext zu extrahieren waren. Die Texte der ersten beiden MUC hatten nautische Sichtungen und Gefechte zum Thema. Informationen, die zu erfassen waren, waren zum Beispiel Typ des Vorfalls, Zeit und Ort des Vorfalls und einige weitere.

Die Regelmäßigkeit (anfangs alle zwei Jahre, dann jährlich) sowie der Wettbewerbscharakter der MUC sorgten immer wieder für Fortschritte und Innovationen im Bereich der IE. Um eine Unabhängigkeit vom Kontext zu erreichen und trotzdem noch Informationen zu erhalten, die mit einer hohen Genauigkeit ermittelt werden, wurde für die MUC-6 das Teilgebiet NER geschaffen. Ziel hierbei war es, alle Personen, Organisationen und geographischen Orte im Text zu markieren. Man erhält durch NER

somit eine kontextunabhängige Vorverarbeitung des Textes, der danach anhand auf den Kontext abgestimmter Systeme weiterverarbeitet werden kann. Häufig ist es jedoch so, dass NER-Systeme trotzdem noch kontextabhängig trainiert werden.

2.2 Einordnung der NER in den Bereich KDD

KDD ist die Bezeichnung für die Extraktion bisher unbekannter, möglichst interessanter Informationen aus (oft großen) Datenbanken oder Datensätzen. *Data Mining* (DM) beschreibt wiederum die Methoden der KDD, die mittels möglichst effizienter Algorithmen automatisch Muster aus den betrachteten Daten liefern.

Es gibt verschiedene Verfahren des DM, wobei alle Verfahren in zwei Gruppen einzuteilen sind. Auf der einen Seite sind die beschreibenden Verfahren, und auf der anderen Seite gibt es die vorhersagenden Verfahren. Beschreibende Verfahren erklären die kausalen Zusammenhänge der Daten. *Clustering* ist zum Beispiel ein solches Verfahren, das ähnliche Daten zu Gruppen zusammenfasst. In Datenbanken sind Daten praktisch gesehen durch ihre Attribute definiert. Das beim Clustering beschriebene Zusammenfassen ähnlicher Daten könnte beispielsweise gerade die Daten zusammenfassen, die in einem bestimmten Attribut ähnliche Werte aufweisen.

Vorhersagende Verfahren ordnen die Daten vorher festgelegten Klassen zu. Diese Verfahren nennt man daher auch *Klassifikation*. Allerdings ist noch zu definieren, welche Daten in welche Klasse einzuordnen sind. Das Ziel der Klassifikation ist es, Funktionen zu entwickeln, die anhand der Struktur der Daten erkennen können, in welche Klasse sie einzuteilen sind. Damit diese Funktionen jedoch erzeugt werden können, benötigt die Klassifikation Beispieldaten, die bereits korrekt klassifiziert sind. In der KDD werden diese Beispieldaten Trainingsdatensatz genannt. Dieser Datensatz ist – wie erwähnt - bereits klassifiziert und sollte möglichst die gleichen Merkmale im gleichen Verhältnis aufweisen wie der Datensatz, der noch zu klassifizieren ist. Die DM-Verfahren, die klassifizieren sollen, werden dann mithilfe des Trainingsdatensatzes trainiert und können anschließend den nicht klassifizierten Datensatz klassifizieren. Dieser nicht klassifizierte Datensatz wird auch Testdatensatz genannt.

Man kann im Grunde NER als eine Art des KDD bezeichnen. Auch bei der NER behandelt man einen „Datensatz“. Dieser „Datensatz“ liegt allerdings nicht in einer Datenbank sondern als Text vor. Und das Verfahren, das man zur NER benutzt – in dieser Arbeit CRF - ist dann ein Klassifikationsverfahren.

Die vorgegebenen Klassen sind bei der NER die NE. Auch bei der NER benutzt man einen Trainings- sowie einen Testdatensatz (-text).

Eine Methode zur NER – in diesem Fall CRF – und alle ihre Bestandteile soll im weiteren Verlauf dieser Arbeit nur noch als *System* bezeichnet werden. Nachdem ein Klassifiationssystem trainiert wurde, wird dieses auf den Testdatensatz angewandt. Dieser Testdatensatz ist möglichst disjunkt zum Trainingsdatensatz, da nur auf diese Weise das gelernte Verfahren korrekt bewertet werden kann. Des Weiteren sollten dem Verfahren natürlich die vorherzusagenden Klassen, falls diese im Testdatensatz

vorhanden sind, verborgen bleiben. Erst zur Bewertung des Verfahrens werden dann die vorhergesagten Daten mit den „echten“ verborgenen Daten verglichen. Bei Klassifikationsverfahren auf Datenbanken fehlt beim Testdatensatz zum Beispiel die Spalte des vorherzusagenden Eintrags in der Datenbank.

Enthielt der Datensatz bei der NER im Training noch die NE-Markierungen (*NE-tags*), so enthält der Testdatensatz schlicht den Text, der vom System zu markieren ist. Als Beispiel sei nun ein Text einschließlich seiner NE-tags gegeben:

<PER>	<O>	<O>	<LOC>
Felix	geht	nach	Hamburg

Abbildung 1: Beispieltext inklusive Markierungen

NE sind in diesem Fall unter anderem PERSON, was im Text mit <PER> markiert wird, und LOCATION, was im Text mit <LOC> markiert wird. Abbildung 1 kann als – wenn auch sehr kleiner – Trainingsdatensatz angesehen werden. Ein trainiertes Pseudo-System kann man sich nun so vorstellen, dass es die Worte „Felix geht nach“ als Indiz dafür ansieht, dass das darauf folgendes Wort eine LOCATION ist. Somit würde in dem Testdatensatz in Abbildung 2 das Wort „Dortmund“ als LOCATION markiert werden.

Felix	geht	nach	Dortmund

Abbildung 2: Zu markierender Beispieltext

Problematisch an NER ist, dass man einerseits auf Lexikoneinträge zurückgreifen kann, andererseits jedoch gezwungen ist, die Semantik des zu untersuchenden Textes zu verstehen. Das soeben benutzte Wort „Dortmund“ galt in diesem Zusammenhang als LOCATION, steht es aber in dem Kontext „Kronen Brauerei Dortmund“, so ändert sich die Situation. Intuitiv gesehen ist es zwar immer noch ein Indikator für eine Ortsbezeichnung, im NER-Zusammenhang jedoch gelten die drei Worte zusammen als ein (Firmen-)NAME. Allerdings impliziert die oben genannte Regel, dass „Felix geht nach“ ein Indiz für eine LOCATION ist, das Verständnis der Semantik. Im Deutschen ist „gehen nach“ nämlich – auch wenn es umgangssprachlich manchmal benutzt wird - kein Indiz für einen (Firmen-)NAMEN. In 2.3 soll auf die auftretenden Probleme und ihre Lösung eingegangen werden.

2.3 Probleme der NER

Probleme treten bei der NER auf, wenn im Text unbekannte oder mehrdeutige Wörter vorkommen. Als unbekannt sieht man Wörter an, die nicht im Trainingsdatensatz vorkommen und auch nicht in vom Verfahren benutzten Lexika zu finden sind. Mehrdeutige Wörter sind zwar bekannt, jedoch sind mehrere Bedeutungen für das Wort – mehrere NE im Bereich des NER – vorhanden. Die korrekte Bedeutung ergibt sich erst, wenn man den Zusammenhang betrachtet, in dem sie benutzt werden. Als Beispiel kann hier das deutsche Wort „Bad“ angeführt werden, das einerseits ein Teil eines Stadtnamens sein kann, andererseits kann es auch als Kurzform für „Ba-

dewanne“ oder „Badezimmer“ benutzt werden. (Siehe hierzu und zur NER allgemein auch [29].)

2.3.1 Interne und externe Evidenz

In [22] wurden zur Erläuterung dieser Tatsachen die Begriffe interne sowie externe Evidenz definiert, die zwei Möglichkeiten beschreiben, die Semantik eines Wortes zu erfassen.

Mit interner Evidenz sind Informationen gemeint, die sich aus dem Wort selbst ergeben. Hierzu gehören Informationen, die Lexikon-Anfragen zu diesem Wort liefern, als auch Informationen, die sich durch die Beschaffenheit des Wortes (Großschreibung, reguläre Ausdrücke, Länge etc.) ergeben. Weiterhin können auch Teile des Wortes zum Informationsgewinn beitragen – das Suffix „-burg“ weist zum Beispiel meistens auf einen Ortsnamen hin.

Externe Evidenz hingegen sind Informationen, die sich durch das Wortumfeld - seinen Kontext – ergeben. Der Text „der sibirische Kaufmann“ weist zum Beispiel darauf hin, dass das folgende Wort, was in diesem Fall „Oljuschin“ und zudem unbekannt sei, ein Personenname ist.

Externe und interne Evidenz hilft also in dem Fall, dass man ein unbekanntes oder mehrdeutiges Wort vorfindet, weiter. Um jedoch ein gutes NER-System zu entwickeln, muss man die interne als auch die externe Evidenz betrachten, da nur das Zusammenspiel beider eine korrekte Klassifizierung verspricht. Denn selbst bei dem oben betrachteten Beispiel für die externe Evidenz greift man auf die interne Evidenz des folgenden Wortes zu, da schon die Tatsache, dass das Wort „Oljuschin“ unbekannt ist, der internen Evidenz zuzuschreiben ist.

2.3.2 Generalisierung

[29] stellt zur Vermeidung von Problemen bei der NER das Prinzip der Generalisierung dar, das dazu dient, Worte und Texte zu verallgemeinern und somit von bekannten Wortformen auf unbekannte schließen zu können. Generalisierung entspricht dem menschlichen Vermögen, unbekannte Worte aus dem Kontext oder auch dem Wort selbst zu erschließen, indem aus bekannten Wortteilen oder Kontexten auf die Bedeutung geschlossen wird. Hierbei werden zwei Arten der Generalisierung unterschieden:

- Modellorientierte Generalisierung
- Datenorientierte Generalisierung

Bei der modellorientierten Generalisierung werden Theorien und Modelle benutzt. Bei der NER sind dies zum Beispiel sprachwissenschaftliche Theorien. Die datenorientierte Generalisierung betrachtet hingegen ausschließlich die in den Daten vorhandenen Merkmale.

Modellorientierte Generalisierung

Die modellorientierte Generalisierung ist in die morphologische sowie die syntaktische Generalisierung zu unterteilen. Bei der morphologischen Generalisierung werden Worte auf ihren Stamm reduziert und somit verallgemeinert. Die syntaktische

Generalisierung findet die Wortklassen zu einzelnen Worten. Diese Klassen werden im Englischen als *part-of-speech* (POS) bezeichnet und beinhalten die grammatikalische Funktion des betrachteten Wortes.

Datenorientierte Generalisierung

Die datenorientierte Generalisierung versucht, nur anhand des betrachteten Wortes die morphologischen oder syntaktischen Eigenschaften zu erfassen. In [1] werden zum Beispiel so genannte „word features" benutzt, die aussagen, ob ein Wort großgeschrieben ist, Sonderzeichen enthält oder unter anderem ausschließlich aus Großbuchstaben besteht. Schon hierdurch werden linguistische Eigenschaften dargestellt.

Zudem kann die Wortform, wie schon erwähnt, zur Erfassung morphologischer Eigenschaften dienen. Zum Beispiel weisen gewisse Prä- und Suffixe im Deutschen auf Verben beziehungsweise Nomen hin. So deuten auf „-t", „-st" oder „-en" endende Worte auf Verben hin, wohingegen „-ung", „-ion" und „-burg" auf Nomen hinweisen. Natürlich sind diese Hinweise nicht eindeutig und müssen jeweils kombiniert betrachtet werden. So genannte *Stemming-Verfahren* machen sich diese Eigenschaften zunutze und reduzieren Worte durch Abtrennen von Prä- und/oder Suffixen auf ihren Wortstamm. Allerdings werden nur Prä- und Suffix abgetrennt, die als solche erkannt werden, so dass meist ein korrekter Stamm bleibt. Der Nachteil an diesen Verfahren ist, dass die Informationen, die die Prä- und Suffixe enthalten, verloren gehen. Zwar erhält man den Wortstamm, aber man verliert unter Umständen genauso wichtige Informationen wie zum Beispiel den Numerus des Wortes. Kann man mehrere Merkmale parallel verwalten, so könnte man natürlich das Wort und den Wortstamm weiterverwenden.

Einen Kompromiss bildet die Zerlegung des Wortes in N-Gramme. Hierbei wird das Wort in (M-N)+1 Teilworte – wobei M die Länge des Wortes ist - der Länge N zerlegt, indem beginnend mit dem N-langen Präfix jeweils das nächste N-Gramm betrachtet wird, bis man das N-lange Suffix erreicht. Durch dieses Verfahren behält man einerseits die Prä- und Suffixe und erhält durch die N-Gramme zudem einen Pseudo-Stamm, da übereinstimmende N-Gramme bei unbekannten und bekannten Worten auf denselben Stamm hindeuten.

Die datenorientierte Generalisierung syntaktischer Eigenschaften kann auch mittels N-Grammen funktionieren. Allerdings bildet man hier N-Gramme über den Worten des Textes, so dass man jeweils ein Fenster von zum Beispiel vier Worten betrachtet. Somit sammelt man Worte oder Wortsequenzen, die auf bestimmte Namen hinweisen. „Prof.", „Dr.", „Herr" und „Frau" weisen zum Beispiel auf einen Personennamen hin.

Semantische Kategorien

[29] weist zudem darauf hin, dass man so genannte semantische Kategorien bilden kann, um Worte zu generalisieren. Anstatt die Beobachtungen, dass nach „Maurer", „Maler", „Tischler" ein Personenname folgt, zu registrieren, kann man semantische Kategorien formulieren, die Beobachtungen verallgemeinern. So lautet die semantische Kategorie in diesem Fall „Berufsbezeichnung". Somit erhält man eine allgemei-

nere (generalisierte) Beobachtung, die besagt, dass nach Worten der semantischen Kategorie „Berufsbezeichnung“ ein Personenname folgt.

2.4 Bewertungsmaße

Bewertungsmaße sind Funktionen, die die Güte eines KDD-Systems darstellen. Sie werden einerseits benutzt, um ein System, das auf einen Testdatensatz angewandt wurde, zu bewerten. Andererseits dienen sie dazu, das Training eines KDD-Systems zu unterstützen, indem so trainiert wird, dass das System anhand der Bewertungsmaße optimiert wird.

Für NER-Aufgaben haben sich drei Bewertungsmaße etabliert, die jedoch im Grunde genommen ziemlich ähnlich sind:

- *Precision*
- *Recall*
- *F-Measure*

Precision ist ein Maß, das die Korrektheit der gesetzten NE-tags bewertet. Precision ist formal:

Precision = (Anzahl NE korrekt) / (Anzahl NE gefunden)

Recall bewertet zudem den Umfang der gefundenen NE-tags und ist formal: *Recall = (Anzahl NE korrekt) / (Anzahl NE im Text)*

F-Measure verknüpft beide Bewertungsmaße und schreibt sich formal:

*F-Measure = (2*Precision*Recall) / (Precision+Recall)*

Man kann nun einerseits die Bewertungsmaße für jede spezielle NE bilden, indem man jeweils nur diese spezielle NE betrachtet. Das Maß Precision für die NE PERSON ist somit die Anzahl der vom System korrekt als NE PERSON markierten Worte geteilt durch die Anzahl der vom System als NE PERSON markierten Worte. Andererseits ist es auch möglich, die Maße kumuliert zu berechnen. Hierzu betrachtet man einfach alle NE zusammen. Das Maß Precision ergibt sich dann, indem man die Anzahl der vom System korrekt als (irgendeine) NE markierten Worte durch die Anzahl der vom System als (irgendeine) NE markierten Worte teilt.

In [2] wird erwähnt, dass in den MUC sowie den *Multilingual Entity Tasks* (MET) darauf Wert gelegt wird, dass auch mehrwortige NE korrekt erkannt werden. Mehrwortige NE sollen also nur als korrekt angesehen werden, wenn sie komplett sind. Daher muss nicht nur die Bezeichnung eines NE korrekt sein, sondern auch seine zwei Grenzen (Anfang, Ende). Ein Evaluierungswerkzeug zur Bestimmung der soeben vorgestellten Maße, das auch die korrekte Bewertung mehrwortiger NE vornimmt, wird im Laufe dieser Diplomarbeit entwickelt.

Das Markieren von Texten bezeichnet man in Anlehnung an die NE-tags auch als *tagging*. Es gibt eine tagging-Art, die besonders für mehrwortige NE sinnvoll ist, und nun vorgestellt werden soll.

2.4.1 iob-tagging

iob ist die Abkürzung für inner, outer und beginning. Der Beginn einer NE wird in besonderer Weise gekennzeichnet: zusätzlich zur eigentlichen Markierung (zum Beispiel PERSON) wird noch eine Kennzeichnung für den Beginn einer NE hinzugefügt („B-„), so dass die neue Markierung dann B-PERSON ist. Dies weist darauf hin, dass eine NE mit Markierung PERSON vorliegt. Jedoch ist keinesfalls sicher, dass das vorangehende Wort nicht die Markierung PERSON hat, da auch mehrere gleich benannte NE hintereinander vorkommen können. Zudem kann man auch nicht davon ausgehen, dass das folgende Wort auch die Markierung PERSON hat, da selbst einwortige NE mit dem beginning-tag gekennzeichnet werden. Alle folgenden Markierungen in einer mehrwortigen NE werden mit der Kennzeichnung „I-„ versehen. Bei diesen Markierungen weiß man also zusätzlich zum eigentlichen NE-tag noch, dass das vorangegangene Wort dasselbe NE-tag haben muss. Alle Worte, die keiner NE angehören bekommen das outer-tag, werden also mit „O“ markiert.

2.5 Bekannte NER-Verfahren

Es gibt zwei grundsätzliche Arten, der automatisierten NER. Einerseits sind dies regelbasierte Verfahren, die den zu untersuchenden Text anhand von sozusagen „von Hand erstellten“ Regeln bearbeiten. Andererseits gibt es die Verfahren des maschinellen Lernens, die versuchen, die komplexe Arbeit des Regelschreibens zu automatisieren, oder aber vorhandene Regelsätze zu optimieren. Einzig die Verfahren des maschinellen Lernens sind in den Bereich der aktuellen Wissensentdeckung einzuordnen, da sie im Vergleich zu regelbasierten Verfahren mit viel weniger Expertenwissen auskommen.

2.5.1 Regelbasierte Verfahren

Zu Beginn der NER war dieser Bereich eher der Linguistik als der Informatik zuzuordnen. Somit wurden bei den ersten automatischen Verfahren Regeln, die zur Kategorisierung der NE dienen, ausschließlich von Hand eingegeben. Diese Aufgabe war natürlich einerseits extrem langwierig und andererseits nur für Experten – also Linguisten – durchführbar. Anfängliche Systeme wie zum Beispiel SPARSER, der in [22] vorgestellt wird, beruhten auf endlichen Automaten oder kontextfreien Grammatiken (*context free grammar* (CFG)). Das System SPARSER benutzte zur NER drei Schritte, die von so gut wie allen regelbasierten Ansätzen gleichfalls benutzt wurden:

1. Erkennung der Worte, die NE darstellen
2. Kennzeichnung der NE als Person, Firma oder Ortsangabe
3. Speicherung der NE in einem Diskursmodell

Vor SPARSER wurde NER ausschließlich über Listenverfahren realisiert, die NE ausschließlich über eine Suche in Lexika ausfindig machen konnten. Die in 2.3.1 beschriebene externe Evidenz hingegen vermeidet erstmals Fehler, die durch mehrdeutige Worte bei den Listenverfahren auftauchen konnten.

Die Frage, ob CFG oder endliche Automaten zu benutzen sind, beantwortete [13], indem die Nachteile von CFG dargestellt werden, wodurch endliche Automaten zu favorisieren sind:

- CFG sind viel zu langsam
- CFG, die global konditioniert sind, führen zu lokalen Fehlern
- CFG auf neue Kontexte einzustellen ist sehr schwer

2.5.2 Verfahren des maschinellen Lernens

Die Verfahren des maschinellen Lernens versuchen anhand eines Beispieltextes ein Modell zu lernen, was die Auszeichnung späterer Texte (Testdatensatz) übernehmen soll. Ausschlaggebend für diese Entwicklung war die Tatsache, dass der Aufwand des Regelschreibens einfach viel zu groß war. Auch entstand die Idee, sich nicht ausschließlich auf linguistische Weise dem Problem zu nähern, sondern andere Verfahren zu benutzen und somit automatisch gewisse linguistische Eigenschaften auszuarbeiten.

Transformationsbasiertes Regellernen

Erste Verfahren, wie das in [1] vorgestellte, benutzten zwar immer noch von Hand erzeugte Regeln, jedoch dienten diese nur zur Manipulation schon getroffener Entscheidungen. Basierend auf einer Art POS-tagger machte das System triviale Aussagen ob ein Wort ein NE war oder nicht, indem es zum Beispiel Worte, die schon im Trainingskorpus als NE vorkamen wieder dementsprechend markierte. Der Regelsatz, der dann benutzt wurde, enthielt Regeln der Form

Ändere Bezeichnung X in Y, wenn aktuelle Wortart W und die folgende Bezeichnung Z ist

Es wurden nun für X, Y, W und Z alle möglichen oder vorhandenen Kombinationen ausprobiert und über dem Trainingsdatensatz getestet. Mögliche Regeln wurden als Kandidaten übernommen und evaluiert. Brachte eine dieser Transformationsregeln eine Verbesserung ein, so wurde sie in das System aufgenommen.

Diesem Verfahren sind aufgrund der vielen möglichen Kombinationsmöglichkeiten bei den Transformationsregeln natürlich Grenzen gesetzt. Andererseits waren die automatisch generierten Regeln im Vergleich zu von Hand erzeugten Regeln noch so schlecht, dass für das System, das schließlich zur Teilnahme an der MUC-6 entwickelt wurde, ausschließlich von Hand entwickelte Regeln benutzt wurden.

Hidden Markov Modelle

Die HMM, die noch ausführlich in 3.2 vorgestellt werden, waren das erste maschinelle Lernverfahren, das annähernd die Ergebnisse regelbasierter Ansätze erreichen konnte ([2]). Das Verfahren von [2] benutzt neben Merkmalen des Wortes auch das Bigramm des vorangestellten Wortes und arbeitet somit mit einer gewissen externen Evidenz.

[43] hingegen übertreffen alle Ergebnisse für MUC-6 und MUC-7. Im Vergleich zu [2] benutzen sie ein erweitertes Tagset, das nicht nur die NE enthält sondern auch darstellt, ob es sich bei mehrwortigen NE um den Beginn, das Ende oder ein NE in der

Mitte handelt. Des Weiteren werden durch bestimmte Beobachtungen semantische Kategorien gebildet, indem zum Beispiel Namenslisten durchsucht werden, oder indem überprüft wird, ob ein Wort im aktuellen Text schon als bestimmte NE klassifiziert wurde.

Maximum Entropy Markov Modelle

Zur Definition und Vorgehensweise der MEMM sei auf 3.3 verwiesen. MEMM sind im Gegensatz zu HMM darauf ausgelegt, mehrere Beobachtungen (Evidenzen) für ein Wort aufzunehmen.

So werden in [3], einem der ersten Systeme, die MEMM benutzten, eine Vielzahl von Merkmalen benutzt. Unter anderem sind dies binäre Merkmale, die anzeigen, ob ein Wort großgeschrieben wird oder Sonderzeichen und ähnliches enthält. Des Weiteren werden so genannte lexikalische Merkmale benutzt, die den Kontext des aktuell betrachteten Wortes beachten. Die Absatz-Merkmale liefern Informationen über den Textbereich, in dem sich das Wort befindet (Titel, Text, Datumsangabe). Die Merkmale werden durch ein Lernverfahren gewichtet und somit bewertet. Diese Tatsache ermöglicht die Nutzung von „Regeln“, die in regelbasierten Verfahren als zu riskant verworfen worden wären. So wäre zum Beispiel das Wort „Storm“ in regelbasierten Verfahren nicht als Vorname angesehen worden. Mit MEMM ist dieses Vorgehen möglich, da die Trainingsgewichte die Semantik des Wortes eher in seiner Bedeutung als Sturm denn als Vornamen deklarieren werden. Ein weiterer Vorteil ist, dass die bis jetzt vorgestellten Merkmale in keinster Weise kontextgebunden sind. Somit ist die bisherige Vorgehensweise auf jeden möglichen Kontext portierbar.

Natürlich werden für bestimmte Kontexte noch zusätzliche Merkmale benutzt. So wurden außerdem Lexikon-Merkmale als auch Merkmale externer Systeme eingebunden. Lexikon-Merkmale bestanden hierbei aus der Suche in vorher erstellten Lexikon-Listen. Als Merkmale externer Systeme wurden die Ausgaben von drei verschiedenen regelbasierten Systemen (siehe auch [13]) benutzt. Die Ergebnisse lagen für die MUC-7 bei einem F-Measure von 97,12%, was dem menschlichen Vermögen der Textklassifikation annähernd entspricht.

Support Vector Machine

Es ist ebenfalls möglich, Support Vector Machines(SVM) als NER-Verfahren anzuwenden. Diese Verfahren können eine große Anzahl Attribute verarbeiten und sind robust gegenüber *overfitting* (siehe hierzu auch 13.4). Allerdings lösen SVM nur binäre Klassifikationsprobleme, indem eine Hyperebene in den Raum gelegt wird, der durch die Beispiele aufgespannt wird und in seinen Dimensionen der Anzahl der Attribute entspricht. Diese Hyperebene wird so gelegt, dass sie die Beispiele bestmöglich separiert und somit klassifiziert. Sofern die Beispiele nicht linear separierbar sind, können so genannte Kernel-SVM die Hyperebene in höherdimensionale Räume abbilden und somit die Beispiele separieren. Die Tatsache, dass nur binäre Klassifikationsprobleme bearbeitet werden können, führt dazu, dass für jede NE eine SVM existieren muss: somit klassifiziert man zum Beispiel für jede NE, ob ein Wort diese NE erhält oder nicht.

NER-Verfahren mittels SVM waren das am meisten benutzte Verfahren der JNLPBA04 ([17]). Allerdings wurden Verfahren, die ausschließlich SVM benutzten von vielen Verfahren übertroffen. Unter anderem waren SVM mit HMM, MEMM und CRF besser (vergleiche Kapitel 11.2).

Weitere maschinell basierte Lernverfahren

Des Weiteren seien noch Entscheidungsbäume sowie ähnlichkeitsbasiertes Lernen genannt. Allerdings spielen diese Verfahren kaum eine große Rolle, da sie eklatante Nachteile zu anderen Verfahren im Bereich der NER aufweisen.

Entscheidungsbäume sind nicht für große Merkmalsräume konzipiert und dienen daher aus oben genannten Gründen kaum zur NER.

Auch ähnlichkeitsbasierte Lernverfahren - auf NER-Aufgaben angewandt - arbeiteten, obwohl sie große Merkmalsräume verarbeiten können, oftmals schlecht. [29] führt dies auf die Probleme, die ähnlichkeitsbasierte Lernverfahren mit redundanten Merkmalen haben, zurück.

3 Entstehung der Conditional Random Fields

Da es bei der NER um die Untersuchung von sequentiellen Daten (Texte sind sequentielle Daten) geht, muss man Verfahren anwenden, die mit dieser Art von Daten umgehen können.

Die Entwicklung von CRF entsprang der steten Weiterentwicklung der HMM.

Um diese Entwicklung nun vollständig aufzuzeigen, werden folgende Verfahren für die NER dargestellt:

- HMM
- MEMM
- CRF – diese sind ein Spezialfall der MRF

Als erstes müssen jedoch *Markov Prozesse* (MP) definiert werden, da sie grundlegend für HMM sind.

3.1 Markov Prozesse

MP dienen dazu, konkrete oder reale Vorgänge zu beschreiben. Diese Vorgänge haben zudem einen gewissen Zeitbezug. Als Beispiel für einen MP kann man das Wetter beziehungsweise die Vorhersage desselben ansehen. Man geht beispielhaft davon aus, dass es drei verschiedene Wetterarten gibt:

- Regnerisch
- Bewölkt
- Sonnig

In diesem Fall stellen die Wetterarten die Zustandsmenge des MP dar, die ein MP nach Definition hat. Aus dieser Zustandsmenge ist zu jedem Zeitpunkt jeweils ein Zustand „aktiv“ – man könnte diesen „aktiven“ Zustand als das aktuelle Wetter bezeichnen. MP stellen also für einen Zeitraum die Wetter an jedem Zeitpunkt dieses Zeitraumes dar.

Für MP gilt zudem die *Markov Eigenschaft*. Das bedeutet, dass der aktuelle Zustand nur von einer gewissen Menge (Vorgänger-)Zustände abhängig ist. Hierdurch ist der Zeitbezug gegeben, denn die Vorgängerzustände entsprechen in der Vergangenheit liegende Zustandsvorkommen. In dem Beispiel resultiert aus der Markov Eigenschaft, dass das aktuelle Wetter vom Wetter der vorangegangen Zeitpunkte (als Zeitpunkte nimmt man ab jetzt Tage an) abhängt.

Bezieht man für die Vorhersage des Wetters an einem bestimmten Tag nur das Wetter des Vortages mit ein, so handelt es sich bei dem MP um einen MP erster Ordnung. Werden die Wetter von n vorangegangenen Tagen miteinbezogen, so handelt es sich um einen MP der Ordnung n.

Die Übergänge von einem Zustand zum Folgezustand werden Transitionen genannt.

Um bei dem einfach Fall eines MP erster Ordnung zu bleiben: hat ein MP m Zustände, so gibt es gleichzeitig m^2 Transitionen, da von jedem der m Zustände m Transiti-

onen zu den (m-1) anderen Zuständen und dem aktuellen Zustand gehen. Von jedem Zustand existieren also Transitionen zu allen Zuständen der Zustandsmenge. Allerdings sind im Allgemeinen und im Wetterbeispiel im Besonderen ja einige Transitionen wahrscheinlicher als andere, was dazu führt, dass Transitionen Wahrscheinlichkeiten erhalten. Diese Wahrscheinlichkeiten werden Eintrittswahrscheinlichkeiten genannt.

Die Transitionen und ihre Eintrittswahrscheinlichkeiten werden in einer Transitionsmatrix A wie in Abbildung 3 dargestellt. Die dargestellte Transitionsmatrix ist die eines MP der ersten Ordnung. Da hierbei die aktuellen Zustände nur von den Vorgängerzuständen abhängig sind, werden auch nur die Transitionen zwischen aktuellem und Vorgängerzustand dargestellt. Die Zeilen enthalten hierbei die Eintrittswahrscheinlichkeiten für eine Transition vom Vorgänger zum aktuellen Zustand.

weather yesterday	weather today: Sun	Cloud	Rain
Sun	0.5	0.25	0.25
Cloud	0.375	0,25	0.375
Rain	0.125	0,5	0.375

Abbildung 3: Beispiel einer Transitionsmatrix

In diesem Beispiel besteht also eine 50%ige Wahrscheinlichkeit, dass es sonnig wird, sofern es am Vortag schon sonnig war.

Da es sich hier um Wahrscheinlichkeiten handelt, summieren sich die ausgehenden Wahrscheinlichkeiten eines Zustandes zu 1 auf. Somit summieren sich also die Zeilen einer Transitionsmatrix zu 1 auf.

Um einen MP weitergehend zu beschreiben, muss man auch seinen Anfangszustand definieren, der durch den $\vec{\pi}$-Vektor gegeben ist. Dieser Vektor gibt die Wahrscheinlichkeiten an, mit der der MP zum Zeitpunkt 0 in einem der Zustände ist.

Hier wäre ein $\vec{\pi}$-Vektor wie in Abbildung 4 denkbar.

Sun	Cloud	Rain
1.0	0.0	0.0

Abbildung 4: $\vec{\pi}$-Vektor

Somit ist ein MP definiert durch:

- seine Zustände (hier sonnig, bewölkt und regnerisch)
- seinen $\vec{\pi}$-Vektor
- seine Transitionsmatrix

Andererseits ist jedes System, was durch diese Punkte beschrieben werden kann, ein MP. (Vergleiche hierzu [16])

3.2 Hidden Markov Modelle

Die HMM waren eines der ersten Verfahren, die zur Untersuchung von sequentiellen Daten angewandt wurden (vergleiche [27] und [28]).

Ihren Namen tragen diese Modelle aus zwei simplen Gründen:

- Das Modell besteht aus MP.
- Diese Prozesse sind versteckt (hidden).

Da die MP schon definiert sind, muss nun auf die Versteckttheit der Zustände eingegangen werden. Es gibt Situationen, in denen MP einfach nicht ausreichen. Im Grunde entsprechen HMM den MP, allerdings sind die Zustände, die schon bei den MP definiert wurden, versteckt. Es gibt jedoch zusätzliche Zustände, die beobachtbar sind. Die versteckten Zustände sind deshalb versteckt, da man die Regeln, nach denen sich die versteckten Zustände ändern, nicht kennt.

Als Beispiel kann man hier wieder das Wetter anführen. Wenn man sich einen Beobachter in einem abgeschlossenen System (zum Beispiel Keller) vorstellt, dann seien die beobachtbaren Zustände dadurch gegeben, dass ein Mitarbeiter, der den Keller betritt,

- einen Regenschirm
- keinen Regenschirm

dabei hat. Der Beobachter kann also nur versuchen, anhand der beobachtbaren Zustände auf die versteckten Zustände zu schließen, die in diesem Beispiel wieder durch

- Regnerisch
- Bewölkt
- Sonnig

gegeben sind.

HMM sind generative Modelle, da die versteckten Zustände die Zustände, die beobachtbar sind, generieren. HMM dienen im Grunde dazu, aus einer Menge beobachtbarer Zustände (*Beobachtungssequenz*) die wahrscheinlichste Menge versteckter Zustände (*Zustandssequenz*) zu bestimmen, die diese generiert hat. Hierfür muss eine Reihe von Problemen bewältigt werden, die im Laufe dieses Kapitels noch erwähnt werden.

Die Zustandssequenz, die die versteckten Zustände repräsentiert, sei gegeben durch den Vektor $\vec{y}$. Die Zustände stammen hierbei aus dem endlichen Zustandsalphabet Y. Wichtig ist die Tatsache, dass die Zustände in einer Zustandssequenz logischerweise mehrmals auftauchen können. Daher bezeichnet man die Zustände des Zustandsalphabet mit q_i. Diese Zustände sind dann eindeutig. Um eine klare Trennung in der Begrifflichkeit zu erreichen, werden Sequenzen $\vec{y}$, die Beobachtungssequenzen $\vec{x}$ beschreiben, fortan als *Labelsequenzen* bezeichnet. Ein Element y_t der Labelsequenz soll dann - im Gegensatz zu den Zuständen q_i - *Label* heißen. Nimmt man die bei den MP definierten Zustände als Beispiel, so sind die Zustände des Zu-

standsalphabets: $\{q_1 = sonnig,\ q_2 = bewölkt,\ q_3 = regnerisch\}$. Zur Verdeutlichung: eine Labelsequenz kann zum Beispiel wie folgt aussehen: $\{y_1 = bewölkt, y_2 = sonnig, y_3 = sonnig,\ y_4 = regnerisch \ldots\}$. Die Beobachtungssequenz, die durch die Labelsequenz generiert wird, sei als Vektor $\vec{x}$ bezeichnet. Die Sequenzen entsprechen sich natürlich in der Länge. Die Länge der Sequenzen sei in allen folgenden Algorithmen mit T bezeichnet. Die hier definierten Begriffe sollen für die gesamte Arbeit gelten.

Ein HMM ist somit durch das Tripel ($\vec{\pi}$, A, B) definiert.

$\vec{\pi}$ und A sind bereits durch die MP bekannt: $\vec{\pi}$ ist der Startvektor und A ist die Transitionsmatrix, die die Wahrscheinlichkeiten für Übergänge von einem Zustand q_i zu einem anderen Zustand q_j beschreibt. Neu ist die Generierungsmatrix B, die die Wahrscheinlichkeiten beschreibt, dass ein beobachtbarer Zustand x_t von einem Zustand q_i generiert wird.

Formal schreibt man:

$A = (a_{q_i q_j})$, wobei die Felder der Matrix mit den Wahrscheinlichkeiten $P(q_j \mid q_i)$ gefüllt werden. q_i ist hierbei der Vorgängerzustand von q_j.

$B = (b_{q_i}(x_t))$, wobei die Felder der Matrix mit den Wahrscheinlichkeiten $P(x_t \mid q_i)$ gefüllt werden. x_t ist hierbei ein Zustand, der von q_i generiert wird.

Abbildung 5 beschreibt exemplarisch das Modell eines HMM, wobei die Pfeile zwischen Labeln den Transitionen und Pfeile zwischen Labeln und Beobachtungen den Generierungen entsprechen.

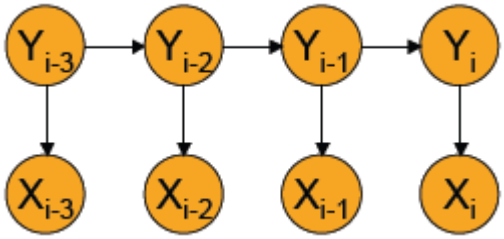


Abbildung 5: Modell eines HMM

Zum Einsatz von HMM sind drei verschiedene Arten von Aufgaben effizient zu lösen:

1. Bei gegebener Beobachtungssequenz $\vec{x}$ und Modell λ=($\vec{\pi}$, A, B) muss die Wahrscheinlichkeit $P(\vec{x} \mid \lambda)$ berechnet werden. In dem oben beschriebenen Beispiel entspräche diese Aufgabe der Tatsache, dass ein Beobachter eine bestimmte Beobachtungssequenz macht (Regenschirm: {ja, nein, nein, ja}). Anschließend wird die Wahrscheinlichkeit berechnet, die darstellt, wie gut das simulierende HMM-Modell ist. Dies ist im Grunde ein Zwischenschritt zum Trainieren eines simulierenden HMM-Modells, da bei einer zu schlechten resultierenden Wahrscheinlichkeit das Modell verändert werden sollte.
2. Für gegebene Beobachtungssequenz $\vec{x}$ und Modell λ=($\vec{\pi}$, A, B) muss die beste Labelsequenz $\vec{y}$ gefunden werden, wobei die beste Labelsequenz $\vec{y}$ die

Wahrscheinlichkeit $P(\vec{x} \mid \vec{y}, \lambda)$ maximiert. Im Beispiel entspricht diese Aufgabe dem Problem, herauszufinden, welche Zustandssequenz (Wettersequenz) eine Beobachtungssequenz mit größter Wahrscheinlichkeit generiert.

3. Die Parameter für Modell $\lambda=(\vec{\pi}$, A, B) müssen so eingestellt werden, dass die Wahrscheinlichkeit $P(\vec{x} \mid \lambda)$ maximiert wird. Diese Aufgabe entspricht dem Problem ein HMM-Modell so einzustellen, dass es oben beschriebenes Beispiel bestmöglich simuliert.

Da CRF auf den HMM aufbauen, sind diese drei Punkte auch für CRF relevant. Zwar wird in späteren Kapiteln noch genauer auf die spezielle Herangehensweise an diese Probleme bei CRF eingegangen, jedoch sollen schon hier die grundsätzlichen Verfahren und Algorithmen zur Lösung der oben angesprochenen Aufgaben vorgestellt werden, da sie bei CRF nur geringfügig modifiziert werden.

3.2.1 Forward-Backward-Algorithmus

Zur Lösung des ersten Problems muss aus der Beobachtungssequenz $\vec{x}$ mithilfe des Modells λ die Wahrscheinlichkeit $P(\vec{x} \mid \lambda)$ berechnet werden.

Ein triviales Verfahren zur Lösung dieses Problems ist die Betrachtung jeder möglichen Labelsequenz, die in der Länge T der Länge der Beobachtungssequenz entspricht. Man berechnet dann für jede mögliche Labelsequenz $\vec{y}$ vorerst die Wahrscheinlichkeit $P(\vec{x} \mid \vec{y}, \lambda)$, die anzeigt, wie wahrscheinlich die Generierung der Beobachtungssequenz aus der Labelsequenz ist. Hiefür benötigt man ausschließlich die Generierungsmatrix B. Somit ergibt sich

$$P(\vec{x} \mid \vec{y}, \lambda) = b_{y_1}(x_1) b_{y_2}(x_2) ... b_{y_T}(x_T)$$

Wahrscheinlichkeit 1: Labelsequenz generiert Beobachtungssequenz

Zusätzlich muss man dann noch die Wahrscheinlichkeit $P(\vec{y} \mid \lambda)$ bestimmen, mit der die Labelsequenz überhaupt auftaucht. Hierfür benötigt man dann die Transitionsmatrix A. Daher entsteht

$$P(\vec{y} \mid \lambda) = \pi \cdot a_{y_1 y_2} a_{y_2 y_3} ... a_{y_{T-1} y_T}$$

Wahrscheinlichkeit 2: Wahrscheinlichkeit für die Existenz der Labelsequenz

Die kombinierte Wahrscheinlichkeit $P(\vec{x}, \vec{y} \mid \lambda)$ gibt schließlich an, mit welcher Wahrscheinlichkeit beide Sequenzen simultan auftauchen. Hierfür müssen die soeben definierten Wahrscheinlichkeiten multipliziert werden. Somit entsteht

$$P(\vec{x}, \vec{y} \mid \lambda) = P(\vec{x} \mid \vec{y}, \lambda) \cdot P(\vec{y} \mid \lambda)$$

Formel 1: Kombinierte Wahrscheinlichkeit

Um nun die Wahrscheinlichkeit $P(\vec{x} \mid \lambda)$ zu erhalten, müssen alle diese Einzelwahrscheinlichkeiten aufsummiert werden. Also erhält man

$$P(\vec{x} \mid \lambda) = \sum_{alle\ \vec{y}} P(\vec{x}, \vec{y} \mid \lambda)$$

Formel 2: Wahrscheinlichkeit, dass Beobachtung aus HMM erzeugt wird

Das Problem bei diesem trivialen Vorgehen ist die Tatsache, dass sich zur Bestimmung der Wahrscheinlichkeit eine Laufzeit von $O(2 \cdot T \cdot N^T)$ ergibt, wobei N die Anzahl der verschiedenen Zustände des Zustandsalphabets ($N = |Y|$) und T die schon bekannte Länge der Sequenzen ist. $O(2 \cdot T)$ ergibt sich aus der Berechnung der beiden oben beschriebenen Wahrscheinlichkeiten Wahrscheinlichkeit 1 und Wahrscheinlichkeit 2. Der exponentielle Term $O(N^T)$ ergibt sich aus der Tatsache, dass über alle möglichen Labelsequenzen aufsummiert werden muss (Formel 2). Dies ist ein uneffizientes Vorgehen, so dass nach einem anderen Weg gesucht werden muss.

Ein effizienteres Verfahren ist der so genannte *Forward-Backward-Algorithmus*. Dieser Algorithmus beruht auf dem Prinzip der dynamischen Programmierung: ein komplexes Problem wird in viele kleine lösbare Probleme aufgeteilt.

Man definiert eine *forward-Variable* α, die der Wahrscheinlichkeit (gegeben Modell λ) entspricht, dass bis zur Länge t die Beobachtungssequenz $x_1, x_2, ..., x_t$ aufgetaucht ist, und an Stelle t das Label y_t in der Labelsequenz der Zustand q_i ist. q_i ist der i-te Zustand im Zustandsalphabet Y.

α ist formal wie folgt definiert:

$$\alpha_t(i) = P(x_1, x_2, ..., x_t, y_t = q_i \mid \lambda)$$

Wahrscheinlichkeit 3: forward-Variable

$\alpha_t(i)$ lässt sich induktiv bestimmen, indem man logischerweise vorerst annimmt, dass

1. $\alpha_1(i) = \pi_i \cdot b_i(x_1)$, für $1 \le i \le N$

Formel 3: Bestimmung der initialen forward-Variablen

Hier wird die initiale forward-Variable erzeugt, indem die Wahrscheinlichkeit berechnet wird, dass vom Zustand q_i die Beobachtung x_1 erzeugt wird. Dies muss für jeden möglichen Zustand q_i geschehen, da diese kollektiv im nächsten Schritt gebraucht werden.

Dann werden jeweils die folgenden forward-Variablen bestimmt.

2. für $1 \le j \le N$ und $t = 1,2,...,T-1$, $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{q_i q_j} \right] b_{q_j}(x_{t+1})$

Formel 4: Rekursionsschritt für die forward-Variable

Für einen bestimmten Zustand q_j an der Stelle t+1 werden alle direkten Vorgänger-forward-Variablen mit der Wahrscheinlichkeit für eine Transition zu diesem Zustand multipliziert und summiert. Dieser Term muss schließlich noch mit der Wahrscheinlichkeit für das Generieren der Beobachtung x_{t+1} multipliziert werden.

Die Wahrscheinlichkeit $P(\vec{x} \mid \lambda)$ ist dann einfach die Summe über alle endgültigen forward-Variablen.

3. $P(\vec{x} \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$

Dies ist so durchführbar, da nach Definition der forward-Variablen gilt: $\alpha_T(i) = P(x_1, x_2, ..., x_T, y_T = q_i \mid \lambda)$

Das Aufsummieren dient schließlich nur noch dem Sicherstellen, dass an der letzten Labelsequenzstelle jedes beliebige Label stehen kann.

Betrachtet man die Laufzeit, so sieht man, dass dieses Verfahren mit $O(T \cdot N^2)$ wesentlich effizienter ist, als das ganz oben beschriebene triviale Vorgehen. Die reduzierte Laufzeit resultiert aus der Tatsache, dass man zum Berechnen an jedem der T Stellen in den Sequenzen jeweils nur N Variablen sowie ihre N Vorgänger betrachten muss.

Äquivalent zu den forward-Variablen werden *backward-Variablen* definiert. Allerdings dienen diese Variablen nicht zur Lösung der 1. Aufgabe von HMM. Sie werden vielmehr später benutzt, um auch Aufgabe zwei und drei effizient zu lösen.

$\beta_t(i) = P(x_{t+1}, x_{t+2}, ..., x_T \mid y_t = q_i, \lambda)$

Wahrscheinlichkeit 4: backward-Variable

Es handelt sich sozusagen um die Wahrscheinlichkeit, dass die partielle Beobachtungssequenz von x_T bis x_{t+1} bei Zustand q_i an Stelle t der Labelsequenz gegeben ist. Man arbeitet im Gegensatz zu den forward-Variablen die Beobachtungssequenz von hinten ab – daher auch der Name. Auch diese Variablen werden induktiv erzeugt, indem man initial annimmt, dass

1. $\beta_T(i) = 1$, für $1 \le i \le N$

Dieser initiale Schritt definiert β_T willkürlich als 1 für jedes q_i und dient dazu, die backward-Variable auf 1 zu normieren.

Danach werden die restlichen Variablen anhand der Vorgänger erzeugt.

2. für $t = T-1, T-2, ..., 1$ für $1 \le i \le N$ gilt $\beta_t(i) = \sum_{j=1}^{N} a_{q_i q_j} b_{q_j}(x_{t+1}) \beta_{t+1}(j)$

Für jedes $\beta_t(i)$ werden alle N möglichen Vorgänger betrachtet. Diese werden natürlich mit der Wahrscheinlichkeit für eine Transition von q_i nach q_j sowie der Wahrscheinlichkeit für eine Generierung der Beobachtung x_{t+1} durch Zustand q_j multipliziert.

Die Laufzeit beträgt dann wie schon bei der Erzeugung der forward-Variablen $O(T \cdot N^2)$. Beide soeben vorgestellten Variablen-Arten sind, wie schon erwähnt, auch für die kommenden Aufgaben von Bedeutung.

3.2.2 Viterbi-Algorithmus

Um für eine gegebene Beobachtungssequenz $\vec{x}$ die optimale Labelsequenz $\vec{y}$ zu finden, kann man zwei verschiedene Herangehensweisen wählen.

Beim ersten Verfahren wählt man jeweils das optimale Label y_t, das am ehesten die Beobachtung x_t generiert. Logischerweise maximiert dieses Verfahren die Anzahl der korrekten einzelnen Zustände. Als Hilfsvariable wird

$$\gamma_t(i) = P(y_t = q_i \mid \vec{x}, \lambda)$$

Formel 5: Hilfsvariable für den Viterbi-Algorithmus

definiert, was die Wahrscheinlichkeit darstellt, an Stelle t der Labelsequenz in Zustand q_i zu sein – gegeben die Beobachtungssequenz $\vec{x}$ sowie Modell λ.

Benutzt man nun die soeben definierten forward- sowie backward-Variablen, so kann man die Formel 5 wie folgt darstellen

$$\gamma_t(i) = \frac{\alpha_t(i) \cdot \beta_t(i)}{P(\vec{x} \mid \lambda)}$$

Formel 6: Hilfsvariable für den Viterbi-Algorithmus dargestellt durch forward-backward-Variablen

Die Gleichungen entsprechen einander, da die forward-Variable $\alpha_t(i)$ der Wahrscheinlichkeit für die Beobachtungen $x_1, x_2, ..., x_t$ sowie das Label $y_t = q_i$ entspricht. Hinzu kommt $\beta_t(i)$, was der Wahrscheinlichkeit für die Beobachtungen $x_{t+1}, x_{t+2}, ..., x_T$ und Label $y_t = q_i$ entspricht. Erläuternd sei auf die folgende Umformung verwiesen:

$$\alpha_t(i) \cdot \beta_t(i) = P(x_1, x_2, ..., x_t, y_t = q_i \mid \lambda) \cdot P(x_{t+1}, x_{t+2}, ..., x_T \mid y_t = q_i, \lambda) = P(\vec{x} \mid \lambda) \cdot P(y_t = q_i \mid \vec{x}, \lambda)$$

Um nun wieder die Hilfsvariable für den Viterbi-Algorithmus zu erhalten, muss man das Produkt der forward- und backward-Variablen nur noch durch $P(\vec{x} \mid \lambda)$ teilen:

$$\frac{P(\vec{x} \mid \lambda) \cdot P(y_t = q_i \mid \vec{x}, \lambda)}{P(\vec{x} \mid \lambda)} = \gamma_t(i)$$

$P(\vec{x} \mid \lambda)$ ist gleichzeitig ein Normalisierungsfaktor, der $\gamma_t(i)$ zu einer bedingten Wahrscheinlichkeit macht, da nämlich $\sum_{i=1}^{N} \gamma_t(i) = 1$.

Wenn man nun das jeweils wahrscheinlichste Label für eine Beobachtung aus einer Beobachtungssequenz erfassen möchte, benutzt man $\gamma_t(i)$ wie folgt:

$y_t = \underset{1 \le i \le N}{\arg\max} [\gamma_t(i)]$, für alle $1 \le t \le T$. y_t wird also der Zustand q_i zugeordnet, der den höchsten $\gamma_t(i)$-Wert hat.

Allerdings kann das soeben beschriebene Verfahren dann Probleme aufwerfen, wenn zwischen zwar einzeln optimalen Zuständen q_i und q_j keine Transition, beziehungsweise eine Transition von $a_{q_i q_j} = 0$, besteht. Man kann nun zwar versuchen, aufeinander folgende erlaubte Labelpaare y_t, y_{t+1} oder –tripel y_t, y_{t+1}, y_{t+2} zu finden. Der Vollständigkeit halber muss allerdings ein zweites Verfahren entwickelt werden, das die komplette optimale Labelsequenz direkt erstellt, indem es die un-

möglichen Transitionen respektiert. Dieses nun vorgestellte Verfahren ist der *Viterbi-Algorithmus*.

1. Schritt: Initialisierung

   $\delta_1(i) = \pi_{q_i} b_{q_i}(x_1)$

   $\psi_1(i) = 0$

   $\delta_1(i)$ enthält die Wahrscheinlichkeit, dass Zustand q_i Startzustand ist und durch q_i die Beobachtung x_1 generiert wird. $\psi_1(i)$ wird mit 0 initialisiert.

2. Schritt: Rekursion

   $for \quad 2 \le t \le T, \quad 1 \le j \le N$

   $$\delta_t(j) = \max_{1 \le i \le N} \left[\delta_{t-1}(i) a_{q_i q_j}\right] b_{q_j}(x_t)$$

   $$\psi_t(j) = \arg\max_{1 \le i \le N} \left[\delta_{t-1}(i) a_{q_i q_j}\right]$$

   Für jeden weiteren Punkt t der Beobachtungssequenz sowie für jeden möglichen Zustand j wird $\delta_t(j)$ gefüllt, indem die höchsten Wahrscheinlichkeiten des vorhergegangenen Punktes $\delta_{t-1}(i)$ mit der Transitionswahrscheinlichkeit zu q_j sowie Generierungswahrscheinlichkeit von x_t multipliziert werden. Man erzeugt somit immer Kandidaten für den nächsten Schritt, aus denen dann jeweils der beste ausgewählt wird. $\psi_t(j)$ enthält jeweils die Zustandsnummer i, als Hinweis auf die wahrscheinlichste Transition zwischen q_i und q_j zu Stelle t.

3. Schritt: Terminierung

   $$P^* = \max_{1 \le i \le N} [\delta_T(i)]$$

   $$i_T^* = \arg\max_{1 \le i \le N} [\delta_T(i)]$$

   Abschließend enthält P^* die optimale Wahrscheinlichkeit für Beobachtungssequenz $\vec{x}$. i_T^* enthält den letzten Zustand der optimalen Labelsequenz.

4. Schritt: Backtracking

   $for \quad t = T-1, T-2, \ldots, 1$

   $i_t^* = \psi_{t+1}(i_{t+1}^*)$

   In diesem abschließenden Backtracking-Schritt wird ausgehend vom letzten Zustand der optimalen Labelsequenz jeweils der nächste (voranstehende) optimale Zustand bestimmt. i^* enthält schließlich die optimale Labelsequenz.

Formel 7: Viterbi-Algorithmus

Die eingeführte Variable δ wird als *quantity* bezeichnet und kann auch beschrieben werden als :

$$\delta_t(i) = \max_{y_1, y_2, ..., y_{t-1}} P[y_1, y_2, ..., y_t = i, x_1, x_2, ..., x_t \mid \lambda],$$

was im Grunde der forward-backward-Berechnung entspricht, wobei der Backtracking-Schritt dort natürlich fehlt. Dieser Schritt ist hier jedoch notwendig, da zuerst ein *forward-scan* ($1 \le t \le T$) über die Sequenz gemacht wird, um mögliche Sackgassen zu vermeiden. Würde man die optimale Labelsequenz schon bei diesem forward-scan festlegen, wäre es möglich, dass man bis Stelle t eine optimale Sequenz hat, danach aber nicht mehr optimal bleibt, da von t keine optimalen Transitionen mehr ausgehen. Nach dem forward-scan hat man durch die quantity-Variable die optimale letzte Stelle in der Labelsequenz. Durch die Variable ψ, die anhand der quantity die optimale Transition zum nächsten Label bestimmt, erhält man dann mithilfe eines *backward-scans* ($T \ge t \ge 1$) jeweils das optimale Vorgängerlabel.

Somit hat man einen Algorithmus, der die komplette optimale Labelsequenz bestimmt.

3.2.3 Baum-Welch-Algorithmus

Die schwierigste Aufgabe ist die, die Parameter λ=($\vec{\pi}$, A, B) des Modells so einzustellen, dass die Wahrscheinlichkeit der Beobachtungssequenz maximal wird. Leider ist es nicht möglich, dieses Problem analytisch zu lösen. Daher müssen zur Optimierung Gradientenverfahren oder iterative Methoden wie der *Baum-Welch-Algorithmus* angewandt werden. Hierbei werden die Parameter so eingestellt, dass $P(\vec{x} \mid \lambda)$ lokal für die Trainingsdaten maximal ist. In diesem Kapitel soll vorerst nur das iterative Vorgehen beschrieben werden, allerdings werden später für die CRF noch andere Verfahren vorgestellt.

Als erstes definiert man nun die Wahrscheinlichkeit, an Stelle t in Zustand q_i und an der darauf folgenden Stelle in Zustand q_j zu sein – gegeben die Beobachtungssequenz $\vec{x}$ sowie das Modell λ:

$$\xi_t(i, j) = P(y_t = q_i, y_{t+1} = q_j \mid \vec{x}, \lambda)$$

Wahrscheinlichkeit 5: (Hilfs-)Wahrscheinlichkeit des den Baum-Welch-Algorithmus

Diese Annahme impliziert eine Transition zwischen q_i und q_j. Auch erinnert diese Variable an die in 3.2.2 definierte Variable $\gamma_t(i)$. Man kann daher auch diese Variable durch die forward- und backward-Variablen beschreiben:

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{q_i q_j} b_{q_j}(x_{t+1}) \beta_{t+1}(j)}{P(\vec{x} \mid \lambda)}$$

Formel 8: Wahrscheinlichkeit des Baum-Welch-Algorithmus durch forward- und backward-Variablen ausgedrückt

Einzig die Tatsache, dass man die Transition zwischen q_i und q_j beachten muss, unterscheidet diese Variable von der eben angesprochenen Variable $\gamma_t(i)$. Die Transi-

tion wirkt sich in der Gleichung durch den Term $a_{q_i q_j} b_{q_j}(x_{t+1})$ aus, der der Transitionswahrscheinlichkeit multipliziert mit der Generierungswahrscheinlichkeit für x_{t+1} entspricht. Die Wahrscheinlichkeit $\gamma_t(i)$, an Stelle t in Zustand q_i zu sein, bei gegebenem Modell und gegebener Beobachtungssequenz, kann nun auch durch $\xi_t(i, j)$ dargestellt werden:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i, j)$$

Formel 9: Darstellung der in 3.2.2 dargestellten Hilfsvariable durch die Wahrscheinlichkeit des Baum-Welch-Algorithmus

Summiert man die Wahrscheinlichkeiten $\gamma_t(i)$ über die Sequenzlänge (bis auf das letzte Label) auf, so erhält man die erwartete Anzahl von Transitionen, die von Zustand i ausgehen. Die Summe der Wahrscheinlichkeiten $\xi_t(i, j)$ über die Sequenzlänge kann als erwartete Anzahl von Transitionen von q_i nach q_j angesehen werden.

$\sum_{t=1}^{T-1} \gamma_t(i)$ = erwartete Anzahl von Transitionen ausgehend von Zustand q_i.

$\sum_{t=1}^{T-1} \xi_t(i, j)$ = erwartete Anzahl von Transitionen von Zustand q_i zu Zustand q_j.

Der Baum-Welch-Algorithmus ändert nun die Modell-Parameter wie folgt:

$$\bar{\pi}_{q_i} = \gamma_1(i), \quad 1 \le i \le N$$

Wahrscheinlichkeit 6: Neue Anfangswahrscheinlichkeit durch den Baum-Welch-Algorithmus

Man definiert neue Anfangswahrscheinlichkeiten insofern, als dass man sie durch die Wahrscheinlichkeiten ersetzt, an Stelle 1 in Zustand i zu sein, bei gegebenem Modell und, was sehr wichtig ist, bei gegebener Beobachtungssequenz.

$$\bar{a}_{q_i q_j} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

Formel 10: Neue Transitionswahrscheinlichkeit durch den Baum-Welch-Algorithmus

Man ändert die Transitionswahrscheinlichkeiten, indem man sie durch die erwartete Anzahl der Transitionen zwischen Zustand q_i zu Zustand q_j geteilt durch die erwartete Anzahl der Transitionen ausgehend von Zustand q_i ersetzt.

$$\bar{b}_{q_j}(k) = \frac{\sum_{\substack{t=1 \\ x_t = k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

Formel 11: Neue Generierungswahrscheinlichkeit durch den Baum-Welch-Algorithmus

Hier wird die Generierungswahrscheinlichkeit, dass Zustand q_j die Beobachtung k generiert, so verändert, dass er der erwarteten Anzahl entspricht, in der man in Zustand j ist und Beobachtung k macht, geteilt durch die erwartete Anzahl, in Zustand j zu sein.

Formel 12: Baum-Welch-Algorithmus

Da diese soeben benutzten Erwartungswerte abhängig sind von der aktuellen Beobachtungssequenz, kann man davon ausgehen, dass das neue Modell $\overline{\lambda}$, was durch $(\overline{\pi}, \overline{A}, \overline{B})$ definiert ist, für die benutzte Beobachtungssequenz mindestens genauso gut arbeitet wie das initiale Modell λ.

Somit gibt es zwei mögliche Fälle, die eintreten können:

1. $\lambda = \overline{\lambda}$
2. $\lambda \neq \overline{\lambda}$, wobei $P(\vec{x} \mid \overline{\lambda}) > P(\vec{x} \mid \lambda)$

Der erste Fall stellt einen kritischen Punkt dar, da anscheinend keine weiteren Verbesserungen auftreten können. Punkt zwei beschreibt den Fall, dass das neue Modell eine Verbesserung der Wahrscheinlichkeit, die Beobachtungssequenz zu generieren, ergibt - gegeben das neue Modell. In dem Fall benutzt man $\overline{\lambda}$ als Modell und wiederholt das beschriebene Verfahren bis zu einem Abbruchkriterium.

Durch den forward-backward-Algorithmus findet man lokale Maxima, und da viele Probleme sehr komplex sind und viele lokale Maxima haben, bedeutet das, dass man unter Umständen in einem lokalen Maximum stecken bleiben kann, ohne bis zum globalen Maximum vorzustoßen. Um dies zu vermeiden können verschiedene Gradientenverfahren angewandt werden. Diese sollen jedoch, wie schon erwähnt, erst im Laufe der Arbeit näher betrachtet werden.

3.2.4 Abschließende Bemerkung

Der große Nachteil der HMM ist die Tatsache, dass es eine strikte Unabhängigkeitsannahme gibt. Und zwar hängt jede Beobachtung x_i nur von dem generierenden Label y_i ab. Wäre dies nicht der Fall, müsste man zur Berechnung von der bestmöglichen Labelsequenz abermals über die Länge der Sequenz iterieren, da jede Beobachtung theoretisch von jedem Label abhängen könnte. Dieses Vorgehen wird allerdings aus Effizienzgründen vermieden. Aufgrund der Unabhängigkeitsannahme können jedoch leider keine komplexen Merkmale in Beobachtungen vorkommen.

Diese Nachteile versuchen die MEMM, die im folgenden Kapitel vorgestellt werden, zu vermeiden.

3.3 Maximum Entropy Markov Modelle

MEMM sind eine Weiterentwicklung der soeben erwähnten HMM. MEMM wurden in [20] vorgestellt und zeichnen sich primär dadurch aus, dass die einzelnen Beobachtungen x_t im Gegensatz zu den HMM aus willkürlichen, sich überlappenden Merkmalen bestehen.

Es gibt im Gegensatz zu HMM zwei explizite Unterschiede:

1. Komplexe, überlappende Merkmale: Viele Einsatzbereiche würden von einer komplexen Beschreibung für Beobachtungen x_t profitieren. HMM ordnen einem Label y_t jeweils nur eine Beobachtung zu, die meistens der Identität der Beobachtung entspricht. Allerdings ist die Tatsache, dass das beobachtete Wort großgeschrieben, ein Nomen und an einer bestimmten Stelle des Textes ist, unter Umständen viel aussagekräftiger, als eine schlichte Identität. Somit dienen diese so genannten überlappenden Merkmale einer detaillierteren Beschreibung der Beobachtung.
2. Bedingte Wahrscheinlichkeit: HMM bilden eine kombinierte Wahrscheinlichkeit aus Beobachtungs- und Labelsequenzen, was allerdings in keinster Weise intuitiv ist. Denn eigentlich möchte man die Labelsequenz in Abhängigkeit von der gegebenen Beobachtungssequenz bestimmen.

Bei MEMM handelt es sich wie bei HMM um gerichtete Modelle, allerdings sind diese bedingt. Das bedeutet, dass die Labelsequenz von bestimmten Faktoren abhängig ist. Formal gesagt, ist ein Label aufgrund der Markov-Eigenschaft von mindestens einem Vorgängerlabel sowie der Beobachtung abhängig. Eine beispielhafte Grafik für MEMM ist in Abbildung 6 zu sehen.

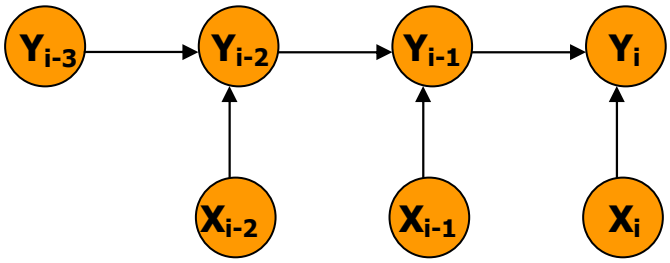


Abbildung 6: Modell eines MEMM

Die bedingten Wahrscheinlichkeiten sind durch exponentielle Modelle gegeben, die auf dem *Maximum Entropy*-Prinzip beruhen. Dieses Prinzip, das in 3.3.2 näher erläutert wird, behandelt die überlappenden Merkmale korrekt und erspart das Iterieren über alle möglichen Beobachtungssequenzen.

Somit bildet man – im Gegensatz zur kombinierten Wahrscheinlichkeit bei HMM – die abhängige (konditionale) Wahrscheinlichkeit durch folgenden Term:

$$P(\vec{y} \mid \vec{x}) = \prod_{t=1}^{T} P(y_t \mid y_{t-1}, x_t)$$

Wahrscheinlichkeit 7: Bedingte Wahrscheinlichkeit bei MEMM

Das aktuelle Label hängt also einerseits vom Vorgänger-Label sowie von der aktuell betrachteten Beobachtung ab.

3.3.1 Lösung der klassischen HMM-Aufgaben

Obwohl sich HMM und MEMM unterscheiden, kann man dennoch die drei klassischen Aufgaben und Abwandlungen ihrer Lösung von den HMM übernehmen. Bei

HMM werden die Variablen $\alpha_t(i)$ mit den Wahrscheinlichkeiten, an Stelle y_t in Zustand q_i zu sein, gefüllt. Dies geschieht mittels dynamischer Programmierung und Rekursion (Formel 4).

Da es sich bei MEMM um abhängige Modelle handelt – die Beobachtung also gegeben ist -, sieht der obige Rekursionsschritt hierbei wie folgt aus:

$$\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i) P(q_j \mid q_i, x_{t+1})$$

Formel 13: Bestimmung der forward-Variable bei MEMM

In beiden Schritten entspricht q_i einem Zustand, der eine Transition zu q_j besitzt. Die neue backward-Variable wird äquivalent aufgebaut. Zudem sollte klar sein, dass auch die anderen HMM-spezifischen Aufgaben leicht auf MEMM übertragbar sind.

3.3.2 Exponentielles Modell für Wahrscheinlichkeiten

Das *Maximum Entropy*-Prinzip besagt, dass das beste Modell für die zu untersuchenden Daten das ist, welches konsistent zu den Charakteristika ist, die man aus den Trainingsdaten bezieht. Jede dieser Charakteristika beschreibt auf seine Weise die Trainingsdaten. Die Charakteristika sollen in diesem Fall aus n binären Merkmalen bestehen. Zwar sind auch real-wertige Merkmale möglich, die binären dienen jedoch der vereinfachten Darstellung.

Die Merkmale sind Aussagen der Form „die Beobachtung ist *New York*“, „das beobachtete Wort ist großgeschrieben“, „das beobachtete Wort ist ein Nomen“, etc. In bedingten Maximum Entropy-Modellen hängen die Merkmale nicht nur von der Beobachtung sondern auch von dem Resultat der zu beschreibenden Funktion ab. Die Funktion ist in diesem Fall die *Transitionsfunktion* $P(q_j \mid q_i, x_t)$, die bei gewissen Beobachtungen eine Transition von q_i nach q_j vollzieht, und das Resultat ist der Zustand q_j.

Die Merkmale werden beschrieben als $f_a(x_t, q_j)$, wobei x_t eine Beobachtung und q_j ein möglicher nächster Zustand ist. a ist ein Paar $a = <b, s>$, so dass b eine binäre Eigenschaft der Beobachtung und s ein Zielzustand ist.

Somit gilt:

$$f_{<b,s>}(x_t, y_t) = \begin{cases} 1 \, wenn \, b(x_t) \, ist \, true \, und \, s = y_t \\ 0 \, sonst \end{cases}$$

Dann geht man beim Maximum Entropy-Verfahren davon aus, dass der erwartete Wert jedes Merkmals dem Durchschnitt über der Trainings-Beobachtungssequenz $\langle x_1, ..., x_T \rangle$ mit der zugehörigen Labelsequenz $\langle y_1, ..., y_T \rangle$ entspricht. Formal wird diese Aussage wie folgt dargestellt:

$$\frac{1}{m_{q'}} \sum_{k=1}^{m_{q'}} f_a(x_{t_k}, y_{t_k}) = \frac{1}{m_{q'}} \sum_{k=1}^{m_{q'}} \sum_{q \in Y} P(q \mid q', x_{t_k}) \, f_a(x_{t_k}, q),$$

Formel 14: Maximum Entropy-Bedingung

wobei $t_1,...,t_{m_{q'}}$ den Stellen entsprechen, an denen die Labelsequenz das Label $y_{t_k} = q'$ hat – also die Stellen, die die Transitionsfunktion $P(q \mid q', x_{t_k})$ betreffen.

Laut [15] hat somit jede Transition folgende Wahrscheinlichkeit:

$$P(q_i \mid q_j, x_t) = \frac{1}{Z(x_t, q_j)} \exp\left[\sum_a \lambda_a f_a(x_t, q_i)\right]$$

Formel 15: Exponentielles Modell für Transitionswahrscheinlichkeiten bei MEMM

Pro Zustandsübergang erhält man also ein exponentielles Modell.

Da der exponentielle Term auf der rechten Formelseite reale Zahlen enthalten kann, handelt es sich nicht um eine echte beziehungsweise normalisierte Wahrscheinlichkeit. Damit eine echte Wahrscheinlichkeit entsteht, normalisiert man das Ergebnis des exponentiellen Terms anhand von $\frac{1}{Z(x_t, q_j)}$. Zur Erläuterung dieses *Normalisierungsterms* sei auf die folgenden Kapitel (vor allem 4.2) verwiesen, da dieser Term auch bei CRF eine wichtige Rolle spielt.

MEMM können zwar komplexe und überlappende Merkmale verarbeiten, allerdings besitzen sie im Gegensatz zu HMM einen anderen Nachteil, der als *Label Bias Problem* bezeichnet wird.

Dies soll nun näher erläutert werden.

3.3.3 Label Bias Problem

Für MEMM gilt, dass die Summe der eingehenden Wahrscheinlichkeiten der Summe der ausgehenden Wahrscheinlichkeiten entspricht. Formal beschrieben gilt also: $\sum_{q_i} P(q_i \mid q_j) = 1$, wobei eine Transition von q_j zu q_i verläuft.

Diese Tatsache wird in der Literatur auch als *conservation of score mass* bezeichnet. Umgangssprachlich gesagt, muss ein Zustand die Masse oder Werte, die in ihn eingehen auch wieder ausgeben. Daraus folgt jedoch, dass Übergänge von Zuständen mit wenigen Nachfolgern im Schnitt automatisch höhere Wahrscheinlichkeiten erhalten als Übergänge zu Zuständen mit vielen Nachfolgern, da die Wahrscheinlichkeiten nur auf wenige ausgehende Transitionen verteilt werden müssen. Im Umkehrschluss bedeutet das allerdings: hat ein Zustand nur einen Nachfolger, so wird die Beobachtung vollkommen ignoriert, da der Pfad zum Nachfolger auf jeden Fall beschritten wird. Daraus folgt, dass der Viterbi-Algorithmus unter Umständen keine korrekten Ergebnisse liefert.

Abbildung 7 beschreibt eine typische Label Bias Problem Situation.

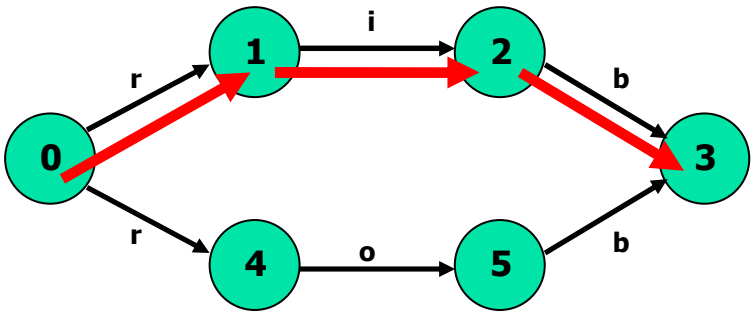


Abbildung 7: Label Bias Problem

Werden die Wahrscheinlichkeitsterme aufgeschlüsselt, so wird deutlich, dass die Wahrscheinlichkeit für Zustandsfolge 0123 bei Beobachtung ‚rib' oder Beobachtung ‚rob' jeweils der Wahrscheinlichkeit entspricht, dass bei Beobachtung r in Zustand 0 zu Zustand 1 gewechselt wird:

$$P(0123|rib) = P(1|0,r)P(2|1,i)P(3|2,b) = P(1|0,r)$$

$$P(0123|rob) = P(1|0,r)P(2|1,o)P(3|2,b) = P(1|0,r)$$

Somit hängt bei diesem speziellen Beispiel die Wahl der zu beschreitenden Zustände nur davon ab, in welchen Zustand von Zustand 0 gewechselt wird. Taucht im Training also häufiger ‚rib' als ‚rob' auf, so wird der Pfad von Zustand 0 zu Zustand 1 auch in der Anwendung beschritten.

Es gibt eine Reihe von Möglichkeiten, das Label Bias Problem zu vermeiden:

1. Verknüpfung relevanter Zustände
2. Vollverknüpfte Modelle benutzen
3. CRF benutzen

Bei der Verknüpfung relevanter Zustände werden die Zustände verbunden, an denen das Label Bias Problem auftritt. In dem oben angesprochenen Fall wären dies die Zustände 1 und 4. Diese Vorgehensweise ist allerdings nicht immer ohne weiteres möglich.

Möchte man vollverknüpfte Modelle benutzen, so fügt man Übergänge ein, so dass jeder Zustand mit jedem anderen Zustand verbunden ist. Dies erzeugt natürlich extrem komplexe Modelle, und, was noch relevanter ist, zudem gibt man das explizite Wissen auf, das unter Umständen schon durch die Zustandsübergänge modelliert war.

Bei CRF handelt es sich im Gegensatz zu HMM und MEMM um ungerichtete Modelle. Diese betrachten zudem nicht nur einzelne Zustände sondern immer die komplette Beobachtungssequenz. Außerdem können einzelne Transitionen verschieden stark bewertet werden, so dass Übergänge nicht unbedingt unabhängig von Beobachtungen beschritten werden.

3.4 Markov Random Fields

Bevor man sich mit CRF befasst, muss man MRF betrachten, die eine Vorform von CRF sind. MRF sind ungerichtete graphische Modelle. Bei dem Modell handelt es

sich um einen azyklischen Graph G mit Knotenmenge V sowie Kantenmenge E. (G=(V, E))

Die Knoten entsprechen den Zuständen in einer eins-zu-eins Beziehung, und es gilt die Markov-Eigenschaft. In diesem Fall heißt das, dass die bedingte Wahrscheinlichkeit für einen Zustand – gegeben alle Nachbarzustände – der bedingten Wahrscheinlichkeit für den Zustand – gegeben alle anderen Zustände – entspricht.

Formal schreibt man:

$$P(q_v \mid q_w : v \neq w) = P(q_v \mid q_w : v \sim w)$$

Formel 16: Markov-Eigenschaft in Markov Random Fields

Da es sich bei MRF um ungerichtete Modelle handelt, kann man die bedingte Wahrscheinlichkeit für einen Zustand nicht durch die bedingten Wahrscheinlichkeiten der Vorgänger berechnen. Vielmehr werden bedingte Wahrscheinlichkeiten durch das Produkt lokaler Funktionen parametrisiert.

Aufgrund der geltenden Markov-Eigenschaft kann man die bedingte Wahrscheinlichkeit für einen Zustand mithilfe der in direkter Nachbarschaft zu diesem Zustand stehenden Zustände berechnen.

Eine Clique bezeichnet eine Knotenmenge eines Graphen, die vollständig verknüpft ist. Die soeben erwähnte Nachbarschaft von Knoten ist also in einer Clique gegeben, daher benötigt man zur Berechnung der bedingten Wahrscheinlichkeit eines Zustandes in einem MRF nur die Clique, zu der der Zustand gehört. Zur Berechnung der bedingten Wahrscheinlichkeit werden so genannte *Potentialfunktionen* über Cliquen definiert.

Die Menge aller Potentialfunktionen sei:

$$\mathbf{\Phi} = \Phi_{c_i} : A_{c_i} \rightarrow \Re^+$$

Formel 17: Menge der Potentialfunktionen in Markov Random Fields

Dies bedeutet, dass die Potentialfunktion über Clique c_i allen Belegungen einen real-wertigen positiven Zahlenwert zuordnet.

Daher erhält man wie bei den MEMM keine echte Wahrscheinlichkeit, da der Wert eine Zahl über 1 annehmen kann. Je höher der resultierende Wert ist, umso wahrscheinlicher ist die Belegung der Clique.

Die Wahrscheinlichkeit für eine Labelsequenz ist dann:

$$P(\vec{y}) = \frac{1}{Z} * \prod_{\mathbf{\Phi}} \Phi_{c_i}(y_{i_1}, ..., y_{i_{|c_i|}})$$

Formel 18: Wahrscheinlichkeit für ein Labelsequenz in Markov Random Fields

Auch hier benutzt man einen Normalisierungsfaktor (Z), um die real-wertigen Zahlen in eine korrekte Wahrscheinlichkeit zu transformieren.

Der Normalisierungsfaktor ist definiert als:

$$Z = \sum_{Y^T} \left[\prod_{\Phi} \Phi_{c_i} \left(y_{i_1}, ..., y_{i_{|c_i|}} \right) \right]$$

Formel 19: Normalisierungsfaktor bei Markov Random Fields

Da der Normalisierungsfaktor bei den CRF (Kapitel 4.2) noch detaillierter beschrieben wird, sei hier nur die formale Schreibweise erwähnt.

4 Conditional Random Fields

4.1 Definition

CRF wurden erstmals in [18] vorgestellt. Wie der Name schon sagt, sind CRF konditional – also abhängig – und zwar von der Beobachtungssequenz $\vec{x}$. Somit wird anstatt der Wahrscheinlichkeit für eine Labelsequenz $\vec{y}$ die bedingte Wahrscheinlichkeit für die Labelsequenz abhängig von der Beobachtungssequenz $\vec{x}$ berechnet. Ansonsten entsprechen CRF den MRF.

Formal wird also $P(\vec{y} \mid \vec{x})$ berechnet. Nimmt man an, dass es sich bei dem Graphen um eine *first-order Markov-Kette* (äquivalent zu MP erster Ordnung (3.1)) handelt, so bestehen die Cliquen jeweils aus einem Knoten sowie dem Nachfolger beziehungsweise dem Vorgänger.

Abbildung 8 zeigt einen exemplarischen CRF-Graphen, wobei die gelbe Ellipse eine Clique kennzeichnet. Wie aus der Abbildung ersichtlich ist, benutzt jeder CRF-Zustand die komplette Beobachtungssequenz.

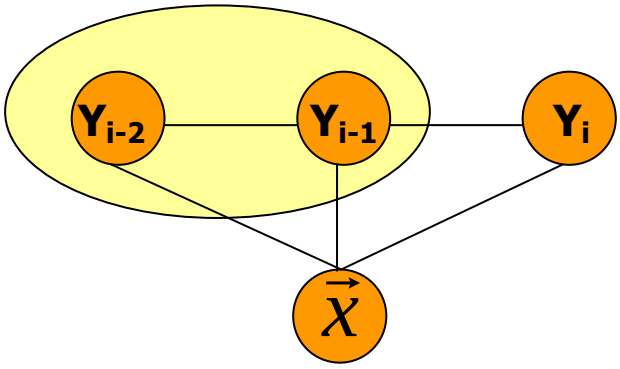


Abbildung 8: Graph eines CRF

Die Potentialfunktionen, die bereits bei den MRF beschrieben wurden, haben bei den CRF die folgende Form: $\Phi_k(y_{i-1}, y_i, \vec{x}, i)$

Um ein anschauliches Beispiel zu liefern, sei
$\Phi_k(y_{i-1}, y_i, \vec{x}, i) = b(\vec{x}, i)$, wenn $y_{i-1} = [ORG]$ und $y_i = [LOC]$.
Und $b(\vec{x}, i)$ sei 1, wenn die Beobachtung an Position i „Deutschland“ ist und ansonsten 0.

Abbildung 9: Beispiel für eine konkrete Potentialfunktion bei CRF

Ist eine Funktion erfüllt, hier würde sie dann zum Beispiel 1 liefern, nennt man sie eine aktive Potentialfunktion.

Man beschränkt sich der Einfachheit halber auf binäre Potentialfunktionen, wobei natürlich auch real-wertige Funktionswerte zugelassen sind. Eine Potentialfunktion kann auch nur ausschließlich das aktuelle Label bearbeiten und somit das voranstehende Label ungeachtet lassen. Ist dies der Fall, so handelt es sich um ein *state-feature*, bei Betrachtung auch des voranstehenden Labels spricht man von einem

feature, bei Betrachtung auch des voranstehenden Labels spricht man von einem *transition-feature*.

Zur Erstellung der Potentialfunktionen wird die Trainingsmenge benutzt, indem zum Beispiel Funktionen erstellt werden, die prüfen ob und mit welchem Zustand das zu beobachtende Wort in der Trainingsmenge aufgetaucht ist. Zudem werden viele (atomare) Tests erstellt, die primär aus Lexikonanfragen bestehen.

Des Weiteren fallen Tests wie „das Wort ist großgeschrieben“, was mit einem Test auf reguläre Ausdrücke analysiert wird, oder *Timeshifts* unter diesen Punkt. Zur Erläuterung: unter Timeshift versteht man das Vorgehen, den Blickpunkt auf ein zum Beispiel zwei Worte entferntes Wort zu richten, und dieses dann mit dem aktuell betrachteten Wort in Verbindung zu bringen.

Die gesamte Menge der Potentialfunktionen wird schließlich über dem Trainingsdatensatz trainiert, so dass jede Potentialfunktion ein Gewicht erhält. Dieses Gewicht gibt an, wie relevant eine bestimmte Potentialfunktion ist.

4.2 Bedingte Wahrscheinlichkeit

Die Formel für die bedingte Wahrscheinlichkeit ist genau wie bei den MRF:

$$P(\vec{y} \mid \vec{x}) = \frac{1}{Z(\vec{x})} \prod_{k} \sum_{i=1}^{t} \Phi_k(y_{i-1}, y_i, \vec{x}, i)$$

Formel 20: Bedingte Wahrscheinlichkeit für eine Labelsequenz in Conditional Random Fields

Nach dem *Hammersley-Clifford-Theorem* [15] kann man diese Formel, sofern es sich bei dem Graphen G um einen Baum handelt (was hier der Fall ist, da es sich bei Sequenzen um primitive Bäume handelt), in folgende Formel umwandeln:

$$P(\vec{y} \mid \vec{x}) = \frac{1}{Z(\vec{x})} \exp\left[\sum_{k} \sum_{i=1}^{t} \Phi_k(y_{i-1}, y_i, \vec{x}, i) \right]$$

Formel 21: Bedingte Wahrscheinlichkeit für eine Labelsequenz in Exponentialschreibweise

Zum Trainieren fügt man nun noch einen Vektor $\vec{\lambda}$ hinzu, der die einzelnen Potentialfunktionen unterschiedlich gewichtet – je nachdem, welche Potentialfunktion relevanter ist.

$$P(\vec{y} \mid \vec{x}, \vec{\lambda}) = \frac{1}{Z(\vec{x}, \vec{\lambda})} \exp\left[\sum_{k} \sum_{i=1}^{t} \lambda_k \Phi_k(y_{i-1}, y_i, \vec{x}, i) \right]$$

Formel 22: Bedingte Wahrscheinlichkeit mit Trainingsgewichten

Wie schon bei den MRF und MEMM erwähnt, ist Z ein Normalisierungsfaktor, der den Wahrscheinlichkeitsterm erst zu einer echten Wahrscheinlichkeit normalisiert.

Um die Formel für den Normalisierungsfaktor zu begreifen, muss man sich überlegen, wie Z arbeiten soll. Z muss - umgangssprachlich gesagt - die perfekte Labelsequenz zu 1 normieren. Z ist somit sozusagen die Summe aller „Treffer“ aller möglichen Labelsequenzen. Wobei mit „Treffer“ eine Potentialfunktion gemeint ist, die eine 1 ausgibt.

Formal ist Z also:

$$Z(\vec{x},\vec{\lambda}) = \sum_{Y^T} \exp\left[\sum_k \sum_{i=1}^{t} \lambda_k \Phi_k(y_{i-1}, y_i, \vec{x}, i)\right]$$

Formel 23: Normalisierungsfaktor Z für die bedingte Wahrscheinlichkeit in Conditional Random Fields

Somit kann man die bedingte Wahrscheinlichkeit P bezeichnen als die Anzahl der Regeln, die für eine bestimmte Labelsequenz 1 liefern, geteilt durch die Anzahl der Regeln, die für alle möglichen Labelsequenzen das Resultat 1 liefern.

Der gravierende Unterschied zu den MEMM ist die Tatsache, dass bei CRF ein einziges exponentielles Modell zur Berechnung der bedingten Wahrscheinlichkeit dient, wohingegen bei MEMM für jeden Zustand der Labelsequenz ein exponentielles Modell vorliegt. CRF können auch als finite Zustandsmodelle mit unnormalisierten Transitionswahrscheinlichkeiten angesehen werden. Finite Zustandsmodelle sind sie daher, da sie eine begrenzte Zustandsmenge vorweisen. Die Transitionen sind unnormalisiert, da eine Transition von vielen verschiedenen Faktoren abhängig ist.

4.3 Trainieren von CRF mittels maximum likelihood

Um schließlich die bestmöglichen λ-Werte zu trainieren, muss man natürlich das CRF so einstellen, dass die Wahrscheinlichkeit für gegebene Trainingsbeispiele $\vec{x}$ und $\vec{y}$ maximal wird. Dieses Vorgehen wird in der Literatur als *maximum likelihood* bezeichnet. Zusätzlich bildet man den Logarithmus der bedingten Wahrscheinlichkeit und maximiert diesen - dies wird dann als maximum-log-likelihood bezeichnet. Die Funktion, die bei maximum-log-likelihood zu maximieren ist, wird dargestellt als

$$L(\vec{\lambda}) = \sum_{h \in E} \log P_{\vec{\lambda}}(\vec{y}^{(h)} \mid \vec{x}^{(h)}),$$

Formel 24: log-likelihood-Funktion

wobei E die Menge der Trainingsbeispiele und h jeweils eines dieser Beispiel bezeichnet.

Der Einfachheit halber stellt man die Potentialfunktionen – aufsummiert über die gesamte Labelsequenz – wie folgt dar:

$$F_k(\vec{y},\vec{x}) = \sum_{i=1}^{T} \left(\Phi_k(y_{i-1}, y_i, \vec{x}, i)\right)$$

Formel 25: Über die Sequenzlänge aufsummierte Potentialfunktion

Dadurch erhält man als Gleichung für die log-likelihood-Funktion:

$$L(\vec{\lambda}) = \sum_{h \in E}\left[\log(\frac{1}{Z(\vec{x}^{(h)})}) + \sum_k \lambda_k F_k(\vec{y}^{(h)}, \vec{x}^{(h)}) \right]$$
$$= \sum_{h \in E}\left[\sum_k \left(\lambda_k F_k(\vec{y}^{(h)}, \vec{x}^{(h)})\right) - \log\left(Z(\vec{x}^{(h)})\right) \right]$$

Formel 26: log-likelihood-Funktion (aufgelöst)

Laut [18] ist diese Funktion konvex, solange zwischen Label und Zustand eine eins-zu-eins-Beziehung herrscht. Durch die Konvexität ist das Finden des globalen Maximums garantiert! Um das globale Maximum zu finden, muss man die Ableitung dieser Gleichung nullsetzen. Die Ableitung der log-likelihood-Funktion nach einem λ_k ist:

$$\nabla L(\lambda_k) = \sum_h F_k(\vec{y}^{(h)}, \vec{x}^{(h)}) - \sum_h E_{p(Y^T|\vec{x}^{(h)}, \vec{\lambda})}[F_k(Y^T, \vec{x}^{(h)})]$$
$$= \sum_{h \in E} E_{\tilde{p}(\vec{y}^{(h)}, \vec{x}^{(h)})}[F_k(\vec{y}^{(h)}, \vec{x}^{(h)})] - \sum_{h \in E} E_{p(Y^T|\vec{x}^{(h)}, \vec{\lambda})}[F_k(Y^T, \vec{x}^{(h)})],$$

Formel 27: Gradient der log-likelihood-Funktion

wobei $\tilde{p}\left(\vec{y}^{(*)}, \vec{x}^{(*)}\right)$ die empirische Verteilung über den Trainingsdaten ist. $E_p[\cdot]$ ist der Erwartungswert in Hinsicht auf die Verteilung p. Setzt man diese Gleichung null, dann erhält man das Maximum der log-likelihood-Funktion. Allerdings entspricht die Gleichung dann auch der Maximum Entropy-Bedingung, die besagt, dass die Erwartung über die Trainingsverteilung der Erwartung jedes Funktionswertes über die Modellverteilung entspricht.

Die Erwartung der Funktionswerte über die Trainingsverteilung ist trivial zu berechnen. Probleme bereitet jedoch die Erwartung über die Modellverteilung:

$$E_{p(Y^T|\vec{x}^{(h)}, \vec{\lambda})}\left[F_k(Y^T, \vec{x}^{(h)})\right] = \sum_{\vec{y} \in Y^T} P(\vec{y}|\vec{x}^{(h)}, \vec{\lambda}) F_k(\vec{y}, \vec{x}^{(h)})$$

Formel 28: Erwartungswert für jede Potentialfunktion bezogen auf das CRF-Modell

Laut [38] ist es umständlich, die rechte Seite der Funktion auf triviale Art zu berechnen. Wie aus Formel 28 zu ersehen ist, müsste man über alle möglichen Labelsequenzen summieren: bei einer Beobachtungssequenz $\vec{x}^{(k)}$ mit n Elementen gäbe es $|Y|^n$ mögliche zugehörige Labelsequenzen.

Daher müssen die Parameter auf andere Weise bestimmt werden. Dies geschieht entweder durch

- *iterative Techniken* oder
- *gradientenbasierte* Methoden.

Von [18] werden zwei iterative Techniken vorgestellt, die jedoch laut [40] von gradientenbasierten Methoden in punkto Schnelligkeit übertrumpft werden. In Kapitel 6 werden diese Optimierungsmethoden vorgestellt.

5 Grundlegende Aufgaben zur Anwendung der CRF

Ein Rückblick auf Kapitel 3.2 zeigt noch einmal die drei Aufgaben auf, die von HMM zu lösen sind. Natürlich sind diese Aufgaben auch für CRF relevant und lösbar. Sicherlich müssen die Aufgaben leicht modifiziert werden, was der Tatsache entspricht, dass CRF sich von HMM in den schon erwähnten Bereichen unterscheiden. Somit werden die drei Aufgaben der HMM zu drei Aufgaben für CRF, die wie folgt lauten und in Kapitel 5.1, 5.2 sowie 5.3 erläutert werden:

1. Für eine gegebene Beobachtungssequenz $\vec{x}$, gegebene Labelsequenz $\vec{y}$, die Potentialfunktionen $\{\Phi_k \mid \forall k\}$ und Gewichtsvektor $\vec{\lambda}$ soll die Wahrscheinlichkeit $P(\vec{y} \mid \vec{x}, \vec{\lambda})$ berechnet werden.
2. Für eine gegebene Beobachtungssequenz $\vec{x}$, Potentialfunktionen $\{\Phi_k \mid \forall k\}$ und Gewichtsvektor $\vec{\lambda}$ soll die beste Labelsequenz $\vec{y}$ gefunden werden, wobei die beste Labelsequenz $\vec{y}$ die Wahrscheinlichkeit $P(\vec{y} \mid \vec{x}, \vec{\lambda})$ maximiert.
3. Der Gewichtsvektor $\vec{\lambda}$ soll so eingestellt werden, dass die Wahrscheinlichkeit $P(\vec{y} \mid \vec{x}, \vec{\lambda})$ bei gegebener Beobachtungssequenz $\vec{x}$, gegebener Labelsequenz $\vec{y}$ und den Potentialfunktionen $\{\Phi_k \mid \forall k\}$ maximiert wird.

5.1 Wahrscheinlichkeitsberechnung mittels Matrizen

Zur Lösung der ersten Aufgabe und somit zur Berechnung der bedingten Wahrscheinlichkeit bedient man sich Matrizen. Und zwar wird für jede Stelle i (und eine weitere (T+1)) einer Beobachtungssequenz $\vec{x}$ eine Matrix der Form $|Y| \times |Y|$ erstellt, wobei Y wie schon bei den MP definiert das Zustandsalphabet darstellt. Hierdurch können alle möglichen Transitionen für jede Stelle der Beobachtungssequenz abgedeckt werden. Die Matrizen sind definiert durch

$$M_i(\vec{x}) = [M_i(y', y \mid \vec{x})], \text{ wobei}$$

$$M_i(y', y \mid \vec{x}) = \exp\left(\sum_k \lambda_k \Phi_k(y', y, \vec{x}, i)\right)$$

Formel 29: Matrizen zur Berechnung der Wahrscheinlichkeit bei Conditional Random Fields

Im Gegensatz zu generativen Modellen muss man bei CRF nicht über alle möglichen Beobachtungssequenzen iterieren sondern kann diese Matrizen direkt aus den Trainingsdaten mithilfe des Vektors $\vec{\lambda}$ füllen. Für jede Beobachtung werden also alle möglichen Potentialfunktionen betrachtet und in der Matrix gespeichert.

Der Normalisierungsfaktor Z für die bedingte Wahrscheinlichkeit ergibt sich dann durch:

$$Z(\vec{x}, \vec{\lambda}) = (M_1(\vec{x}) M_2(\vec{x}) ... M_{T+1}(\vec{x}))$$

Formel 30: Berechnung des Normalisierungsfaktors mithilfe von Matrizen

wobei Stelle 0 der Beobachtungssequenz ein Startsymbol und Stelle T+1 ein Stopsymbol enthält.

Entsprechend Formel 23 berechnet diese Formel alle aktiven Potentialfunktionen für alle möglichen Labelsequenzen.

Die bedingte Wahrscheinlichkeit ergibt sich durch:

$$P(\vec{y} \mid \vec{x}, \vec{\lambda}) = \frac{\prod_{i=1}^{T+1} M_i(y_{i-1}, y_i \mid \vec{x})}{Z(\vec{x}, \vec{\lambda})}$$

Formel 31: Berechnung der bedingten Wahrscheinlichkeit mithilfe von Matrizen

Es ist darauf zu achten, dass bei der Berechnung der bedingten Wahrscheinlichkeit für eine bestimmte Labelsequenz nur die Einträge aus den Matrizen genommen werden, die den Elementen beziehungsweise den Transitionen der Labelsequenz entsprechen. Mit den Matrizen kann man somit die bedingten Wahrscheinlichkeiten aller möglicher Labelsequenzen berechnen. Somit hat man mithilfe von Matrizen ein effizientes Verfahren gefunden, das Formel 22 berechnen kann.

5.1.1 Beispielberechnung mithilfe von Matrizen

Um die Vorgehensweise zu verdeutlichen, soll sie nun an einem Beispiel dargestellt werden:

Man gehe davon aus, dass die zu untersuchende Beobachtungssequenz

$\vec{x}$:= Felix geht nach Hamburg

sei. Die in diesem Fall zu untersuchende Labelsequenz sei:

$\vec{y}$:= [PER] [-] [-] [LOC]

Um der Einfachheit halber seien drei triviale Potentialfunktionen gegeben:

$f_1 = 1$, wenn y_{i-1} = [-], y_i = [LOC], x_i = „Hamburg“

$f_2 = 1$, wenn y_{i-1} = [-], y_i = [PER], x_i = „Felix“

$f_3 = 1$, wenn y_{i-1} = [PER], y_i = [LOC], x_i = „Hamburg“

Da die Sequenzen die Länge vier haben, werden fünf Matrizen der Form von Tabelle 1 erstellt.

	[LOC]	[PER]	[-]
[LOC]			
[PER]			
[-]			

Tabelle 1: Beispiel-Tabelle zur Berechnung der bedingten Wahrscheinlichkeit

Anschließend wird jede Matrix entsprechend ihres Index gefüllt. Nimmt man die erste Stelle der Sequenzen, so entsteht eine Matrix der Form von Tabelle 2. Zur Erläuterung: man befindet sich an Stelle i=1. Außerdem muss man alle Felder der Matrix betrachten. Allerdings gibt es nur eine Potentialfunktion, die relevant ist, also für Beobachtung „Felix“ ein Resultat liefert. Diese Potentialfunktion ist Funktion f_2. Diese Potentialfunktion liefert 1, wenn das aktuelle Label [PER] sowie das Vorgängerlabel [-] ist. Somit befindet sich nur an dieser Stelle der Matrix ein e^1.

	[LOC]	[PER]	[-]
[LOC]	1	1	1
[PER]	1	1	e^1
[-]	1	1	1

Tabelle 2: Matrix für Stelle 1 der Beispiel-Sequenz

Da die Beobachtungen an Stelle 2 („geht“) sowie 3 („nach“) nicht in den gegebenen Potentialfunktionen auftauchen, sind die Matrizen dieser Stellen lediglich 1-Matrizen. Auch die Matrix für Stelle 5 ist eine 1-Matrix. Die Beobachtung an Stelle 4 („Hamburg“) taucht allerdings in zwei Potentialfunktionen auf, und zwar in Funktion f_1 und f_3. Potentialfunktion f_1 liefert eine 1, sofern das aktuelle Label [LOC] und das Vorgängerlabel [-] ist. Potentialfunktion f_3 liefert 1, bei [LOC] als aktuellem und [PER] als Vorgängerlabel.

Es entsteht somit Tabelle 3.

	[LOC]	[PER]	[-]
[LOC]	1	e^1	e^1
[PER]	1	1	1
[-]	1	1	1

Tabelle 3: Matrix für Stelle 4 der Beispiel-Sequenz

Betrachtet man Formel 31 so erhält man folgende (unnormalisierte) bedingte Wahrscheinlichkeit:

$$P^*(\vec{y} \mid \vec{x}, \vec{\lambda}) = \prod_{i=1}^{T+1} M_i(y_{i-1}, y_i \mid \vec{x}) = e^2$$

Zur konkreteren Erläuterung werden noch einmal alle Einzelschritte betrachtet:

$$P^*(\vec{y} \mid \vec{x}, \vec{\lambda}) = M_1(y_0, y_1 \mid \vec{x}) * M_2(y_1, y_2 \mid \vec{x}) * M_3(y_2, y_3 \mid \vec{x}) * M_4(y_3, y_4 \mid \vec{x}) * M_5(y_4, y_5 \mid \vec{x})$$

$$M_1([start],[PER] \mid "Felix") = e^1$$

$$M_2([PER],[-] \mid "geht") = 1$$

$$M_3([-],[-] \mid "nach") = 1$$

$$M_4([-],[LOC] \mid "Hamburg") = e^1$$

$$M_5([LOC],[end] \mid stop) = 1$$

Wie schon vorher erwähnt wurde, betrachtet man bei der Bestimmung der unnormierten bedingten Wahrscheinlichkeit nur die jeweils benachbarten Label. In diesem konkreten Beispiel wird das Feld [PER]x[LOC] aus Tabelle 3 überhaupt nicht betrachtet, da es in der betrachteten Labelsequenz nicht auftaucht. Die Matrizen dienen schließlich dazu, alle möglichen Potentialfunktionsbelegungen abzudecken.

Und man benötigt sie auch zur Berechnung von Z, was man in Formel 30 sieht. Da man für die Berechnung von Z schlicht alle Matrizen miteinander multipliziert, erhält man hier das Ergebnis

$Z(\vec{x}, \vec{\lambda}) = e^3$, da es in allen Matrizen nur drei Belegungen e^1 gibt.

Dadurch erhält man als bedingte Wahrscheinlichkeit

$$P(\vec{y} \mid \vec{x}, \vec{\lambda}) = \frac{P^*(\vec{y} \mid \vec{x}, \vec{\lambda})}{Z(\vec{x}, \vec{\lambda})} = \frac{e^2}{e^3} = e^{-1} \approx 37\%$$

5.2 Viterbi-Algorithmus für CRF

Der Viterbi-Algorithmus für CRF arbeitet im Grunde genauso, wie für HMM. Allerdings werden die soeben eingeführten Matrizen zur Berechnung benutzt. Analog zum Vorgehen für HMM wird nun auch hier der Viterbi-Algorithmus vorgestellt:

1. Schritt: Initialisierung

   $\delta_1(i) = 1 \quad wenn \quad y_1 = q_i \quad sonst \quad 0$

   $\psi_1(i) = 0$

2. Schritt: Rekursion

   $for \quad 2 \le t \le T, \quad 1 \le j \le N$

   $$\delta_t(j) = \max_{1 \le i \le N} \left[\delta_{t-1}(i) M_t(q_i, q_j \mid \vec{x}) \right]$$

   $$\psi_t(j) = \arg\max_{1 \le i \le N} \left[\delta_{t-1}(i) M_t(q_i, q_j \mid \vec{x}) \right]$$

3. Schritt: Terminierung

$$P^* = \max_{1 \le i \le N} [\delta_T(i)]$$

$$i_T^* = \arg\max_{1 \le i \le N} [\delta_T(i)]$$

4. Schritt: Backtracking

   $for \quad t = T-1, T-2, ..., 1$

   $i_t^* = \psi_{t+1}(i_{t+1}^*)$

Formel 32: Viterbi-Algorithmus für CRF

5.3 Trainieren des Gewichtsvektors

Ruft man sich Formel 28 ins Gedächtnis, so wird deutlich, dass ein naives Vorgehen unattraktiv ist: man müsste über alle möglichen Labelsequenzen aufsummieren und hätte bei einer Sequenzlänge von T eine Anzahl von $|Y|^T$ Labelsequenzen.

Man bedient sich allerdings eines Tricks, indem man die Formel 28:

$$E_{p(Y^T|\vec{x}^{(h)},\vec{\lambda})}\left[F_k(Y^T,\vec{x}^{(h)})\right] = \sum_{\vec{y}\in Y^T} P(\vec{y}\,|\vec{x}^{(h)},\vec{\lambda}) F_k(\vec{y},\vec{x}^{(h)})$$

umformt. Anstatt über alle möglichen Labelsequenzen aufzusummieren, betrachtet man für jede Stelle der Labelsequenz jede mögliche Transition. Anstatt der globalen Potentialfunktion über komplette Labelsequenzen benutzt man nur die Potentialfunktionen über einzelne Transitionen:

$$E_{p(Y^T|\vec{x}^{(h)},\vec{\lambda})}\left[F_k(Y^T,\vec{x}^{(h)})\right] = \sum_{i=1}^{T}\sum_{q',q} P(y_{i-1}=q', y_i=q \mid \vec{x}^{(h)},\vec{\lambda})\Phi_k(q',q,\vec{x}^{(h)},\vec{\lambda})$$

Formel 33: Erwartungswert für jede Potentialfunktion bezogen auf das CRF-Modell (vereinfacht)

Des Weiteren benutzt man zur Berechnung von $P(y_{i-1}=q', y_i=q \mid \vec{x}^{(k)},\vec{\lambda})$ dynamische Programmierung, indem man forward- und backward-Variablen für CRF entwickelt, die den forward- und backward-Variablen bei HMM entsprechen. Hierfür macht man sich die soeben schon benutzten Matrizen zu Nutzen.

Die forward-Variable $\alpha_i(q)$ entspricht der unnormalisierten Wahrscheinlichkeit, bei Beobachtung $\langle x_1, x_2, ..., x_i \rangle$ in Zustand q zu sein. Die Variable ist wie folgt definiert:

$$\alpha_0(q) = \begin{cases} 1, & q = start \\ 0, & ansonsten \end{cases}$$

und

$$\alpha_{i+1}(q) = \sum_{q'} \alpha_i(q') M_i(q', q \mid \vec{x})$$

Die backward-Variable ist äquivalent definiert, und zwar gilt:

$$\beta_T(q) = \begin{cases} 1, & q = stop \\ 0, & ansonsten \end{cases}$$

sowie

$$\beta_{i-1}(q') = \sum_{q} M_i(q', q \mid \vec{x})\beta_i(q)$$

Somit lässt sich die Wahrscheinlichkeit $P(y_{i-1} = q', y_i = q \mid \vec{x}^{(k)}, \vec{\lambda})$ wie folgt darstellen:

$$P(y_{i-1} = q', y_i = q \mid \vec{x}^{(h)}, \vec{\lambda}) = \frac{\alpha_{i-1}(q')M_i(q', q \mid \vec{x}^{(h)})\beta_i(q)}{Z(\vec{x}, \vec{\lambda})}$$

Dadurch ergibt sich ein effizientes Verfahren zur Berechnung des Gradienten des log-likelihood-Wertes:

$$\nabla L(\lambda_k) = \sum_{h}\left[E_{\tilde{p}(\vec{y}^{(h)}, \vec{x}^{(h)})}[F_k(\vec{y}^{(h)}, \vec{x}^{(h)})] - \sum_{k}\sum_{i=1}^{T}\sum_{q',q} \frac{\alpha_{i-1}(q')M_i(q', q \mid \vec{x}^{(h)})\beta_i(q)}{Z(\vec{x}, \vec{\lambda})} \Phi_k(q', q, \vec{x}^{(h)}, \vec{\lambda}) \right]$$

Formel 34: Effizientes Verfahren zur Berechnung des Gradienten des log-likelihood-Wertes

Nach Berechnung des Gradienten muss mit seiner Hilfe Lambda optimiert werden. Optimierungsmethoden hierzu werden im nächsten Kapiteln vorgestellt.

6 Optimierungsmethoden für CRF-Parameter

Zum Trainieren von CRF wird jeweils der log-likelihood-Wert anhand der aktuellen Lambda-Werte bestimmt. Anhand des log-likelihood-Wertes wird dann der Lambda-Wert so verändert, dass sich daraus ein neuer, optimalerer log-likelihood-Wert berechnen lässt. Das ganze Verfahren wird solange wiederholt, bis eine Abbruchbedingung erreicht ist. In diesem Kapitel sollen nun einige Optimierungsmethoden für Lambda vorgestellt werden.

6.1 Iterative Scaling

Beim *iterative scaling* geht man so vor, dass in jedem Schritt der Methode die Modellparameter so eingestellt werden, dass die resultierenden neuen Parameter näher an der maximum likelihood-Lösung liegen:

$\lambda_k \leftarrow \lambda_k + \delta\lambda_k$

Die von [18] vorgestellten Techniken *Generalised Iterative Scaling* (GIS) und *Improved Iterative Scaling* (IIS) bauen auf diesem Prinzip auf.

Die grundsätzliche Annahme beim iterative scaling ist die, dass man ein Modell $P(\vec{y} \mid \vec{x}, \vec{\lambda})$ hat, wobei neue Parameter $\vec{\lambda} + \vec{\Delta}$ zu finden sind, mit $\vec{\Delta} = (\delta\lambda_1, \delta\lambda_2, ...)$. Diese neuen Parameter sollen, wie schon gesagt, so beschaffen sein, dass sie den neuen log-likelihood-Wert erhöhen. Ein weiterer Punkt ist jedoch der, dass sie den log-likelihood-Wert maximal erhöhen sollen, es sollte also kein $\vec{\lambda} + \vec{\Delta}_*$ geben, das einen höheren log-likelihood-Wert ergibt als $\vec{\lambda} + \vec{\Delta}$. Dieses Verfahren wird dann so lange angewandt, bis es konvergiert.

Um die optimalen neuen Parameter zu finden, erstellt man sich eine Hilfsfunktion, die eine untere Schranke für den log-likelihood-Gewinn darstellt. Um diese Schranke herzuleiten, muss man vorerst den log-likelihood-Gewinn formal definieren:

$$L(\vec{\lambda} + \vec{\Delta}) - L(\vec{\lambda}) = \sum_{h \in E} \left[\sum_{i=1}^{T+1} \sum_{k} \lambda_k \Phi_k \left(y_{i-1}^{(h)}, y_i^{(h)}, \vec{x}^{(h)}, i \right) \right] -$$

$$\log \left(\sum_{h \in E} \tilde{p}(\vec{x}^{(h)}) p(\vec{y}^{(h)} \mid \vec{x}^{(h)}, \vec{\lambda}) \sum_{i=1}^{T+1} \exp \sum_{k} \left(\delta\lambda_k \Phi_k ({y^{(h)}}_{i-1}, {y^{(h)}}_i, \vec{x}^{(h)}, i) \right) \right)$$

Formel 35: log-likelihood-Gewinn (Schritt 1/3)

Folgt man der Definition von [8], muss man nun folgende Ungleichung anwenden:

$\log x \leq x - 1$, für alle x>0

Formel 36: Hilfs-Ungleichung zur Berechnung des log-likelihood-Gewinns

Somit löst man den logarithmischen Term in Formel 35 auf und erhält Formel 37. Der (untere) Term in Formel 37, ist nun größer oder gleich dem ursprünglichen logarithmischen Term. Daher ist der log-likelihood-Gewinn größer (oder gleich) der rechten Seite der Ungleichung.

$$L(\vec{\lambda}+\vec{\Delta})-L(\vec{\lambda}) \geq \sum_{h\in E}\left[\sum_{i=1}^{T+1}\sum_{k}\lambda_k \Phi_k\left(y_{i-1}^{(h)}, y_i^{(h)}, \vec{x}^{(h)}, i\right)\right] +$$
$$1-\left(\sum_{h\in E}\tilde{p}(\vec{x}^{(h)})p(\vec{y}^{(h)} \mid \vec{x}^{(h)}, \vec{\lambda})\sum_{i=1}^{T+1}\exp\sum_{k}\left(\delta\lambda_k \Phi_k(y^{(h)}{}_{i-1}, y^{(h)}{}_i, \vec{x}^{(h)}, i)\right)\right)$$

Formel 37: log-likelihood-Gewinn (Schritt 2/3)

Anschließend benutzt man eine Zusatzfunktion
$V(\vec{x}^{(h)}, \vec{y}^{(h)})$, die in 6.1.1 und 6.1.2 jeweils verschiedene Formen hat. Nun ergibt sich:

$$L(\vec{\lambda}+\vec{\Delta})-L(\vec{\lambda}) \geq \sum_{h\in E}\left[\sum_{i=1}^{T+1}\sum_{k}\lambda_k \Phi_k\left(y_{i-1}^{(h)}, y_i^{(h)}, \vec{x}^{(h)}, i\right)\right] +$$
$$1-\left(\sum_{h\in E}\tilde{p}(\vec{x}^{(h)})p(\vec{y}^{(h)} \mid \vec{x}^{(h)}, \vec{\lambda})\sum_{i=1}^{T+1}\exp\sum_{k}\left(\delta\lambda_k\left[\frac{\Phi_k(y^{(h)}{}_{i-1}, y^{(h)}{}_i, \vec{x}^{(h)}, i)}{V(\vec{x}^{(h)}, \vec{y}^{(h)})}V(\vec{x}^{(h)}, \vec{y}^{(h)})\right]\right)\right)$$

Formel 38: log-likelihood-Gewinn (Schritt 3/3)

Aufgrund der Jensen-Ungleichung, die folgendes besagt:

$$\exp\left(\sum_k t_k \alpha_k\right) \leq \sum_k t_k \exp(\alpha_k)$$

Formel 39: Jensen's Ungleichung

ergibt sich als Hilfsfunktion und untere Grenze für den log-likelihood-Gewinn schließlich:

$$\mathrm{A}(\vec{\lambda}, \vec{\Delta}) \hat{=} \sum_{h\in E}\left[\sum_{i=1}^{T+1}\sum_{k}\lambda_k \Phi_k\left(y_{i-1}^{(h)}, y_i^{(h)}, \vec{x}^{(h)}, i\right)\right]$$
$$+1-\sum_{h\in E}\tilde{p}(\vec{x}^{(h)})p(\vec{y}^{(h)} \mid \vec{x}^{(h)}, \vec{\lambda})\left[\sum_{i=1}^{T+1}\sum_{k}\left(\frac{\Phi_k(y^{(h)}{}_{i-1}, y^{(h)}{}_i, \vec{x}^{(h)}, i)}{V(\vec{x}^{(h)}, \vec{y}^{(h)})}\right)\exp\left(\delta\lambda_k V(\vec{x}^{(h)}, \vec{y}^{(h)})\right)\right]$$

Formel 40: Hilfsfunktion für iterative scaling

Aus $L(\vec{\lambda}+\vec{\Delta})-L(\vec{\lambda}) \geq \mathrm{A}(\vec{\lambda}, \vec{\Delta})$ folgt, dass, sofern man ein $\vec{\Delta}$ gefunden hat, was $\mathrm{A}(\vec{\lambda}, \vec{\Delta})$ maximiert, dieses $\vec{\Delta}$ auch die Veränderung im log-likelihood-Wert maximiert. Man bestimmt nun die $\delta\lambda_k$-Werte, indem man – bis das Verfahren konvergiert – die Hilfsfunktion jeweils nach $\delta\lambda_k$ ableitet und nullsetzt. Da das neue Modell immer noch die Maximum Entropy-Bedingung erfüllen muss, ergibt sich folgende Funktion:

$$E_{\tilde{p}(\vec{x}^{(h)}, \vec{y}^{(h)})}[\Phi_k] \hat{=} \sum_{h\in E} p(\vec{x}^{(h)}, \vec{y}^{(h)})\sum_{i=1}^{T+1}\Phi_k(y_{i-1}^{(h)}, y_i^{(h)}, \vec{x}^{(h)}, i)$$
$$= \sum_{h\in E}\tilde{p}(\vec{x}^{(h)})\, p(\vec{y}^{(h)} \mid \vec{x}^{(h)}, \vec{\lambda})\sum_{i=1}^{T+1}\Phi_k(y_{i-1}^{(h)}, y_i^{(h)}, \vec{x}^{(h)}, i)\exp\left(\delta\lambda_k V(\vec{x}^{(h)}, \vec{y}^{(h)})\right) \hat{=} \frac{\partial \mathrm{A}(\vec{\lambda}, \vec{\Delta})}{\partial \lambda_k}$$

Formel 41: Bestimmung des neuen Gewichtsvektors mittels iterative scaling

Wenn man nun den Term $V(\vec{x}^{(h)}, \vec{y}^{(h)})$ definiert, kann man Formel 41 lösen. Dies geschieht in den folgenden Kapiteln.

6.1.1 Generalised Iterative Scaling (GIS)

Um nun konkret die $\delta\lambda_k$-Werte zu berechnen, muss man den Term $V(\vec{x},\vec{y})$ näher bestimmen. Im Falle des GIS sind an diesen Term zwei Bedingungen geknüpft:

1. $V(\vec{x},\vec{y}) = C$, wobei C eine Konstante ist, die dafür sorgt, dass der neue Gewichtsvektor analytisch berechnet werden kann.
2. $V(\vec{x},\vec{y}) \hat{=} \sum_{i=1}^{T+1}\sum_{k}\Phi_k(y_{i-1}, y_i, \vec{x}, i)$

Damit allerdings beide Bedingungen immer erfüllt sind, muss eine globale zusätzliche Potentialfunktion erstellt werden:

$$s(\vec{x},\vec{y}) \hat{=} C - \sum_{i=1}^{T+1}\sum_{k}\Phi_k(y_{i-1}, y_i, \vec{x}, i)$$

Formel 42: Korrekturterm für GIS

Man ersetzt nun den Term $V(\vec{x},\vec{y})$ in Formel 41 durch die Konstante C und bildet zusätzlich den Logarithmus, wodurch sich folgende Gleichung ergibt:

$$\log E_{\tilde{p}(\vec{x}^{(h)},\vec{y}^{(h)})}[\Phi_k] = \log\left[\sum_{h=1}^{|E|}\tilde{p}(\vec{x}^{(h)})\, p(\vec{y}^{(h)} \mid \vec{x}^{(h)},\vec{\lambda})\sum_{i=1}^{T+1}\Phi_k(y_{i-1}^{(h)}, y_i^{(h)}, \vec{x}^{(h)}, i)\exp(\delta\lambda_k C)\right]$$

Der rechte Teil der Gleichung entspricht bis auf den C-Term dem Erwartungswert $E_{p(\vec{y}^{(h)}|\vec{x}^{(h)},\vec{\lambda})}[\Phi_k]$. Somit lässt sich die rechte Seite schreiben als:

$$\log E_{p(\vec{y}^{(h)}|\vec{x}^{(h)},\vec{\lambda})}[\Phi_k] + \delta_k C,$$

was dazu führt, dass die Differenz zum neuen Gewichtsvektor folgender Form entspricht:

$$\delta\lambda_k = \frac{1}{C}\log\left[\frac{E_{\tilde{p}(\vec{x}^{(h)},\vec{y}^{(h)})}[\Phi_k]}{E_{p(\vec{y}^{(h)}|\vec{x}^{(h)},\vec{\lambda})}[\Phi_k]}\right]$$

Formel 43: Gewichtungsschritt für Potentialfunktion k

Wie man leicht sieht, wirkt sich die gewählte Größe für C auf die Konvergenz-Schritte aus. Während ein großes C kleinere Schritte verursacht, und es somit länger dauert, bis eine Konvergenz erreicht ist, verursachen kleine Werte für C größere Schritte, so dass das Verfahren schneller konvergiert. Dieses Verfahren ist jedoch aufgrund des oben beschriebenen Korrekturterms ineffizient, da der Korrekturterm jetzt wie jede andere Potentialfunktion anzusehen ist. Das bedeutet, dass auch für s die Formel 43 angewandt wird. In [40] wird nachgewiesen, dass diese Berechnung aufgrund der Tatsache, dass über alle möglichen Labelsequenzen aufsummiert werden muss, ineffizient ist. Außerdem werden Lafferty und Kollegen ([18]) widerlegt, die behaupten, diese Ineffizienz mithilfe dynamischer Programmierung zu vermeiden. Somit ist es nachteilig, den in [18] vorgestellten GIS-Algorithmus zum Trainieren von CRF zu benutzen.

6.1.2 Improved Iterative Scaling (IIS)

IIS ist eine Verbesserung des GIS, da der Korrekturterm für GIS nicht benötigt wird und zudem eine schnellere Konvergenz gegeben ist. IIS benutzt die Tatsache, dass Formel 41 nach einigen Umformungen polynomiell in $\exp(\delta\lambda_k)$ ist und für $\delta\lambda_k$ somit mit simplen Methoden wie der *Newton-Raphson-Methode* (siehe [41]) gelöst werden kann. Man nimmt nicht mehr an, dass Term V (die Summe der aktiven Potentialfunktionen) eine Konstante ist, sondern man approximiert den Term für jede Beobachtungs- und Labelsequenz durch die maximal positive Summe der aktiven Potentialfunktionen für die aktuelle Beobachtungssequenz. Formal schreibt man:

$V(\vec{x},\vec{y}) \approx V(\vec{x}) \hat{=} \max_{\vec{y}} V(\vec{x},\vec{y})$

Formel 41 schreibt man nun also als:

$$E_{\tilde{p}(\vec{x}^{(h)},\vec{y}^{(h)})}[\Phi_k] = \sum_{h=1}^{|E|} \tilde{p}(\vec{x}^{(h)})\, p(\vec{y}^{(h)} \mid \vec{x}^{(h)},\vec{\lambda}) \sum_{i=1}^{T+1} \Phi_k(y_{i-1}^{(h)}, y_i^{(h)}, \vec{x}^{(h)}, i) \exp\left(\delta\lambda_k V(\vec{x}^{(h)})\right)$$

Diese Gleichung kann nun als polynomiell in $\exp(\delta\lambda_k)$ beschrieben werden, wenn man annimmt, dass jedes Beobachtungs- und Labelsequenzpaar $(\vec{x},\vec{y})$ in eins von $V_{\max}$ sich nicht überlappende Teilmengen einteilen lässt. $V_{\max}$ ist hierbei der größte Wert, den V für die Beobachtungssequenzen $\vec{x}^{(*)}$ annehmen kann. Dann beschreibt man den Erwartungswert wie folgt:

$$E_{\tilde{p}(\vec{x}^{(h)},\vec{y}^{(h)})}[\Phi_k] = \sum_{m=0}^{V_{\max}} \sum_{h=1|V(x)=m}^{|E|} \tilde{p}(\vec{x}^{(h)})\, p(\vec{y}^{(h)} \mid \vec{x}^{(h)},\vec{\lambda}) \sum_{i=1}^{T+1} \Phi_k(y_{i-1}^{(h)}, y_i^{(h)}, \vec{x}^{(h)}, i) \left[\exp\left(\delta\lambda_k\right)\right]^m$$

Formel 44: Erwartungswert beim improved iterative scaling

Mit $a_{k,m}$ beschreibt man nun den Erwartungswert von Φ_k, sofern $V(\vec{x}) = m$:

$$a_{k,m} = \sum_{h=1}^{|E|} \tilde{p}(\vec{x}^{(h)})\, p(\vec{y}^{(h)} \mid \vec{x}^{(h)},\vec{\lambda}) \sum_{i=1}^{T+1} \Phi_k(y_{i-1}^{(h)}, y_i^{(h)}, \vec{x}^{(h)}, i)\, \delta\left(m, V(\vec{x}^{(h)})\right)$$

Somit lässt sich schließlich Formel 44 als polynomiell in $\exp(\delta\lambda_k)$ beschreiben:

$E_{\tilde{p}(\vec{x}^{(h)},\vec{y}^{(h)})}[\Phi_k] = \sum_{m=0}^{V_{\max}} a_{m,k} \exp\left(\delta\lambda_k\right)^m$, was sich mit Hilfe der Newton-Raphson-Methode lösen lässt.

6.2 Numerische Optimierung

In [34] wird dargestellt, dass numerische Optimierungstechniken besser zur Einstellung der Parameter bei Maximum-Entropy-Modellen sind als iterative-scaling-Methoden. Iterative-scaling hat vor allem mit vielen korrelierenden Merkmalen Probleme. In [40] werden numerische Optimierungstechniken und iterative-scaling-Methoden beim Trainieren von CRF verglichen. Hierbei benötigte IIS (6.1.2) über 188 Sekunden bis zum Finden des Maximums (auf fünf ausgewählten Sätzen aus dem CoNLL-2000 Datensatz), *Conjugate Gradient* (6.2.1) benötigte in einer Form 124,67 in anderer Form 176,55 Sekunden, *Limited memory variable metric* (6.2.2) benötigte schließlich nur 29,72 Sekunden. Numerische Optimierungsmethoden, wobei es sich

bei Conjugate gradient und limited memory variable metric handelt, sind Verfahren, die Gradienten der zu optimierenden Funktion benutzen, um die Funktion zu optimieren. Grundsätzlich unterscheidet man zwei Arten von numerischen Optimierungstechniken:

- First- und
- Second-order Techniken

Diese sollen nun vorgestellt werden.

6.2.1 First-order numerische Optimierung

First-order Optimierungstechniken bilden die erste Ableitung der zu optimierenden Funktion, um sich anhand dieses Gradienten in Richtung des Nullpunktes zu bewegen und somit das Funktionsoptimum zu erreichen. Triviale Techniken dieses Typs bewegen sich starr anhand des Gradienten. Allerdings landen solche Methoden häufig in lokalen Maxima, so dass die globalen Maxima unter Umständen nicht gefunden werden. Die so genannten *conjugate direction*-Methoden bilden einen ganzen Satz nichtleerer *conjugate sets*, mit denen die Funktion maximiert wird. Als Beispiel sei hier die *non-linear conjugate gradient*-Methode genannt, bei der jeder neue Optimierungsvektor $\vec{p}_j$ eine Linearkombination des Gradienten der Funktion sowie des vorangegangenen Optimierungsvektors $\vec{p}_{j-1}$ ist. Der Optimierungsschritt sieht nun wie folgt aus:

$\lambda_k^{(j+1)} = \lambda_k^{(j)} + \alpha^{(j)} \vec{p}_j$, wobei $\alpha^{(j)}$ die optimale Vektorlänge beschreibt.

Die Beschreibung konkreter Algorithmen zur Maximierung konvexer Funktionen wie der log-likelihood-Funktion würde hier den Rahmen sprengen. Allerdings werden in [40] der *Fletcher-Reeves* sowie der *Polak-Ribière-Positive* Algorithmus erwähnt, die diese Funktion optimieren. In der Implementierung kann man diese Algorithmen meist als Blackbox benutzen und muss sich um die Programmierung keine Gedanken machen.

6.2.2 Second-Order numerische Optimierung

Second-order Optimierungstechniken übertreffen first-order-Verfahren, indem sie zusätzlich zum Gradienten die zweite Ableitung der zu optimierenden Funktion betrachten. Approximiert wird der Optimierungsschritt $L(\vec{\lambda} + \vec{\Delta})$ durch die Taylorreihe der zweiten Ordnung:

$$L(\vec{\lambda} + \vec{\Delta}) \approx L(\vec{\lambda}) + \vec{\Delta}^T G(\vec{\lambda}) + \frac{1}{2} \vec{\Delta}^T H(\vec{\lambda}) \vec{\Delta}$$

Formel 45: Optimierung der log-likelihood Funktion mittels second-order numerischer Optimierung

$G(\vec{\lambda})$ ist der Gradient, die erste Ableitung, der zu optimierenden log-likelihood Funktion. $H(\vec{\lambda})$ ist die so genannte Hesse-Matrix, oder auch zweite Ableitung, der zu optimierenden log-likelihood Funktion.

Bildet man nun die Ableitung dieser approximierten Funktion, und setzt diese null, um sie zu optimieren, erhält man:

$$\vec{\Delta}^{(k)} = H^{-1}(\vec{\lambda}^{(k)})G(\vec{\lambda}^{(k)})$$

Formel 46: Finden der neuen Gewichte mittels Gradient und Hesse-Matrix

Obwohl dieses Verfahren schnell konvergiert, ist das Erzeugen der inversen Hesse-Matrix für Probleme wie *natural language processing* (NLP) oder auch NER sehr teuer. Daher wurden *variable-metric* oder *quasi-Newton-Methoden* entwickelt, die, anstatt mit der Hesse-Matrix zu rechnen, ausschließlich den Gradienten betrachten. Die Hauptidee der variable-metric Methoden ist es, die Hesse-Matrix der Taylor-Approximation durch $B(\vec{\lambda})$ zu approximieren. Dadurch erhält man:

$$\vec{\Delta}^{(k)} = B^{-1}(\vec{\lambda}^{(k)})G(\vec{\lambda}^{(k)})$$

Formel 47: Finden der neuen Gewichte mittels Gradient und Approximation der Hesse-Matrix

Anstatt jedoch $B^{-1}(\vec{\lambda}^{(k)})$ immer wieder neu zu bestimmen, wird die Matrix aktualisiert, indem man nur den aktuellen und den Vorgänger-Gradienten benutzt:

$$\vec{\Delta}^{(k-1)} = B^{-1}(\vec{\lambda}^{(k)})\left(G(\vec{\lambda}^{(k)}) - G(\vec{\lambda}^{(k-1)})\right)$$

Formel 48: Vereinfachte Berechnung der approximierten Hesse-Matrix durch die neuen Gewichte im Vorgänger-Zeitpunkt sowie den Vorgänger- sowie aktuellen Gradienten

Diese Approximation gewährleistet eine schnellere Konvergenz als die konservativen Newton-Methoden.

Es besteht allerdings immer noch das Problem, die Matrix zu speichern beziehungsweise im Speicher zu halten. Bei CRF ist es inakzeptabel, die komplette Matrix im Speicher zu halten. Daher entwickelte man eine Approximation der Hesse-Matrix, die nur eine gewisse Anzahl der Parameter beinhalten. Diese Methoden nennt man limited-memory variable-metric. Anstatt alle Iterationen zu speichern, hält man immer nur m Iterationen im Speicher. Aus Formel 48 folgt dann, dass man nur folgende Paare im Speicher halten muss: $\left\{\vec{\Delta}^{(i)}, G(\vec{\lambda}^{(i)}) - G(\vec{\lambda}^{(i-1)}) \mid i = k-m,...,k-1\right\}$, um die Approximation der Hesse-Matrix zu bestimmen. Mit jeder neuen Iteration fällt das älteste Paar aus dem Speicher. Laut [40] ist ein Wert für m zwischen drei und 20 ausreichend, um gute Ergebnisse zu erzielen.

7 CRF außerhalb der NER

Natürlich ist die Benutzung von CRF nicht auf die NER beschränkt. Grundsätzlich sind CRF auf alle möglichen sequentiellen Daten anwendbar. Zudem sind CRF nach Definition auf alle Graphen anwendbar.

Hier sollen nun einerseits die Möglichkeiten für die Nutzung von CRF außerhalb der NER sowie andererseits die Weiterentwicklung der CRF aufgezeigt werden.

7.1 Nutzungsmöglichkeiten von CRF

CRF werden in verschiedenen Bereichen benutzt, in denen Sequenzen analysiert oder vorhergesagt werden sollen. Dies umfasst Textverarbeitung, Bioinformatik und zum Beispiel auch maschinelles Sehen (vergleiche [35]). Beispielhaft sollen nun einige der Anwendungen und Anwendungsgebiete für CRF vorgestellt werden.

Sha und Pereira ([34]) benutzen CRF für das so genannte *Shallow parsing*, wodurch die Worte im Text mit ihren grammatikalischen Einheiten versehen werden.

Das System von Culotta und Kollegen ([7]) untersucht zuerst den Mail-Posteingang eines Benutzers und findet Personennamen. Daraufhin werden dann Webseiten gefunden, die diese Personennamen enthalten. Mittels CRF wird schließlich die Webseite durchsucht, um Kontaktdaten (Adresse, Telefonnummer, etc.) für den aktuell betrachteten Personennamen zu finden. Mit diesen Kontaktdaten wird dann das Adressbuch des Benutzers gefüllt. Zusätzlich findet das System neue Personen, um ein soziales Netzwerk für den Benutzer aufzubauen.

Die Gruppe um Yejin Choi ([4]) nutzt CRF, um in Texten die Meinungsquellen zu finden. Meinungsquellen sind hierbei Personen oder Objekte, die eine Meinung haben oder Meinungen beeinflussen. Dieses System dient schließlich als Verarbeitungsschritt für Systeme, die Fragen beantworten, die Meinungen betreffen – wie zum Beispiel „Was denkt [A] über [B]?“.

Die Tatsache, dass in vielen asiatischen Sprachen, wie zum Beispiel im Chinesischen, die geschriebenen Worte nicht durch Leerzeichen voneinander getrennt werden, veranlasste die Gruppe um Fuchun Peng zu einem System zur Segmentierung und Entdeckung neuer Worte im Chinesischen mit CRF([25]).

Ariadna Quattoni und andere benutzen CRF zur Erkennung von Objekten in Bildern ([26]).

Gregory und Altun wiederum verwenden CRF, um in gesprochenem Englisch die Akzentuierung beziehungsweise den Taktabstand zwischen Worten vorherzusagen ([12]), wobei ihr System einerseits für die Spracherkennung als auch für die Computer-Spracherzeugung sinnvoll ist.

7.2 Weiterentwicklung der CRF

Im Bereich der CRF gibt es bereits eine Vielzahl von Weiterentwicklungen. Exemplarisch sollen hier einige vorgestellt werden, wobei zwischen grundsätzlichen Entwicklungen und internen Veränderungen unterschieden werden soll.

7.2.1 Interne Veränderungen

Interne Veränderungen sollen die Erweiterungen der traditionellen CRF abdecken, wobei versucht wird, durch diese Veränderungen Verbesserungen entweder in der Laufzeit oder in der Güte zu erzielen.

Die aktuellste Veränderung – vor allem im Vergleich zur in dieser Diplomarbeit benutzten Implementierung – ist die von Vishwanathan und Kollegen, die die quasi-Newton-Methode (L-BFGS) zum Trainieren von CRF durch eine stochastische Gradientenoptimierungsmethode ersetzen ([38]). Zwar bessert sich nicht die Güte der CRF, allerdings ist die Konvergenz viel schneller erreicht.

7.2.2 Grundsätzliche Entwicklungen

Grundsätzliche Entwicklungen sollen Entwicklungen darstellen, die weiter gehen als die in dieser Arbeit gehende Definition von der Funktion der CRF.

In [36] präsentieren Sutton und McCallum die so genannten *Dynamic Conditional Random Fields* (DCRF), die *Dynamic Bayesian Networks* (DBN) mit CRF verknüpfen. Die Vorteile beider Verfahren sollen auf diese Weise kombiniert werden. Da HMM eine Spezialform der DBN sind, kann man sich ein DCRF wie ein CRF vorstellen, das mit versteckten Zuständen arbeitet.

Sarawagi und Cohen haben die *semi-markov-CRF* entwickelt ([32]), die eine Segmentierung der Beobachtungssequenz vornimmt. Anschließend werden diese Untersequenzen und nicht einzelne Elemente der Beobachtungssequenz mit - zum Beispiel - NE-tags versehen. Das Verfahren benötigt zwar eine längere Laufzeit, liefert aber Verbesserungen in der erzielten Güte.

In [19] werden *Kernel-CRF* (KCRF) vorgestellt. Diese verknüpfen die Stärken von DBN, Kernel-Maschinen sowie diskriminativer linearer Verfahren (zum Beispiel SVM). Der Ansatz versucht eher graphentheoretisch vorzugehen, als stur wie beim ‚sliding window'-Ansatz Stück für Stück die Sequenz abzuarbeiten. Allerdings hängen gute Ergebnisse stark davon ab, dass der für die Daten korrekte Kernel benutzt wird.

Von Zhu und anderen wird das *zweidimensionale CRF* (2DCRF) vorgestellt ([44]), das grundsätzlich zur Erfassung so genannter *WebObjects* aus dem Internet konzipiert ist. WebObjects sind die zusammengefassten Informationen über zum Beispiel Personen, Organisationen oder Objekten. Da die Informationen größtenteils auf Webseiten vorliegen, wurde die traditionelle CRF erweitert, um nicht nur eindimensionale (links/rechts) sondern auch zweidimensionale Nachbarschaftsbeziehungen („Text unter Bild") zu verarbeiten. In Versuchen mit Daten, die auch – aber nicht nur – zweidimensionale Nachbarschaftsbeziehungen aufweisen, ist die Güte der 2DCRF besser als die der traditionellen CRF.

8 Benutzte Implementierung

8.1 Einführung

Als grundlegende Java-Implementierung wurden die Klassen von Sunita Sarawagi von der IIT Bombay benutzt. Diese Implementierung liefert bereits viele Algorithmen wie den Forward-Backward- (3.2.1) sowie den Viterbi-Algorithmus (3.2.2). Des Weiteren sind auch schon die nötigen Datenstrukturen zur Verarbeitung der Matrizen vorhanden, die eine effiziente Bearbeitung erst möglich machen. Grundsätzlich richtet sich die Implementierung nach [34], allerdings werden auch semi-markov CRF ([32]) unterstützt. Zwar werden semi-markov CRF in dieser Arbeit erläutert (siehe 7.2), jedoch werden sie nicht in den späteren Versuchen eingesetzt, da diese Arbeit auf die ursprünglichen CRF beschränkt ist. Für die Zukunft ist die Untersuchung der semi-markov CRF jedoch denkbar und wünschenswert.

Als Referenz sei hier die *CRF Project page* ([6]) genannt, auf der der grundlegende Quellcode sowie eine einleitende Dokumentation zugänglich ist.

Als nächstes soll versucht werden, darzustellen, inwiefern die schon vorgestellten theoretischen Grundsätze implementierungstechnisch umgesetzt wurden, und an welchen Stellen unter Umständen Handlungsbedarf beziehungsweise Optimierungsmöglichkeiten bestehen.

8.2 Trainieren eines Modells

Ein Modell, das im späteren Verlauf eines Versuches – beim Anwenden auf einen Testdatensatz – wieder verwendet werden soll, besteht in der vorliegenden Implementierung aus zwei Komponenten:

1. dem *Feature-Modell*,
2. dem *CRF-Modell*.

Das Feature-Model enthält alle Potentialfunktionen, die für das CRF vorhanden sind. Das CRF-Model enthält für jede der Potentialfunktionen im Feature-Model ein Gewicht, das dem λ_k in Formel 22 entspricht.

Das Trainieren eines Modells sieht in Pseudocode ausgedrückt wie folgt aus:

```
FeatureGenImpl fgi = new FeatureGenImpl(…);
CRF crf = new CRF(…,fgi,…);
fgi.train(DataIter);
fgi.write(PotentialFunktionsDatei);
crf.train(SequenzIterator);
crf.write(CRFmodellDatei);
```

Code 1: Trainieren und herausschreiben aller Potentialfunktionen sowie ihrer Gewichte

Die Klasse `FeatureGenImpl` enthält im Grunde die Potentialfunktionen, die schließlich zum Trainieren des CRF benutzt werden sollen. Daher wird beim Aufruf des Konstruktors der Klasse `CRF` auch das Objekt der Klasse `FeatureGenImpl` mit-

übergeben. Zum Trainieren muss dem `FeatureGenImpl`-Objekt ein Objekt einer Klasse, die das Interface `DataIter` implementiert, übergeben werden. Dieses Interface modelliert die Handhabung eines zu benutzenden Datensatzes. Die implementierende Klasse, die vom Diplomanden erstellt wurde, heißt `SequenzIterator`. Der Datensatz wird in Sequenzen zerlegt, wobei ein Satz einer Sequenz entspricht. `Sequenz` ist hierbei auch wieder eine Klasse, die vom Diplomanden entwickelt wurde, und die das Interface `DataSequence` implementiert. `DataSequence` bietet primär Methoden zum Zugriff auf die Worte sowie die Markierungen der Worte eines Textes an. Die Klasse `Sequenz` bietet noch einige weitere Methoden, auf die im Laufe der Diplomarbeit noch eingegangen wird. Intern arbeitet die Klasse `FeatureGenImpl` dann mithilfe des `SequenzIterator`s den Datensatz Satz für Satz oder Sequenz für Sequenz ab. Die Methode `train()` erstellt im Grunde nur alle Potentialfunktionen. Mittels `write()` werden diese Funktionen schließlich an dem angegeben Datei-Ort gespeichert. Die `train()` und `write()` Methoden der Klasse `CRF` funktionieren ähnlich. Die Methoden beider Klassen werden in den folgenden Unterkapiteln ausführlich vorgestellt.

8.2.1 Trainieren und speichern der Potentialfunktionen

Ein Objekt der Klasse `FeatureGenImpl` enthielt anfangs eine Reihe von Features (Potentialfunktionen), die statisch hinzugefügt wurden. Allerdings musste dieser Punkt in Hinsicht auf variablere Versuche vom Diplomanden geändert werden, so dass nun mittels Parameter im Konstruktor eingestellt wird, welche Potentialfunktionen benutzt werden sollen. Sobald alle Potentialfunktionen, die benutzt werden sollen, hinzugefügt wurden, wird ein Lauf über den Datensatz durchgeführt. Für jedes besuchte Wort werden dann alle benutzten Potentialfunktionen überprüft und gegebenenfalls die entsprechende konkrete Potentialfunktion erstellt. Es wird hier klar zwischen Potentialfunktionen und konkreten Potentialfunktionen unterschieden. Eine Potentialfunktion kann man sich als die Idee oder das Interface vorstellen, wodurch dann mittels konkreter Daten die konkreten Potentialfunktionen erzeugt werden. Als Beispiel kann man die Potentialfunktion `Word-Features` anführen. Diese Potentialfunktion soll dann ihren Funktionswert liefern, wenn das im Testdatensatz aktuell betrachtete Wort bereits im Trainingsdatensatz vorkam. Diese Idee sei als Potentialfunktion bezeichnet und muss natürlich implementiert werden. Beim Lauf über die Daten wird dann für jedes Wort eine eigene konkrete Potentialfunktion angelegt. Alle konkreten Potentialfunktionen werden dann in der folgenden Form abgespeichert:

```
******* Features ***********
69306
0W_L:491514:1:1 56115
0W_tumour:1277937:4:1 40125
…
```

Abbildung 10: Format der Speicherung konkreter Potentialfunktionen

Die erste Zeile der Datei, die die Potentialfunktionen enthält, beinhaltet nur das Wort ‚Features'. Die darauf folgende Zeile enthält die Anzahl der folgenden Potentialfunktionen. Die nächsten beiden Zeilen enthalten exemplarisch jeweils eine konkrete Potentialfunktion der Art `Word-Features`. Zu lesen sind die Einträge nun wie folgt: die Nummer 0 weist darauf hin, dass die Potentialfunktion keinen Timeshift enthält. Das Wort und das korrespondierende NE-tag verweisen also auf denselben Index in der Wort- sowie in der NE-Sequenz. Das Kürzel ‚W_' weist auf eine Potentialfunktion des Typs `Word-Features` hin. Dann folgt das eigentliche Wort, das in diesem Fall ‚L' beziehungsweise ‚tumour' ist. Nach dem Doppelpunkt folgt eine ID, mit der man auf die ID schließen kann, unter der das Wort in dem Wörterbuch (*Dictionary*) zu finden ist. Nach dem nächsten Doppelpunkt folgt dann die Kennnummer für das NE-tag, das für die aktuelle Potentialfunktion gültig ist. Schließlich folgt noch die Häufigkeit des Wortes im Trainingsdatensatz, für das die Potentialfunktion erstellt wurde. Die Ziffer am Ende jeder Zeile ist eine ID, die intern zur Haltung der Potentialfunktionen benötigt wird.

Es ist eine Tatsache, dass die Potentialfunktionen das entscheidende Element der CRF sind. Die Potentialfunktionen dienen nämlich der technischen Umsetzung linguistischer Merkmale. Das bedeutet, dass hier der Ansatzpunkt zur Verbesserung der CRF zu finden ist. Zwar sind in der vorhandenen Implementierung schon einige grundlegende Potentialfunktionen bereitgestellt, allerdings reichen diese unter Umständen nicht aus. Die Überlegung liegt nahe, Potentialfunktionen zu suchen, die bessere Trainingserfolge liefern als die vorhandenen Potentialfunktionen. Bevor im kommenden Kapitel auf die vom Diplomanden entwickelten Potentialfunktionen eingegangen wird, sollen vorerst die bereits vorliegenden Potentialfunktionen vorgestellt werden.

8.2.2 Architektur der Potentialfunktionsklassen

Alle Potentialfunktionen erben jeweils von der Klasse `FeatureTypes`. Man kann im Prinzip zwei Arten von Potentialfunktionen unterscheiden: auf der einen Seite die Potentialfunktionen, die mit einem Wörterbuch (Dictionary) arbeiten, und auf der anderen Seite die Potentialfunktionen, die kein Wörterbuch benutzen. Bevor auf diesen Unterschied eingegangen wird, sollen vorerst noch die grundsätzlichen Funktionen erläutert werden, die für jede Potentialfunktion gültig sind:

- `boolean startScanFeaturesAt(DataSequence data, int prevPos, int pos)`
- `boolean hasNext()`
- `void next(FeatureImpl f)`
- `void setFeatureIdentifier(int fId, int stateId, Object name, FeatureImpl f, int count)`

`startScanFeaturesAt` ist die Methode, die dem Weg über die einzelnen Sätze entspricht. Die Sequenz – in diesem Fall ein Satz – wird ebenso wie die aktuelle Position (`pos`) und die vorherige Position (`prevPos`) übergeben. Liefert diese Methode ‚false', so wird für die aktuelle Position im aktuellen Satz keine konkrete Potential-

funktion erstellt. Das Problem, das in diesem Zusammenhang zu betrachten ist, besteht darin, dass diese Methode im Training sowie im Testfall gleich arbeiten muss. Um diesen Punkt näher zu erläutern, wird als Beispiel die Potentialfunktion `WordFeatures` exemplarisch vorgestellt. Wie oben erwähnt wurde, soll diese Potentialfunktion ihren Funktionswert liefern, wenn im Testdatensatz ein Wort auftaucht, das bereits im Trainingsdatensatz vorkam. Allerdings besteht für die implementierten Potentialfunktionen zwischen der Lern- und der Anwendungsphase kein Unterschied, so dass diese Methode die Worte des Trainingsdatensatz nicht erst mit dem Lauf über die Daten erfassen darf. Diese Daten müssen vielmehr schon vorher übergeben worden sein, und zwar geschieht dies mit den schon angesprochenen Dictionaries. Diese enthalten jedes Wort (oder Wortteil) mit den entsprechenden NE-tags, wie sie im Training vorkamen. `startScanFeaturesAt` liefert dann in dem Fall ‚true', wenn das Wort, das aktuell betrachtet wird, welches also durch den Index `pos` gekennzeichnet ist, im Dictionary vorkommt.

`hasNext` liefert jeweils dann ‚true', wenn noch eine konkrete Potentialfunktion des gleichen Potentialfunktionentyps für die aktuelle Position vorhanden ist – zum Beispiel wenn ein Wort im Dictionary mit unterschiedlichen NE-tags vorkommt.

`next` erhält als Parameter eine konkrete Potentialfunktion und erzeugt dann für diese übergebene Potentialfunktion die Bezeichnung sowie die aktuelle und unter Umständen die vorhergegangene NE-tags. Des Weiteren wird auch der Funktionswert erzeugt, so dass hier beim Trainieren die Bezeichnung für die konkrete Potentialfunktion erzeugt wird, wobei beim Testen eine bestimmte Potentialfunktion ihren Funktionswert liefert.

`setFeatureIdentifier` existiert mit mehreren Parametern und dient einerseits zur lesbaren Bezeichnung der Potentialfunktionen und andererseits zur eindeutigen, programminternen Identifizierung. Die lesbare Bezeichnung ist durch den Parameter `name` und die Identifizierung durch den Parameter `fId` gegeben. Beim Parameter `fId` ist darauf zu achten, dass dieser wirklich eindeutig ist. Vor allem bei der Erstellung der konkreten Potentialfunktionen ist auf diese Eindeutigkeit zu achten, da meist mithilfe der Identifikationsnummern der Worte aus den Dictionaries die Identifikationsnummern für die Potentialfunktionen berechnet werden. Obwohl ein Wort natürlich auch mit verschiedenen NE-tags vorkommen kann, müssen die auf diesem Wort beruhenden Potentialfunktionen auf jeden Fall unterschiedliche Identifikationsnummern haben.

Wie soeben beschrieben wurde, können Worte mit verschiedenen NE-tags vorkommen. Dies führt dazu, dass eine Potentialfunktionsart an einer aktuellen Position im Satz mehrere konkrete Potentialfunktionen erzeugen kann. Wenn zum Beispiel das Wort ‚Bad' im Text auftaucht, und im Dictionary steht das Wort ‚Bad' mit den NE-tags ‚Ort' und ‚Gerät', so werden auch zwei konkrete Potentialfunktionen für das Wort ‚Bad' erzeugt. Meist wird über alle NE-tags iteriert, die mit diesem Wort vorkommen, und nur, wenn alle Markierungen besucht wurden, liefert die Methode `hasNext` ‚false'.

Nun werden die bereits vorhandenen Potentialfunktionen vorgestellt.

8.2.3 Bereits vorhandene Potentialfunktionen

Die Implementierung bietet bereits sechs Potentialfunktionen, die für NER-Aufgaben benutzt werden können. Diese sechs Funktionen sind:

- `WordFeatures`
- `ConcatRegexFeatures`
- `EdgeFeatures`
- `EndFeatures`
- `StartFeatures`
- `UnknownFeatures`

`WordFeatures` wurden bereits zur Genüge erläutert und werden daher hier nicht mehr näher betrachtet.

`ConcatRegexFeatures` überprüfen anhand einer Liste von regulären Ausdrücken, ob diese in den betrachteten Worten vorkommen. Ist dies der Fall, so wird eine konkrete Potentialfunktion aus der Bezeichnung des regulären Ausdrucks selbst sowie des aktuellen NE-tags erstellt. Der Unterschied zu den `WordFeatures` ist also, dass bei den `ConcatRegexFeatures` keinerlei Bezug zu den Worten der Sequenz, sondern ausschließlich zu den NE-tags hergestellt wird, um herauszufinden, welche besonderen regulären Ausdrücke auf bestimmte NE hinweisen. Somit kann die Anzahl der konkreten Potentialfunktionen der Art `ConcatRegexFeatures` höchstens R*Y betragen, wobei R die Anzahl der regulären Ausdrücke und Y die Anzahl des NE-Alphabets ist.

`EdgeFeatures` betrachten ausschließlich die Transitionen zwischen den CRF-Zuständen. Für jede Transition zwischen den NE-tags, die ja den CRF-Zuständen eins-zu-eins entsprechen, wird jeweils eine konkrete Potentialfunktion erzeugt, so dass ihre Anzahl der Anzahl der CRF-Zustände oder unterschiedlicher NE-tags zum Quadrat entspricht.

`EndFeatures` erstellen jeweils konkrete Potentialfunktionen für die letzte Stelle im betrachteten Satz. Allerdings wird das Wort an sich nicht betrachtet, so dass nur die NE-tags in die Potentialfunktion mit einfließen. Also entspricht auch hier die Anzahl der konkreten Potentialfunktionen höchstens der Anzahl unterschiedlicher NE-tags.

`StartFeatures` arbeiten äquivalent zu den `EndFeatures`, jedoch werden hier ausschließlich die Stellen am Satzanfang betrachtet.

8.2.4 Aufbau eines Dictionaries

Das Dictionary war schon immer ein viel benutztes Werkzeug zur Textanalyse. Viele Methoden der Textanalyse binden zum Beispiel selbst externe Wörterbücher oder Wortlisten ein, um bessere Ergebnisse zu liefern. Bei der vorhandenen Implementierung wird allerdings nicht auf externe Wortlisten zurückgegriffen, sondern es wird jeweils aus dem Trainingsdatensatz ein Dictionary erstellt, auf das dann im Training sowie im Test zugegriffen wird. Hierzu wird jedes Wort des Trainingsdatensatzes mitsamt der Anzahl an Vorkommen mit bestimmten NE-tags gespeichert. Ein Eintrag in einer Dictionary-Datei sieht dann wie folgt aus:

```
promyelocytic 4993 0:1 1:14 2:10 3:1 4:34 5:3
```

Abbildung 11: Dictionary-Eintrag mit ID und Vorkommenshäufigkeiten

Zu lesen ist dieser Eintrag nun so, dass das Wort ‚promyelocytic' unter der ID 4993 im Dictionary gespeichert ist. Des Weiteren erfährt man, dass das Wort einmal in Zustand 0, 14-mal in Zustand 1, zehnmal in Zustand 2, einmal in Zustand 3, 34 mal in Zustand 4 und dreimal in Zustand 5 vorkommt. Der Einfachheit halber sind die NE programmintern als `Integer`-Werte kodiert. Die Potentialfunktion `WordFeatures` greift auf dieses Dictionary zurück, um für ein bestimmtes Wort die NE-tags zu finden, mit denen es im Trainingsdatensatz vorkam. Erreicht man beim Trainieren zum Beispiel das Wort ‚promyelocytic', so werden direkt alle konkreten Potentialfunktionen (sechs, für jeden Zustand eine) erzeugt. In der vorhandenen Implementierung wird das Dictionary noch zusammen mit den konkreten Potentialfunktionen abgespeichert. Der Diplomand hat allerdings das Dictionary insofern erweitert, dass nicht nur Worte sondern auch Wortbestandteile abgespeichert werden können. Um eine bessere Übersichtlichkeit zu erreichen hat der Diplomand zudem die Speicherung der konkreten Potentialfunktionen sowie der Dictionaries in unterschiedliche Dateien realisiert, was jedoch später noch näher erläutert wird.

8.2.5 Trainieren und speichern eines CRF

Dadurch, dass das `CRF`-Objekt schon bei der Generierung eine Referenz auf das `FeatureGenImpl`-Objekt erhält, hat es somit auch Zugriffsmöglichkeiten auf die Menge der konkreten Potentialfunktionen. Das Training erfolgt mit Hilfe des in Kapitel 6.2.2 beschriebenen Optimierungsverfahren. Die Bestimmung der Gewichte erfolgt als Blackbox-Verfahren, und benötigt als Input jeweils den aktuellen Gradienten. Das Verfahren terminiert entweder, wenn die Höchstzahl an Iterationen oder die Nullstelle erreicht ist. Die Höchstzahl an Iterationen ist schon bei der ursprünglichen Implementierung variabel durch den Benutzer einstellbar gewesen, wohingegen die Nullstelle mit einem Wert von 0,001 fest definiert war – dies bedeutet, dass das Verfahren stoppt, sofern die Lösung der Formel 26 den Wert 0,001 erreicht. Der Diplomand hat nach einigen Versuchen, die noch erwähnt werden, auch den Wert für die Nullstelle des Gradienten durch den Benutzer variabel einstellbar definiert, da ein höherer Wert für die Nullstelle unter Umständen eine schnellere Konvergenz bei gleichen Ergebnissen verspricht.

Zum Trainieren wird also ausschließlich die Berechnung des Gradienten benötigt, die, wenn man Formel 27 betrachtet, wiederum einerseits von den Funktionswerten über alle Beispiele sowie andererseits vom Erwartungswert über die Modellverteilung abhängt. Zur Erläuterung sei noch einmal die Definition des Gradienten gegeben:

$$\nabla L(\lambda_k) = \sum_h F_k(\vec{y}^{(h)}, \vec{x}^{(h)}) - \sum_h E_{p(Y^T|\vec{x}^{(h)},\vec{\lambda})}[F_k(Y^T, \vec{x}^{(h)})]$$

Formel 49: Gradient der bedingten Wahrscheinlichkeit in Abhängigkeit von den Funktionswerten der Beispiele wie vom Erwartungswert über die Modell-Verteilung

Betrachtet man die rechte Seite der Formel 34, so kann man den rechten Teil der Formel 49 auch wie folgt ausdrücken:

$$\sum_h E_{p(Y^T|\vec{x}^{(h)},\vec{\lambda})}[F_k(Y^T,\vec{x}^{(h)})] = \sum_k \sum_{i=1}^{T} \sum_{q',q} \frac{\alpha_{i-1}(q')M_i(q',q\mid\vec{x}^{(h)})\beta_i(q)}{Z(\vec{x},\vec{\lambda})}\Phi_k(q',q,\vec{x}^{(h)},\vec{\lambda})$$

Formel 50: Erwartungswert aller Labelsequenzen über die Modellverteilung

Das Vorgehen sieht (ausgedrückt in Pseudocode) nun wie folgt aus:

```
Do{  GradientenBestimmung(){
          for(all Sentences){
               //Backward-Scan
               //Forward-Scan
                    //visit all features for every Word
          }
     };
     LBFGS.LambdaOptimieren();
}until(LBFGS.Abbruch||MaxIters erreicht);
```

Code 2: Trainieren der Gewichte für alle vorhandenen Potentialfunktionen

Das Verfahren besteht aus zwei Schritten,

- der Gradienten-Bestimmung
- der Lambda-Optimierung

Bei der Lambda-Optimierung handelt es sich um ein limited memory quasi Newton-Verfahren (L-BFGS), was schon in Kapitel 6.2.2 erwähnt wurde. Dieses Verfahren wird wie schon angesprochen als BlackBox benutzt, da es als freies Paket vorliegt und eine gute, einfache Schnittstelle hat.

Die Gradienten-Bestimmung läuft so ab, dass zuerst über alle vorhandenen Trainingssätze iteriert wird. Über jedem Satz wird zuerst ein *//Backward-Scan* durchgeführt, durch den die β-Werte berechnet werden. Danach findet ein *//Forward-Scan* statt, durch den die α-Werte sowie die Matrizeneinträge M_i berechnet werden. Parallel wird im //Forward-Scan bei jedem Wort die komplette Menge an konkreten Potentialfunktionen überprüft. Für jede konkrete Potentialfunktion wird ein gradient-Wert verwaltet, der durch Formel 49 berechnet werden soll. Dafür wird der Wert einer konkreten Potentialfunktion, sofern diese für das aktuelle Wort erfüllt ist, zu dem gradient-Wert der konkreten Potentialfunktion addiert. Weiterhin werden für jede konkrete Potentialfunktion ein Erwartungswert-Wert und ein Normalisierungswert-Wert verwaltet. Der Erwartungswert-Wert wird um den aktuellen Erwartungswert ohne Normalisierung (vergleiche dazu Formel 50) erhöht. Der Normalisierungswert-Wert wird im Anschluss erstellt, indem alle α-Werte aufsummiert werden. Dann muss nur noch der gradient-Wert jeder konkreten Potentialfunktion um den Erwartungswert-Wert der Potentialfunktion geteilt durch den Normalisierungswert vermindert werden, was im Grunde Formel 49 entspricht. Somit hat man den Gradienten errechnet, der wiederum als Eingabe für die Optimierung von Lambda benutzt wird.

Sofern die Nullstelle oder die maximale Anzahl an Iterationen erreicht wurde, wird das Training abgebrochen und die Gewichte für Lambda werden gespeichert, so dass sie in Verbindung mit den konkreten Potentialfunktionen zum Testen benutzt werden können.

8.3 Anwenden eines gelernten Modells

Der Pseudo-Code zum Anwenden eines gelernten Modells sieht wie folgt aus:

```
fgi.read(Potentialfunktions-Datei);
crf.read(CRFmodell-Datei);
SequenzIterator seqIterTest = Text2Sequences.getSeqIter();
SequenzIterator output = new SequenzIterator();
seqIterTest.startScan();
while (seqIterTest.hasNext()) {
     Sequenz tempSeq = seqIterTest.next();
     crf.apply(tempSeq);
     output.add(tempSeq);
}
```

Code 3: Laden und anwenden aller Potentialfunktionen sowie ihrer Gewichte

Man erzeugt wie schon beim Lernen eines Modells ein `FeatureGenImpl`- sowie ein `CRF`-Objekt. Die Funktionen `read()` lesen logischerweise die gelernten Modelle ein. Mittels eines `SequenzIterator`-Objekts, das mit dem Testdatensatz initialisiert ist, kann man den Text Satz für Satz betrachten. Solange noch Sätze zu betrachten sind, benutzt man die `CRF`-Methode `apply()`, die den Sequenzen, die den Sätzen entsprechen, die wahrscheinlichsten NE-tags hinzufügt. Um auch eine weiterverarbeitbare Ausgabe zu erhalten, werden die Sätze nach Hinzufügen der NE-tags einem neuen, leeren `SequenzIterator`-Objekt hinzugefügt.

8.3.1 Finden der wahrscheinlichsten NE-Sequenz

Die wahrscheinlichste NE-Sequenz findet man natürlich - wie in den Kapiteln 3.2.2 und 5.2 beschrieben ist – mit dem Viterbi-Algorithmus. Da der Algorithmus kaum anders implementiert ist, als wie er in Kapitel 5.2 beschrieben ist, soll hier nicht näher auf die Programmierung eingegangen werden.

9 Eigene Erweiterungen der Implementierung

Bevor die Einbindung der Implementierung in die Versuchsumgebung YALE vorgestellt wird, sollen die Veränderungen an der Implementierung, die der Diplomand vorgenommen hat, vorgestellt werden. Natürlich ist das Ziel aller Veränderungen eine Verbesserung der vorliegenden Implementierung. Es können zwei mögliche Arten der Verbesserung identifiziert werden:

1. Verbesserungen in der Güte der erzielten Ergebnisse
2. Verbesserungen in der Laufzeit

Ob die nun vorgestellten Veränderungen in der Implementierung in Verbesserungen resultieren, wird in den kommenden Kapiteln vorgestellt.

Vorerst sollen jedoch Überlegungen und umgesetzte Veränderungen in der Implementierung dargestellt werden.

9.1 Veränderungen zur Verbesserung der Ergebnisgüte

Wie schon in Kapitel 8.2.1 beschrieben wurde, wurden die Potentialfunktionen als möglicher Ansatzpunkt für bessere Ergebnisse definiert. Änderungen in anderen Bereichen der vorhandenen Implementierung können höchstens eine Verbesserung der Laufzeit bewirken.

In Kapitel 2.3.2 wird das Prinzip der Generalisierung als Möglichkeit zur Minimierung von Fehlern im Bereich der NER vorgestellt. Die Überlegung liegt nun nahe, dass neue Potentialfunktionen sich also vor allem das Prinzip der Generalisierung zu Nutze machen sollten. Die vom Diplomand erstellten Potentialfunktionen sollen nun vorgestellt werden.

9.1.1 Neue Potentialfunktionen

AdditionalDictionaryFeatures

Die in Kapitel 11.2 vorgestellten Verfahren nutzen unter anderem auch externe Wortlisten (so genannte gazetteers). Das mit dem System des Diplomanden vergleichbare System, das in [33] beschrieben ist, benutzt zum Beispiel eine Wortliste aus griechischen Buchstaben, da diese auf gewisse NE hinweisen. Diese Möglichkeiten sollen auch in der Implementierung des Diplomanden geschaffen werden. Jedenfalls muss für jede externe Wortliste ein Objekt der Klasse `AdditionalDictionaryFeatures` erstellt werden. Sofern im Training dann ein Wort auftaucht, das in der entsprechenden Wortliste enthalten ist, wird eine konkrete Potentialfunktion erstellt, die mit der NE-Markierung verknüpft ist, die auch das aktuelle Wort hat. Auch hier können somit für jede Wortliste höchstens die Anzahl konkreter Potentialfunktionen erzeugt werden, die der Menge an unterschiedlichen NE entspricht.

NGramFeatures

Eine Möglichkeit der Generalisierung ist die Zerlegung eines unbekannten Wortes in N-Gramme, um durch bereits bekannte N-Gramme auf die Bedeutung des unbe-

kannten Wortes zu schließen. Diese Potentialfunktion ist so parametrisiert, dass die N-Gram-Länge variabel durch den Benutzer einstellbar ist. Diese Potentialfunktion arbeitet zudem mit einem NGram-Dictionary, was bedeutet, dass vor dem eigentlichen Training erst der komplette Trainingsdatensatz in seine N-Gramme zerlegt werden muss. Bis auf die Tatsache, dass die Worte in mehrere Dictionary-Einträge zerfallen, entspricht dieses Dictionary in seiner Form dem Dictionary für ganze Worte. Allerdings wurde das Dictionary-Konzept vom Diplomanden insofern erweitert, dass eine Klasse sowohl für Wort- als auch für N-Gram-Dictionaries benutzt wird.

Die konkreten Potentialfunktionen entsprechen dann im Grunde auch im Verhalten den `WordFeatures`, wobei natürlich nicht das Wort sondern seine NGramme auf ihr Vorkommen im Dictionary überprüft werden.

PosFeatures

POS-tags liefern grammatikalische Informationen über ein Wort. Es ist denkbar – und belegt (siehe 11.2), dass gerade bestimmte grammatikalische Formen für bestimmte NE kennzeichnend sind. Deshalb wurde die Potentialfunktion `PosFeatures` entwickelt, die ähnlich zu den `Word-` und `NGramFeatures` das Vorkommen des POS-tags des aktuellen Wortes mit den POS-tags des Trainingsdatensatzes abgleicht. Auch diese Potentialfunktion benötigt ein Dictionary, das vor dem Training der Potentialfunktion aus dem Trainingsdatensatz erstellt werden muss.

PrefixFeatures

`PrefixFeatures` entsprechen im Grunde den `NGramFeatures`, wobei natürlich jedes Wort auch nur ein Präfix enthält, im Gegensatz zu den NGrammen, von denen mehrere in einem Wort auftauchen können. Ansonsten entsprechen sich beide Potentialfunktionen. Allerdings kann diese Potentialfunktion nicht unbedingt durch die `NGramFeatures` abgedeckt werden, da man ja unterschiedliche Präfix- und NGram-Längen einstellen kann.

SuffixFeatures

`SuffixFeatures` sind in ihrer Arbeitsweise äquivalent zu den `PrefixFeatures`.

9.1.2 Einbindung externer Evidenz

Die bisher vorgestellten neuen Potentialfunktionen sind ausschließlich Veränderungen, die der internen Evidenz zuzuordnen sind. Aber natürlich sollte auch eine Möglichkeit gefunden werden, externe Evidenz in den Trainingsprozess mit einzubeziehen. Es werden nun zwei Verfahren dargestellt, die diesen Punkt beachten.

Benutzung von relativen Positionen

Bei einigen Potentialfunktionen ist es möglich, *relative Positionen* anzugeben, die dazu dienen, auf das zu betrachtende Objekt hinzuweisen. WordFeatures unterstützen zum Beispiel diese Möglichkeit. Eine relative Position von ‚-1' bedeutet dann, dass, wenn man über den aktuellen Satz wandert und sich zum Beispiel gerade an Wort Nummer drei befindet, man zwar das NE-tag an Stelle drei erhält. Diese Markierung ist jedoch dann nicht mehr mit dem Wort an Stelle drei, sondern mit dem Wort an

Stelle drei plus die relative Position – in diesem Fall zwei – verknüpft. Die Benutzung relativer Positionen wird von allen Potentialfunktionen unterstützt, die Dictionaries benutzen. Dies sind momentan die Potentialfunktionen `WordFeatures`, `PrefixFeatures`, `SuffixFeatures`, `NGramFeatures`, `AdditionalDictionaryFeatures` und `PosFeatures`. Werden relative Positionen benutzt, so muss jeweils für jede relative Position ein eigenes Dictionary erzeugt werden. Man kann nun zum Beispiel Regelmäßigkeiten erkennen wie:

‘Eine Position vor der aktuellen Position mit NE xy steht oft das Wort z’ oder ‚Zwei Positionen vor der aktuellen Position mit NE xy steht oft das Wort w’

Da diese Regelmäßigkeiten parallel auftauchen können, weil man mehrere relative Positionen auf einmal benutzen kann, würde man intuitiv schließen, dass nach der Wortsequenz ‚w z’ oft ein Wort mit NE xy folgt. Dieses Verhalten wird jedoch erst durch die Benutzung von Sequenzen unterstützt.

Benutzung von Sequenzen

Die Benutzung von Sequenzen soll explizit dafür sorgen, dass komplette Sequenzen von Worten als konkrete Potentialfunktion benutzt werden. Dieses Verfahren kann mit mehreren Sequenzen parametrisiert werden. Die Sequenzen ähneln dabei den relativen Positionen, da auch die Sequenzen in ihrer Position relativ zur aktuellen Position betrachtet werden. Möchte man zum Beispiel die beiden Positionen vor sowie nach dem aktuellen Wort, als Merkmale benutzen, muss ‚-2,-1;1,2’ als Parameter übergeben werden. Wie schon erwähnt wurde, ist der Unterschied zu den relativen Positionen der, dass ausschließlich die komplette Sequenz vor, nach oder über einem Wort als konkrete Potentialfunktion erzeugt wird, wohingegen bei den relativen Positionen jede Position eine konkrete Potentialfunktion ergibt. Auch hier müssen für die jeweiligen Sequenzen eigene Dictionaries erzeugt werden. Die unterstützten Potentialfunktionen sind vorerst nur `WordFeatures`, `PrefixFeatures`, `SuffixFeatures` und `NGramFeatures`.

9.2 Veränderungen zur Verbesserung der Laufzeit

Große Veränderungen in den implementierten Algorithmen sind nicht vorgenommen worden, da die gemachten Versuche in einigen Fällen keine zeitliche Verbesserungen versprachen und da nicht soviel Zeit zur Verfügung stand, um die korrekt implementierten Algorithmen auf ihre Laufzeit zu untersuchen, beziehungsweise sie programmiertechnisch zu verbessern.

Es wird vielmehr Wert darauf gelegt, die einzelnen NER-Schritte klar voneinander abzugrenzen. Dies geschieht vor allem durch die explizite Trennung der Vorverarbeitung von dem späteren Lern- beziehungsweise Anwendungsschritt.

9.2.1 Abtrennung der Vorverarbeitung

Gerade die Einführung der eigenen Potentialfunktionen führt zu einer Menge neuer Wortmerkmale, die es zu bearbeiten gilt. Die erste Überlegung war die, die Wortmerkmale aus dem bestehenden Wort-Dictionary zu extrahieren.

Dass dieses Unterfangen – vor allem während der Laufzeit – eine viel zu lange Rechenzeit mit sich bringt, ist verständlich. Die nächste Idee war die, für jedes zusätzliche Merkmal ein eigenes Dictionary zu erzeugen. Diese Idee wurde realisiert, wobei auf die Dictionaries noch im nächsten Kapitel eingegangen wird. Die ersten Versuche, die noch dokumentiert werden, zeigten allerdings immer noch einen Handlungsbedarf, um die enorme Bearbeitungszeit zu vermindern. Daraufhin wurde das in Tabelle 5 auf Seite 68 beschriebene Datenformat entwickelt und seine Erzeugung in die Vorverarbeitung portiert. Somit muss (nach Abschluss der Vorverarbeitung) zum Beispiel zum Erfassen des Präfixes nur noch das entsprechende Attribut in dem aktuellen Tabelleneintrag nachgeschlagen werden, was im Vergleich zu String-Operationen während der Laufzeit eindeutig den kleineren Zeitaufwand benötigt.

In diesem Kapitel sollen nun noch die Veränderungen in den Dictionaries dargestellt werden.

9.2.2 Neue Dictionaries

Im Gegensatz zur vorhandenen Implementierung wurden Dictionary und konkrete Potentialfunktionen vom Diplomanden getrennt. Ein weiterer Unterschied ist natürlich auch durch die Vielzahl neuer Dictionaries gegeben. Hatte die vorhandene Implementierung nur das reine Wort-Dictionary, so existieren nun noch weitere Dictionaries wie zum Beispiel das Prefix- oder Suffix-Dictionary. Zudem wird durch die Einführung der externen Evidenz der Umfang der eigentlichen Dictionaries nochmals vergrößert. Gibt man zum Beispiel im NERpreprocessing-Operator an, dass man drei Stellen vor, sowie drei Stellen nach dem aktuellen Wort als externe Evidenz betrachten möchte, so werden zusätzlich zu dem bereits bekannten Dictionary noch sechs weitere Dictionaries jeweils des gleichen Typs angelegt, die dann die externe Evidenz darstellen.

Als Beispiel sei hier ein Satz aus sieben Worten gegeben:

```
Felix geht schnell mit Tom ins Kino
```

Abbildung 12: Beispielsatz zur Darstellung der externen Evidenz

Befindet man sich bei der Erstellung der Wort-Dictionaries nun an Stelle vier (Wort='mit'), so wird in sieben Dictionaries, die jeweils mit der relativen Position gekennzeichnet sind, das jeweilige Wort eingetragen und mit dem NE-tag des Wortes ‚mit' verknüpft. Das Wort ‚Felix' wird zum Beispiel ins ‚-3'-Wort-Dictionary eingetragen. In Abbildung 13 sieht man zum Beispiel, dass das Wort ‚mit' mit Timeshift -1 mit der NE Nummer 3 verknüpft ist. Diese Nummer steht für die NE PERSON. Somit wurde modelliert, dass das Wort ‚mit' vor einer NE des Typs PERSON stehen kann.

Im Vergleich zum in Kapitel 8.2.4 vorgestellten Dictionary-Aufbau sieht der neue Aufbau wie folgt aus:

```
Words:45028
…
false mit 43788 1 5:1
```

```
false mit 43785 0 5:1
false mit 43782 -1 3:11
…
fWords:132420
true:2 geht schnell 119603 -2 -1 5:1
true:3 Tom ins Kino 31506 1 2 3 5:1
…
```

Abbildung 13:Neues Format der Dictionaries

Exemplarisch wurde hier nur das normale Wort-Dictionary in Verbindung mit der Benutzung von Sequenzen (vergleiche Kapitel 9.1.2) dargestellt. Die Worte mitsamt den Einträgen für die relativen Positionen werden in dem Bereich ‚Words' gespeichert. Sequenzen fallen in den Bereich ‚fWords'. Alle anderen Dictionaries wie zum Beispiel Prefix- oder Suffix-Dictionaries werden natürlich adäquat benannt.

Hinter den Überbegriffen wie ‚Words' und ‚fWords' steht dann die Anzahl der folgenden Einträge. Vor jedem Eintrag steht entweder ‚true' oder ‚false', wobei ‚false' einen „normalen" Eintrag (mit oder ohne relativer Position) bezeichnet. ‚true' bezeichnet eine Sequenz, wobei direkt hinter dem folgenden Doppelpunkt die Länge der Sequenz definiert ist. Die letzten Stellen jedes Eintrags definieren immer noch die NE-tags und ihre Häufigkeit, allerdings handelt es sich um die NE-tags der aktuellen Position. Die vorletzte Stelle enthält die relative Position beziehungsweise die relativen Positionen der Sequenz. Die Stelle davor enthält die Identifikationsnummer des Eintrags, und davor steht das Wort selbst.

‚true:2 geht schnell 119603 -2 -1 5:1' steht für das einmalige Vorkommen der Worte ‚geht schnell' vor einem Wort mit dem NE-tag Nummer 5 – in diesem Fall dem Wort ‚mit'.

9.2.3 Normalisierte Funktionswerte

Es bestand die Überlegung, anstatt der binären Funktionswerte normalisierte Funktionswerte ausgeben zu lassen. Zu Anfang der Arbeit hat der Diplomand sogar darüber nachgedacht, unnormalisierte Funktionswerte, die schlicht der Häufigkeit der Vorkommen bestimmter Elemente im Trainingsdatensatz (zum Beispiel der Häufigkeit von Worten bei `WordFeatures`) entsprechen, einzusetzen.

Allerdings wäre dieser Weg zu extrem gewesen, da bestimmte häufige Vorkommen die Trainingsphase zu sehr zu sprunghaftem Verhalten angeregt hätten.

Die Benutzung normalisierter Funktionswerte war jedoch einerseits sehr schwer umzusetzen, da bis dato nur die Häufigkeiten der einzelnen konkreten Potentialfunktionselemente und nicht die der gesamten Potentialfunktionen persistent gemacht wurden. Erste Versuche brachten zudem nicht den erhofften Erfolg, so dass die Möglichkeit, normalisierte Funktionswerte zu benutzen zwar noch existiert, aber nicht empfohlen wird.

Durch das Training der Gewichte für jede Potentialfunktion erhält man im Grunde ja auch die Gewichtung, die man durch den normalisierten Funktionswert einbringen möchte. Einzig eine schnellere Konvergenz wäre von einer Nutzung von nicht binären Funktionswerten zu erwarten.

10 CRF in YALE

10.1 Einführung in YALE

YALE ist ein Programm, das am Lehrstuhl für künstliche Intelligenz in Dortmund entwickelt wurde. Es handelt sich hierbei um eine Umgebung zur Analyse sowie Manipulation von Datensätzen.

Eine umfassende Präsentation von YALE würde den Rahmen dieser Arbeit sprengen, so dass an dieser Stelle vertiefend auf [11] verwiesen sei.

Grundsätzlich erstellt man in YALE jedoch Versuchsketten, die – einfach gesagt – einen Trainingsdatensatz einlesen, aufgrund dieses Datensatzes ein Modell lernen und schließlich dieses Modell auf einen unbekannten Testdatensatz anwenden.

Der Versuchsaufbau für einen NER-Versuch sieht wie folgt aus:

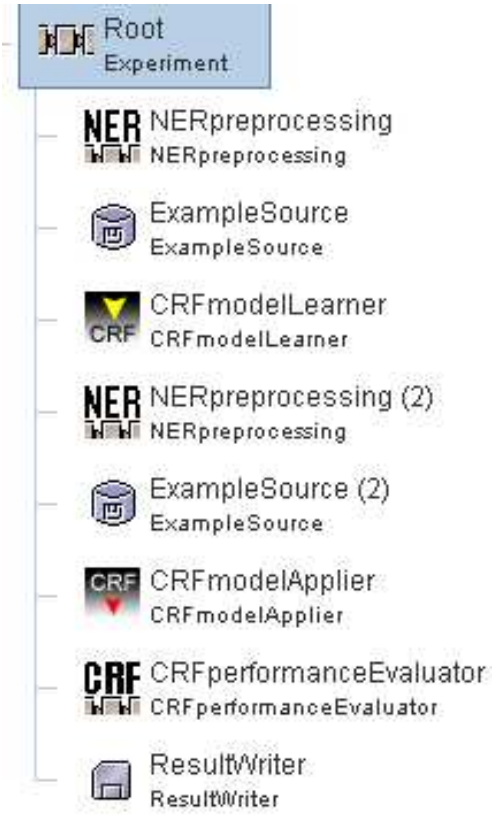


Abbildung 14: YALE-Versuch

Bis auf die Operatoren Root, ExampleSource und PerformanceWriter mussten alle Operatoren, sowie ihre Funktionalität natürlich, implementiert werden.

Die Operatoren

- *NERpreprocessing*
- *CRFmodelLearner*
- *CRFmodelApplier*
- *CRFperformanceEvaluator*

sollen in den folgenden Kapiteln näher beschrieben werden.

10.2 NERpreprocessing

Der Vorverarbeitungsoperator NERpreprocessing dient dazu, einen Datensatz in YALE einzulesen.

Man geht grundsätzlich davon aus, dass dieser Datensatz als Datei vorliegt. Daher muss man als Parameter in diesem Operator den Dateinamen der einzulesenden Datei angeben.

Weiterhin kann man mithilfe dieses Operators den eingehenden Datensatz vorverarbeiten. Als Vorverarbeitung wird hierbei das Anreichern der Daten um bereits vorhandene Merkmale gesehen. Vorhandene Merkmale sind zum Beispiel Prä- oder Suffixe eines Wortes. Diese Merkmale liegen nach der Verwendung des Operators als Attribute im Datensatz vor. Daher kann man direkter und somit schneller auf diese Merkmale zugreifen. Dies dient dazu, den Aufwand zur Extraktion der Merkmale während der Laufzeit des Programms zu vermindern.

Praktisch bedeutet dies, dass ein Datensatz, der vor der Anwendung des NERpreprocessing-Operators vielleicht nur die Worte sowie deren NE-tags enthielt, nach Benutzung des Operators für jedes Wort noch die Prä- und Suffixe enthält. Als nächstes werden alle Parameter des Operators mitsamt ihrer Bedeutung erläutert:

Parameter-Name	**Parameter-Art**	**Parameter-Funktion**
file	Datei	Datei, die den Datensatz enthält
outfileTagged	Datei	Datei, die den um alle Merkmale angereicherten Datensatz enthält
outfileUntagged	Datei	Datei, die nur die Worte des Datensatzes enthält
outXMLfile	Datei	Datei, die die Referenz auf outfileTagged sowie alle Merkmalbezeichnungen enthält
Dictionaryfile	Datei	Datei, die alle Dictionaries enthält
sampling	Boolean	Soll ein Sampling des Datensatzes erstellt werden?
samplingSize	Integer	Sofern ein Sampling erstellt werden soll, wird hier die Sample-Größe in Prozent angegeben.
cutchar	String	Hier müssen reguläre Ausdrücke angegeben werden, die im Datensatz die Worte von den NE-tags trennen.
training	Boolean	Sollen die Daten fürs Training oder fürs Testen aufbereitet werden?

External evidence	String	Hier wird kommasepariert die externe Evidenz, die benutzt werden soll, erwartet (z.B. -3,-2,-1,0,1,2).
External following evidence	String	Hier müssen die Sequenzen semikolonsepariert angegeben werden, die zum Training benutzt werden sollen (z.B. -2,-1;1,2).
Wordposition	Integer	An welcher Position befindet sich das Wort im Datensatz?
NERposition	Integer	An welcher Position befindet sich der NE-tag im Datensatz?
beginentity	String	Um welche Buchstabenfolge wird ein Start-tag einer NE erweitert (z.B. „B-„ bei iob-tagging)?
innerentity	String	Um welche Buchstabenfolge wird ein innerer tag einer NE erweitert (z.B. „I-„ bei iog-tagging)?
Prefix-Feature	Boolean	Sollen die Präfixe extrahiert werden?
prefixLength	Integer	Wie lang sollen die Präfixe sein?
Suffix-Feature	Boolean	Sollen die Suffixe extrahiert werden?
suffixLength	Integer	Wie lang sollen die Suffixe sein?
nGram-Feature	Boolean	Sollen die nGramme extrahiert werden?
nGramLength	Integer	Wie lang sollen die nGramme sein?
POS-Feature	Boolean	Sollen POS-tags extrahiert werden?
Regex-Feature	Boolean	Sollen reguläre Ausdrücke extrahiert werden?
additional Regex	List of String	Welche regulären Ausdrücke sollen zusätzlich benutzt werden?
additional Lists	List of Datei	Welche zusätzlichen Wortlisten sollen benutzt werden?
stopword	String	Welches Wort soll zur Trennung der einzelnen Sätze benutzt werden?
entity	List of String	Hier müssen alle NE-tags, die vorkommen können, angegeben werden.

Tabelle 4: Parameter für NERpreprocessing

10.2.1 Texte in YALE

Texte wurden bisher in YALE so gut wie gar nicht behandelt.

Es gibt zwar schon ein Werkzeug namens *Word Vector Tool*, das zur Verarbeitung von Texten dient. Allerdings wird bei diesem Werkzeug der sequentielle Charakter von Texten überhaupt nicht beachtet. Einzig die Analyse der Worthäufigkeiten ist mit der Methode *bag of words* des Word Vector Tools möglich.

Da YALE mit dem *attribute-value-format* arbeitet, können die Sequenzaspekte von Texten zudem nicht ohne weiteres verarbeitet werden. YALE ist auf die Verarbeitung von nicht – sequentiellen Daten ausgerichtet. Bei der Verwendung von sequentiellen Daten (wie zum Beispiel Texten) muss man vor allem zwei Punkte beachten:

- Der Erhalt der Sequentialität. Die Texte liegen in der originalen Datei zum Beispiel so vor, dass pro Zeile ein Wort mitsamt seines NE-tags vorkommt. Eine Leerzeile kennzeichnet beispielsweise das Ende eines Satzes. Um die Sequentialität zu erhalten, darf man natürlich keine Worte innerhalb eines Satzes vertauschen. Vor allem beim *Sampling* muss darauf geachtet werden, dass ausschließlich gesamte Sätze benutzt werden und nicht die interne Satzstruktur zerstört wird.
- Attribute-value-format umsetzen. Jedes Wort erhält einen Eintrag mit einem Label, das das NE-tag ist, sowie einer Anzahl von Attributen, die den im Vorverarbeitungsoperator eingestellten Merkmalen entsprechen. Wichtig ist hierbei, dass die in Datensätzen häufig verwendete Art, mit einer Leerzeile einen Satz zu beenden, im attribute-value-format nicht anwendbar ist. Dieser Fakt wird mithilfe des oben vorgestellten Stopwortes gelöst, das anstatt einer Leerzeile in die Daten eingefügt wird. Ist ein Satz beendet, wird er YALE-intern mit einem Stopwort beschlossen, und dieses Stopwort enthält auch als Label sowie in allen Attributen das Stopwort. Dies ist notwendig, um den Datensatz konsistent zu halten.

  Ein möglicher Datensatz kann YALE-intern dann wie folgt aussehen:

0word	**0prefix**	**0suffix**	**0ngrams**	**0pos**	**0regex**	**label**
…	…	…	…	…	…	…
unique	uni	ue	uni;niq;iqu;que	JJ	0:isWord;10:isAllSmallCase;11:isAlpha;12:isAlphaNumeric	5
Stopp	stop	stop	stop	stop	Stopp	stop
…	…	…	…	…	…	…

Tabelle 5: YALE-interne Datenhaltung

In diesem speziellen Fall wird nur das aktuelle Wort (die aktuelle Stelle wird mit 0 bezeichnet, daher wird bei den Wertbezeichnungen eine 0 vorangestellt) betrachtet, die externe Evidenz wird sozusagen nicht beachtet. Würde man die externe Evidenz in diesem Fall mit einbeziehen, so hätte man natürlich entsprechend mehr Attribute.

Besonders der Punkt, die Sequentialität zu erhalten, ist vor allem dann wichtig, wenn man – wie oben schon angesprochen – ein Sample aus dem Datensatz extrahieren

möchte. Aufgrund dieser Problematik wurde ein spezielles Sampling-Verfahren entwickelt, das nun vorgestellt werden soll.

10.2.2 Stratifiziertes Satz Sampling

Die Verteilung der NE-tags in Texten ist keineswegs gleichverteilt. Im Vergleich zu nicht markierten Worten, also *nicht-NE*, sind die einzelnen NE in den Texten eindeutig in der Minderheit. Dieser Tatsache muss man, sofern man nur eine Auswahl eines Textes betrachten will, auf jeden Fall Rechnung tragen. Bei Klassifikationsaufgaben würde man in diesem Fall das *stratifizierte Sampling* anwenden, da hiermit die Klassenverteilung im Datensatz verhältnismäßig auch im Sample vorliegt. Allerdings geht stratifiziertes Sampling grundsätzlich so vor, dass per Zufall einzelne Daten aus dem Datensatz gezogen werden. Dies ist natürlich bei Texten nicht möglich, da man die Struktur der Sequenzen beibehalten muss. Die Lösung dieses Problems besteht darin, nicht einzelne Worte sondern jeweils einzelne Sätze zu betrachten.

Das implementierte *stratifizierte Satz Sampling* benötigt als Eingabe einen Wert, der die prozentuale Größe des Samples im Vergleich zum originalen Datensatz vorgibt. Anhand dieses Wertes wird eine obere Grenze berechnet, die die Anzahl an gezogenen Sätzen nicht überschreiten darf. Diese Obergrenzen werden auch jeweils für die NE berechnet, so dass im späteren Sample jede NE im gleichen Verhältnis wie im originalen Datensatz vorliegt. Per Zufallsverfahren werden nun Sätze gezogen, und die Vorkommen der einzelnen NE werden vermerkt. Überschreitet nun das Vorkommen einer NE eine obere Grenze, so wird das Sampling für diesen Kandidaten abgebrochen. Solange kein Abbruch erfolgt, wird der Kandidat gespeichert. Das Verfahren wird zehnmal wiederholt, und aus den zehn Kandidaten, die dann vorliegen, wird anhand der folgenden Formel derjenige bestimmt, der dem originalen Datensatz am ähnlichsten ist.

$$Sample = \arg\min_i \left(\sum \left[\left(\frac{tags_i}{words_i} - \frac{tags}{words} \right)^2 \right] \right)$$

Formel 51: Bestimmung des besten Samples im NERpreprocessing-Operator

Jedes prozentuale Vorkommen einer NE im originalen Datensatz wird vom prozentualen Vorkommen der gleichen NE im Sample subtrahiert. Dieser Wert wird, um einen positiven Wert zu erhalten, quadriert und schließlich mit den Werten für alle anderen NE aufsummiert. Das beste Sample ist dann das, welches den niedrigsten Wert – aus Formel 51 resultierend – aufweist.

10.3 CRFmodelLearner /-Applier

Die Operatoren CRFmodelLearner sowie –Applier dienen dazu, ein CRF-Modell zu trainieren, beziehungsweise, ein gelerntes Modell anzuwenden. Aus der YALE-Versuchskette heraus erwarten diese beiden Operatoren einen Datensatz, der bereits vorverarbeitet ist. Daher muss der NERpreprocessing-Operator vor Benutzung eines dieser Operatoren angewandt worden sein. Ein weiterer wichtiger Punkt ist, dass im

CRFmodelApplier nur die Potentialfunktionen ausgewählt werden, die auch schon im CRFmodelLearner benutzt wurden. Ein Modell, das zum Beispiel als Potentialfunktionen die Präfix-Potentialfunktionen benutzt hat, kann auch nur mit der Präfix-Potentialfunktion im CRFmodelApplier benutzt werden.

Die Parameter, die bei CRFmodelLearner angegeben werden können, sind:

Parameter-Name	**Parameter-Art**	**Parameter-Funktion**
entities	Integer	Anzahl der NE-tags
epsForConvergence	Double	Wert für die Nullstelle des Gradienten (z.B. 0.1)
Dictionaryfile	Datei	Die im Training erstellte Dictionary-Datei
feature_model_file	Datei	Datei, in die die gelernten Potentialfunktionen geschrieben werden
crf_model_file	Datei	Datei, in die die gelernten Gewichte für die Potentialfunktionen geschrieben werden
tempIter	Integer	Nach dieser Anzahl von Iterationen werden die temporären Gewichte gespeichert
Binary values	Boolean	Sollen die Potentialfkt. binäre oder relative Werte liefern
External ev.	Boolean	Soll externe Evidenz berücksichtigt werden?
External evidence	String	Hier wird kommasepariert die externe Evidenz, die benutzt werden soll, erwartet (z.B. -3,-2,-1,0,1,2).
External following ev.	Boolean	Sollen Sequenzen berücksichtigt werden?
External following evidence	String	Hier müssen die Sequenzen semikolonsepariert angegeben werden, die zum Training benutzt werden sollen (z.B. -2,-1;1,2).
prefix-Features	Boolean	Sollen Präfix-Potentialfunktionen gelernt werden?
Prefix Evidence	Boolean	Sofern relative Positionen benutzt werden, kann deren Benutzung noch für jede Potentialfunktion einstellen.
suffix-Features	Boolean	(siehe prefix-Features)
Suffix Evidence	Boolean	(siehe prefix-Features)

word-Features	Boolean	(siehe prefix-Features)
Word Evidence	Boolean	(siehe prefix-Features)
gram-Features	Boolean	(siehe prefix-Features)
Gram Evidence	Boolean	(siehe prefix-Features)
fWord-Features	Boolean	(siehe prefix-Features)
fgram-Features	Boolean	(siehe prefix-Features)
fPrefix-Features	Boolean	(siehe prefix-Features)
fSuffix-Features	Boolean	(siehe prefix-Features)
start-Features	Boolean	(siehe prefix-Features)
end-Features	Boolean	(siehe prefix-Features)
edge-Features	Boolean	(siehe prefix-Features)
regex-Features	Boolean	(siehe prefix-Features)
pos-Features	Boolean	(siehe prefix-Features)
unknown-Features	Boolean	(siehe prefix-Features)
additional-Dictionary-Features	Boolean	(siehe prefix-Features)
iterations	Integer	Hier wird definiert, nach wie vielen Iterationen auf jeden Fall abgebrochen wird.
stopword	String	Das schon im Training benutzte Stopword muss hier angegeben werden.

Tabelle 6: Parameter für CRFmodelLearner

Die Potentialfunktionen wie zum Beispiel „prefix-Features“ werden im Kapitel 8.2.3 und 9.1.1 näher beschrieben.

CRFmodelApplier enthält im Grunde dieselben Parameter, wobei noch ein Parameter zum Speichern des markierten Testdatensatzes hinzukommt.

10.4 CRFperformanceEvaluator

Der Operator CRFperformanceEvaluator dient dazu, die Güte eines markierten Testdatensatzes zu bestimmen. Dies funktioniert grundsätzlich, indem im Testdatensatz die korrekten NE-tags, die als Attribut „label“ vorliegen, mit der eigenen Vorhersage, die als Attribut „predicted label“ vorliegt, verglichen werden. Der Operator respektiert das iob-tagging (2.4.1) und berechnet die vorgestellten Güte-Maße precision, recall und F-Measure (2.4).

Die einzustellenden Parameter sind folgende:

Parameter-Name	Parameter-Art	Parameter-Funktion
precision	Boolean	Soll der precision-Wert für jede NE berechnet werden?
recall	Boolean	Soll der recall-Wert für jede NE berechnet werden?
f-measure	Boolean	Soll der f-measure-Wert für jede NE berechnet werden?
overall-precision	Boolean	Soll ein globaler precision-Wert (über alle NEs) berechnet werden?
overall-recall	Boolean	Soll ein globaler recall-Wert berechnet werden?
overall-f-measure	Boolean	Soll ein gloaber f-measure-Wert berechnet werden?
stopword	String	Das benutzte Stopwort muss angegeben werden, damit es nicht in die Berechnung mit einbezogen wird.
not-NER-tag	String	Welche der NE-tags bezeichnet die Nicht-NE?
iob	Boolean	Wird iob-tagging benutzt?

Tabelle 7: Parameter für CRFperformanceEvaluator

Die Ergebnisse werden nach Benutzung des Operators YALE-intern angezeigt, allerdings kann die Ausgabe auch mithilfe eines nachgeschalteten ResultWriter-Operators in eine Datei geschrieben werden.

11 Datensätze

Das implementierte und in YALE eingebundene Verfahren soll nun natürlich auch an einem konkreten Datensatz getestet werden. Der nun vorgestellte Datensatz ist mehrmals mit verschiedenen Verfahren bearbeitet worden. Das vom Diplomanden entwickelte System soll mit diesen Verfahren verglichen werden. Den Abschluss dieses Kapitels bildet die Vorstellung eines weiteren Datensatzes, der gegen Ende der Arbeit mit dem vom Diplomanden entwickelten System untersucht wurde.

11.1 JNLPBA-Datensatz

11.1.1 Einleitung

Der JNLPBA-Datensatz ist ein Datensatz aus der biomedizinischen Domäne (vergleiche [17]).

JNLPBA ist die Abkürzung für *Joint workshop on natural language processing in biomedicine and its applications.* Dieser Workshop fand im Jahr 2004 in Genf statt und befasste sich ausschließlich mit der Analyse biomedizinischer Texte. Dies geschah aus dem Grund, da die NER auf anderen Domänen wie zum Beispiel Nachrichtentexten schon sehr gut arbeitete. Auf biomedizinischen Texten war dies noch nicht der Fall, da vor allem Abkürzungen oder sehr kurze unbekannte Wort in diesen Texten sehr häufig sind und bis dato noch nicht effizient und gut erkannt werden konnten. Vor dem Workshop wurden in einem Wettbewerb die besten Verfahren zur NER auf dem JNLPBA-Datensatz gesucht, die in Kapitel 11.2 vorgestellt werden.

Der Datensatz besteht aus einem Auszug aus dem *Genia-Korpus*, der neu markiert wurde. Der Genia-Korpus besteht aus Ergebnissen von *MEDLINE-Anfragen* mit den *MeSH-Termen* 'human', 'blood cells' und 'transcription factors'. MEDLINE ist die Datenbank der Nationalen Bibliothek für Medizin in den USA (vergleiche [23]). Diese Datenbank enthält Referenzen auf ungefähr 13 Millionen Artikel mit biomedizischen Themen. MeSH wiederum ist ein so genannter Vokabel-Thesaurus (vergleiche [24]). Durch MeSH ist es möglich, zu den gestellten Schlagwörtern – in diesem Fall ‚human', ‚blood cells' und ‚transcription factors' – Texte in MEDLINE zu finden, die entweder direkt oder aber indirekt mit den Schlagwörtern übereinstimmen. Zum Beispiel findet man mithilfe von MeSH Anfragen mit dem Schlagwort ‚vitamin c' auch Texte, die unter Umständen gar nicht den Begriff ‚vitamin c' sondern ‚ascorbic acid' (was Vitamin C entspricht) enthalten.

Der Genia-Korpus enthält 36 verschiedene Markierungen (tags).

11.1.2 Datenanalyse

Der JNLPBA-Datensatz enthält nur sechs tags, wobei eine dieser sechs tags das Outer-tag (siehe 2.4.1) ist. Die sechs tags werden nun vorgestellt:

O	protein	DNA	cell_type	cell_line	RNA

Tabelle 8: Tags im JNLPBA-Datensatz

Der Datensatz ist weiterhin eingeteilt in einen Trainings- und eine Testdatensatz. Bevor der Workshop stattfand, war für alle Teilnehmer natürlich nur der nicht markierte Testdatensatz verfügbar. Nun (nach dem Workshop) liegt allerdings eine markierte Version vor, so dass man die Güte eigener Versuche mit der Güte vorhandener Lösungen vergleichen kann.

Der Trainingsdatensatz enthält 492.551 Worte, wobei sich die prozentuale Verteilung der tags wie folgt ergibt:

Markierung	O	protein	DNA	cell_type	cell_line	RNA
Vorkommen	77,75 %	11,19 %	5,14 %	3,14 %	2,28 %	0,50 %

Tabelle 9: Prozentuales Vorkommen der tags im JNLPBA-Trainingsdatensatz

Um schneller Trainingsergebnisse zu erzielen, wurde ein 10%iges Sample aus dem Trainingsdatensatz gezogen, mit dem die ersten Trainingsläufe gemacht wurden, um später die vielversprechendsten Versuche auf dem kompletten Trainingsdatensatz zu wiederholen.

Das Sample enthält 52.186 Wörter, deren Markierungen wie folgt verteilt sind:

Markierung	O	protein	DNA	cell_type	cell_line	RNA
Vorkommen	77,66 %	11,26 %	5,29 %	3,11 %	2,16 %	0,52 %

Tabelle 10: Prozentuales Vorkommen der tags im Sample des JNLPBA-Trainingsdatensatz

Der Testdatensatz enthält 101.039 Worte, und entspricht somit in seinem Umfang 20,51% des Trainingsdatensatzes. Allerdings umfasst das Sample nur knapp die Hälfte der Daten des Testdatensatzes. Die prozentuale Verteilung im Testdatensatz ergibt sich wie folgt:

Markierung	O	protein	DNA	cell_type	cell_line	RNA
Vorkommen	80,81 %	9,74 %	2,82 %	4,86 %	1,47 %	0,30 %

Tabelle 11: Prozentuales Vorkommen der Markierungen im JNLPBA-Testdatensatz

11.2 Erfolgreiche Verfahren auf dem JNLPBA-Datensatz

Die drei ersten Plätze bei der Bearbeitung des JNLPBA-Datensatzes belegten folgende Verfahren:

1. HMM (Zhou und Su)
2. MEMM (Finkel und andere)
3. CRF (Settles)

Die Besonderheiten der Verfahren sollen nun vorgestellt werden. Inwiefern das vom Diplomanden entwickelte Verfahren mit diesen Verfahren vergleichbar ist, beziehungsweise wo die Unterschiede liegen, wird in Kapitel 13.7 erörtert.

11.2.1 CRF

Besonders interessant ist natürlich das Vorgehen von Burr Settles (vergleiche [33]), da es direkt mit dem vom Diplomanden entwickelten Verfahren vergleichbar ist. Al-

lerdings hat Settles kaum Implementierungsarbeit geleistet, da er die vorhandene CRF-Implementierung MALLET benutzte. Das von Andrew McCallum implementierte Verfahren nutzt wie auch das vom Diplomanden benutzte Verfahren von Sunita Sarawagi L-BFGS als Optimierungsmethode der Gewichte für die konkreten Potentialfunktionen. In diesem Punkt sind die Verfahren also identisch. Die Menge der Potentialfunktionen teilt Settles in zwei verschiedene Bereiche:

- *orthographic features*
- *semantic features*

Mit orthographic features sind Merkmale gemeint, die aus den Worten selbst extrahiert werden können. Natürlich zählt das Wort an sich hierzu. Zudem werden 17 reguläre Ausdrücke überprüft, wie zum Beispiel ob das Wort alphanumerisch ist, Punkte enthält oder ob es aus römischen Ziffern besteht. Auch Prä- und Suffixe zählen zu dieser Art Merkmal. Des Weiteren wird das Word wie in [4] generalisiert, indem Großbuchstaben durch ‚A', Kleinbuchstaben durch ‚a', Ziffern durch ‚0' und alle anderen Zeichen durch ‚_' ersetzt werden. Das Wort ‚Animal' würde also zu ‚Aaaaaa' werden. Eine weitere in [4] vorgestellte Generalisierung benutzt für aufeinanderfolgende gleiche Generalisierungszeichen jeweils nur einmal das Zeichen, so dass das Wort ‚Animal' zu ‚Aa' würde. Weiterhin wird zu jedem Wort noch das benachbarte vorherige sowie folgende Wort gespeichert.

Die semantic features bezeichnen so genannte *semantic word groups*, die im Grunde genommen zusätzlichen Wörterbüchern entsprechen. Settles benutzt 17 dieser semantic word groups, wobei sieben davon manuell erstellt wurden. Beispiele dieser Gruppen sind griechische Buchstaben, die auf Proteine hinweisen können, Aminosäuren, chemische Elemente und bekannte Viren. Für die NE cell-type benutzt Settles *Google-Sets*, um ein Wörterbuch für diese NE zu erzeugen. Google-Sets benutzen einige Beispiele, um dann in der Suchmaschine Google weitere Ergebnisse zu diesem Kontext zu finden.

Die Ergebnisse mit beiden Merkmalsarten wurden mit Ergebnissen der orthographic features allein verglichen und zeigen, dass nur bei den NE RNA und cell-line Steigerungen in den F-Measures auftraten, wobei die Verbesserungen und Verschlechterungen aufsummiert zeigen, dass die Nutzung beider Merkmalsarten grundsätzlich erfolgreicher ist.

11.2.2 MEMM

Die Gruppe um Jenny Finkel benutzt MEMM als Verfahren, wobei hier hauptsächlich auf die benutzten Merkmale geachtet werden soll (vergleiche [10]). Die bei Settles noch als orthographic features bezeichneten Merkmale werden hier als *local features* bezeichnet. Die Merkmale, die schon von Settles benutzt werden, werden natürlich auch hier benutzt, wobei einige weitere Merkmale angeführt werden. Einerseits werden POS-tags benutzt, wobei der POS-tagger mit kontextrelevanten – also biomedizinischen – Texten trainiert wurde. Ein weiteres Merkmal sind Abkürzungen, und auch Konjunktionen von Merkmalen sind zu erwähnen. Unter Konjunktionen versteht man die Verbindung von einzelnen Merkmalen, wie zum Beispiel dem vo-

rangegangenen POS-tag, dem vorangegangenen Wort und dem aktuellen Wort. Zudem wird in einem Fenster der Länge vier erkannt, wenn in einem Paar Klammern ‚( )' die Klammern mit jeweils unterschiedlichen NE markiert sind.

Die andere Art benutzter Merkmale wird als *external resources* bezeichnet, die *gazetteers*, *web-querying*, *surrounding abstract* und *frequency counts* beinhalten. frequency counts dienen dazu, mehrdeutige Worte zu erkennen, indem davon ausgegangen wird, dass häufig vorkommende Worte eher mehrdeutig sind. Die Häufigkeit wird aus dem *British national corpus* (BNC) – einem Datensatz, der über 100 Millionen Worte Englischer Texte enthält – bestimmt. Die gazetteers enthalten ausschließlich Gen-Namen, und es wird davon ausgegangen, dass diese Liste nicht bei der Erkennung von NE an sich, aber bei der Erkennung der Start- und Endpunkte mehrwortiger NE nützlich sein kann. web-querying ähnelt den in 11.2.1 erwähnten Google-Sets. Für jede NE-Klasse werden bezeichnende Worte gesucht, mit denen eine Anfrage an Google gestellt wird. Für die Klasse protein wird zum Beispiel ligation benutzt. Vorangestellt wird diesem Wort jeweils das aktuell betrachtete Wort, wenn dieses mit einer Häufigkeit von unter 10 im BNC vorkommt. Liefert die Anfrage ‚X ligation', wobei X das betrachtete Wort ist, eine hohe Trefferzahl, so klassifiziert die Methode das Wort als (in diesem Fall) protein. surrounding abstract markiert ein Wort zusätzlich mit der Information, ob das Wort irgendwo im Text schon einmal als NE markiert ist.

11.2.3 HMM

Die Gruppe um GuoDong Zhou benutzt zur Einbindung der Merkmale HMM als Verfahren, und erweitert das System um eine SVM, um die spärlichen Daten verarbeiten zu können (vergleiche [42]). Zur eigentlich NER benutzt Zhou eine HMM, wie sie von ihm bereits in [43] vorgestellt wurde. Das HMM berechnet intern die log-likelihood aus Beobachtungs- und Labelsequenz wie folgt:

$$\log P(\vec{y} \mid \vec{x}) = \log P(\vec{y}) - \sum_{i=1}^{T} \log P(y_i) + \sum_{i=1}^{T} \log P(y_i \mid \vec{x})$$

Formel 52: Bestimmung der log-likelihood in Zhous HMM-System

Als den kritischen Punkt der log-likelihood-Berechnung stellt Zhou die Berechnung der Wahrscheinlichkeit $P(y_i \mid \vec{x})$ heraus. Für diese Berechnung wird eine SVM benutzt, deren Ausgabe allerdings nicht normalisiert ist. Daher wird zur Bestimmung der Wahrscheinlichkeit ein sigmoider Kernel benutzt. Für jede NE muss jeweils ein Klassifizierer erstellt werden, da SVM (siehe 2.5.2) binäre Klassifizierer sind.

Die Theorie der HMM ist zur Genüge vorgestellt worden, so dass nun vor allem die benutzten Merkmale vorgestellt werden sollen. Zusätzlich zu den bereits vorgestellten Merkmalen werden vor allem die Merkmale, die nicht direkt aus dem aktuellen Wort extrahierbar sind, erweitert.

Worte oder Wortsequenzen, die sich auf vorangegangene Worte oder Sequenzen der gleichen NE beziehen, werden mit einer *Alias-Behandlung* verarbeitet. Dieses Verfahren beruht auf der Annahme, dass relevante Sequenzen öfter im Text vorkommen als

andere. Die Alias-Behandlung verfährt so, dass bereits vorhandene NE in einer Liste gespeichert werden und neue NE mit der Liste verglichen werden. Sofern ein möglicher Alias für ein Element der Liste gefunden wurde, wird überprüft, ob es sich wirklich um einen Alias handelt. Ist dies der Fall, so wird ein Merkmal der Form ENTITYnLm erzeugt, wobei ENTITY den NE-Namen enthält. n steht für die Länge der aktuellen NE, und m ist die Länge der NE, für die die aktuelle NE ein Alias ist. Ein Eintrag in der Liste sei zum Beispiel ‚Bruce Springsteen', während der Alias-Behandlung wird dann ‚der Boss' als möglichen Alias-Namen entdeckt. Das Merkmal hat dann die Form PERSON2L2, da die NE eine NE mit Namen PERSON ist und die originale NE die Länge zwei und die Alias-NE auch die Länge zwei hat.

Eine weitere Art der Alias-Erkennung ist die Entdeckung von Abkürzungen. Da in biomedizinischen Texten häufig Abkürzungen benutzt werden, ist die Erkennung dieser Abkürzung vielversprechend. Zhou benutzt ein Verfahren, das auf der Erkennung von Klammern beruht. Es gibt zwei Arten, wie Abkürzungen vorkommen können: entweder taucht die Abkürzung in Klammern hinter der kompletten Form auf, oder die komplette Form taucht in Klammern hinter der Abkürzung auf. Zur Textbearbeitung wird jeweils nur die komplette Form benutzt, so dass Abkürzungen entweder entfernt oder zu ihrer kompletten Form expandiert werden. Nach der Verarbeitung werden die Abkürzungen – wenn sie in Klammern mit auftauchten – wieder erstellt und mit der gleichen NE wie die komplette Form bezeichnet. Weiterhin wird eine Liste mit den Abkürzungen sowie den entsprechenden kompletten Formen erzeugt, um spätere Vorkommen von Abkürzungen, die nicht mehr von ihren kompletten Formen begleitet werden, zu erkennen.

Da der Genia-Korpus (siehe 11.1.1) zu knapp 17% aus kaskadierenden NE besteht, wird außerdem diese Art der NE-Konstrukte mit Hilfe von sechs Mustern erkannt. Diese Muster sind ähnlich wie Grammatiken aufgebaut, ein Beispiel ist: <ENTITY>:=<ENTITY> + <ENTITY>. Weitere Merkmale sind die Vorkommen in entweder dem offenen oder geschlossenen Dictionary, wobei das geschlossene Dictionary den Worten im Trainingsdatensatz entspricht. Das offene Dictionary besteht aus externen Dictionaries.

Die soeben vorgestellten Merkmale erbringen einen zusätzlichen Gewinn von knapp 12 % F-Measure.

11.3 CoNLL03-Datensatz

11.3.1 Einleitung

Der *Workshop on Computational Natural Language Learning* (CoNLL) befasst sich, wie der Name schon sagt, mit Lernverfahren für natürliche Sprachen. Auch vor jedem CoNLL-Workshop findet ein Wettbewerb statt. Im Jahr 2003 war NER in unterschiedlichen Sprachen das Thema (vergleiche [31]). Nachdem im Jahr 2002 die Sprachen Spanisch und Holländisch untersucht wurden, umfasste der Wettbewerb ein Jahr später die Sprachen Deutsch und Englisch. Der CoNLL03-Datensatz ist somit in einen englischen sowie einen deutschen Teil aufgeteilt. Die Datensätze bestehen aus

Zeitungstexten, wobei der deutsche Teil, der hier betrachtet werden soll, aus Ausschnitten aus der Zeitung „Frankfurter Rundschau“ stammt. Die Ausschnitte stammen allesamt aus einer der letzten Augustwochen des Jahres 1992.

Neben dem Ziel gute NER-Verfahren für beide Sprachen zu entwickeln, galt ein weiteres Augenmerk der Nutzung externer sowie nicht markierter Daten. Aus diesem Grund enthält der Datensatz nicht nur einen Trainings- sowie Testdatensatz sondern auch eine Menge an nicht markierten Daten.

11.3.2 Datenanalyse

Wie soeben erwähnt, wird in dieser Arbeit nur der deutsche Teil des Datensatzes untersucht werden. Der Datensatz enthält fünf tags, wobei eines davon das outer-tag ist. Die fünf tags sind:

O	ORG	MISC	PER	LOC

Tabelle 12: Tags im CoNLL03-Datensatz

Der Trainingsdatensatz enthält 207.484 Wörter, wobei die Verteilung der tags wie folgt ist – in Klammern steht die Anzahl der NE-tags:

Markierung	O	ORG	MISC	PER	LOC
Vorkommen	91,91 %	2,04%(2427)	1,35%(2288)	2,17%(2773)	2,53%(4363)

Tabelle 13:Prozentuales und absolutes Vorkommen der tags im CoNLL03-Trainingsdatensatz

Das prozentuale Vorkommen ist hierbei über jedem markiertem Wort berechnet und nicht über der Anzahl der NE (genau wie in Tabelle 14). Der Datensatz wurde nicht so gründlich untersucht wie der JNLPBA-Datensatz. Daher wird der CoNLL03-Datensatz ausschließlich mit den für den JNLPBA-Datensatz entwickelten Versuchskombinationen (siehe 13) untersucht. Aus diesem Grund wurde auch kein Sample erzeugt.

Es gibt einerseits einen Developer- und einen Testdatensatz, die in Größe ungefähr einem Viertel des Trainingsdatensatzes entsprechen und deren tag-Verteilung (Anzahl der Vorkommen) jetzt dargestellt wird:

Markierung	O	ORG	MISC	PER	LOC
Devel	87,24 %	4,10%(1241)	2,21%(1010)	3,85%(1401)	2,59%(1181)
Test	90,07 %	2,42%(773)	1,54%(670)	3,50%(1195)	2,47%(1035)

Tabelle 14: Anzahl der Vorkommen bestimmter NE-tags im CoNLL03-Developer- und Testdatensatz

Der Developerdatensatz diente ursprünglich dazu, im Stadium der Entwicklung von Systeme zur Bearbeitung des Datensatzes, diese Systeme zu evaluieren. In dieser Arbeit ist dieser Datensatz daher nicht mehr von Belang.

12 Grundsätzliches zu den Versuchen

12.1 Einleitung

Zuerst werden eine Reihe von Versuchen auf dem JNLPBA-Datensatz gemacht. Dieser Datensatz wird sehr genau untersucht werden. Die beste für diesen Datensatz gefundene Versuchskombination soll anschließend auch auf den CoNLL03-Datensatz angewandt werden.

12.2 Hardware

Die Rechner, auf denen die Versuche stattfanden sind jeweils *Dual Opteron 248*-Rechner mit einer Taktfrequenz von 2,2 Gigahertz. Die Rechner verfügen über 4096 Megabyte RAM, wobei Java der maximal mögliche Speicher von 1,5 Gigabyte zugeteilt wurde.

12.3 Evaluierung der Versuchsergebnisse

Bei der Evaluierung der Versuchsergebnisse wird der vom Diplomanden entwickelte CRFperformanceEvaluator-Operator (siehe 10.4) benutzt. Der Operator berechnet einerseits Precision, Recall und F-Measure für jede NE als auch über alle NE kumuliert. In allen Versuchen wird jeweils das F-Measure berechnet, da dies das populärste Maß zur Einordnung von NER-Verfahren ist. Zwar entspricht dieses Vorgehen dem Evaluierungswerkzeugs des JNLPBA-Workshops, jedoch waren die Ergebnisse anfangs nicht hundertprozentig identisch. Dieser Punkt wurde jedoch nicht besonders beachtet, da die korrekten Güteergebnisse höchstens 2% von den Ergebnissen des CRFperformanceEvaluators abwichen. Hier sind nicht zwei absolute Prozentpunkte im F-Measure sondern im Verhältnis gemeint. Erst gegen Ende der Diplomarbeit korrigierte der Diplomand diesen Punkt. Die Problematik lag an der Tatsache, dass aufeinander folgende gleichartige Sequenzen nicht korrekt erkannt wurden. Aufeinander folgende gleichartige Sequenzen können zum Beispiel der Form <B-DNA><I-DNA><B-DNA> sein. In der Programmierung des Diplomanden werden die führenden *Identifikatoren* wie ‚B-, oder ‚I-, beschnitten. Würde man die Identifikatoren nicht abschneiden, so müsste man später darauf achten, dass keine ‚inner'-Markierungen ohne ‚beginning'-Markierung vorkommen. Des Weiteren besteht die Vermutung, dass durch die Nutzung der Identifikatoren schlechtere Ergebnisse erzielt werden, da die Anzahl der zu erkennenden Markierungen verdoppelt wird. Finkel begründet in [10] dasselbe Vorgehen unter Anderem damit, dass man bei der Nichtbeachtung der Identifikatoren die Anzahl von Merkmalen für bestimmte NE erhöht, was dem Verfahren eine bessere Entscheidungsgrundlage liefert.

12.4 Problematik ‚Kompletter Datensatz – Sample'

Je mehr eigene Potentialfunktionen im Laufe der Diplomarbeit entwickelt wurden, umso deutlicher trat das Problem auf, dass die Versuche auf dem kompletten Trainingsdatensatz viel zu lange dauerten. Einige Versuche dauerten bis zu einem Monat, was für die Phase, in der sich das Verfahren noch in der Entwicklung befand, einfach viel zu lange war. Daraufhin wurde das in Kapitel 10.2.2 vorgestellte Sampling entwickelt, woraufhin die späteren Versuche nur noch auf einem Sample des Trainingsdatensatzes durchgeführt wurden.

Zur Demonstration soll hier die zeitliche Performanz des Verfahrens einerseits auf dem kompletten Trainingsdatensatz und andererseits auf einem 10%igen Sample des Trainingsdatensatzes (JNLPBA) dargestellt werden:

Datensatz	**Zeit für Vorverarbeitung**	**Größe Dictionary**	**Größe Datensatz (mit allen Merkmalen)**
Komplett	4559 Sekunden	61,3 MB	283 MB
10%iges Sample	313 Sekunden	16,6 MB	30,0 MB

Tabelle 15: Unterschiede zwischen komplettem und 10%igem Datensatz

In Tabelle 15 wird ausschließlich der NERpreprocessing-Operator (vergleiche 10.2) betrachtet, wobei die Zeit für die Vorverarbeitung die Zeit ist, die zum Einlesen des Datensatzes, unter Umständen ziehen des Samples und schließlich für die Speicherung des mit Merkmalen angereicherten Datensatzes benötigt wird. Die Größe des Datensatzes entspricht ungefähr dem Verhältnis 1:10 das ist logischerweise so, da der Umfang des Samples ja 10% des Originaldatensatzes haben soll. Bei der Größe des Dictionaries ist das Verhältnis ungefähr 1:4, das liegt daran, dass im Dictionary nicht jedes vorkommende Wort einen Eintrag erhält, sondern unter Umständen nur ein schon vorhandener Eintrag für das Wort erhöht wird. Durch das Sample sind viele Worte schon abgedeckt, so dass beim kompletten Datensatz nicht mehr die neunfache Menge an Einträgen hinzukommt. Die Tatsache, dass die Vorverarbeitung des Samples fast 15mal so schnell ist wie die Vorverarbeitung für den kompletten Datensatzes, ist nicht besonders relevant, da die Vorverarbeitung nur einmal durchzuführen ist. Danach können die Daten gespeichert und für alle Versuche immer wieder benutzt werden.

Nun soll am Beispiel der Potentialfunktion `WordFeatures` die Laufzeit für das Lernen eines CRF-Modells auf dem Sample sowie dem kompletten Datensatz dargestellt werden:

Datensatz	**Zeit bis Konvergenz**	**Zeit pro Iteration**	**Iterationen**
Komplett	3271 Sekunden	ca. 16 Sekunden	ca. 200
10%iges Sample	103 Sekunden	ca. 2 Sekunden	ca. 40

Tabelle 16: Vergleich der Laufzeit beim Lernen eines CRF-Modells

Man sieht, dass die Zeit für eine Iteration bei der Untersuchung des kompletten Datensatzes im Vergleich sogar (im Verhältnis zur Größe des Datensatzes) geringer ist, als beim Sample. Allerdings benötigt die Untersuchung des Samples nur knapp 40 Iterationen, bis sie konvergiert, wohingegen die Untersuchung des kompletten Datensatzes ungefähr 200 Iterationen bis zur Konvergenz benötigt. Dieser Umstand führt im Endeffekt dazu, dass die Untersuchung des kompletten Datensatzes mehr als dreißig mal so viel Zeit benötigt wie die Untersuchung des Samples. Um die Art des Anstiegs der Bearbeitungsdauer zu verifizieren, wurden noch Versuche auf einem 33%igem sowie einem 66%igem Sample durchgeführt.

Datensatz	Zeit bis Konvergenz	Zeit pro Iteration	Iterationen
33%iges Sample	451 Sekunden	ca. 5 Sekunden	90
66%iges Sample	1590 Sekunden	ca. 10 Sekunden	ca. 160

Tabelle 17: Vergleich der Laufzeit beim Lernen eines CRF-Modells (33%iges und 66%iges Sample)

Die Ergebnisse werden der Übersichtlichkeit halber noch einmal als Graphen dargestellt:

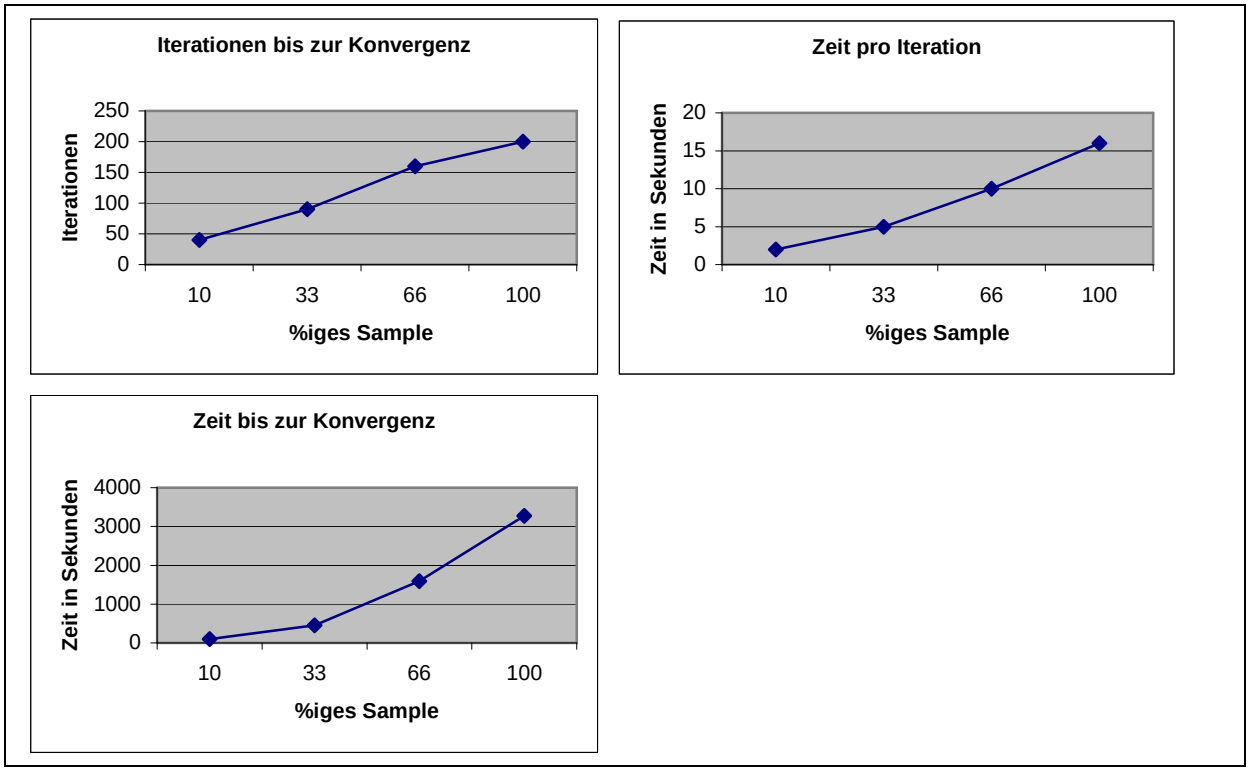


Abbildung 15: Laufzeitvergleich zwischen unterschiedlich großen Samples

Die Tatsache, dass die Iterationen bis zur Konvergenz sowie die Zeit pro Iteration bei größeren Samples linear ansteigen, führt logischerweise zu einem exponentiellen Anstieg der Zeit, die bis zur Konvergenz des Verfahrens vergeht. Somit ist es sehr sinnvoll, für die ersten Versuche das Sample zum Trainieren zu benutzen. Hat man die richtigen Einstellungen bestimmt, so kann der komplette Datensatz benutzt werden, was voraussichtlich noch eine gewisse Steigerung der Performanz liefern wird.

13 Versuche auf den Datensätzen

Kapitel 13.1 bis Kapitel 13.7 behandeln die Versuche auf dem JNLPBA-Datensatz. In Kapitel 13.8 werden einige der auf dem JNLPBA-Datensatz gemachten Versuche auf den CoNLL03-Datensatz übertragen.

Die Zeit zur Bestimmung der Güte über dem Testdatensatz beträgt je nach Testdatensatz ungefähr eine Minute, sofern YALE erst gestartet werden muss. Erzeugt man mehrere Güteergebnisse in einer YALE-Kette, fällt also die Start-Zeit von YALE weg, benötigt eine Gütebestimmung ungefähr eine halbe Minute. Diese Zeit wird im Verlauf dieser Arbeit bei den Versuchen nicht mit aufgeführt, da sie im Vergleich zu den teilweise sehr hohen Trainingszeiten zu vernachlässigen ist.

13.1 Untersuchung der vorhandenen Potentialfunktionen

13.1.1 Vorhandene Potentialfunktionen allein angewandt

Vorerst sollen die einzelnen Potentialfunktionen auf das Sample des Trainingsdatensatzes allein angewandt werden. Es werden vorerst keine externen Evidenzen benutzt.

Die Ergebnisse aller sechs Versuche werden nun vorgestellt:

Potentialfunktion	**Anzahl konkreter Potentialfunktionen**	**Anzahl Elemente (für alle Evidenzen)**	**Zeit/Iteration in Sekunden**	**Iterationen bis Konvergenz**	**Zeit bis Konvergenz**	**F-Measure über Testdaten**
`WordFeatures`	8985	ca. 44000 Worte	ca. 2	ca. 40	103 Sekunden	31,49 %
`ConcatRegexFeatures`	156	26 reguläre Ausdrücke	ca. 4	ca. 550	ca. 37 Minuten	21,58 %
`StartFeatures`	6	6 Zustände	ca. 1	ca. 10	ca. 10 Sekunden	k.A.
`EndFeatures`	6	6 Zustände	ca. 1	ca. 10	ca. 10 Sekunden	k.A.
`EdgeFeatures`	36	6 Zustände	ca. 4	ca. 250	ca. 17 Minuten	k.A.
`UnknownFeatures`	6	6 Zustände	ca. 1	ca. 10	ca. 10 Sekunden	k.A.

Tabelle 18: Vergleich der Güte der vorhandenen Potentialfunktionen

Die Potentialfunktionen `WordFeatures` und `ConcatRegexFeatures` liefern als einzige Potentialfunktionen ein messbares Güteergebnis über dem Testdatensatz. Die Anzahl der konkreten Potentialfunktionen ergibt sich aus dem Vorgehen der Poten-

tialfunktionen, das in Kapitel 8.2.3 beschrieben ist. Die Elemente, die jeweils für die Anzahl der konkreten Potentialfunktionen ausschlaggebend sind, sind hier Worte, reguläre Ausdrücke und Zustände (gleichbedeutend mit unterschiedlichen NE-tags). Diese Elemente können zum Beispiel auch Prä- oder Suffixe sein. Aus der Zeit, die das Verfahren für eine Iteration mit dieser Potentialfunktion benötigt sowie der Anzahl der Iterationen bis zur Konvergenz ergibt sich natürlich die gesamte Dauer des Verfahrens. Diese Dauer ist für `WordFeatures` im Vergleich zu den `ConcatRegexFeatures` sehr schnell. Vor allem wenn man die Tatsache beachtet, dass ein mit `WordFeatures` gelerntes Modell auf dem Testdatensatz ein mehr als 10% höheres F-Measure erreicht, ist dieser Fakt bemerkenswert.

Die restlichen vier Potentialfunktionen liefern allein betrachtet keine messbaren Ergebnisse über dem Testdatensatz – sie markieren keine einzige NE korrekt. Allerdings wird die spätere Kombination der Potentialfunktionen zeigen, dass `EdgeFeatures` in Kombination mit anderen Potentialfunktionen sehr wohl eine Steigerung in den Ergebnissen liefert.

Abbildung 20 zeigt unter anderem die Güte der Potentialfunktionen `WordFeatures` und `ConcatRegexFeatures` im Verlauf des Trainings bei steigender Anzahl von Iterationen.

13.1.2 Vorhandene Potentialfunktionen kombiniert

Erst durch die Kombination einzelner Potentialfunktionen wird man gute Ergebnisse erzielen können. Daher sollen nun mehrere Potentialfunktionen kombiniert werden, um herauszufinden, welche Kombination die besten Ergebnisse liefert. Vorerst werden nur die vorhandenen Potentialfunktionen kombiniert, woraufhin dann später diese Kombination um eigene Potentialfunktionen erweitert wird, um möglichst noch bessere Ergebnisse zu erzielen.

Als Grundlage aller Kombinationen wird die Potentialfunktion WordFeatures benutzt, da ein mit dieser Potentialfunktion allein gelerntes Modell schon fast einen F-Measure von 31,5 % liefert. Vor allem von der Kombination mit der Potentialfunktion ConcatRegexFeatures werden gute Ergebnisse erwartet.

Tabelle 19 zeigt die Versuche mit verschiedenen Potentialfunktionkombinationen. Die jeweils beste Ergebnisgüte sowie die schnellste Zeit bis zur Konvergenz werden in dieser und in allen folgenden Tabellen besonders markiert. Abbildung 16 zeigt dazu den Verlauf der Güte während des Lernprozesses.

Potentialfunktionkombinationen (*)	**Anzahl konkreter Potentialfunktionen (**)**	**Zeit/Iteration in Sekunden (***)**	**Iterationen bis Konvergenz (****)**	**Zeit bis Konvergenz (*****)**	**F-Measure über Testdaten (******)**
`W_R_St_En_E_U`	9195	ca. 7,5	ca. 425	ca. 52 Minuten	**55,64 %**
`W_R`	9141	ca. 5	ca. 350	ca. 30 Minuten	33,60 %

W_R_St	9147	ca. 5	ca. 400	ca. 35 Minuten	32,98 %
W_R_En	9147	ca. 5	ca. 375	ca. 30 Minuten	33,12 %
W_R_U	9147	ca. 5	ca. 500	ca. 40 Minuten	33,08 %
W_R_E	9177	ca. 7	ca. 450	ca. 55 Minuten	55,21 %
W_E	9021	ca. 3	ca. 300	**ca. 15 Minuten**	52,70 %
W_E_St	9027	ca. 4	ca. 250	ca. 17 Minuten	52,21 %
W_E_En	9027	ca. 4	ca. 270	ca. 18 Minuten	52,59 %
W_E_U	9027	ca. 3,5	285	ca. 17 Minuten	51,70 %

Tabelle 19: Kombination der vorhandenen Potentialfunktionen (W=WordFeatures, R=ConcatRegexFeatures, St=StartFeatures, En=EndFeatures, E=EdgeFeatures, U=UnknownFeatures)

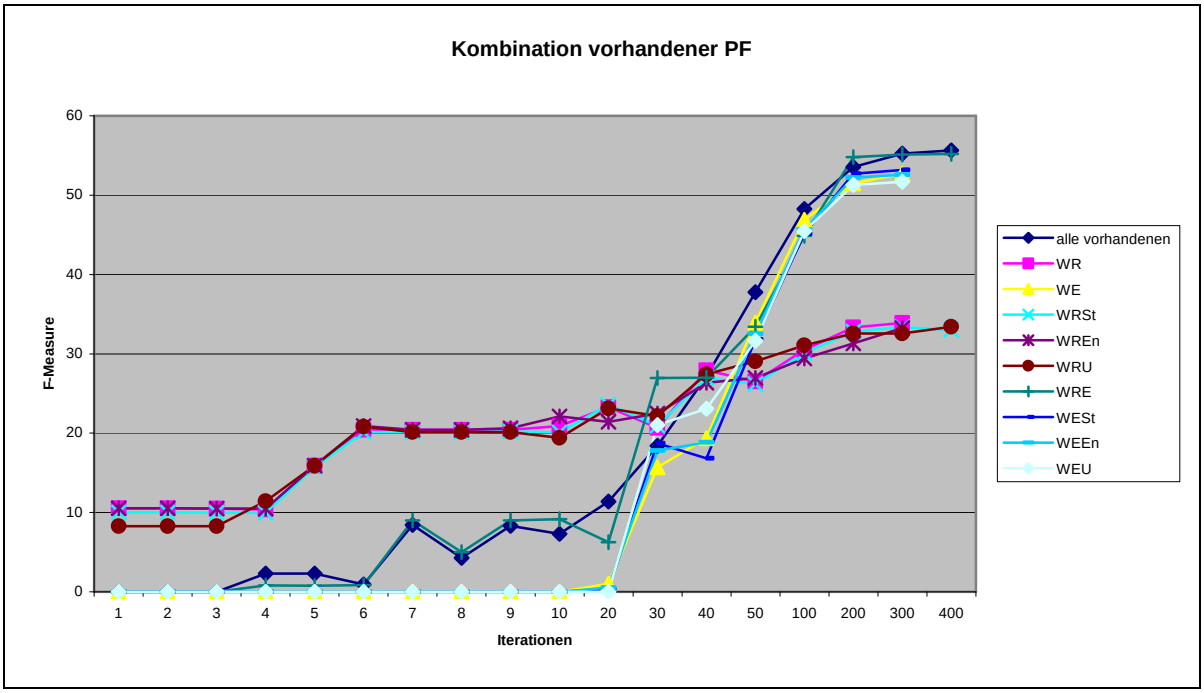


Abbildung 16: Vergleich der Kombinationen aus vorhandenen Potentialfunktionen (W=WordFeatures, R=ConcatRegexFeatures, St=StartFeatures, En=EndFeatures, U=UnknownFeatuers, E=EdgeFeatures)

Als Baseline-Kombination wird die Kombination aller vorhandenen Potentialfunktionen angesehen. Vielleicht ist es aber auch möglich die Güte der Baseline-Kombination schon durch die Kombination von wenigen vorhandenen Potentialfunktionen zu erreichen.

Zwar liefert die Kombination von `WordFeatures` mit `ConcatRegexFeatures` einen kleinen Gewinn im F-Measure, jedoch ist dieser Gewinn nicht so hoch wie erwartet, und vor allem liegt er noch 20% unter dem F-Measure der Baseline-Kombination. `StartFeatures`, `EndFeatures` oder `UnknownFeatures` mit `WordFeatures` und `ConcatRegexFeatures` zu kombinieren bringt sogar noch schlechtere Ergebnisse.

Kombiniert man allerdings `WordFeatures`, `ConcatRegexFeatures` und `EdgeFeatures`, so erhält man ein F-Measure, das fast an das der Baseline-Kombination heranreicht. Diese Kombination wäre also ein guter Ersatz für die Baseline-

Kombination, wobei die Baseline-Kombination etwas schneller konvergiert, was im Endeffekt zu einer kürzeren Laufzeit führt.

Anschließend wurde noch die Kombination von `WordFeatures` mit `EdgeFeatures` untersucht, und hier sind wirklich interessante Ergebnisse zu beobachten. Das resultierende F-Measure der Kombination liegt nur knapp unter dem F-Measure der Baseline-Kombination, wobei es mehr als dreimal so schnell konvergiert (zeitlich) als die Baseline-Kombination. Das Hinzufügen der `StartFeatures`, `EndFeatures` oder `UnknownFeatures` bringt allerdings keine Verbesserung in der Güte oder in der Laufzeit.

Daraus folgt nun, dass für alle folgenden Versuche die Baseline-Kombination, die Kombination W_R_E und die Kombination W_E (siehe Tabelle 19) benutzt wird, um durch das Hinzufügen der selbst entwickelten Potentialfunktionen möglichst bessere Ergebnisse, sei es in Güte oder Laufzeit, zu erzielen.

13.2 Untersuchung der selbst entwickelten Potentialfunktionen

13.2.1 Selbst entwickelte Potentialfunktionen allein angewandt

Um nun die Performanz der vom Diplomanden entwickelten Potentialfunktionen einzuschätzen, sollen auch diese Potentialfunktionen jeweils erst einmal allein auf dem Sample des Trainingsdatensatzes angewandt werden. Allerdings gibt es bei den `Prefix-`, `Suffix-` und `NGramFeatures` die Möglichkeit, die Potentialfunktionen zu parametrisieren: man muss vor der Versuchsdurchführung zuerst die Länge der Präfixe etc. einstellen.

Vorversuche mit unterschiedlichen Längen haben folgende Resultate erzielt:

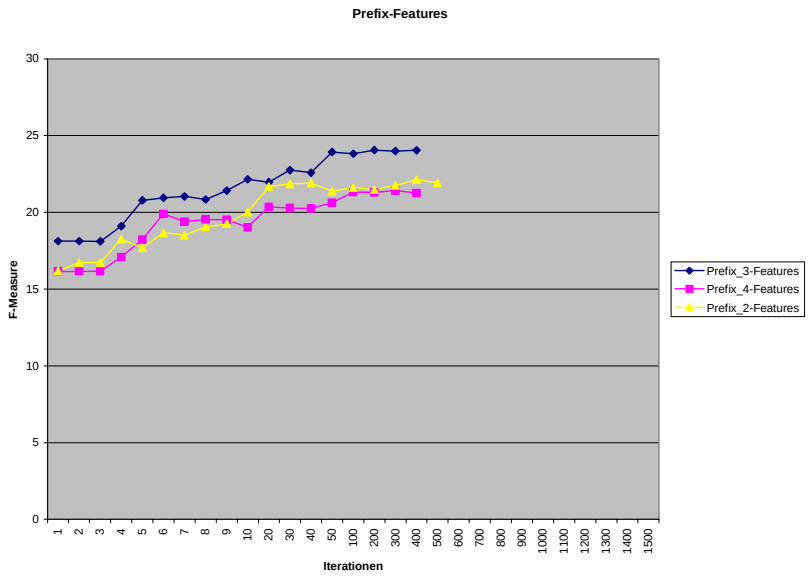


Abbildung 17: Vergleich der erzielten Güte mit 2-, 3- und 4-Präfixen

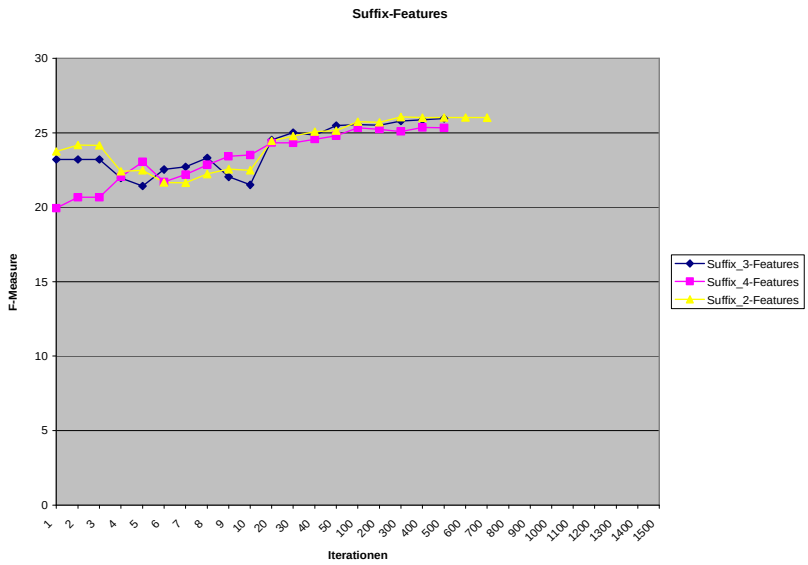


Abbildung 18: Vergleich der erzielten Güte mit 2-, 3- und 4-Suffixen

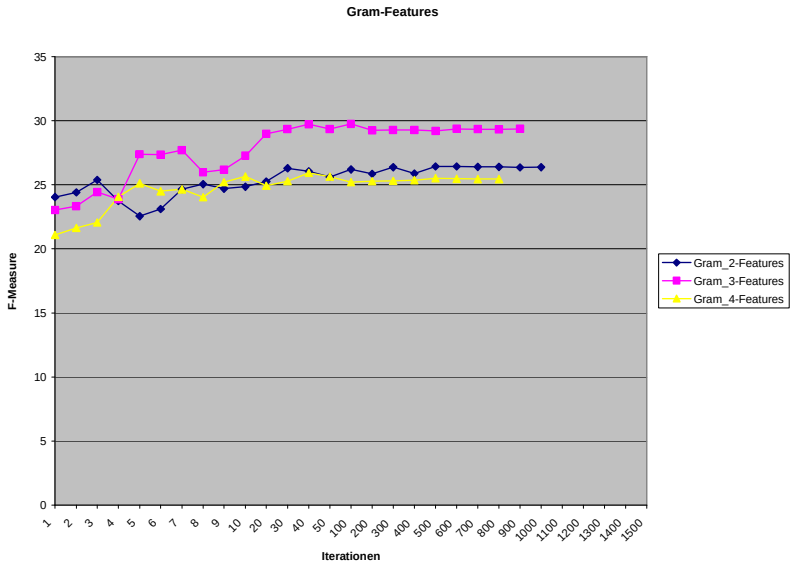


Abbildung 19: Vergleich der erzielten Güte mit 2-, 3- und 4-Grammen

Abbildung 17 und Abbildung 19 zeigen eindeutig die Dominanz einer bestimmten Länge von Präfixen beziehungsweise NGrammen. Die Präfixe der Länge drei dominieren die Präfixe der Länge zwei oder vier. Denselben Effekt sieht man bei den NGrammen. Daher werden in den folgenden Versuchen jeweils NGramme sowie Präfixe der Länge drei benutzt. In Abbildung 18 ist allerdings keine besondere Dominanz auszumachen, so dass sich der Diplomand dazu entschlossen hat, bei allen folgenden Versuchen Suffixe der Länge 2 zu benutzen, da diese eine kürzere Rechenzeit sowie weniger konkrete Potentialfunktionen nach sich ziehen.

Die Versuche auf den vom Diplomanden entwickelten Potentialfunktionen ergaben nun folgende Ergebnisse:

Potential-funktion	**(**)**	**Anzahl Elemente (für alle Eviden-zen)**	**(***)**	**(****)**	**(*****)**	**(******)**
`GramFea-tures`	18951	ca. 23000 3-Gramme	ca. 4	ca. 135	ca. 550 Se-kunden	**33,96 %**
`Prefix-Features`	156	ca. 12000 3-Präfixe	ca. 2	ca. 70	**110 Sekun-den**	26,16 %
`Suffix-Features`	3239	ca. 2000 2-Suffixe	ca. 2	ca. 75	ca. 150 Se-kunden	25,91 %
`PosFeatu-res`	6	ca. 1200 POS-tags	ca. 1,5	ca. 180	ca. 280 Se-kunden	20,09 %

Tabelle 20: Vergleich der Güte der selbst entwickelten Potentialfunktionen

Die selbst entwickelten Potentialfunktionen konvergieren zwar alle nicht so schnell wie die `WordFeatures`, allerdings liefern sie viel versprechende Ergebnisse. Vor allem, wenn man bedenkt, dass sie zwar langsamer konvergieren als die `WordFea-tures` aber trotzdem noch schneller sind als Versuche mit `ConcatRegexFeatures`, die als einzige bereits vorhandene Potentialfunktion neben den `WordFeatures` messbare Ergebnisse liefern.

Abbildung 20 zeigt zudem deutlich, dass die 3Gramme bessere Ergebnisse liefern, als die `WordFeatures.` Dieser Punkt wird in späteren Kombinationen der beiden Potentialfunktionen noch untersucht.

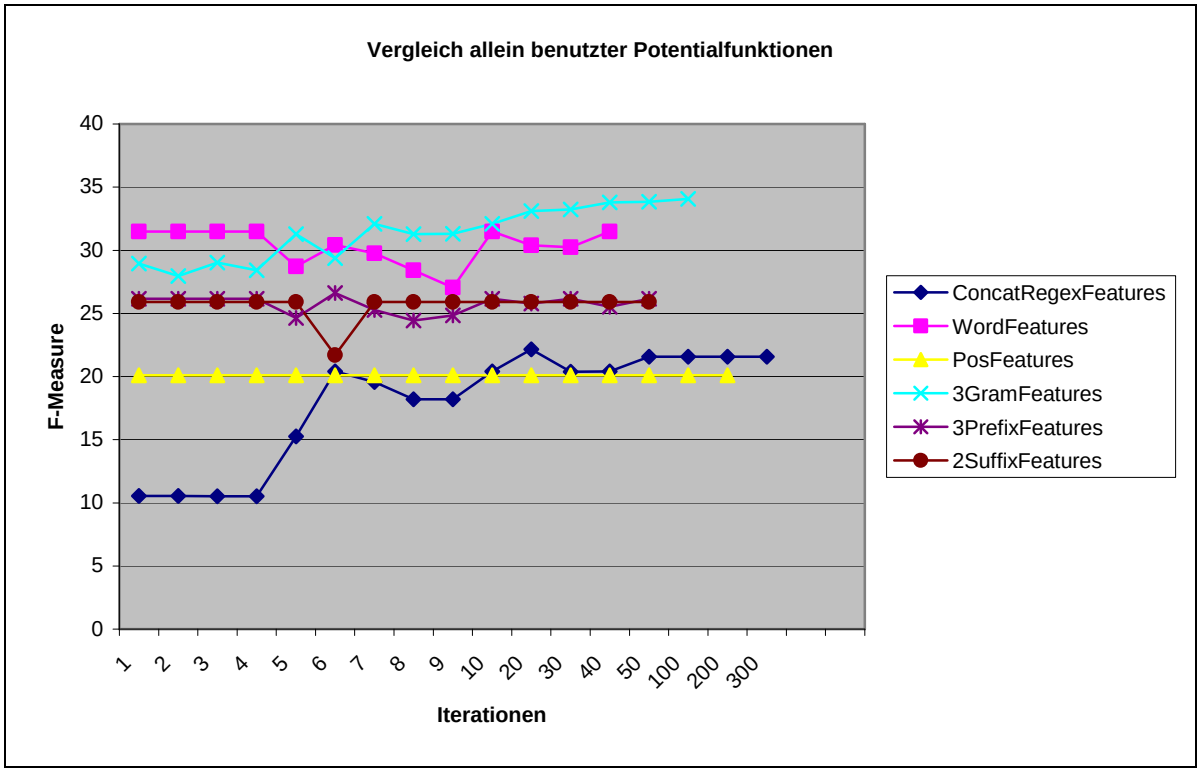


Abbildung 20: Vergleich aller Potentialfunktionen allein auf das Sample angewandt

13.2.2 Selbst entwickelte Potentialfunktionen kombiniert

13.2.1 zeigt, dass NGramFeatures leicht bessere Ergebnisse liefern als die WordFeatures. Allerdings sollen nun vorerst die selbst entwickelten Potentialfunktionen miteinander kombiniert werden, bevor die vorhandenen mit den selbst entwickelten Potentialfunktionen kombiniert werden.

(*)	(**)	(***)	(****)	(*****)	(******)
G_Po	19094	ca. 5,5	ca. 180	ca. 17 Minuten	33,92 %
G_S	22190	ca. 5,5	ca. 160	ca. 15 Minuten	33,66 %
G_P	24850	ca. 5	ca. 150	**ca. 13 Minuten**	34,18 %
G_P_S	28089	ca. 6,5	ca. 150	ca. 17 Minuten	**34,68 %**

Tabelle 21: Kombination der eigenen Potentialfunktionen (G=GramFeatures, Po=PosFeatures, S=SuffixFeatures, P=PrefixFeatures)

Betrachtet man die Laufzeit, so sind alle Kombinationen schlechter als die NGramFeatures allein, aber das war auch zu erwarten. Allerdings verbessert sich selbst die Güte der Ergebnisse nur in Kombination mit PrefixFeatures sowie in Kombination mit Prefix- und SuffixFeatures. Ein Grund, dass sich die Ergebnisse nicht so sehr verbessern, könnte die Tatsache sein, dass sich die Prefix- sowie SuffixFeatures nicht allzu sehr von den NGramFeatures unterscheiden. Im Grunde sind Prä- und Suffixe ja N-Gramme, die eine bestimmte Lokalität innehaben (Wortanfang und –ende). Somit könnte durch die N-Gramme der Wissensgewinn, den Prä- und Suffixe bringen, schon abgedeckt sein.

PosFeatures wurden auch schon von erfolgreichen Verfahren auf dem JNLPBA-Datensatz angewandt. Allerdings wurden dort POS-tagger benutzt, die auf der biomedizinischen Domäne trainiert wurden. Der POS-tagger, der vom Diplomanden benutzt wurde, wurde auf Zeitungstexten trainiert und liefert somit – wie die Versuche zeigen – keinen Ergebniszuwachs.

Zusammenfassend ist zu sagen, dass GramFeatures allein in Kombinationen sicherlich eine Ergebnisverbesserung liefern werden. Ob und inwiefern die Hinzunahme von Prefix- und SuffixFeatures noch weiteren Erfolg bringt, sei abzuwarten.

13.3 Kombination vorhandener und selbst entwickelter Potentialfunktionen

13.3.1 Vergleich GramFeatures - WordFeatures

Abbildung 20 zeigt, dass GramFeatures etwas bessere Ergebnisse liefern als WordFeatures. Allerdings konvergiert ein CRF, das mit GramFeatures arbeitet im Vergleich auch viel später als ein CRF, das mit WordFeatures arbeitet. Ausgewählte Kombinationen früherer Versuche sollen nun mit GramFeatures anstatt mit WordFeatures durchgeführt werden.

(*)	(**)	(***)	(****)	(*****)	(******)
G_E	18987	ca. 6	ca. 230	**ca. 24 Minuten**	54,95 %
G_R_E	19143	ca. 10	ca. 450	ca. 72 Minuten	**56,23 %**
G_R_E_St_En_U	19161	ca. 11	ca. 410	ca. 74 Minuten	55,80 %

Tabelle 22: Kombinationen mit GramFeatures anstatt mit WordFeatures

Vergleicht man Tabelle 22 mit Tabelle 19, so sieht man, dass die Ergebnisse der Kombinationen mit `GramFeatures` besser sind als mit `WordFeatures`. Allerdings benötigen die `GramFeatures`-Kombinationen erheblich länger, bis sie konvergieren.

13.3.2 Erweiterung erfolgreicher Kombinationen vorhandener Potentialfunktionen

In Kapitel 13.1.2 wurden die Kombinationen W_E, W_R_E sowie W_R_E_St_En_U (siehe Abbildung 16) als viel versprechende Kombinationen angesehen. Nachdem diese Kombinationen in Kapitel 13.3.1 in Hinblick auf ihre Abhängigkeit von der Potentialfunktion `WordFeatures` betrachtet wurden, sollen sie nun vorerst um die Potentialfunktion `GramFeatures` erweitert werden.

(*)	(**)	(***)	(****)	(*****)	(******)
W_G_E	27972	ca. 7,5	ca. 210	**ca. 26 Minuten**	56,06 %
W_G_R_E	28128	ca. 11	ca. 340	ca. 63 Minuten	**56,88 %**
W_G_R_E_St_En_U	28146	ca. 11	ca. 350	ca. 64 Minuten	56,77 %

Tabelle 23: Erweiterung der erfolgreichen Kombinationen durch die Potentialfunktion GramFeatures

Das Erweitern der in Tabelle 19 vorgestellten erfolgreichen Kombinationen um `GramFeatures` ergibt grundsätzlich bessere Ergebnisse. Auch konvergieren die Verfahren grundsätzlich schneller, wenn man ausschließlich die Iterationen betrachtet. Natürlich ist der Zeitraum, der bis zur Konvergenz vergeht, größer, da die Zeiten pro Iteration steigen.

Abschließend soll nun die aus Tabelle 23 beste Kombination noch zusätzlich durch die Potentialfunktionen `Prefix`- und `SuffixFeatures` erweitert werden:

W_G_R_E_P_S	37266	ca. 13,5	ca. 310	ca. 70 Minuten	56,66 %

Tabelle 24: Hinzufügen der Potentialfunktionen Prefix- und SuffixFeatures

13.4 Untersuchung relativer Positionen und Sequenzen

13.4.1 Untersuchung relativer Positionen

Die Ergebnisse durch die Generalisierung mit Hilfe der `Gram`-, `Prefix`- und `SuffixFeatures` haben zwar den erwarteten Erfolg gebracht, allerdings ist die Verbesserung im Vergleich zu den vorhandenen Potentialfunktionen nicht so gut wie erhofft. Als nächstes sollen nun die relativen Positionen untersucht werden. Hierfür

wurde als erstes die Potentialfunktion `WordFeatures` um unterschiedliche relative Positionen erweitert. Die Ergebnisse sind in Tabelle 25 aufgeführt.

(*)	(**)	(***)	(****)	(*****)	(******)
W_-3_-2_-1_0	37849	ca. 4	56	246 Sekunden	30,04 %
W_-2_-1_0	28059	ca. 3,5	61	216 Sekunden	31,37 %
W_-1_0	18299	ca. 3	ca. 50	ca. 150 Sekunden	31,24 %
W_0	8985	ca. 2	47	**80 Sekunden**	31,49 %
W_0_1	17784	ca. 2,5	56	145 Sekunden	**39,05 %**
W_0_1_2	26925	ca. 3	53	175 Sekunden	37,29 %
W_0_1_2_3	36132	ca. 4	53	230 Sekunden	37,47 %

Tabelle 25: Untersuchung der relativen Positionen anhand der WordFeatures

Wie zu erwarten war, steigt bei der Benutzung von relativen Positionen einerseits die Anzahl der konkreten Potentialfunktionen sowie die Zeit pro Iteration. Allerdings steigt bei einigen Versuchen auch erheblich die Güte der Ergebnisse. Interessant ist, dass nur die positiven relativen Positionen, sofern man sie betrachtet, Verbesserungen liefern. Die besten Ergebnisse liefert also die Hinzunahme des auf die aktuelle Position folgenden Wortes.

Bevor nun Kombinationen untersucht werden, sollen auch die `GramFeatures` mit relativen Positionen untersucht werden.

(*)	(**)	(***)	(****)	(*****)	(******)
G_-3_-2_-1_0	75761	ca. 17	Ca. 70	ca. 20 Minuten	29,29 %
G_-2_-1_0	57103	ca. 12,5	Ca. 90	ca. 19 Minuten	31,46 %
G_-1_0	38164	ca. 8	Ca. 100	ca. 13 Minuten	34,31 %
G_0	18951	ca. 4	Ca. 135	**ca. 9 Minuten**	33,96 %
G_0_1	37075	ca. 8	Ca. 100	ca. 13 Minuten	**40,11 %**
G_0_1_2	55229	ca. 12,5	Ca. 80	ca. 17 Minuten	38,16 %
G_0_1_2_3	73182	ca. 17,5	Ca. 80	ca. 23 Minuten	36,78 %

Tabelle 26: Untersuchung der relativen Positionen anhand der GramFeatures

Auch die Versuche mit der Potentialfunktion GramFeatures zeigen, dass sich relative Positionen positiv auf die Güte der Ergebnisse auswirken. Allerdings ergibt auch die relative Position vor der aktuellen Position eine Verbesserung. Ob die relativen Positionen um die aktuelle Position herum zusammen Verbesserungen bringen, wird durch die Kombination G_-1_0_1 untersucht:

G_-1_0_1	56288	ca. 13	Ca. 70	ca. 15 Minuten	39,34 %

Eigentlich sollte man davon ausgehen, dass durch zusätzliche Informationen, die durch die relativen Positionen gegeben sind, auch bessere Ergebnisse zu erzielen

sind. Doch Tabelle 25 und Tabelle 26 zeigen dies nur teilweise. Um alle Eventualitäten abzudecken soll nun noch die erzielte Güte, die erreicht wird, bevor das Verfahren konvergiert, dargestellt werden.

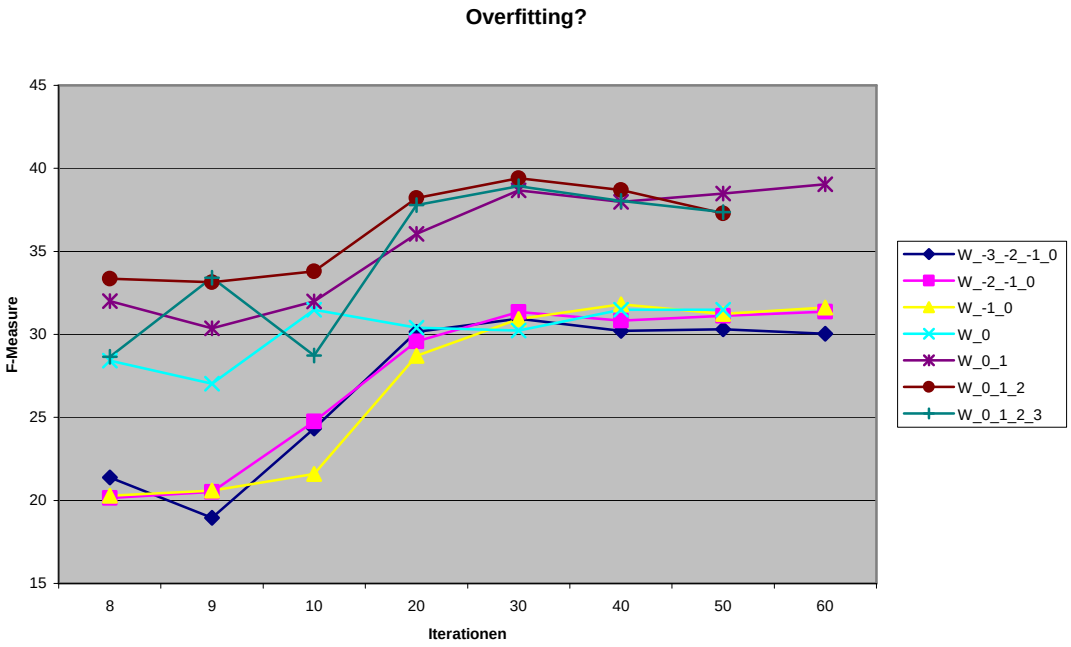


Abbildung 21: WordFeatures mit relativen Positionen im Verlauf des Versuchs

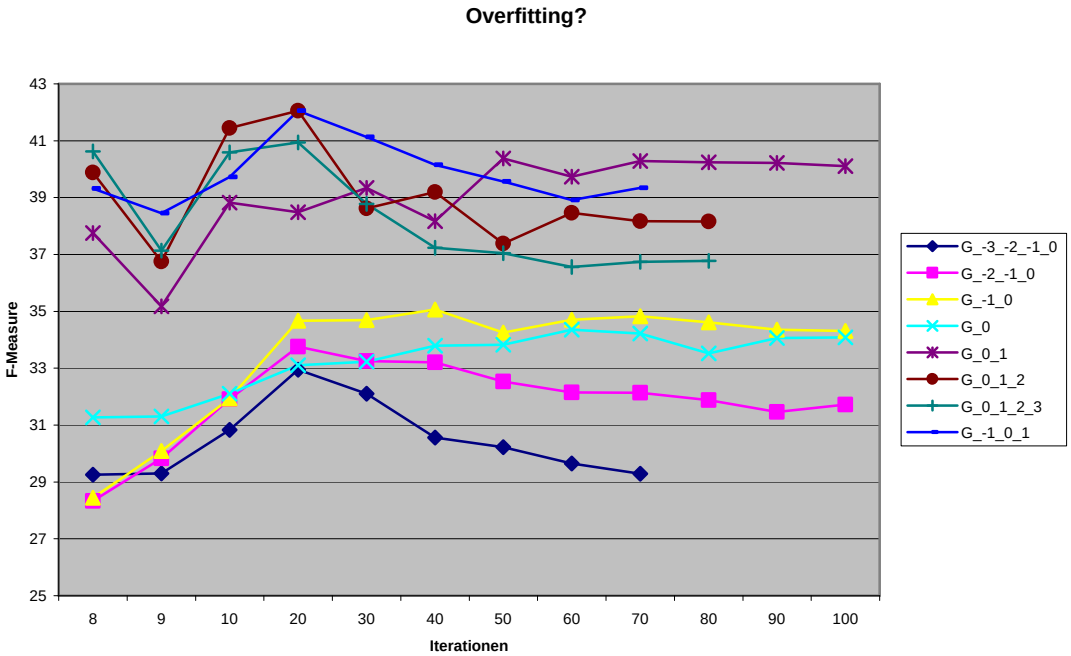


Abbildung 22: GramFeatures mit relativen Positionen im Verlauf des Versuchs

Auf Abbildung 21 und noch stärker in Abbildung 22 ist deutlich zu sehen, dass bei der Benutzung von mehreren relativen Positionen der maximale Wert des F-Measures grundsätzlich erreicht wird bevor das Verfahren konvergiert. Beim Erreichen der Konvergenz ist die Güte des Verfahrens in vielen Fällen sogar wieder gefal-

len. Es scheint sogar so, als ob die Verfahren nach erreichen des Maximums schneller wieder abfallen, wenn sie viele relative Positionen enthalten.

Dieses Verhalten weist eindeutig auf eine Überangepasstheit (overfitting) des Verfahrens hin. Overfitting tritt nach Definition dann auf, wenn eine Hypothese auf einem Datensatz zwar optimal ist, auf einem anderen Datensatz jedoch von einer anderen Hypothese in der erzielten Güte übertrumpft wird. In diesem Fall ist das trainierte Verfahren zwar auf dem Trainingsdatensatz optimiert. Jedoch ist es zu sehr auf die Trainingsdaten angepasst, so dass auf dem Testdatensatz, der anders beschaffen ist, als der Trainingsdatensatz, schlechtere Ergebnisse erzielt werden.

Diese Vermutung wird auch durch die nächste Beobachtung gestützt: obwohl das CRF auf dem Trainingsdatensatz noch nicht optimal ist, also noch nicht konvergiert, nimmt die Güte der erreichten Ergebnisse über dem Testdatensatz nach Erreichen des Optimums wieder ab. Im Grunde ist dieses Verhalten jedoch nachvollziehbar, da durch die verschiedenen relativen Positionen ein Überangebot an sehr ähnlichen konkreten Potentialfunktionen vorliegt. Diese alle zu trainieren begünstigt scheinbar overfitting.

Als endgültiger Beweis wurde ein mit `GramFeatures` und relativen Positionen gelerntes Modell einmal auf den Trainings- und einmal auf den Testdatensatz angewandt. Die Ergebnisse sind in Abbildung 23 zu sehen. Man sieht deutlich, dass die Güte der Ergebnisse über dem Testdatensatz nach dem Erreichen ihres Maximums bis zur Konvergenz wieder abfallen, wobei die Güte der Ergebnisse über dem Trainingsdatensatz bis zur Konvergenz immer weiter ansteigt. Und nach Definition ist genau dieses Verhalten ein Indiz für Überangepasstheit.

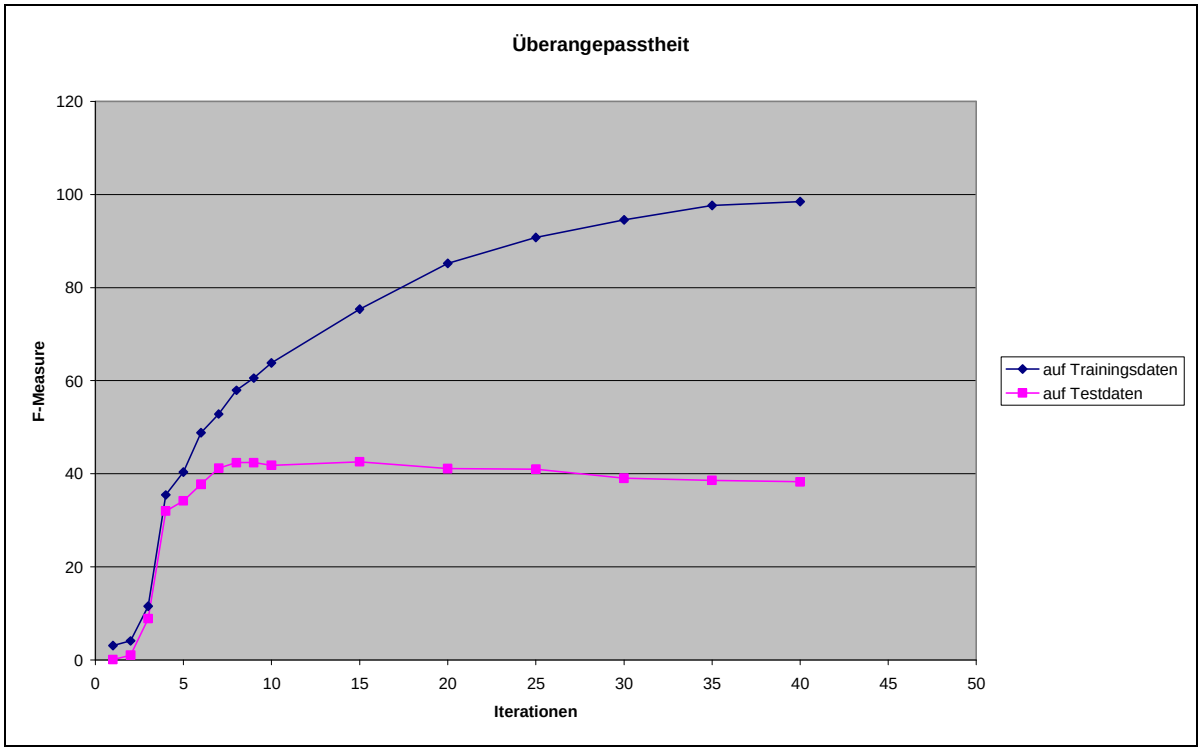


Abbildung 23: Vergleich der Güte über dem Trainings- sowie dem Testdatensatz

13.4.2 Vermeidung von overfitting

Es stellt sich nun die Frage, ob overfitting vermieden werden kann.

Laut [34] wird Überangepasstheit durch einen Bestrafungsterm bei der Berechnung des log-likelihood-Terms (siehe 4.3) vermieden. Da allerdings die vorhandene Implementierung sich genau auf diesen Artikel beruft, ist diese Aussage nicht korrekt. Allerdings sind `GramFeatures` mit relativen Positionen auch ein Sonderfall. Bei Versuchen mit vielen verschiedenen Potentialfunktionen (siehe 13.5 und folgende) tritt das Problem nicht mehr auf.

In [21] wird eher umgangssprachlich formuliert, dass bei der Nutzung von Quasi-Newton-Methoden, die hier benutzt werden, nur eine gewisse Anzahl von Iterationen benutzt werden sollten. Welchen Wert diese Anzahl haben sollte, wird jedoch nicht erläutert. Abbildung 22 zeigt allerdings, dass bei ungefähr 20 Iterationen das Maximum erreicht ist. Jedoch besteht die Erwartung, dass diese Grenze nicht immer dieselbe ist. Um diese Grenze jeweils zu bestimmen, müsste man den Trainingsdatensatz so zerlegen, dass man einerseits ein Modell trainieren kann und andererseits genug Daten hat, um die soeben erwähnte Grenze zu evaluieren. Der Diplomand allerdings kann diese Grenze mit Hilfe des Testdatensatzes evaluieren.

Ein anderer Ansatz zur Vermeidung von Überanpassung ist die Verknüpfung von CRF mit DBN, was ausführlich in Kapitel 7.2 erläutert wird.

Eigene Ansätze zur Vermeidung von Überangepasstheit

Betrachtet man die CRF-Theorie, so wird deutlich, dass relevante konkrete Potentialfunktionen durch ein hohes (positiv oder negativ) Gewicht (λ) definiert sind. Um Überangepasstheit zu vermeiden, sind nun zwei Vorgehen denkbar.

Beide Verfahren nutzen einen λ-Schwellwert. Das erste Verfahren betrachtet die trainierten Gewichte nach der Trainingsphase. Gewichte, die kleiner oder gleich dem λ-Schwellwert sind, werden auf 0 gesetzt, und fallen somit beim Anwenden eines Modells heraus. Man versucht also, nur relevante konkrete Potentialfunktionen zu benutzen. Bei der zweiten Methode werden nach einer vorgegebenen Anzahl von Iterationen die Gewichte betrachtet, und jede Potentialfunktion mit einem Gewicht, das kleiner oder gleich dem λ-Schwellwert ist, wird gelöscht. Um dann keine Trainingsfehler zu erzeugen, muss die Trainingsphase neu gestartet werden. Dieses Vorgehen sorgt dafür, dass nur mit den konkreten Potentialfunktionen trainiert wird, die nach einer gewissen Anzahl von Iterationen – unter Umständen auch sehr wenigen – bereits relevant sind. Später könnten – was somit vermieden wird - durchs Training auch nicht relevante konkrete Potentialfunktionen über den Schwellwert kommen, was schließlich zur Überangepasstheit führen kann.

Allerdings verspricht eher das zweite Verfahren Erfolg, da nur dort vor dem Auftreten der Überangepasstheit die Potentialfunktionen manipuliert werden.

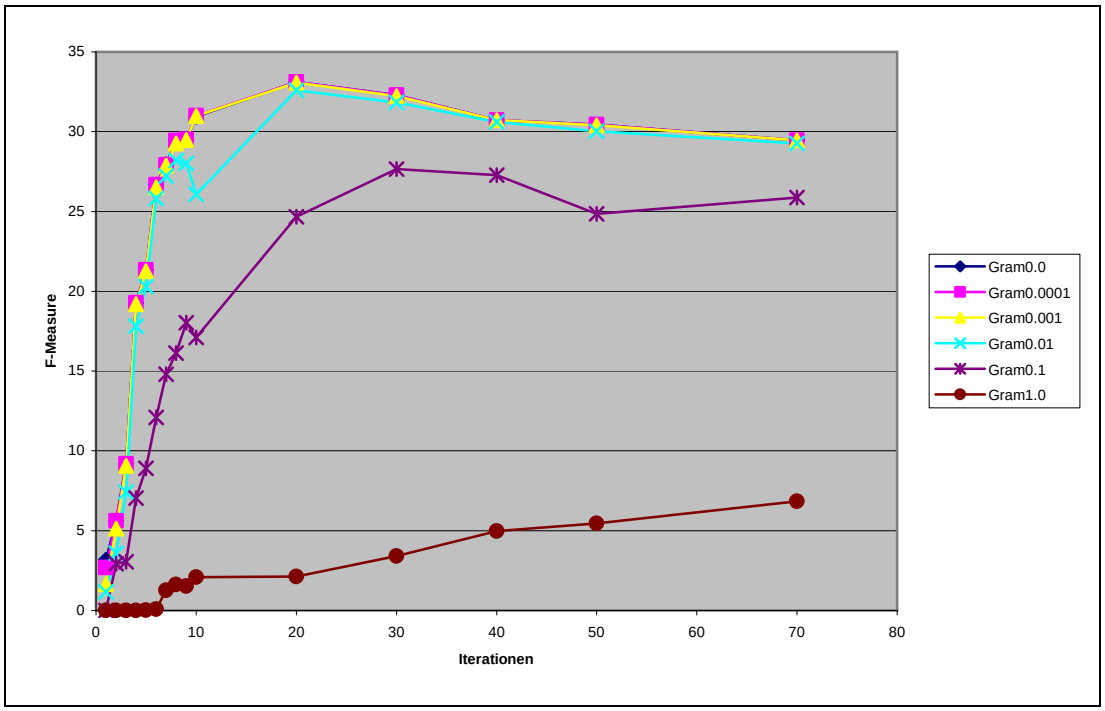


Abbildung 24: Nutzung relativer Positionen mit Beschneidung der Gewichte an verschiedenen Punkten

Abbildung 24 zeigt die Ergebnisse der Versuche des ersten Verfahrens mit Kombination ‚G_-3-2-10' und verschiedenen λ-Schwellwerten. Allerdings werden, wie schon vermutet, keine Verbesserungen in der Güte erzeugt. Erst bei sehr hohen λ-Schwellwerten wird zudem die Überanpassung vermieden, dies resultiert jedoch in sehr niedriger Ergebnisgüte (λ-Schwellwert=1.0).

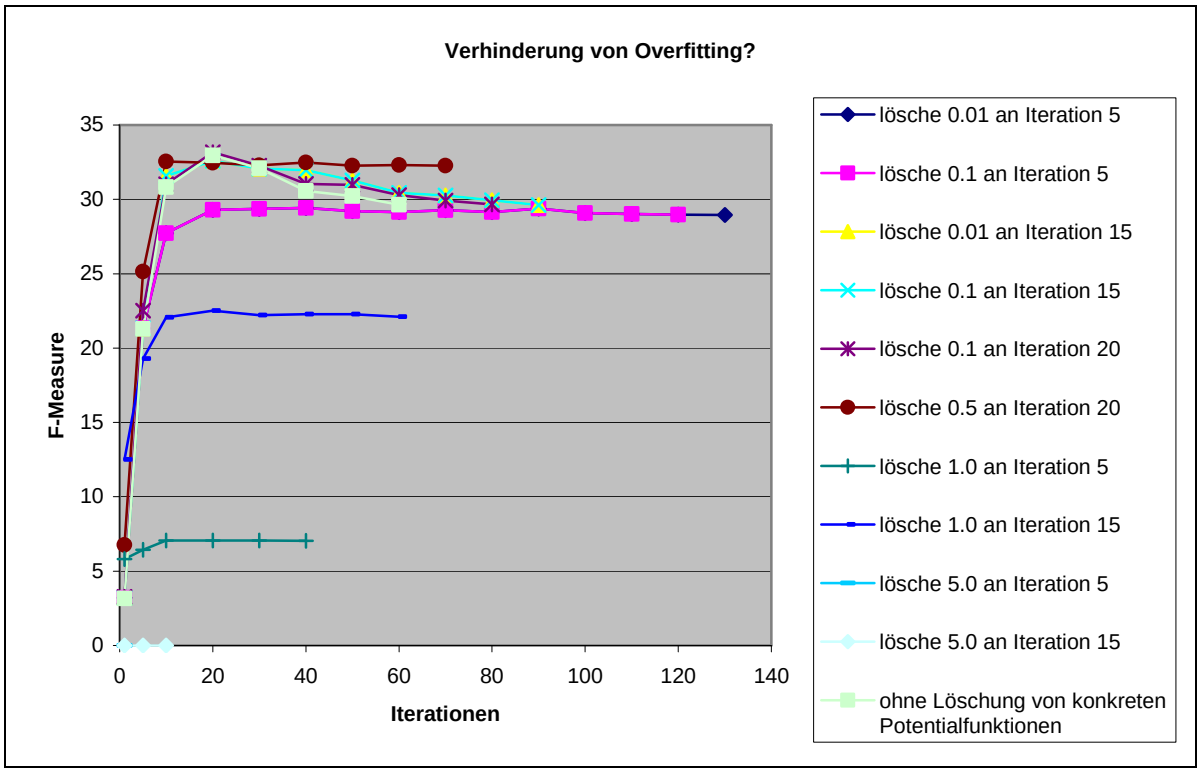


Abbildung 25: Nutzung relativer Positionen mit Beschneidung der Gewichte an verschiedenen Punkten nach einer gewissen Anzahl von Iterationen

Abbildung 25 zeigt Ergebnisse des zweiten Verfahrens. Die weiße Linie zeigt deutlich die Überangepasstheit – hier die Kombination ‚G_-3_-2_-1_0'. Vermieden wird die Überangepasstheit fast vollständig, wenn man nach 20 Iterationen jede konkrete Potentialfunktion mit einem Gewicht, das kleiner gleich 0,5 ist, herausnimmt. Trainiert man mit den übrig gebliebenen Potentialfunktionen neu, so umgeht man das Problem der Überangepasstheit. Die übrigen Versuche bringen alle entweder viel schlechtere Ergebnisse oder enden auch in Überangepasstheit. Es ist daher sehr schwer, einerseits die Anzahl der Iterationen, nach denen gestoppt werden soll, und andererseits die Gewichtsgröße, bei der beschnitten werden soll, herauszufinden.

Anscheinend ist der einzige Weg, Überangepasstheit einfach und sicher zu vermeiden, der, die Anzahl der Iterationen zu evaluieren, bei der das Maximum in der Güte erreicht ist.

13.4.3 Untersuchung von Sequenzen

Relative Positionen bewirken zwar Verbesserungen in der Güte der Ergebnisse, allerdings tritt auch vermehrt das Problem der Überanpassung auf.

Nun soll untersucht werden, ob durch die Benutzung von Sequenzen erstens auch Verbesserungen in der Güte entstehen und zweitens die Überanpassung reduziert wird.

(*)	(**)	(***)	(****)	(*****)	(******)
W_-2-1	38560	ca. 3	39	113 Sekunden	30,29 %
W_-10	10469	ca. 3,5	51	182 Sekunden	31,26 %
W_-101	9761	ca. 3,5	49	176 Sekunden	31,07 %
W_01	10884	ca. 2	46	**81 Sekunden**	30,66 %
W_12	38151	ca. 3	41	120 Sekunden	**35,01 %**
W_123	49456	ca. 6	45	286 Sekunden	30,75 %
W_12_23_123	80205	ca. 6	33	183 Sekunden	27,67 %

Tabelle 27: Kombination der Potentialfunktion WordFeatures mit WordFeatures-Sequenzen

Die Versuche, deren Ergebnisse in Tabelle 27 vorgestellt sind, benutzen jeweils die bekannte Potentialfunktion WordFeatures. Zusätzlich enthalten die Versuche noch eine oder mehrere Sequenzen der Potentialfunktion WordFeatures. Die Kombinationen sind folgendermaßen zu lesen: ‚W_-2-1' bedeutet, dass es sich um die Potentialfunktion WordFeatures handelt und außerdem noch eine Sequenz von Position ‚-1' bis ‚-2' vorhanden ist. Die Versuche zeigen (siehe Abbildung 26 und Abbildung 27), dass bei der Benutzung von Sequenzen das Problem der Überanpassung nicht extrem auftritt. Einzig die Sequenz von Position ‚1' bis ‚2' bringt einen Zugewinn in der Güte. In Abbildung 26 wird deutlich, dass die Benutzung von vielen Sequenzen eine schnellere Konvergenz (gemessen an den Iterationen) zur Folge hat, jedoch auch eine Verschlechterung der Güte nach sich zieht. Abschließend sollen Sequenzen noch anhand von GramFeatures überprüft werden.

(*)	(**)	(***)	(****)	(*****)	(******)
G_-2-1	22628	ca. 4,5	121	**9,5 Minuten**	33,76 %
G_-10	21226	ca. 4,5	130	10 Minuten	33,42 %
G_-101	20640	ca. 4,5	128	**9,5 Minuten**	33,29 %
G_01	21235	ca. 5	129	10 Minuten	33,72 %
G_12	22562	ca. 4,5	158	12 Minuten	33,13 %
G_123	21647	ca. 5	117	**9,5 Minuten**	**33,90 %**
G_12_23_123	27367	ca. 5	137	12 Minuten	33,11 %

Tabelle 28: Kombination der Potentialfunktion GramFeatures mit GramFeatures-Sequenzen

Tabelle 28 zeigt, dass keine der Kombinationen mit Sequenzen die Güte der Versuche mit GramFeatures allein erreicht. Der Verlauf der Versuche soll Aufschluss darüber geben, ob vielleicht durch Überanpassung die maximalen Gütewerte schon vorher vorlagen.

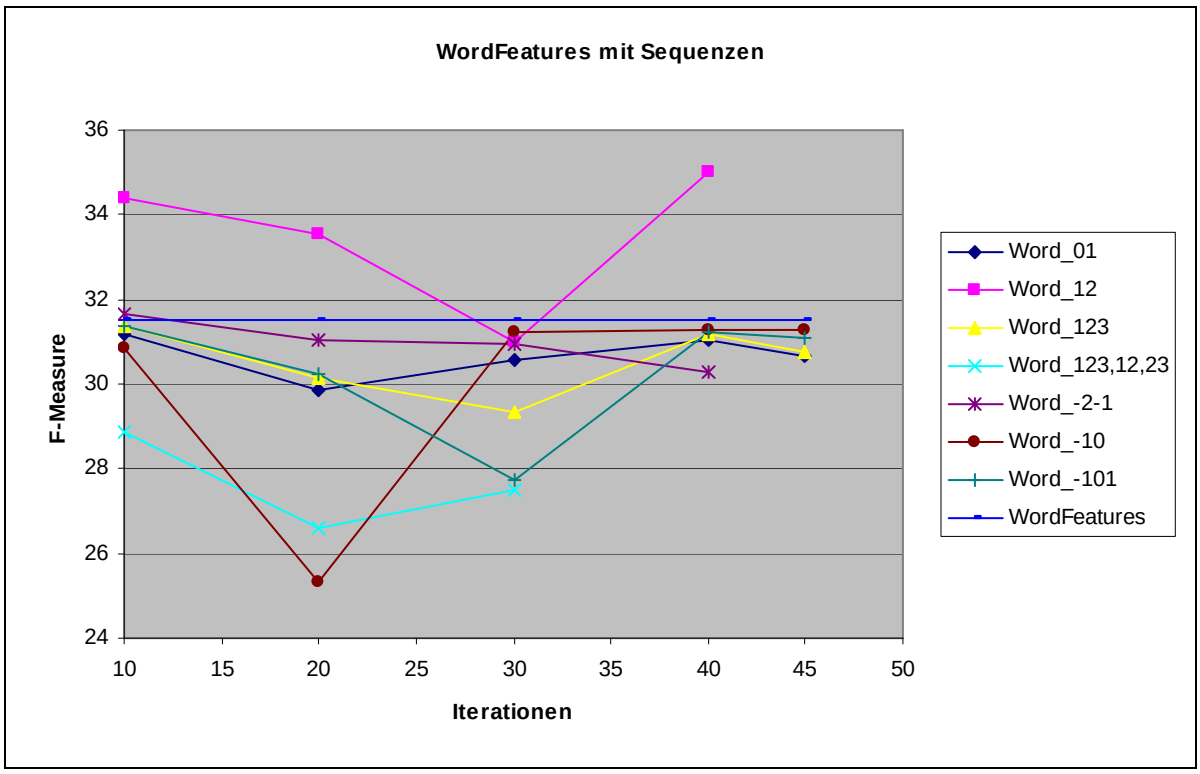


Abbildung 26: WordFeatures mit Sequenzen

Abbildung 26 und Abbildung 27 zeigen, dass, auch wenn einige Punkte über der Ergebnisgüte der Versuche ohne Sequenzen liegen, kaum Verbesserungen durch die Benutzung von Sequenzen zu erzielen sind. Die Verbesserungen durch die relativen Positionen sind eindeutig höher einzustufen. Aus diesem Grund sollen in späteren Versuchen nur relative Positionen benutzt werden.

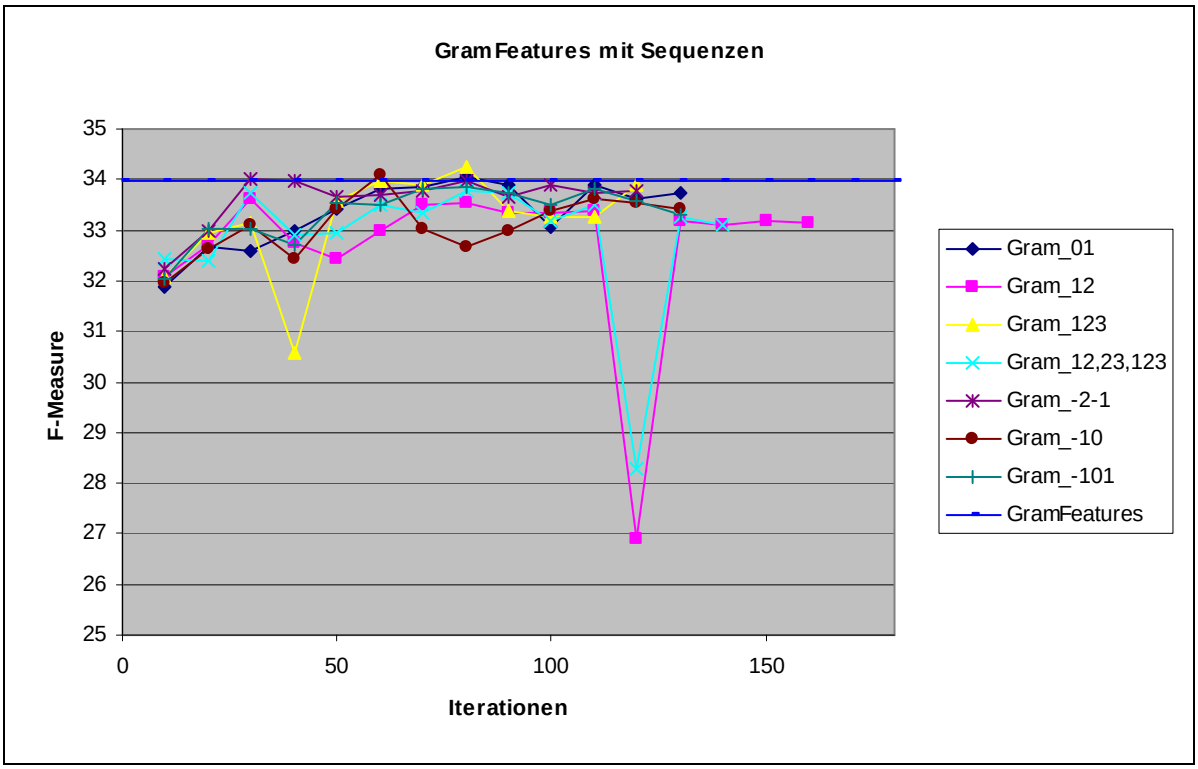


Abbildung 27: GramFeatures mit Sequenzen

13.5 Verknüpfung relativer Positionen mit erfolgreichen Kombinationen

Abschließend sollen noch Versuche mit relativen Positionen und bereits erfolgreichen Kombinationen gemacht werden. Kapitel 13.3 ergab, dass die Kombination aus `WordFeatures`, `GramFeatures`, `ConcatRegexFeatures`, `EdgeFeatures`, `PrefixFeatures` und `SuffixFeatures` die besten Ergebnisse liefert. Daher soll nun diese Kombination um relative Positionen erweitert werden. Abbildung 28 zeigt die Ergebnisse dieser Versuche.

Augenscheinlich ist das Problem der Überanpassung bei vielen relativen Positionen auch in dem Sinne vorhanden, dass die Ergebnisse schlechter werden, sofern die Anzahl der relativen Positionen ansteigt. Die Kombination ‚W012_G012_R_E_P012_S012' müsste ansonsten bessere Ergebnisse liefern als die Kombination ‚W012_G012_R_E_P_S'. Jedoch sind die Ergebnisse sogar schlechter. Es ist sogar so, dass beide Kombinationen immer noch schlechter sind, als die ursprüngliche Kombination ‚W_G_R_E_P_S'. Zur Überprüfung sollen nun noch einige andere Kombinationen untersucht werden.

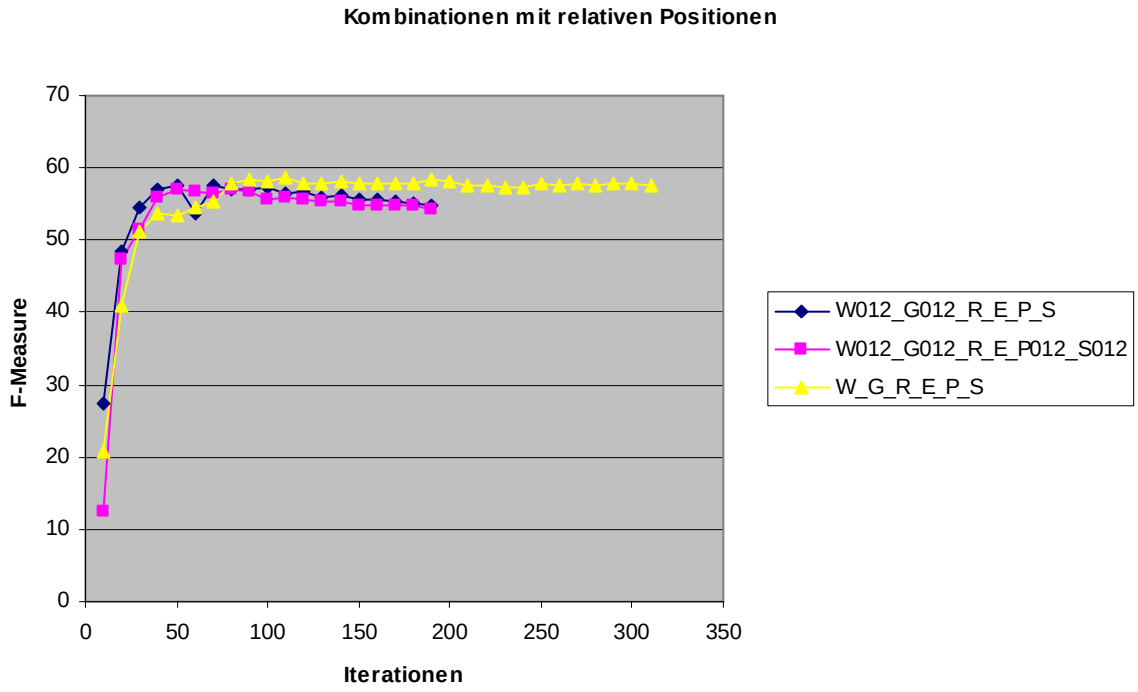


Abbildung 28: Erfolgreiche Kombinationen erweitert um relative Positionen (W012=WordFeatures mit relativen Positionen an Stelle 0, 0+1 und 0+2 usw.)

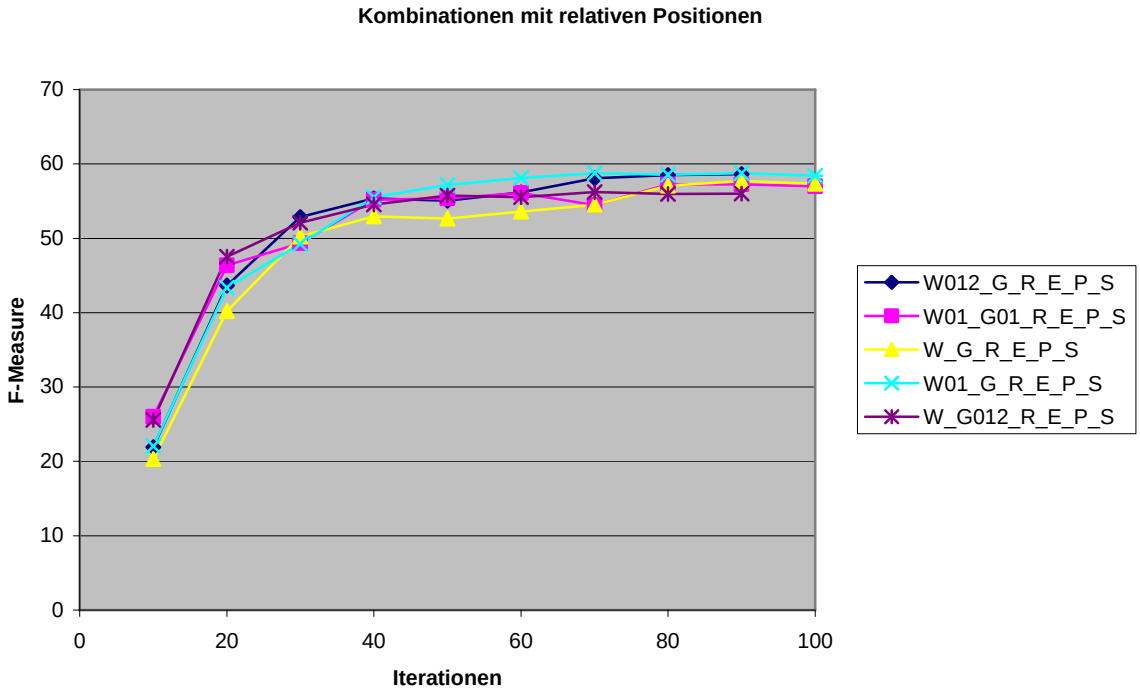


Abbildung 29: Erfolgreiche Kombinationen erweitert um relativen Positionen (W012=WordFeatures mit relativen Positionen an Stelle 0, 0+1 und 0+2 usw.)

(*)	(**)	(***)	(****)	(*****)	(******)
W_G_R_E_P_S	37266	ca. 13,5	ca. 310	ca. 70 Minuten	56,66 %
W01_G01_R_E_P_S_1.0	64189	ca. 19	ca. 110	ca. 36 Minuten	56,94 %
W012_G_R_E_P_S_1.0	55206	ca. 16	ca. 90	**ca. 26 Minuten**	**58,65 %**
W01_G_R_E_P_S_1.0	46065	ca. 16	ca. 110	ca. 30 Minuten	58,40 %
W_G012_R_E_P_S_1.0	73544	ca. 23	ca. 90	ca. 35 Minuten	55,99 %

Tabelle 29: Vergleich von Kombinationen mit relativen Positionen

Frühere Versuche (zum Beispiel Abbildung 28) haben gezeigt, dass, obwohl das Maximum bereits erreicht wurde, die Konvergenz – also das Abbruchkriterium – noch nicht erreicht wurde. Um eine frühere Konvergenz zu erreichen, wurde die Nullstelle für diese Versuche mit 1.0 parametrisiert. Dieser Umstand erklärt die Tatsache, dass die Versuche – bis auf ‚W_G_R_E_P_S', der nicht mit der anderen Nullstelle arbeitet - allesamt früher beendet sind. Tabelle 29 und Abbildung 29 enthalten die Ergebnisse der Kombinationen mit relativen Positionen. Weiterhin ist das Maximum der Kombinationen ‚W01_G_R_E_P_S' und ‚W012_G_R_E_P_S' um fast 2% größer als bei der Kombination ‚W_G_R_E_P_S' (siehe Tabelle 29). Aus diesem Grund werden anschließend die Kombinationen ‚W01_G_R_E_P_S' und ‚W012_G_R_E_P_S' auf den kompletten Trainingsdatensatz angewandt, um die Ergebnisse schließlich über dem Testdatensatz zu evaluieren.

13.6 Nutzung des kompletten Trainingsdatensatzes

Die Kombination ‚W01_G_R_E_P_S' ergibt auf dem Trainingsdatensatz gelernt folgende Ergebnisse.

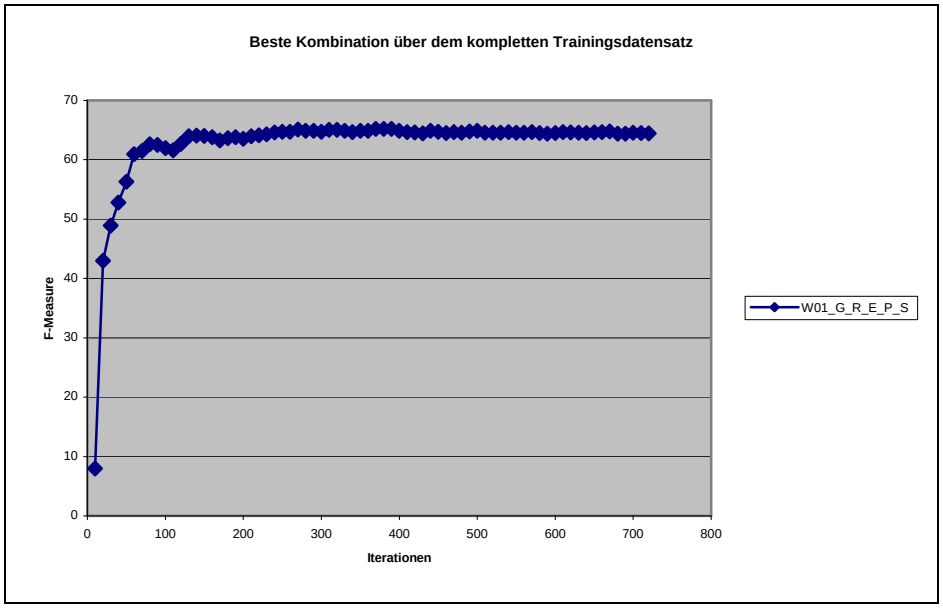


Abbildung 30: Ergebnisse des über dem kompletten Trainingsdatensatz gelernten Modells

Es ist offensichtlich, dass die Konvergenz im Hinblick auf den optimalen F-Measure-Wert schon nach 200 bis 300 Iterationen erreicht ist. Bei der internen Berechnung ist das Optimum (Nullstelle des Gradienten) allerdings noch nicht erreicht. Daher wird der Versuch mit einer höher parametrisierten (Wert=5.0) Nullstelle wiederholt.

(*)	(**)	(***)	(****)	(*****)	(******)
W01_G_R_E_P_S_1.0	129420	ca. 163	ca. 720	ca. 33,5 Stunden	64,45 %
W012_G_R_E_P_S_5.0	161463	ca. 195	ca. 250	**ca. 13,5 Stunden**	64,46 %
W01_G_R_E_P_S_5.0	129420	ca. 162	ca. 300	**ca. 13,5 Stunden**	**64,86 %**

Tabelle 30: Vergleich der Versuche über dem kompletten Datensatz mit unterschiedlichen Nullstellen

Tabelle 30 zeigt die Versuche im Vergleich, wobei zwischen gleichen Kombinationen mit unterschiedlichen Nullstellenparametern natürlich nur Unterschiede in den Iterationen und der Laufzeit auftreten, da beide Versuchsläufe unterschiedlich schnell konvergieren. Die Unterschiede in der Güte resultieren einfach aus den schwankenden Ergebnissen gegen Ende des Versuchs (siehe Abbildung 30). Auf dem kompletten Trainingsdatensatz trainiert, ist die Kombination W012_G_R_E_P_S im Gegensatz zum Sample schlechter als die Kombination W01_G_R_E_P_S.

13.7 Einordnung des eigenen Verfahrens

Die in Kapitel 11.2 vorgestellten Verfahren sollen nun mit dem vom Diplomanden entwickelten Verfahren verglichen werden. Nachdem die Verfahren miteinander verglichen wurden, sollen die Vorteile und Mängel des vom Diplomanden entwickelten Verfahrens ausgearbeitet werden.

13.7.1 Vergleich der vorgestellten Verfahren

Als erstes wird nun das Verfahren des Diplomanden mit den vorgestellten Verfahren verglichen, indem die Ergebnisse der Verfahren mit denen des Diplomanden (siehe Kapitel 13.6) in Relation gesetzt werden.

Autor	Verfahren	F-Measure (über Test-daten)	Iterationen bis Kon-vergenz	Zeit bis Konvergenz
B. Settles	CRF - orthographic features	69,8 %	230	18 Stunden
	CRF – complete fea-tureset	69,5 %	152	9 Stunden
J. Finkel	MEMM	70,1 %	n. v.	k. A.
G. Zhou	SVM + HMM	72,6 %	n. v.	k. A.
F. Jungermann	CRF – 10%iges Sample	58,4 %	110	0,5 Stunden
	CRF – kompletter Trainingsdatensatz	64,86 %	300	13,5 Stun-den

Tabelle 31: Vergleich bekannter Verfahren auf dem JNLPBA-Datensatz mit dem Verfahren des Diplomanden

Die erzielten F-Measure-Werte des vom Diplomanden entwickelten Systems liegen knapp 5% unter dem von Settles entwickelten System. Erklärungen hierfür werden im nächsten Kapitel gegeben.

13.7.2 Erläuterung der eigenen Ergebnisse

Der größte Unterschied des vom Diplomanden entwickelten Systems zu den in Kapitel 11.2 vorgestellten Verfahren ist die Tatsache, dass alle vorgestellten Systeme explizit auf den Datensatz abgestimmte Merkmale (features) benutzen. Das Verfahren von Settles, das konkret mit dem in dieser Arbeit vorgestellten System verglichen werden kann, da es auch CRF benutzt, benutzt beispielsweise 17 Dictionaries, die ausschließlich für den konkreten Datensatz erzeugt wurden. Ein weiteres nicht benutztes Merkmal ist die Generalisierung der Worte (siehe 11.2.1). Dieses Verfahren wäre allerdings mit einem gewissen Zeitaufwand problemlos implementierbar.

Das von Finkel benutzte Verfahren (11.2.2) benutzt auch ein Dictionary beziehungsweise gazetteer. Zusätzlich wird auch ein POS-tagger verwandt, der allerdings auf biomedizinischen Daten trainiert ist. Zudem greift Finkel auf das Internet als Wissensbasis zu, um zusätzliche Daten zu erfassen.

Zhou benutzt zudem noch ein Verfahren zur Behandlung von Synonymen sowie Abkürzungen. Aber auch hier werden externe Dictionaries verwendet.

Abschließend kann man sagen, dass die schlechteren Ergebnisse des vom Diplomanden entwickelten Systems auf die allgemeine Verwendbarkeit dieses Systems zurückzuführen sind. Da das System möglichst auf jede Domäne anwendbar bleiben sollte, wurde auf speziell für einen Datensatz entwickelte Potentialfunktionen verzichtet. Dies hat natürlich zur Folge, dass die Ergebnisse nicht so gut sind, wie speziell für die Aufgabe entwickelte Konkurrenzsysteme. Allerdings könnte man allgemein nutzbare Werkzeuge wie das web-querying (siehe 11.2.2) übernehmen, um das Internet als Wissensbasis zur Anreicherung des Datensatzes zu benutzen.

13.8 Übertragung der Versuche auf den CoNLL03-Datensatz

13.8.1 Ergebnis mit der besten Kombination der JNLPBA-Daten

Bevor tiefergehende Untersuchungen auf dem CoNLL03-Datensatz angestellt werden, soll die auf dem JNLPBA-Datensatz entwickelte Kombination ‚W01_G_R_E_P_S' auf den CoNLL03-Daten trainiert und schließlich auf den Testdatensatz angewandt werden. Über dem kompletten CoNLL03-Trainingsdatensatz trainiert, lieferte die Kombination mit einer parametrisierten Nullstelle von 0.1 folgendes Ergebnis:

(*)	(**)	(***)	(****)	(*****)	(******)
W01_G_R_E_P_S_0.1	122979	ca. 69	ca. 220	ca. 4,25 h	60,66 %

Tabelle 32: Ergebnis der besten Kombination von Potentialfunktionen auf dem JNLPBA-Datensatz angewandt auf den CoNLL03-Datensatz

13.8.2 Ergebnisse mit einzelnen Potentialfunktionen

Um unter Umständen noch bessere Ergebnisse als das in 13.8.1 erzielte Ergebnis zu erhalten, werden einige der schon für den JNLPBA-Datensatz gemachten Versuche auf dem CoNLL03-Datensatz wiederholt. Die einzelnen Potentialfunktionen (vergleiche 13.1.1) werden nun auf dem CoNLL03-Datensatz angewandt. Die Ergebnisse sind jedoch sehr schlecht, was die folgende Tabelle 33 zeigt:

(*)	(**)	(***)	(****)	(*****)	(******)
Word	35030	ca. 7,5	ca. 70	ca. 8,5 Minuten	2,02 %
ConcatRegex	150	ca. 16	ca. 640	ca. 3 Stunden	0,90 %
Gram	37295	ca. 19	ca. 110	ca. 35 Minuten	1,81 %
Pos	127	ca. 6	ca. 270	ca. 25 Minuten	0,32 %
Prefix	11534	ca. 11	ca. 60	ca. 10 Minuten	2,01 %
Suffix	3001	ca. 7,5	ca. 150	ca. 20 Minuten	1,42 %

Tabelle 33: Versuche mit einzelnen Potentialfunktionen auf dem CoNLL03-Datensatz

Diese Ergebnisse sind im Grunde indiskutabel. Den einzigen Nutzen, den man aus ihnen ziehen kann ist die Erkenntnis, dass auf einigen Datensätzen die Ergebnisse mit einzelnen Potentialfunktionen extrem schlechte Ergebnisse liefern und somit keine Aussagekraft haben.

13.8.3 Verschiedene Kombinationen auf dem CoNLL03-Datensatz

Einige Kombinationen von Potentialfunktionen erwiesen sich auf dem JNLPBA-Datensatz als erfolgreich. Vorerst sollen die Kombinationen aus nicht vom Diplomanden erstellten Potentialfunktionen untersucht werden (vergleiche 13.1.2).

(*)	(**)	(***)	(****)	(*****)	(******)
W_E	35066	ca. 17	ca. 290	ca. 85 Minuten	0,43 %
W_R	35180	ca. 21,5	ca. 450	ca. 2,5 Stunden	1,21 %
W_R_E	35216	ca. 30,5	ca. 450	ca. 4 Stunden	0,28 %
W_R_E_St_En_U	35234	ca. 31	ca. 400	ca. 3,5 Stunden	0,35 %

Tabelle 34: Kombinationen von Potentialfunktionen auf den CoNLL03-Datensatz angewandt

Tabelle 34 zeigt hingegen, dass die Erfolge auf dem CoNLL03-Datensatz ausbleiben. Wäre nicht durch Tabelle 32 die Gewissheit gegeben, dass adäquate Ergebnisse zu erzielen sind, könnte das benutzte System bei solch schlechten Ergebnissen in Frage gestellt werden.

In Tabelle 35 sind die Ergebnisse aufgeführt, die mit Kombinationen aus bereits vorhandenen und vom Diplomanden entwickelten Potentialfunktionen erzielt werden.

(*)	(**)	(***)	(****)	(*****)	(******)
W_G	72325	ca. 29	ca. 85	**ca. 40 Minuten**	45,24 %
G_P_S	51830	ca. 28	ca. 125	ca. 60 Minuten	44,90 %
W_G_P_S	86860	ca. 38	ca. 100	ca. 65 Minuten	**46,33 %**

Tabelle 35: Kombination von eigenen und vorhandenen Potentialfunktionen auf dem CoNLL03-Datensatz

Man sieht in Tabelle 35, dass erst durch das Hinzufügen der vom Diplomanden entwickelten Potentialfunktionen akzeptable Ergebnisse erzielt werden. Auch entwickelt sich erst bei diesen Kombinationen eine respektable Laufzeit. Ohne die vom Diplomanden entwickelten Potentialfunktionen wären auf diesem Datensatz also nur sehr schlechte Ergebnisse zu erzielen gewesen. Nun wird noch der Einfluss der Potentialfunktionen `ConcatRegexFeatures` und `EdgeFeatures` auf die Kombination ‚W_G_P_S' überprüft.

W_G_E_P_S	86885	ca. 46	ca. 165	**ca. 120 Minuten**	**58,19 %**
W_G_R_P_S	86985	ca. 54	ca. 310	ca. 270 Minuten	47,59 %

Tabelle 36:Untersuchung des Einflusses der Potentialfunktionen ConcatRegexFeatures sowie EdgeFeatures auf die Kombination W_G_P_S

Tabelle 36 zeigt, dass - wie auch schon auf den JNLPBA-Daten (vergleiche 13.1.2) - die Hinzunahme der Potentialfunktion `EdgeFeatures` einen enormen Gütegewinn mit sich bringt. Allerdings muss hierfür anscheinend, was auch in den früheren Kapiteln deutlich wurde, eine Kombination erweitert werden, deren Ergebnisse an sich schon eine gewisse Güte liefern.

Kapitel 13.8.1 zeigt die Ergebnisse mit der Versuchskombination ‚W01_G_R_E_P_S' bei einer parametrisierten Nullstelle von 0.1. Abschließend soll nun noch eine Versuchsreihe derselben und zwei ähnlichen Kombinationen bei einer parametrisierten Nullstelle von 1.0 gemacht werden. Die Ergebnisse sind in Tabelle 37 zu sehen.

(*)	(**)	(***)	(****)	(*****)	(******)
W01_G_R_E_P_S	122979	ca. 66	ca. 135	**ca. 2,5 Stunden**	**60,67 %**
W012_G_R_E_P_S	157849	ca. 91	ca. 125	ca. 3 Stunden	60,20 %
W_G01_R_E_P_S	122670	ca. 91	ca. 135	ca. 3,5 Stunden	59,69 %

Tabelle 37: Versuche mit unterschiedlichen Kombinationen und parametrisierter Nullstelle von 1.0 auf dem CoNLL-Datensatz

Die Versuche zeigen, dass der höhere Parameter für die Nullstelle eine früherer Konvergenz bei gleich bleibenden Ergebnissen erreicht. Auch wird deutlich, dass die beste Kombination immer noch die schon in 13.8.1 benutzte Kombination ist, die auch auf dem JNLPBA-Datensatz die besten Ergebnisse erzielt.

13.8.4 Erzielte Ergebnisse während des CoNLL-Workshops 2003

Für den CoNLL-Workshop 2003 wurden 16 Systeme zur Bearbeitung des Datensatzes entwickelt, deren F-Measure über dem Testdatensatz zwischen 47,74% und 72,41% lagen (siehe [31]). Zur Veranschaulichung der Ergebnisse des Diplomanden wurden sie in Tabelle 38 in die Ergebnisse der im CoNLL-Workshop 2003 benutzten Systeme eingeordnet.

Autor des Systems	Erzielte Güte (F-Measure)
1.) Florian	72,41 %
…	…
13.) Hendrickx	63,02 %
Jungermann	**60,67 %**
14.) De Meulder	57,27 %
15.) Whitelaw	54,43 %
16.) Hammerton	47,74 %

Tabelle 38: Einordnung der Ergebnisse des Diplomanden auf den CoNLL03-Daten

13.8.5 Einordnung der eigenen Ergebnisse

Um die Handhabung des vom Diplomanden entwickelten Systems zu überprüfen, wurde es ohne besondere Untersuchung der Daten auf den CoNLL03-Datensatz angewandt. Einzig die Tatsache, dass die Datensätze nicht mit dem originalen iob-tagging versehen sind, verursachte geringe Schwierigkeiten. Allerdings wurden die Datensätze mittels eines Skripts ins iob-tagging überführt, so dass sie problemlos bearbeitet werden konnten.

Zwar liegt die erzielte Güte zwischen den Gütewerten des dritt- und viertschlechtesten Systems des Workshops. Allerdings muss man auch die Umstände betrachten, und dafür sind die Ergebnisse doch sehr beachtlich.

Es wurden keinerlei Vorversuche auf dem Datensatz gemacht. Dass die Kombination ‚W01_G_R_E_P_S' auf dem CoNLL-Datensatz gut arbeitet, war nicht von vornherein

klar. Allerdings zeigen die nachträglich gemachten Versuche (13.8.2 und 13.8.3), dass dies doch der Fall ist. Diese Versuche zeigen auch deutlich, dass erst durch die vom Diplomanden entwickelten Potentialfunktionen respektable Ergebnisse erzielt werden konnten.

Viele der auf den CoNLL03-Datensatz angewandten Verfahren nutzen externe Datenquellen und nicht-markierte Daten, was ja auch explizit für den Workshop gewünscht war. Dass das vom Diplomanden entwickelte Verfahren hierauf verzichtet, relativiert die erzielten Ergebnisse.

Die geringe Laufzeit des Versuchs ist etwas überraschend, allerdings beruht diese auf der Tatsache, dass der JNLPBA-Trainingsdatensatz mehr als zweieinhalb mal so groß ist wie der CoNLL03-Trainingsdatensatz.

14 Abschluss

14.1 Rückblick

In dieser Arbeit wurden die Grundsätze der NER erläutert. Nach der Ausarbeitung der möglichen Problemfelder im Bereich der NER wurde das Prinzip der Generalisierung als Lösungsansatz definiert. CRF wurden daraufhin als das aktuelle Verfahren zur NER dargestellt. Die Entstehung der CRF als Weiterentwicklung der Prinzipien der HMM sowie MEMM wurde anschließend erläutert. Hierbei wurde vor allem auf die HMM ausführlich eingegangen, da viele Algorithmen und Verfahren der HMM auch für die CRF von Belang sind. Die Theorie der CRF wurde mit der Definition der CRF als Spezialisierung von MRF begonnen und umfasste zudem die Definition der drei effizient zu bearbeitenden Hauptpunkte der CRF:

- die Bestimmung der bedingten Wahrscheinlichkeit für eine Beobachtungssequenz
- die Bestimmung der am besten passenden Labelsequenz für eine Beobachtungssequenz
- die Gewichte eines CRF für bestimmte Label- und Beobachtungssequenzen optimal einstellen.

Neben der NER sind CRF natürlich auch in anderen Gebieten einsetzbar. Diese Gebiete wurden vorgestellt. Außerdem wurden Weiterentwicklungen der CRF vorgestellt, denen man sich unter Umständen in weiteren Arbeiten widmen könnte.

Die Optimierung des Gewichtsvektors umfasste ein zusätzliches Kapitel, da hier seit Entstehung der CRF umfassende Entwicklungen stattfanden. Aufgrund des knappen Zeitrahmens dieser Arbeit wurde als Implementierungsgrundlage die Implementierung von Sunita Sarawagi benutzt. Diese Implementierung, die sich im Laufe der Diplomarbeit als Basis für vergleichende Versuche entwickelt hat, wurde natürlich ebenso dargestellt, wie die vom Diplomanden vorgenommenen Erweiterungen sowie Implementierungen. Die Implementierungen des Diplomanden richteten sich vor allem nach den anfangs dargelegten Prinzipien der Generalisierung.

Um eine intuitive und nutzbare Oberfläche für die Nutzung der Implementierung zu haben, wurde das Programm in YALE – eine Versuchsumgebung für maschinelles Lernen – als Baustein integriert. Zusätzliche Operatoren wurden geschaffen, um NER in YALE überhaupt möglich zu machen. Zu diesen Operatoren zählen zusätzlich zu den CRF-Operatoren ein Operator zur Einbindung von Texten, der auch das Anreichern der Textdatensätze um NER-spezifische Merkmale wie zum Beispiel Prä- und Suffixe anbietet, sowie ein Evaluierungsoperator, der die Bewertung gemachter Versuche ermöglicht. Die soeben erwähnten Operatoren wurden mitsamt ihrer möglichen Parameter vorgestellt.

Die Implementierung sollte natürlich auch angewandt und verglichen werden. Hierfür wurden der JNLPBA- und CONLL03-Datensatz vorgestellt. Der JNLPBA-Datensatz aus dem Bereich der Biomedizin ist bereits öfters untersucht worden, so

dass er sich gut zur Einordnung des in dieser Arbeit entwickelten Verfahrens eignet. Drei auf diesem Datensatz angewandte Verfahren wurden vorgestellt.

Die Implementierung des Diplomanden wurde zuerst auf den JNLPBA-Datensatz angewandt, wobei die vorhandenen Bausteine der Implementierung mit den Entwicklungen des Diplomanden in Relation gesetzt wurden. Die Entwicklung eines Verfahrens zur Extraktion von Samples aus dem Datensatz sowie die Anwendung dieses Verfahrens zeigten, dass die Benutzung eines zum Beispiel 10%igen Samples eine überproportionale Laufzeitverbesserung ergibt. Spätere Versuche zeigten zudem, dass die aus diesem Sample resultierende Güte der Ergebnisse nur knapp 6% unter der Güte der Ergebnisse aus Versuchen über dem kompletten Datensatz liegen.

In vergleichenden Versuchen zeigte sich, dass die auf dem Prinzip der Generalisierung beruhenden Erweiterungen des Diplomanden Sinn machen, da sie bessere Ergebnisse liefern als die vorhandene Implementierung. Das Prinzip der externen Evidenz, das in dieser Arbeit als relative Positionen beschrieben wird, erbrachte einerseits gute Ergebnisse. Allerdings entstand hierbei das Problem des overfitting, so dass mögliche Methoden zur Vermeidung des overfitting untersucht wurden. Pragmatische Methoden wie die Evaluierung des Abbruchzeitpunktes blieben hierbei die beste Wahl. Es wurde allerdings auch deutlich, dass bei vielen unterschiedlichen Potentialfunktionen ohne viele relative Positionen das overfitting nicht auftritt.

Der Vergleich mit bereits auf dem Datensatz angewandten Methoden zeigte, dass das vom Diplomanden entwickelte Verfahren unabhängig vom zu bearbeitenden Datensatz ist, da es keine domänspezifischen Wörterbücher oder Wortlisten benutzt. Die verglichenen Methoden benutzten allesamt speziell für den Datensatz erzeugte Merkmale oder Wortlisten. Obwohl die Ergebnisse des vom Diplomanden entwickelten Systems im F-Measure 5% schlechter als die Ergebnisse des vergleichbaren CRF-Systems waren, kann man die Ergebnisse aufgrund der Unabhängigkeit der Merkmale vom benutzten Datensatz als vergleichbar bezeichnen. Abschließend wurden einige bereits auf dem JNLPBA-Datensatz benutzte Kombinationen von Potentialfunktionen auf den CoNLL03-Datensatz angewandt. Zwar waren auch hier die Ergebnisse nicht sehr gut, doch die strikte Unabhängigkeit des Verfahrens von bestimmten Datensätzen und die (noch-)Nicht-Benutzung von externen Ressourcen relativieren diese Ergebnisse. Bei Versuchen auf dem CoNLL03-Datensatz zeigte sich, dass erst die vom Diplomanden entwickelten Potentialfunktionen Erfolge einbringen.

Der Diplomand hat somit die CRF nicht nur theoretisch und praktisch vorgestellt, sondern auch ein Werkzeug implementiert, das intuitiv nutzbar ist und ohne besondere Kenntnis eines Datensatzes annähernd gute Ergebnisse erzielt.

14.2 Ausblick

In diesem Kapitel sollen Arbeitsfelder definiert werden, die in dieser Arbeit vielleicht erwähnt, allerdings nicht mehr bearbeitet werden konnten. Diese Arbeitsfelder stel-

len unter Umständen mögliche Ansatzpunkte für zukünftige Diplomarbeiten zum Thema CRF dar.

14.2.1 Neue Potentialfunktionen

Die Potentialfunktionen wurden in dieser Arbeit als der effektivste Ansatzpunkt zur Verbesserung sowie Erweiterung der CRF herausgestellt. Allerdings konzentrierte sich diese Arbeit voll und ganz auf Potentialfunktionen, die Merkmale aus dem zu bearbeitenden Datensatz extrahieren. Es gibt jedoch neben dem Trainingsdatensatz noch weitere Quellen, die zur Informationsgewinnung benutzt werden können. Diese Quellen beziehen sich auf externe Daten und sollen daher externe Quellen genannt werden. Diese Quellen sind:

- Das Internet
- Nicht markierte Daten
- Externe Dictionaries (gazetteers)

Im Wettbewerb auf dem CoNLL03-Datensatz wurde ausdrücklich auf das Interesse der Organisatoren an Lösungen mit externen Quellen hingewiesen (siehe [31]). Einige Systeme erzielten bei der Nutzung von gazetteers und nicht markierten Daten einen F-Measure-Gewinn von über 20% beim englischen Teil der Daten und von 15% beim deutschen Teil der Daten. Die externen Quellen sowie die Systeme, in denen sie bisher zum Einsatz kamen, sollen nun vorgestellt werden, um die Entscheidungsfindung, ob und welche externen Quellen eingebunden werden sollen, zu erleichtern.

Internet

Das Internet wird von vielen aktuellen NER-Verfahren als Wissensbasis benutzt. Der Vorteil ist, dass es einfach benutzbar ist. Leider ist die Fülle an Daten schlecht geordnet, so dass man sich zum Beispiel einer Suchmaschine bedienen muss, um relevante Daten für die Domäne der aktuellen NER-Aufgabe zu erhalten. Von den in Kapitel 11.2 vorgestellten Verfahren, die beim Wettbewerb auf den JNLPBA-Daten die vordersten Plätze belegten, benutzen Settles ([33]) und Finkel ([10]) Anfragen an die Suchmaschine Google, um ihre Datensätze entweder anzureichern (Settles) oder zu klassifizieren (Finkel). Die erzielten F-Measure-Gewinne durch die Hinzunahme des Internets als Wissensbasis werden von Finkel nicht berichtet. Settles allerdings benutzt mehrere externe Lexika sowie das Internet als externe Quellen und verliert an Güte im Vergleich zu Versuchen ohne diese Datenquellen. Somit ist fraglich ob die Benutzung des Internet als Wissensbasis in NER-Versuchen sinnvoll ist. Jedenfalls ist aufgrund der langen Bearbeitungszeiten beim Zugriff auf das Internet lediglich eine Nutzung in der Vorverarbeitung zu sehen.

Nicht markierte Daten

Nicht markierte Daten sind Daten, die derselben Domäne wie der Trainings- und Testdatensatz entstammen. Diese Daten zu benutzen ist höchst interessant, da sie meist in ausreichender Menge zur Verfügung stehen - ganz im Gegenteil zu den markierten Daten, die mühsam „von Hand“ aus einem Teil der unmarkierten Daten erstellt werden müssen. In [30] wird ein Verfahren vorgestellt, was sich die vorhan-

denen unmarkierten Daten zu Nutze macht. Anhand eines markierten Trainingsdatensatzes werden Klassifizierer (SVM – vergleiche 11.2.3) gelernt, die auf die unmarkierten Daten angewandt werden, um diese zu markieren und somit den Trainingsdatensatz zu erhöhen. Durch dieses Verfahren wurde auf den CoNLL03-Daten der bis dato beste F-Measure-Wert für die Klassifikation der NE PERSON um mehr als 5% übertroffen. Allerdings konnten die Werte für die anderen NE nicht übertroffen werden, da Personennamen schlicht einfacher zu klassifizieren sind.

Externe Dictionaries

Gazetteers enthalten Listen von Wörtern, die einer bestimmten NE angehören beziehungsweise voraussichtlich einer bestimmten NE angehören. Sofern alle Einträge sicher einer NE zuzuordnen sind, kann man einen Klassifizierer entwickeln, der Worten im Text, die im gazetteer vorkommen, diese NE zuweist. Man könnte während des Trainings auch überprüfen, welche Worte des Trainingsdatensatzes im gazetteer vorkommen, und dann anhand der beobachteten NE-Vorkommen einen dynamischen Klassifizierer entwickeln. Im Wettbewerb auf dem CoNLL03-Datensatz benutzten alle Systeme gazetteers. Settles zum Beispiel benutzt eine Liste griechischer Buchstaben, um Proteine auf dem JNLPBA-Datensatz zu klassifizieren. Externe Wortlisten können im System des Diplomanden zwar schon eingebunden werden, allerdings wurden bisher noch keine Versuche mit ihnen gemacht.

14.2.2 CRF-Entwicklungen umsetzen

Die in Kapitel 7.2 vorgestellten Entwicklungen könnten unter Umständen umgesetzt werden, um einerseits Verbesserungen in Laufzeit oder Güte zu erzielen.

Die in Kapitel 7.2.1 vorgestellte stochastische Gradientenoptimierung ist momentan noch nicht notwendig. In allen vorgestellten Kapiteln benötigte die benutzte Methode L-BFGS weniger als eine Sekunde zur Berechnung der neuen Gewichte. Da die stochastische Gradientenoptimierung zwar Laufzeitverbesserungen jedoch keine Gewinne in der Güte erzielt, ist es praktisch gesehen unnötig, diese Methode zu implementieren.

Die gravierendste CRF-Entwicklung ist sicher die 2D-CRF ([45]), die allerdings für NER kaum effizient nutzbar sind. Die Einbindung von Lokalitätsmerkmalen in Texten ist zwar denkbar, jedoch ist sie im NER-Kontext eher durch neue Potentialfunktionen realisierbar.

Im Gegensatz dazu ist die semi-CRF ([32]) für den NER-Bereich sehr interessant. Die Einteilung des Textes in Segmente (vergleiche 7.2.2) ist ein wichtiger Punkt, der momentan noch Schritt für Schritt geschieht. Die effiziente, korrekte und direkte Einteilung in Segmente, was bei der semi-CRF zwar mit längerer Laufzeit aber besserer Güte verbunden ist, ist sicher ein Thema, das in Zukunft zu bearbeiten ist – vor allem, da Sarawagi schon eine Implementierung der semi-CRF entwickelt hat.

Literatur

[1] J. Aberdeen, J. Burger, D. Day, L. Hirschman, P. Robinson and M. Vilain: MITRE: Description of the Alembic System used for MUC-6. In Proceedings of the 6th Message Understanding Conference (MUC-6), pages 141-154, 1995.

[2] D. M. Bikel, R. Schwartz and R. M. Weischedel: An Algorithm that Learns What's in a Name. In Machine Learning, 34(1-3), pages 211-231, 1999.

[3] A. Borthwick, J. Sterling, E. Agichtein and R. Grishman: NYU: Description of the MENE Named Entity System as Used in MUC-7. In Proceedings of the Seventh Message Understanding Conference (MUC-7), 1998.

[4] Y. Choi, C. Cardie, E. Riloff and S. Patwardhan: Identifying Sources of Opinions with Conditional Random Fields and Extraction Patterns. In Proceedings of the Human Language Technology Conference/Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP), 2005.

[5] M. Collins: Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In ACL-02, 2002.

[6] CRF Project Page, http://crf.sourceforge.net/, Juni 2006.

[7] A. Culotta, R. Bekkerman and A. McCallum: Extracting social networks and contact information from email and the Web. In First Conference on Email and Anti-Spam (CEAS), Mountain View, CA, 2004.

[8] S. Della Pietra, V. Della Pietra and J. Lafferty: Inducing features of random fields. In IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 4, pages 380-393, 1997.

[9] T. G. Dietterich, A. Ashenfelter and Y. Bulatov: Training Conditional Random Fields via Gradient Tree Boosting. In ICML-04, 2004.

[10] J. Finkel, S. Dingare, H. Nguyen, M. Nissim, G. Sinclair and C. Manning: Exploiting Context for Biomedical Entity Recognition: From Syntax to the Web. In Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA-2004), pages 88-91, 2004.

[11] S. Fischer, R. Klinkenberg, I. Mierswa and O. Ritthoff: YALE 3.0. Yet another learning environment. User guide - Operator Reference - Developer Tutorial, 2005.

[12] M. L. Gregory and Y. Altun: Using Conditional Random Fields to Predict Pitch Accents in Conversational Speech. In 42nd Annual Meeting of the Association for Computational Linguistics (ACL), 2004.

[13] R. Grishman: The NYU System for MUC-6 or Where's the Syntax? In Proceedings of the MUC-6 workshop, Washington, 1995.

[14] R. Grishman and B. Sundheim: Message understanding conference - 6: A brief history. In Proceedings of the International Conference on Computational Linguistics, 1996.

[15] J. Hammersley and P. Clifford: Markov fields on finite graphs and lattices. Unpublished Manuscript, 1971.

[16] Hidden Markov Models – Introduction. http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html, Dezember 2005.

[17] J.-D. Kim, T. Ohta, Y. Tsuruoka and Y. Tateisi: Introduction to the Bio-Entity Recognition Task at JNLPBA. In Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA-2004), pages 70-75, 2004.

[18] J. Lafferty, F. Pereira and A. McCallum: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In International Conference on Machine Learning (ICML'01), 2001.

[19] J. Lafferty, X. Zhu, and Y. Liu: Kernel conditional random fields: representation and clique selection. In Proceedings of ICML-2004, 2004.

[20] A. McCallum, D. Freitag and F. Pereira: Maximum entropy Markov models for information extraction and segmentation. In proceedings of ICML-2000, 2000.

[21] A. McCallum: Efficiently inducing features of conditional random fields. In Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI03), 2003.

[22] D. D. McDonald: Internal and External Evidence in the Identification and Semantic Categorization of Proper Names. In B. Boguraev and J. Pustejovsky, editors, Corpus processing for lexical acquisition, pages 21-39, 1996.

[23] MEDLINE Fact Sheet. http://www.nlm.nih.gov/pubs/factsheets/medline.html, Juni 2006.

[24] Medical Subject Headings (MeSH®) Fact Sheet. http://www.nlm.nih.gov/pubs/factsheets/mesh.html, Juni 2006.

[25] F. Peng, F. Feng and A. McCallum: Chinese Segmentation and New Word Detection using Conditional Random Fields. In Proceedings of The 20th International Conference on Computational Linguistics (COLING), 2004.

[26] A. Quattoni, M. Collins and T. Darrell: Conditional Random Fields for Object Recognition. In Advances in Neural Information Processing Systems 17 (NIPS 2004), 2005.

[27] L. Rabiner: A tutorial on hidden Markov models and selected applications in speech recognition. In Proceedings of the IEEE 77, pages 257-286, 1989.

[28] L. Rabiner and B. Juang: An introduction to hidden Markov models. IEEE ASSP Magazine, pages 4-16, 1986.

[29] M. Rössler: Named Entity Recognition. Doktorarbeit. Institut für Künstliche Intelligenz, Universität Duisburg, 2006.

[30] M. Rössler, K. Morik: Using Unlabeled Texts for Named-Entity Recognition. In Proceedings of the ICML 2005 Workshop on Learning With Multiple Views, Bonn, 2005.

[31] E. F. T. K. Sang and F. D. Meulder: Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In Proceedings of CoNLL-2003, Edmonton, Canada, pages 142-147, 2003.

[32] S. Sarawagi and W. Cohen: Semimarkov conditional random fields for information extraction. In Proceedings of ICML 2004, 2004.

[33] B. Settles: Biomedical Named Entity Recognition Using Conditional Random Fields and Rich Feature Sets. In Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA-2004), 2004.

[34] F. Sha and F. Pereira: Shallow Parsing with Conditional Random Fields. Technical Report CIS TR MS-CIS-02-35, University of Pennsylvania, 2003.

[35] C. Sutton and A. McCallum: An Introduction to Conditional Random Fields for Relational Learning. In Introduction to Statistical Relational Learning, MIT Press, 2006.

[36] C. Sutton, K. Rohanimanesh, and A. McCallum: Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In Proceedings of the ICML Conference, 2004.

[37] A. Torralba, K. P. Murphy, and W. T. Freeman: Contextual models for object detection using boosted random fields. In Proceedings NIPS, 2004.

[38] S. V. N. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, K. P. Murphy: Accelerated Training of Conditional Random Fields with Stochastic Gradient Methods. In Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, 2006.

[39] H. M. Wallach: Conditional Random Fields: An Introduction. Technical Report MS-CIS-04-21. Department of Computer and Information Science, University of Pennsylvania, 2004.

[40] H. M. Wallach: Efficient Training of Conditional Random Fields. Master's thesis, University of Edinburgh, 2002

[41] The Newton-Raphson Method. http://www.math.ubc.ca/~kliu/104-184/newtonmethod.pdf, Juli 2006.

[42] G. Zhou and J. Su: Exploring Deep Knowledge Resources in Biomedical Name Recognition. In Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA-2004), 2004.

[43] G. Zhou and J. Su: Named Entity Recognition using an HMM-based Chunk-Tagger. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL). Philadelphia, pages 473-480, 2002.

[44] G. Zhou, J. Zhang, J. Su, D. Shen and C. Tan: Recognizing Names in Biomedical Texts: a Machine Learning Approach. In: Bioinformatics, Vol.20, pages 1178-1190, 2004.

[45] J. Zhu, Z. Nie, J.-R. Wen, B. Zhang, W.-Y. Ma: 2D Conditional Random Fields for Web Information Extraction. In Proceedings of the ICML Conference, 2005.