Bauhaus-Universität Weimar
Fakultät Medien
Studiengang Mediensysteme

Named Entity Recognition

Sequenzklassifikation mit Hidden Markov Models und dem Perzeptron-Algorithmus

Bachelorarbeit

Beyer, Dustin
Geboren am 10.05.1986 in Lübben (Spreewald)
Matrikelnummer 51657

1. Gutachter: Prof. Dr. Benno Stein

Datum der Abgabe: 14.06.2010

Erklärung

Hiermit versichere ich, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weimar, den 14.06.2010

....

Beyer, Dustin

Zusammenfassung

Diese Bachelorarbeit widmet sich dem Problem der Named Entity Recognition – die Erkennung und Klassifikation von Entitäten in unstrukturierten Textdaten. Ziel der Arbeit ist die Gegenüberstellung von zwei grundsätzlich verschiedenen Verfahren, die für diese Aufgabe verwendet werden können: Hidden Markov Models und der Perzeptron-Lernalgorithmus. Diese Verfahren wurden implementiert, anhand von Experimenten bezüglich ihrer Performanz ausgewertet und in ihrem theoretischen Hintergrund beleuchtet. Das Beste der implementierten Systeme erreichte dabei in seiner optimalen Konfiguration eine Performanz von $65,98\%$ F_1-Measure auf dem deutschen Teil des CoNLL-2003 Shared Task Datensatz und $78,28\%$ auf dem englischen.

Inhaltsverzeichnis

Kapitel 1

Einführung

Die Vorstellung, Computern die Fähigkeit zu verleihen Sprache genauso effizient zu verarbeiten und zu verstehen wie Menschen es vermögen, ist so alt wie die Idee des Computers selbst (Jurafsky u. Martin, 2008, S. 35). Heute sind sprachverarbeitende Systeme[1] bereits Realität, jedoch noch weit davon entfernt, der Auffassungs- und Reaktionsgabe des Menschen gleich zu kommen. Sprachverarbeitende Systeme unterscheiden sich von normalen datenverarbeitenden System durch ihr Wissen über die Sprache, welches über mehrere Ebenen verteilt ist (siehe Jurafsky u. Martin, 2008, S. 37 f.). Beispielsweise benötigt ein System, das akustische Sprachdaten verabeitet, v.a. Wissen über Phonologie (Lehre der Sprachlaute). Das hier bearbeitete Thema *Named Entity Recognition* (NER) verortet sich auf der semantischen Sprachebene[2].

NER stellt ein Teilproblem des Forschungsgebietes *Information Extraction* (IE) dar. Im weiteren Verlauf der Einführung folgt eine kurze Positionierung von NER innerhalb von IE und ein kurzer Exkurs über frühere Arbeiten, die sich diesem Problem gestellt haben. Kapitel 2 widmet sich der umfassenderen Beschreibung von NER, wobei zugleich auf die Problemstellung sowie auf Lösungsansätze eingegangen wird. Im Rahmen dieser Arbeit wurden zwei spezielle Verfahren zur Lösung des NER-Problems ausgewählt und implementiert. Kapitel 3 beinhaltet den theoretischen Hintergrund der Verfahren. In den darauffolgenden Kapiteln 4 und 5 werden die implementierten Algorithmen vorgestellt und anhand von Experimenten in ihrer Performanz einander gegenübergestellt.

---

[1] *Natural Language Processing*-Systeme.

[2] Die Ebene, in der es um die bedeutungstragenden Inhalte der Sprache geht.

1.1 Information Extraction

In der heutigen Zeit sieht sich der Mensch mit einer exorbitanten Informationsflut konfrontiert. Webseiten, Blogs, E-Mails, Zeitungsartikel, historische Dokumente, Lexika und viele weitere Quellen stellen eine schier unüberschaubare Menge an Texten, Bildern und Multimedia-Inhalten bereit, die fast alle Fragen der Welt zu beantworten scheinen. Doch niemand ist dazu in der Lage, diese Masse an Inhalten zu überschauen. Von Bildern und Multimedia-Inhalten abgesehen, liegen die Informationen meist nur in natürlicher Sprache – also unstrukturiert – vor. Ein erster Ansatz zur Lösung dieses Problems ist im Forschungsgebiet *Information Retrieval* (IR) zu finden, in dem man sich v.a. um das Durchsuchen und Bewerten von Dokumenten bezüglich einer formulierten Anfrage beschäftigt. IR-Applikationen verstehen die Texte als *Bags of Words* und können keine strukturellen Informationen ermitteln (Cowie u. Wilks, 1996). Eine Eignung für *Data Mining* und die Beachtung semantischer Gehalte in der Anfrage besteht nicht (McCallum, 2005).

An dieser Stelle setzt Information Extraction an. Das Ziel von IE ist die Strukturierung und Normalisierung der unstrukturierten Textdaten, um die Einbeziehung semantischer Kriterien und Data Mining zu ermöglichen. Die Strukturierung wird durch die Population einer relationalen Datenbank aus den Textdaten umgesetzt, die die Grundlage für eine effiziente, computerbasierte Weiterverarbeitung bildet (McCallum, 2005).

Datenverarbeitung

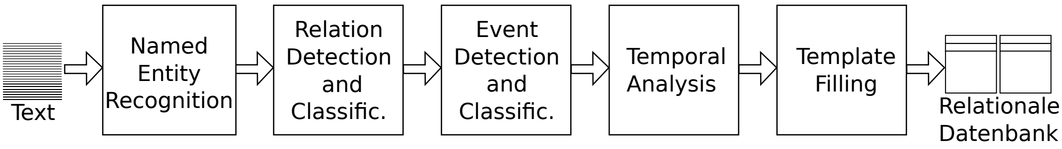


Abbildung 1.1: Die Information Extraction Pipeline mit ihren Teilproblemen (Jurafsky u. Martin, 2008, S. 760 ff.).

Zur Population der relationalen Datenbank aus den unstrukturierten Textdaten wird der IE-Prozess in mehrere Teilprobleme unterteilt, die sequentiell nacheinander abgearbeitet werden. Es entsteht eine pipelineartige Verarbeitungsstruktur. Das Schema der zu erzeugenden Datenbank mit Entitäts- und Beziehungstypen ist eingangs bereits festgelegt. Im IE-Prozess werden die Einträge für die jeweiligen Tabellen und ihre Attribute ermittelt. In Abbildung 1.1 ist die von Jurafsky u. Martin (2008, S. 760 ff.) beschriebene Vorgehensweise dargestellt. McCallum (2005) beschreibt eine abweichende Vorgehensweise.

Die Zielstellung bleibt jedoch die gleiche.

Beispiel 1.1 zeigt ein erstes Beispiel für einen Text, der von einem IE-System verarbeitet werden könnte. Auf dieses Beispiel wird in der Beschreibung der einzelnen Schritte der Pipeline Bezug genommen.

Beispiel 1.1 *Offene Stadtinformationssysteme*
VON [PER Christiane Schulzki-Haddouti] *28. MAI 2010 UM 11:53 UHR*

Das im Herbst erst gegründete [ORG OpenData Network] *hat mit* [MISC OpenBerlin.net] *ein kleines Projekt gestartet, das Keim eines offenen Stadtinformationssystems werden könnte. Es enthält im Moment Maßnahmen des* [MISC Konjunkturpakets II] *sowie statische Karteninformationen zur Sozialstruktur* [LOC Berlins]*. Die Daten bzw. Bilder stammen aus dem* [ORG FIS-Broker der Stadt Berlin].

[MISC OpenBerlin.net] *basiert auf dem Projekt* [MISC Mapnificent] *des Berliner Informatikers* [PER Stefan Wehrmeyer]*, der unter anderem auch die Daten des öffentlichen Nahverkehrs sowie Kriminalitätsdaten verwendet. [...]* [PER Wehrmeyer] *selbst ließ sich vom britischen Projekt* [MISC Mapumental] *inspirieren, die noch in private beta ist.* Quelle: ZEIT ONLINE, im Artikel Offene Stadtinformationssysteme von Christiane Schulzki-Haddouti (28.05.2010)

NER ist der erste Schritt der IE-Abarbeitungsreihenfolge. Dabei besteht die Aufgabe darin, Eigennamen zu finden, die Entitäten einer bestimmten Klasse sind. Was im jeweiligen Fall als Entitätsklasse zu verstehen ist, hängt von der Aufgabe des IE-Systems ab. Normalerweise sind Personen, Organisationen, Ortsangaben usw. von Interesse[3]. In Beispiel 1.1 sind bereits alle Vorkommen von Entitäten von vier Klassen markiert, wobei zu beobachten ist, dass viele Eigennamen aus mehreren Worten bestehen. Alle anderen Worte gehören keiner Klasse an. Sind diese Eigennamen oder „Named Entity Mentions“, wie sie von Jurafsky u. Martin (2008, S. 760) gennant werden, gefunden, erfolgt die Zuordnung jedes von ihnen zu einer Entität. Dieser zusätzliche Prozess ist wichtig, da sich manche Eigennamen oder Teile von ihnen wiederholen und sich nicht immer auf die gleiche Sache in der realen Welt beziehen. Dieser *Reference Resolution*-Prozess ist jedoch nicht Teil dieser Arbeit. Ein solcher Fall tritt in Beispiel 1.1 für das Wort ‘Berlin‘ ein. Einmal ist ‘Berlin‘ eine Ortsangabe und im Satz darauf ist es Teil eines Organisationsnamens.

An diesen ersten Schritt schließt sich das *Auffinden und Klassifizieren von semantischen Relationen*[4] zwischen den Entitäten an. Dabei handelt es sich meist um binäre Relationen wie ein Teil-Ganzes- oder ein Angestelltenverhältnis. ‘OpenBerlin.net basiert auf Mapnificent‘ ist eine solche Relation zwischen

---

[3] Weitere Beispiele sind die Biologie, in der evt. verschiedene Eiweißtypen oder Gene interessieren, oder Preisvergleiche, bei denen die Texte auf Preisangaben, Produktbeschreibungen u.ä. untersucht werden.

[4] *Relation Detection and Classification*

zwei Entitäten. In den ersten beiden Schritten werden folglich Ausprägungen von Entitäts- und Beziehungstypen gefunden, die in die relationale Datenbank eingefügt werden.

Um die Ereignisse und die zeitliche Einordnung zu ermitteln, in deren Zusammenhang die Entitäten und ihre Beziehungen stehen, folgt daraufhin die *Event Detection* und *Temporal Analysis*. Diese Informationen sind entscheidend für die Bedeutung des Inhalts und generieren entscheidende Attributwerte für die Datenbank. Eine zeitliche Information aus Beispiel 1.1 ist ‘im Herbst‘.

Zuletzt können Templates für wiederkehrende Ereignisketten im *Template Filling*-Schritt erstellt und gefüllt werden. Wird eine konkrete Fragestellung untersucht, sind die Templates vorgegeben und enthalten nach dem IE-Prozess alle Informationen, die von Interesse sind. Dieser Schritt ist optional.

Anwendungen

Die Domäne generischer IE-Systeme ist, wie bereits erwähnt v.a., im Beziehungsgeflecht von Personen, Organisationen und Ortsangaben situiert. Zwei Beispiele, die etwas davon abweichen, sind die Webseiten *CiteSeer.org*[5] und *FlipDog.com*[6].

CiteSeer.org extrahiert Titel, Autoren, Veröffentlichungszeitpunkt usw. aus wissenschaftlichen Papers. Es stellt all diese Daten, die in den Papers in unstrukturierter Textform vorliegen, einheitlich dar. Zusätzlich werden Zitierungsinformationen zwischen den einzelnen Papers ausgewertet und anhand dessen eine Art Ranking erstellt. Ein Paper erhält ein hohes Ranking, wenn es selbst viel zitiert ist und die Papers, die es zitieren ebenfalls eine große Zitationsanzahl aufweisen. Das System von CiteSeer.org wandelt also unstrukturierte Textdaten in eine strukturierte Form für Data Mining um, wie es McCallum (2005) beschreibt.

FlipDog.com extrahiert Jobangebote von Firmenwebseiten und stellt diese übersichtlich mit Jobtitel, Tätigkeitsbeschreibung, Organisation, Kontaktinformationen usw. zusammenhängend dar.

1.2 Frühere Arbeiten

DeJong (1982) griff als erster das Thema Information Extraction auf und implementierte eine erste Template Filling-Applikation. Darauf folgten die ersten *Message Understanding Conferences* (MUC) in den späten 80er bis Anfang der

---

[5] http://www.citeseer.org/

[6] http://www.flipdog.com/

90er Jahren. Sundheim u. Chinchor (1993) geben einen Überblick über MUC-1 bis MUC-4, in denen sich zahlreiche Entwicklungen in den Evaluierungsarten, den Testdaten und damit zusammenhängend der Komplexität der Problemstellungen vollzogen haben. In MUC-1 bestanden die bearbeiteten Daten aus zehn Paragraphen als Trainingsmenge und zwei weiteren als Testmenge. Der Inhalt der Texte stammte aus Marinenachrichten. Für die dritte Konferenz wuchs die Anzahl der Texte auf 1400 aus dem Bereich des Terrorismus an, die aus 14 verschiedenen Quellen wie niedergeschriebener Sprache und Zeitungsartikeln stammten. Aufgabe der entwickelten Systeme war weiterhin das Füllen von vorgegebenen Templates. Es wurden v.a. formale, deskriptive Grammatikregelwerke verwendet, was eine Portabilität der Systeme zu anderen Domänen[7] und Sprachen schwierig bis unmöglich machte (Sundheim u. Chinchor, 1993).

Im Laufe der 90er Jahre wurden maschinelle Lernverfahren wie *Hidden Markov Models* der *Driving Factor* im gesamten *Natural Language Processing*. Einer der ersten Ansätze, der für NER konzipiert wurde, ist der von Bikel u. a. (1999). Dieser Ansatz wurde im Rahmen dieser Arbeit zum Teil implementiert und wird in Kapitel 3.2 und 4.3.3 noch näher beschrieben. Darauf folgten weitere statistische Modelle wie *Maximum Entropy Markov Models* (siehe Berger u. a., 1996), *Conditional Random Fields* (siehe Lafferty u. a., 2001) und *Support Vector Machines* (siehe Kudo u. Matsumoto, 2001).

Ab dem Jahr 2000 kam es zur Veröffentlichung neuer Benchmark-Datensätze, die von nun an auch multilingual waren, d.h. Datensätze für verschiedene Sprachen beinhalteten. Der Datensatz, der den Ergebnissen dieser Arbeit zu Grunde liegt, stammt aus dem *CoNLL-2003 Shared Task* (Tjong Kim Sang u. De Meulder, 2003) und umfasst sowohl Trainings- und Testdaten für Englisch als auch für Deutsch. Die Inhalte der Datensätze kommen aus Zeitungsartikeln[8].

User Generated Contents von Blogs und Foren stellt derzeit das am stärksten bearbeitete Feld in IE dar. Dies bedeutet insofern eine große Herausforderung, da die Texte in diesen Quellen meist unsaubere Sprache mit Rechtschreib- und Grammatikfehlern aufweisen, was in Texten von Nachrichtenagenturen selten der Fall ist. Diese Gegebenheit verlangt besonders robuste Systeme, die mit solch verrauschten Daten umgehen können.

---

[7] Z.B. Biologie, Preisvergleiche, Jobangebote.

[8] Für eine ausführliche Beschreibung der Datensätze siehe Abschnitt 4.2

Kapitel 2

Named Entity Recognition

Dieses Kapitel widmet sich einer näheren Beschreibung von Named Entity Recognition. Es wird zunächst auf die Problemstellung eingegangen und im Anschluss daran Lösungsansätze vorgestellt. Am Ende des Kapitels gibt es noch einen Exkurs über die Evaluierung von NER-Systemen.

2.1 Problemstellung

Die allgemeine Problemstellung von NER ist zweigeteilt: Zunächst gilt es, alle im Text vorkommenden Worte ausfindig zu machen, die Eigennamen bilden. Anschließend daran müssen diese Eigennamen einer bestimmten Klasse zugeordnet werden. Es handelt sich also um ein Sequenzklassifikationsproblem: Einer Eingangssequenz $X = x_1x_2 \dots x_{|X|}$ soll eine Klassensequenz $C = c_1c_2 \dots c_{|X|}$ zugeordnet werden. $|X|$ ist dabei die Länge der Eingangssequenz.

Diese Arbeit widmet sich einer noch konkreteren Problemstellung: der NER-Sequenzklassifikation im Kontext maschineller Lernprozesse. Maschinelle Lernprozesse nutzen bereits klassifizierte Daten, die als Trainingsmenge bezeichnet werden[1], um ein Modell zu lernen, dass es ermöglicht, weitere Daten korrekt zu klassifizieren. Bei den gelernten Modellen handelt es sich um probabilistische, deren Ziel es ist, die Wahrscheinlichkeitsverteilung $P(C|X) = P(c_1c_2 \dots c_{|X|}|x_1x_2 \dots x_{|X|})$ zu modellieren. Mit dem gelernten Modell lässt sich nun jeder möglichen Eingangssequenz X die Klassensequenz C zuordnen, die die höchste Wahrscheinlichkeit $\underset{C}{\operatorname{argmax}}\, P(C|X)$ im Modell aufweist (Ratinov u. Roth, 2009). Dieser Schritt wird als Dekodierung bezeichnet[2]. Die Trainings-

---

[1] Meist werden diese Daten direkt von Hand klassifiziert.

[2] Der Begriff Dekodierung stammt aus dem *Noisy Channel*-Theorem von Shannon (1948). Die Eingangsdaten werden als verrauschte Signale verstanden, die es anhand des gelernten

mengen stammen meist aus *Benchmark*-Datensätzen, die ebenfalls Testmengen mit klassifizierten Daten enthalten, um die Performanz der Systeme zu evaluieren.

2.2 Lösungsansätze

Sequenzklassifikation wird meist durch einen *Tagging*-Vorgang umgesetzt, in dem jedes Wort der Eingangssequenz mit einem Klassennamen, zu Grunde liegend einer Menge an Klassen $\mathcal{K}$ (dem *Tag Set*), markiert wird. Ein vereinfachtes Beispiel für NER, in dem nur Worte markiert sind[3], die einer Klasse angehören, zeigt Beispiel 2.1 mit der Klassenmenge $\mathcal{K} = \{‘\mathit{PER}‘, ‘\mathit{ORG}‘, ‘\mathit{MISC}‘, ‘\mathit{LOC}‘\}$.

Beispiel 2.1 *Offene Stadtinformationssysteme*
VON [$_{\text{PER}}$ Christiane Schulzki-Haddouti] *28. MAI 2010 UM 11:53 UHR*

Das im Herbst erst gegründete [$_{\text{ORG}}$ OpenData Network] *hat mit* [$_{\text{MISC}}$ OpenBerlin.net] *ein kleines Projekt gestartet, das Keim eines offenen Stadtinformationssystems werden könnte. Es enthält im Moment Maßnahmen des* [$_{\text{MISC}}$ Konjunkturpakets II] *sowie statische Karteninformationen zur Sozialstruktur* [$_{\text{LOC}}$ Berlins]*. Die Daten bzw. Bilder stammen aus dem* [$_{\text{ORG}}$ FIS-Broker der Stadt Berlin]*.*

[$_{\text{MISC}}$ OpenBerlin.net] *basiert auf dem Projekt* [$_{\text{MISC}}$ Mapnificent] *des Berliner Informatikers* [$_{\text{PER}}$ Stefan Wehrmeyer]*, der unter anderem auch die Daten des öffentlichen Nahverkehrs sowie Kriminalitätsdaten verwendet. [...]* [$_{\text{PER}}$ Wehrmeyer] *selbst ließ sich vom britischen Projekt* [$_{\text{MISC}}$ Mapumental] *inspirieren, die noch in private beta ist.* Quelle: ZEIT ONLINE, im Artikel Offene Stadtinformationssysteme von Christiane Schulzki-Haddouti (28.05.2010)

Für die Entscheidung, welcher Klasse ein Wort zugewiesen wird, gibt es zwei wichtige Lösungsansätze. Der erste, der in dieser Abeit nicht weiter verfolgt wird, ist die Entwicklung eines grammatikbasierten, formalen Regelwerkes (siehe Jurafsky u. Martin, 2008, S. 419 ff.). Dieser Ansatz hat sich als relativ performant bezüglich NER herausgestellt, bringt jedoch den großen Nachteil mit sich, dass die Grammatikregeln meist sehr spezifisch auf ein bestimmtes Anwendungsgebiet zugeschnitten sind. Z.B. bedarf es für ein NER-System einer anderen Sprache eines komplett neuen Regelwerkes. Der zweite Ansatz ist der im voherigen Abschnitt beschriebene maschinelle Lernansatz. Diese Arbeit widmet sich ausschließlich diesem Ansatz in Form von *Hidden Markov Models* (siehe Kapitel 3.2) und des *Perzeptron*-Lernalgorithmus (siehe Kapitel

---

Modells zu dekodieren gilt.

[3] In praktischen Anwendungen gibt es eine weitere Klasse für Worte, die keiner Klasse angehören.

3.3). Beide Modelle werden in einem maschinellen Lernprozess trainiert und machen sich dabei jeweils bestimmte, dem Text inhärente Features zu nutze. Features sind Informationen, die den Worten einer Sequenz innewohnen. Ein Beispiel für solch ein Feature wäre: Auf das Wort ‘Wehrmeyer‘ folgt das Wort ‘selbst‘, was darauf hinweist, dass das Wort ‘Wehrmeyer‘ wahrscheinlich eine Person bezeichnet. Die Wahl der Features ist entscheidend für die Performanz eines NER-Systems, wie sich später im Kapitel 5 herausstellen wird. Essentielle Features der Eingangssequenz sind:

- Worte
- Part Of Speech Tags[4]
- Wortstämme
- Syntactic Base Phrase Chunk Tags[5]
- Präfixe/Suffixe verschiedener Länge
- Informationen über Groß- und Kleinschreibung
- Existenz eines Trennstriches (z.B. FIS-Broker)
- Endung des Wortes auf Punkt (z.B. Corp.)

Vier weitere wichtige Designkriterien, die bei der Entwicklung von NER-Systemen beachtet werden sollen, beschreiben Ratinov u. Roth (2009), die jedoch für diese Arbeit nicht entscheidend waren:

1. Wie sollen NER-Chunks[6] repräsentiert werden?
2. Welcher Dekodierungsalgorithmus soll verwendet werden?
3. Wie sollen nicht lokale Abhängigkeiten modelliert werden?
4. Wie kann externes Wissen für NER eingebracht werden?

Für die Chunking-Repräsentation werden zwei Möglichkeiten vorgestellt: Das weit verbreitete BIO-Schema[7] und das BILOU-Schema[8] (Ratinov u. Roth, 2009).

---

[4] Wortklassen (z.B. Nomen, Verb, Präposition usw.)

[5] Informationen über syntaktische Phrasen (siehe Jurafsky u. Martin, 2008, S. 461 ff.).

[6] Bezüglich NER sind Chunks Wortgruppen innerhalb einer Sequenz, die zur gleichen Entität gehören (z.B. FIS-Broker der Stadt Berlin).

[7] BIO steht für: *before*, *internal*, *outside*.

[8] BILOU steht für: *before*, *internal*, *last*, *unit-length*, *outside*.

Zum zweiten Designkriterium – der Wahl des Dekodierungsalgorithmus – werden einfache *Greedy Search*-Verfahren (siehe Kapitel 3.3.3), *Beam Search*-Verfahren (siehe Furcy u. Koenig, 2005) und der *Viterbi*-Algorithmus (siehe Kapitel 3.2.3) angeführt.

Nicht-lokale Features verkörpern das Wissen über bereits vorgekommene Worte, die schon einmal einer bestimmten Klasse zugeordnet wurden. Dieses Wissen kann dadurch für die Zuordnungsentscheidung bei wiederholtem Auffinden des jeweiligen Wortes genutzt werden. Ratinov u. Roth (2009) stellen dazu drei verschiedene Verfahren vor: *Context Aggregation* (siehe Chieu u. Ng, 2003), *Two-stage Prediction Aggregation* (siehe Krishnan u. Manning, 2006) und *Extended Prediction History*, was von Ratinov u. Roth (2009) selbst beschrieben ist.

Ratinov u. Roth (2009) stellen die These auf, dass NER eine sehr wissenintensive Aufagbe ist und es sich als sinnvoll erweist, externe Quellen mit einzubeziehen. Rohe Texte[9] können die Performance steigern, in dem man sie einem *Word Clustering* (siehe Brown u. a., 1992) unterzieht und diese Informationen miteinbezieht. Entitätslisten – genannt *Gazetteers* – sind eine weitere Quelle externen Wissens. Ratinov und Roth halten sie für sehr wichtig in diesem Kontext und nutzen insgesamt 30 solcher Listen.

2.3 Evaluierung

Die Performanz eines NER-Programms wird meist durch *Precision*, *Recall* und F_1*-Measure* bestimmt. Da es sich bei NER um eine *Multiclass*-Klassifizierung handelt, werden diese Performanzkriterien für jede Klasse einzeln berechnet und diese Ergebnisse anschließend gemittelt, um eine Gesamtbewertung des Programms zu erhalten. Recall (R) ist der Anteil der richtig zugeordneten Entitäten bezüglich der Anzahl aller Entitäten einer Klasse:

$$R = \frac{\text{Anzahl gefundener Entitäten}}{\text{Anzahl aller Entitäten}} \tag{2.1}$$

Precision (P) ist der Anteil der richtig zugeordneten Entitäten bezüglich der Anzahl aller gefundenen Entitäten einer Klasse:

$$P = \frac{\text{Anzahl richtig zugeordneter Entitäten}}{\text{Anzahl gefundener Entitäten}} \tag{2.2}$$

F_1-Measure ist das harmonische Mittel aus Precision und Recall.

$$F_1 = \frac{2PR}{P + R} \tag{2.3}$$

---

[9] Texte, in denen die Worte noch nicht klassifiziert sind.

Kapitel 3

Sequenzklassifikation

Sequenzklassifikation ist ein Vorgang, der jedem Element einer gegebenen Sequenz $X = x_1 x_2 \dots x_{|X|}$ eine Klasse aus einer Menge $\mathcal{K} = \{k_1, k_2, \dots, k_{|\mathcal{K}|}\}$ von $|\mathcal{K}|$ Klassen zuordnet. Im Beispiel 2.1 war diese gegebene Sequenz eine Wortsequenz, deren Worte jeweils einer Entitätsklasse zugewiesen wurden. Viele der Worte wurden jedoch gar nicht markiert, was bedeutet, dass sie gar keiner Klasse aus $\mathcal{K}$ angehören. Dafür kann $\mathcal{K}$ in praktischen Anwendungen eine Klasse hinzugefügt werden, die dies signalisiert.

Im Folgenden werden die Elemente der Eingangssequenz auch Token genannt. Ein Token beinhaltet dabei alle Informationen eines Wortes, die in dem verwendeten Datensatz bereit gestellt werden. Im hier verwendeten CoNLL-2003 Datensatz ist ein Token ein Quadrupel $\langle w, pos, syn, neclass \rangle$[1].

3.1 Einteilung

Die vorliegende Arbeit beschäftigt sich mit einer erweiterten Form – der probabilistischen Sequenzklassifikation im Kontext maschineller Lernprozesse. Die Erweiterung besteht darin, dass diese Verfahren einer gegebenen Sequenz $X = x_1 x_2 \dots x_{|X|}$ die Klassensequenz $C = c_1 c_2 \dots c_{|X|}$ zuweisen, die die höchste Wahrscheinlichkeit $\underset{C}{\operatorname{argmax}} P(C|X)$ hat. Während des maschinellen Lernprozesses entsteht ein Modell der Wahrscheinlichkeitsverteilung $P(C|X)$, mit dem anschließend weitere Daten klassifiziert werden können. Dafür gibt es ganz verschiedene Ansätze, die sich in vier Gruppen einteilen lassen. Es gibt generativ-lokale, generativ-globale, diskriminativ-lokale und diskriminativ-globale Verfahren. Die Unterscheidung global-lokal begründet sich durch die Klassifikationsentscheidungen im Dekodierungsschritt. Lokale Verfahren wie Greedy

---

[1] w = Wort, pos = Part Of Speech Tag, syn = Syntactic Chunk Tag, $neclass$ = Entitätsklasse. Für den deutschen Datensatz kommt noch die Wortstamminformation hinzu.

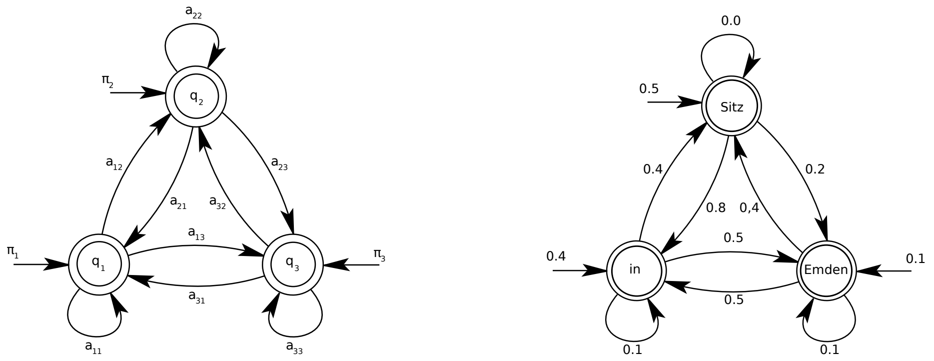
	Lokal	Global
Generativ	Naive Bayes (siehe Manning u. Schütze, 1999, S. 235 ff.)	Hidden Markov Models (siehe Rabiner, 1989)
Diskriminativ	Perzeptron mit Greedy Search (siehe Collins, 2002), Maximum Entropy Markov Models (siehe Berger u. a., 1996), Support Vector Machine (siehe Kudo u. Matsumoto, 2001)	Conditional Random Fields (siehe Lafferty u. a., 2001), Collins-Perzeptron (siehe Ratinov u. Roth, 2009)

Tabelle 3.1: Einteilung der Sequenzklassifikationsverfahren mit Beispielen.

Search (siehe Abschnitt 3.3.3) treffen in jedem Schritt determinierte (gierige) Entscheidungen unter In-Betracht-Ziehung der vorhergehenden, können allerdings keine früheren Entscheidungen ändern. Der Viterbi-Algorithmus (siehe Abschnitt 3.2.3), als Beispiel für ein globales Verfahren, ist dazu in der Lage, frühere Entscheidungen auf Grund aktueller zu revidieren. Viterbi macht also einen *Trade Off* zwischen aktuellen und früheren Entscheidungen. Die Unterteilung generativ-diskriminativ entsteht durch die Lernstrategie des Verfahrens. Generative Verfahren modellieren die Wahrscheinlicheit $P(C|X)$ durch die Umformung mit Bayes Theorem in den Quotient $\frac{P(X,C)}{P(X)}$ (siehe Abschnitt 3.2.1). Diskriminative Verfahren beruhen auf der Approximierung der Wahrscheinlichkeitsverteilung $P(C|X)$ durch eine Funktion $h(X)$. Der Lernprozess ist von einer Minimierung des Klassifikationsfehlers bezüglich der Trainingsmenge gekennzeichnet (siehe Abschnitt 3.3.2). Diese Einteilung ist in Tabelle 3.1 mit Beispielen für jede Gruppe dargestellt.

3.2 Hidden Markov Models

Hidden Markov Models (kurz HMM) sind stochastische Sequenzklassifikationsmodelle, die in maschinellen Lernprozessen eingesetzt werden. Sie errechnen gegeben einer Sequenz X eine Wahrscheinlichkeitsverteilung $P(X, C)$ über alle Sequenzen an Klassenzuordnungssequenzen C und wählen anschließend C_{max} mit der größten Wahrscheinlichkeit $\underset{C}{\operatorname{argmax}} P(X, C)$ aus (siehe Abschnitt 3.2.2 für die Herleitung). Es handelt sich bei einem HMM um ein generativ-globales Modell (Bikel u. a., 1999). Entwickelt wurde dieses Verfahren v.a. von Baum u. Pietre (1966).



(a) Markov-Kette formal.

(b) Markov-Kette Beispiel.

Abbildung 3.1: Markov-Kette mit Transitionswahrscheinlichkeiten a_{ij} und Zuständen q_i, adaptiert von Jurafsky u. Martin (2008, S. 210 ff.). Abbildung 3.1(b) beinhaltet eine beispielhafte Eingangssequenz: ‘Sitz in Emden‘.

3.2.1 Grundlagen

Markov-Ketten bilden den Ausgangspunkt für HMMs. Sie können durch gewichtete, endliche Automaten formalisiert werden, wobei die Transitionsgewichte angeben, wie groß die Wahrscheinlichkeit ist, vom jeweiligen Ausgangszustand in den jeweiligen Zielzustand zu wechseln. Die Zustände $\mathcal{Q}$ des Automaten werden durch die Eingangssequenz X eindeutig bestimmt. Jedes Element x_i aus X erhält einen eigenen Zustand q_i. Zusätzlich ist es möglich, Start- und Endzustände einzuführen, die jedoch auch durch Initialwahrscheinlichkeiten π_i [2] und eine Menge akzeptierender Endzustände $\mathcal{F}$ modelliert werden können. Eine Markov-Kette besteht also aus folgenden Elementen (Jurafsky u. Martin, 2008, S. 207 ff.):

$\mathcal{Q} = \{q_1, q_2, \ldots, q_{\|\mathcal{Q}\|}\}$	Eine Menge aus $\|\mathcal{Q}\|$ Zuständen
$\mathbf{A} \in \mathbf{R}^{\|\mathcal{Q}\| \times \|\mathcal{Q}\|}, a_{ij} = P(q_i \| q_j)$	Die Transitionswahrscheinlickeiten. a_{ij} ist die Wahrscheinlichkeit von q_i in q_j überzugehen. ($\sum_{j=1}^{\|\mathcal{Q}\|} a_{ij} = 1,\ \forall i$)
$\Pi \in \mathbf{R}^{\|\mathcal{Q}\|}, P(q_i \| START) = \pi_i$	Die Initialwahrscheinlichkeiten ($\sum_{i=1}^{\|\mathcal{Q}\|} \pi_i = 1$)
$\mathcal{F} = \{q_i, q_j \ldots\}$	Eine Menge akzeptierender Endzustände

Abbildung 3.1(b) zeigt eine Markov-Kette für eine konkrete Menge an Worten. Es folgt eine Beispielrechnung für diese Markov-Kette:

---

[2] π_i gibt an, wie wahrscheinlich das Element x_i die Sequenz eröffnet.

$$P(\text{‘Sitz in Emden‘}) = \pi_2 a_{21} a_{13} = 0,2$$
$$P(\text{‘Emden in Sitz‘}) = \pi_3 a_{31} a_{12} = 0,02$$

Aus der Beispielrechnung wird die Annahme ersichtlich, dass in einer Markov-Kette die Wahrscheinlichkeit eines Zustands nur von der Wahrscheinlichkeit des vorhergehenden Zustands abhängt[3]. Ausgangspunkt ist eine beliebige Zustandssequenz $S = s_1 s_2 \ldots s_{|X|}$ (Jurafsky u. Martin, 2008, S. 208 ff.):

$$\text{Markov-Annahme: } P(s_t|s_1 \ldots s_{t-1}) = P(s_t|s_{t-1}),\ s \in \mathcal{Q} \tag{3.1}$$

Ereignisse Eine Markov-Kette ist dazu in der Lage, Wahrscheinlichkeiten von Sequenzen *beobachtbarer Ereignisse* zu berechnen. Bei Klassifizierungsaufgaben liegt das Interesse allerdings auf anderen, nicht observierbaren Ereignissen. Bei NER werden z.B. Token beobachtet und es muss von ihnen ausgehend auf ihre Entitätsklasse geschlossen werden. Diese Entitätsklassen sind also *versteckte Ereignisse*. Dazu muss eine Markov-Kette um Beobachtungswahrscheinlichkeiten $b_i(o_t) = P(o_t|q_i)$ augmentiert werden, die angeben, wie wahrscheinlich eine Beobachtung o_t vom Zustand q_i generiert wurde. Die Zustände werden nun durch die versteckten Ereignisse determiniert. Bei NER entspricht dem zufolge jedem Element aus dem Tag Set einem Zustand. Zusammgefasst besteht ein HMM aus folgenden Elementen (Jurafsky u. Martin, 2008, S. 210 ff.):

$\mathcal{Q} = \{q_1, q_2, \ldots, q_{\lvert\mathcal{Q}\rvert}\}$	Eine Menge aus $\lvert\mathcal{Q}\rvert$ Zuständen
$O = o_1 o_2 \ldots o_{\lvert O\rvert}$	Eine Sequenz von $\lvert O\rvert$ Beobachtungen aus einem Alphabet V
$\mathbf{B} \in \mathbf{R}^{\lvert O\rvert \times \lvert\mathcal{Q}\rvert}, b_i(o_t) = P(o_t \mid q_i)$	Die Beobachtungswahrscheinlichkeiten (auch Emissionwahrscheinlichkeiten genant)
$\mathbf{A} \in \mathbf{R}^{\lvert\mathcal{Q}\rvert \times \lvert\mathcal{Q}\rvert}, a_{ij} = P(q_i \mid q_j)$	Die Transitionswahrscheinlickeiten. a_{ij} ist die Wahrscheinlichkeit von q_i in q_j überzugehen. ($\sum\limits_{j=1}^{\lvert\mathcal{Q}\rvert} a_{ij} = 1,\ \forall i$)
$\Pi \in \mathbf{R}^{\lvert\mathcal{Q}\rvert}, P(q_i \mid START) = \pi_i$	Die Initialwahrscheinlichkeiten ($\sum\limits_{i=1}^{\lvert\mathcal{Q}\rvert} \pi_i = 1$)
$\mathcal{F} = \{q_i, q_j \ldots\}$	Eine Menge akzeptierender Endzustände

---

[3] Diese werden diese Markov-Ketten erster Ordnung genannt.

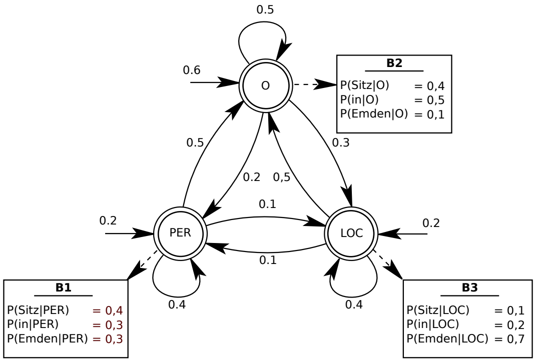


Abbildung 3.2: Ein Hidden Markov Model. Adaptiert von Jurafsky u. Martin (2008, S. 210).

Die Erweiterung der Markov-Kette mit Emissionswahrscheinlichkeiten – ein Hidden Markov Model – ist in Abbildung 3.2 dargestellt. Die Markov-Annahme aus Gleichung 3.1 gilt auch in diesem Fall. Hinzu kommt eine weitere: Die Wahrscheinlichkeit, dass eine Beobachtung o_t vom Zustand $s_t \in \mathcal{Q}$ einer Zustandssequenz $S = s_1, s_2, \ldots, s_{|O|}$ generiert wurde, hängt nur von s_t ab.

$$P(o_t|s_1, \ldots, s_t, \ldots, s_{|O|}, o_1, \ldots, o_t, \ldots, o_{|O|}) = P(o_t|s_t) \tag{3.2}$$

Beispielhaft sollen die Wahrscheinlichkeiten für die folgenden getaggten Sequenzen mit den Werten aus Abbildung 3.2 errechnet werden:

Beispiel 3.1 *Sitz in [$_{LOC}$ Emden]*

Beispiel 3.2 *[$_{PER}$ Sitz] in Emden*

$$P(\text{‘Sitz in [}_{\text{LOC}}\text{ Emden]‘}) = P(\text{‘Sitz‘}|O)P(\text{‘in‘}|O)P(\text{‘Emden‘}|LOC) = 0,14$$

$$P(\text{‘[}_{\text{PER}}\text{ Sitz] in Emden‘}) = P(\text{‘Sitz‘}|PER)P(\text{‘in‘}|O)P(\text{‘Emden‘}|O) = 0,02$$

In diesen beiden Beispielen ist die Zustandssequenz vorgegeben, was normalerweise nicht der Fall ist. Dies weist auf das erste unter drei Problemen hin, die Rabiner (1989) als die drei fundamentalen Probleme im Umgang mit HMMs ermittelt hat:

Problem 1: Gegeben einer Beobachtungssequenz $O = o_1 o_2 \ldots o_{|O|}$ und eines HMM $\lambda = (\mathbf{A}, \mathbf{B}, \mathcal{Q}, \mathcal{F}, \Pi)$, berechne die Wahrscheinlichkeit $P(O|\lambda) = \sum\limits_{S} P(O, S|\lambda)$.

Problem 2: Gegeben einer Beobachtungssequenz $O = o_1 o_2 \ldots o_{|O|}$ und eines HMM $\lambda = (\mathbf{A}, \mathbf{B}, \mathcal{Q}, \mathcal{F}, \Pi)$, ermittle die beste versteckte Zustandssequenz $S = s_1 s_2 \ldots s_{|O|}$ mit $\max\limits_{S} P(O, S|\lambda)$.

Problem 3: Gegeben einer Beobachtungssequenz $O = o_1 o_2 \ldots o_{|O|}$ und einer Menge von Zuständen $\mathcal{Q}$, berechne die Wahrscheinlichkeiten $\mathbf{A}$, $\mathbf{B}$ und Π, die $P(O|\lambda)$ maximieren.

In der vorliegenden Arbeit ist das zweite Problem entscheidend und wird in Abschnitt 3.2.3 erläutert.

3.2.2 Lernprozess

Um die Modellierung der Wahrscheinlichkeitverteilung $P(C|X)$ zu erreichen, wird ein generativer Ansatz verwendet. Dafür wird $P(C|X)$ (C entspricht nun der Zustandssequenz $S = s_1 s_2 \dots s_{|O|}$ ($s_i \in \mathcal{Q}$) und X der Beobachtungssequenz $O = o_1 o_2 \dots o_{|O|}$) durch das Theorem von Bayes umgeformt (siehe Bikel u. a., 1999):

$$P(S|O) = \frac{P(O,S)}{P(O)} = \frac{P(O|S)P(S)}{P(O)} \tag{3.3}$$

Da O bei der Maximumbildung über alle Zustandssequenzen S gleich ist, kann der Nenner dabei vernachlässigt werden. Daraus folgt (siehe Bikel u. a., 1999):

$$\underset{S}{\operatorname{argmax}} P(S|O) = \underset{S}{\operatorname{argmax}} P(O|S)P(S) \tag{3.4}$$

In überwachten Lernprozessen werden die Emissions- und Transitionswahrscheinlichkeiten **B** und **A** meist empirisch durch relative Häufigkeiten berechnet. Die Transitionswahrscheinlichkeit $P(s_t|s_{t-1})$ in der Zustandssequenz $S = s_1 s_2 \dots s_{|O|}$ ist dabei nichts weiter als die Anzahl der Übergange von s_{t-1} zu s_t, geteilt durch die Anzahl aller Vorkommnisse von s_{t-1} in der Trainingsmenge:

$$P(s_t|s_{t-1}) = \frac{count(s_t, s_{t-1})}{count(s_{t-1})} \tag{3.5}$$

Die Emissionswahrscheinlichkeit $P(o_t|s_t)$ ergibt sich aus dem Quotienten der Anzahl, in der Beobachtung o_t von Zustand s_t generiert wurde[4] und der Anzahl aller Vorkommnisse von s_t in der Trainingsmenge:

$$P(o_t|s_t) = \frac{count(o_t, s_t)}{count(s_t)} \tag{3.6}$$

Es folgen Beispielrechnungen für $P(\text{‘}O\text{‘}|\text{‘}PER\text{‘})$, $P(\text{‘}O\text{‘}|\text{‘}O\text{‘})$ und $P(\text{‘der‘}|\text{‘}ORG\text{‘})$[5], bezogen auf den Satz aus Bespiel 2.1.

---

[4] D.h. auf NER bezogen die Anzahl der Vorkommnisse in der das Wort o_t der Klasse aus s_t in der Trainingsmenge zugewiesen ist.

[5] ‘O‘ symbolisiert hierbei die Klasse für alle Worte, die keine Entitäten sind.

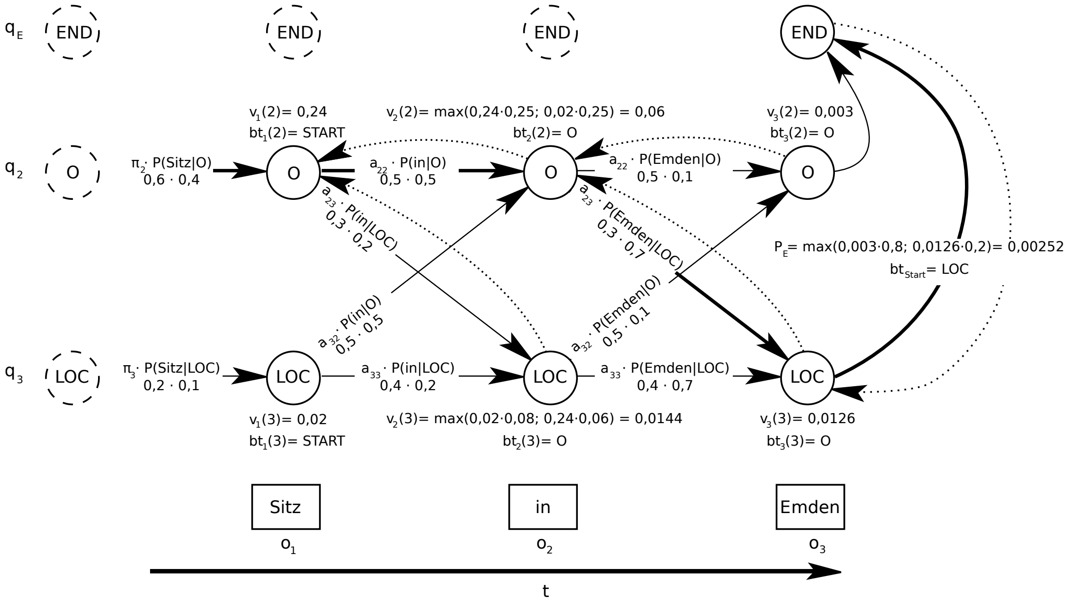


Abbildung 3.3: Der Viterbi-Algorithmus zum Auffinden der bestmöglichen Klassensequenz für Beispiel 3.1. Das HMM verwendet die Wahrscheinlichkeitswerte aus Abbildung 3.2 und es werden nur die Zustände $q_2 = ‘O‘$ und $q_3 = ‘LOC‘$ betrachtet. Ein zusätzlicher Endzustand und Endwahrscheinlichkeiten $\epsilon_2 = 0,8$ und $\epsilon_3 = 0,2$ wurden hinzugefügt. Die gepunkteten Linien stellen die jeweiligen Backpointer dar. Adaptiert von Jurafsky u. Martin (2008, S. 221).

$$P(‘O‘|‘PER‘) = \frac{count(‘O‘,‘PER‘)}{count(‘PER‘)} = \frac{3}{5}$$

$$P(‘O‘|‘O‘) = \frac{count(‘O‘,‘O‘)}{count(‘O‘)} = \frac{61}{72}$$

$$P(‘\text{der}‘|‘ORG‘) = \frac{count(‘\text{der}‘,‘ORG‘)}{count(‘ORG‘)} = \frac{1}{6}$$

3.2.3 Dekodierung

Im Kontext eines HMM ist Dekodierung der im zweiten Problem von Rabiner (1989) beschriebene Vorgang. Wie im vorhergehenden Abschnitt beschrieben, gilt es die Zustandssequenz S zu finden, die die maximale Wahrscheinlichkeit aufweist. Dazu werden die im Lernprozess ermittelten Emissions- und Transitionswahrscheinlichkeiten genutzt.

In HMMs wird dazu meist der in Abbildung 3.3 und Algorithmus 1 dargestellte Viterbi-Algorithmus, der von Viterbi (1967) entwickelt wurde, verwendet. Der Algorithmus arbeitet global (siehe Abschnitt 3.1) und durchläuft die Beobachtungssequenz von links nach rechts und macht in jedem Schritt

Gebrauch von den Ergebnissen aus den vorhergehenden Berechnungen, wobei diese in sogenannten *Trellis*-Spalten[6] $\mathbf{v}_t$ gespeichert werden. Er zählt damit zu den *Dynamic Programming*-Algorithmen. Alle Trellis-Spalten zusammen ergeben einen Trellis-Graphen und ein Trellis-Eintrag $v_t(j)$ sagt aus, wie hoch die Wahrscheinlichkeit ist, in Zustand q_j nach den Beobachtungen $o_1 \dots o_t$ zu sein. $v_t(j)$ berechnet sich folgendermaßen (Jurafsky u. Martin, 2008, S. 218 ff.):

$$v_t(j) = \max_{i=1\dots|\mathcal{Q}|}(v_{t-1}(i)a_{ij}b_j(o_t)) = \max_{i=1\dots|\mathcal{Q}|}(v_{t-1}(i)P(q_i|q_j)P(o_t|q_j)) \tag{3.7}$$

Mit der Berechnung der Trellis-Spalten ist allerdings noch nicht die gesamte Dekodieraufgabe erledigt, da nicht nur eine Wahrscheinlichkeit, sondern auch die bestmögliche Zustandssequenz ermittelt werden muss. Dafür werden im Viterbi-Algorithmus *Backpointer* $bt_t(j)$ angelegt, die, wenn das Satzende erreicht wurde, die beste Zustandssequenz beinhalten. Alle Backpointer zusammen werden *Backtrace* genannt. Ein Backpointer $bt_t(j)$ enthält den Zustand q_i, der nach den Beobachtungen $o_1 \dots o_t$ im Übergang zum Zustand q_j die größte Wahrscheinlichkeit hatte (Jurafsky u. Martin, 2008, S. 218 ff.):

$$bt_t(j) = \underset{i=1\dots|\mathcal{Q}|}{\operatorname{argmax}}(v_{t-1}(i)a_{ij}b_j(o_t)) = \underset{i=1\dots|\mathcal{Q}|}{\operatorname{argmax}}(v_{t-1}(i)P(q_i|q_j)P(o_t|q_j)) \tag{3.8}$$

In praktischen Anwendungen ist es oft sinnvoll, neben den Initialwahrscheinlichkeiten Π auch Wahrscheinlichkeiten $\mathcal{E} = \{\epsilon_1, \epsilon_2, \dots, \epsilon_{|\mathcal{Q}|}\}$ dafür zu berechnen, dass die Sequenz in einem Zustand endet. Dies ist v.a. wichtig, um den Start des Backtrace zu ermitteln, wie in Abbildung 3.3 und Algorithums 1 ersichtlich wird.

Die globalen Dekodierungseigenschaften des Viterbi-Algorithmus sind dadurch gekennzeichnet, dass vorhergehende Entscheidungen durch aktuelle revidiert werden können.

3.2.4 Erweiterungen

Die natürliche Sprache hat eine sehr dynamische Natur. Es entstehen ständig neue Wörter und Namen, was es unmöglich macht, alle Worte bereits zumindest einmal in den Trainingsdaten während des Lernens gesehen zu haben. Das HMM hat also kein Wissen über die Wahrscheinlichkeit $P(o_t|s_t)$, wenn o_t nie in der Trainingsmenge vorkam und kann dem Wort somit auch keine Klasse zuordnen. Um dieses Problem zu lösen, wird ein *Unknown Word Model* auf

---

[6] Trellis sind spezielle Graphen, in denen die Knoten in Spalten angeordnet sind (siehe Abbildung 3.3). Die Knoten haben nur Verbindungen zu Knoten benachbarter Spalten.

den sogenannten *Development*-Daten[7] gelernt und damit die Vorhersagen in der eigentlichen Testmenge getroffen (Jurafsky u. Martin, 2008, S. 129).

Des Weiteren ist es üblich die Modellierung dahingehend zu erweitern, dass mehr als nur die vorhergehende Klasse für die Transitionswahrscheinlichkeiten ins Kalkül gezogen werden kann. Theoretisch kann die Bigram-Modellierung $P(s_t|s_{t-1})$ auf eine N-Gram-Modellierung $P(s_t|s_{t-1}, \dots, s_{t-N})$ augmentiert werden, womit noch mehr Sequenzinformation[8] beachtet werden kann. Die gleiche Erweiterung ist auch für die Emissionswahrscheinlichkeiten möglich, wo $P(o_t|s_t)$ zu $P(o_t|s_t, o_{t-1}, \dots, o_{t-N})$ wird. Durch N-Gram-Modellierung entsteht allerdings ein noch tiefgreifenderes Problem als das unbekannter Worte: Wie soll mit N-Gram-Sequenzen umgegangen werden, die nie in der Trainingsmenge gesehen wurden? Für dieses sogenannte *Sparse Data*-Problem gibt es drei sehr gut erforschte Lösungsansätze, die hier aus Platzgründen nur genannt werden sollen: *Smoothing* (siehe Jeffreys, 1948; Good, 1953), Interpolation (siehe Baum, 1972) und *Backoff* (siehe Katz, 1987).

Algorithm 1 Der Viterbi-Algorithmus

```
Input:  Observationsequence $O$, Set of states $\mathcal{Q}$
        Transition probability matrix $\mathbf{A}$, Emission probability matrix $\mathbf{B}$
        Set of initial probabilities $\Pi$, Set of ending probabilities $\mathcal{E}$

create viterbi-matrix with size $|\mathcal{Q}| \times |O|$
create backtrace with size $|\mathcal{Q}| \times |O|$
for each state $q \in \mathcal{Q}$ do
    viterbi-matrix$[q][1] = \Pi[q]b[q][O[1]]$
    backtrace$[q][1] = START$
end for
for $t = 2$ to $|O|$ do
    for each state $q \in \mathcal{Q}$ do
        viterbi-matrix$[q][t] = \max_{q' \in \mathcal{Q}}$ viterbi-matrix$[q'][t-1]\ A[q'][q]\ B[q][O[q']]$
        backtrace$[q][t] = \operatorname{argmax}_{q' \in \mathcal{Q}}$ viterbi-matrix$[q'][t-1]\ A[q'][q]\ B[q][O[q']]$
    end for
end for
viterbi-matrix$[q_F][|O|] = \max_{q \in \mathcal{Q}}$ viterbi-matrix$[q][|O|]\ \mathcal{E}[q]$
backtrace$[q_F][|O|] = \operatorname{argmax}_{q \in \mathcal{Q}}$ viterbi-matrix$[q][|O|]\ \mathcal{E}[q]$
output bestpath
```

---

[7] Meist wird der Trainingsmenge ein gewisse Menge an Daten entnommen um damit Hyperparameter der Verfahren oder wie hier ein Modell unbekannter Worte zu lernen. Diese Menge wird Development-Menge genannt.

[8] Damit sind Informationen gemeint, die durch die Abfolge von Worten und Klassen in der Sequenz entstehen.

3.3 Perzeptron mit Greedy Search

Der Perzeptron-Lernalgorithmus stammt aus der neurologischen Forschung der 50er und 60er Jahre (siehe Rosenblatt, 1958). Er wurde als eines der ersten maschinellen Lernverfahren für Klassifikationen verwendet, wobei eine diskriminative Funktion $h(X)$ der Eingabedaten X in einem iterativen Prozess ermittelt wird, die schrittweise die Missklassifikationsrate bezüglich der Trainingsmenge minimiert. In dieser Arbeit wird ein Perzeptron mit Greedy Search (siehe Abschnitt 3.3.3) vorgestellt. Es handelt sich also um eine diskriminativ-lokale Methode (Manning u. Schütze, 1999; Carreras u. a., 2003).

3.3.1 Grundlagen

Feature-Vektoren In diskriminativen Verfahren werden Token x als Feature-Vektoren $\mathbf{v}_x$ interpretiert, in denen jede Komponente v_i ein Feature aus einem d-dimensionalen Feature-Raum $\mathbf{F}$ repräsentiert. Meist sind Features Indikatorfunktionen $I : x \rightarrow \{0, 1\}$, die das Token x auf die Menge $\{0, 1\}$ abbilden. Eine Beispielfunktion könnte wie folgt aussehen:

$$v_{10} = I(x) = \begin{cases} 1 & \text{if current word of token } x \text{ is `yes'} \\ 0 & \text{otherwise} \end{cases}$$

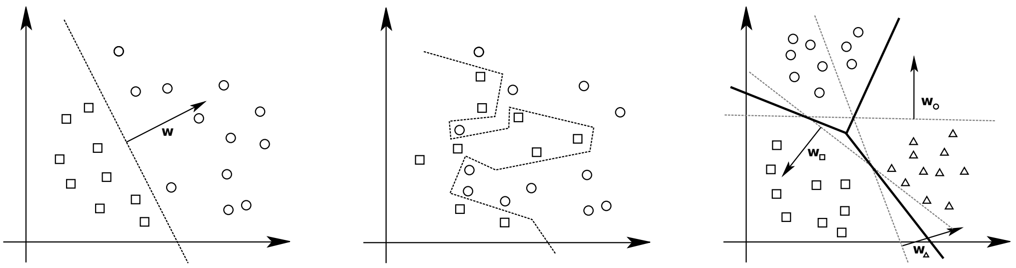
In der vorliegenden Arbeit werden die Features aus sogenannten Templates[9] generiert, die ihrerseits je eine bestimmte Fenstergröße haben. Ein denkbares Template-Fenster wäre $word(-2, 2)$, was bedeutet, dass jedes Wort in einem Fenster vom vorletzten bis zum übernächsten Wort um das aktuell betrachtete Wort herum zu einer Dimension im Feature-Raum werden. Es folgt ein Beispiel für dieses Template mit einem Satzteil aus Beispiel 1.1:

Beispiel 3.3 *„ ... auf dem Projekt $\overline{\text{Mapnificent}}$ des Berliner ... “*
Daraus ergeben sich für das Wort ‘$Mapnificent$‘ die Features $word(-2)$ = ‘dem‘, $word(-1)$ = ‘Projekt‘, $word(0)$ = ‘$Mapnificent$‘, $word(1)$ = ‘des‘ und $word(2)$ = ‘Berliner‘.

Ein Perzeptron ist eine lineare, diskriminative Funktion $h_{\mathbf{w}} : \mathbf{R}^d \rightarrow \mathbf{R}$, die von einem Gewichtsvektor $\mathbf{w}$ parametrisiert ist. Ein Token x mit dem Feature-Vektor $\mathbf{v}_x$ wird dabei mit dem inneren Produkt $h_{\mathbf{w}}(x) = \mathbf{w} \cdot \mathbf{v}_x$ bewertet. In Multiclass-Aufgaben wie NER gibt es demzufolge mehrere Perzeptrons mit den Gewichtsvektoren $\mathcal{W} = \{\mathbf{w}_{k_1}, \mathbf{w}_{k_2}, \ldots, \mathbf{w}_{k_{|\mathcal{K}|}}\}$. Eine Klassenzuordnung für

---

[9] Diese Idee gleicht der des *CRF++*-Systems. Siehe http://crfpp.sourceforge.net/#templ



(a) Linear separierbar. (b) Nicht linear separierbar. (c) Multiclass-Separierung.

Abbildung 3.4: Zweidimensionales Beispiel für das lineare Separierbarkeitsproblem. Die gestrichelten Linien stellen die Separierungsebenen dar. In Abbildung 3.4(c) kennzeichnen die fett konturierten Linien die Bereiche der einzelnen Klassen. Die Abbildungen sind jedoch für die hier benutzten indikativen Features nicht repräsentativ, da in solch einem Koordinatensystem alle Daten in den Eckpunkten liegen würden. Sie dienen somit nur zur Veranschaulichung des allgemeinen Falls.

x gründet sich auf der Bildung des $\underset{\mathbf{w}}{\operatorname{argmax}}\, \mathbf{w} \cdot \mathbf{v}_x$ (Carreras u. a., 2003; Collins, 2002).

Lineare Separierbarkeit Der Perzeptron-Algorithmus ist dazu in der Lage, ein Modell zu lernen, dass die Eingabedaten linear separiert, falls die Daten es zulassen (siehe Abbildung 3.4). Im iterativen Lernprozess über die Trainingsmenge wird dabei für jede Klasse k ein Gewichtsvektor $\mathbf{w}_k$ berechnet, der eine Hyperebene in der Dimension des Feature-Vektors determiniert. Die Gewichtsvektoren $\mathbf{w}_k$ sind dabei die Normalenvektoren der Hyperebenen. Jede dieser Ebenen separiert den Feature-Raum $\mathbf{F}$ in zwei Hälften, wobei die Datenpunkte, die in der einen Hälfte liegen, der Klasse angehören und die anderen nicht. (Manning u. Schütze, 1999, S. 597 ff.).

Bei Multiclass-Aufgaben wie NER begrenzen die Hyperebenen untereinander bestimmte Bereiche in $\mathbf{F}$, die durch die *argmax*-Zuordnungen determiniert sind. Für ein Beispiel mit drei Klassen siehe Abbildung 3.4(c).

Die lineare Separierbarkeit ist die Voraussetzung für das Funktionieren des Perzeptron-Algorithmus. In praktischen Anwendungen wird dies v.a. durch die hohe Dimensionalität des Feature-Raums und die dadurch entstehenden Freiheitsgrade für $\mathbf{w}$ garantiert (siehe Klein u. Taskar, 2005).

3.3.2 Lernprozess

Im Lernprozess des Perzeptron-Algorithmus gilt es die Gewichtsvektoren $\mathcal{W} = \{\mathbf{w}_{k_1}, \mathbf{w}_{k_2}, \dots, \mathbf{w}_{k_{|\mathcal{K}|}}\}$ für alle Klassen $\mathcal{K} = \{k_1, k_2, \dots, k_{|\mathcal{K}|}\}$ zu erlernen, die die Missklassifikationsrate in der Trainingsmenge (den Trainingsfehler) minimieren. Der Trainingsfehler kann durch eine Fehlerfunktion formal dargestellt werden. Beim Perzeptron-Algorithmus gilt es die sogenannte *Zero-One-Loss*-Fehlerfunktion $E(X)$ der Trainingsmenge X zu minimieren[10] (Collins, 2002):

$$E(X) = \sum_{x \in X} 1 - \mathbb{1}(k_x = \operatorname*{argmax}_{\mathbf{w}} \mathbf{w} \cdot \mathbf{v}_x)$$

$$\mathbb{1}(b) = \begin{cases} 1 & \text{if b is true} \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 2 Der Perzeptron-Lernalgorithmus

```
Input:   Training Set $X$ with Feature Vectors $\mathcal{V} = v_1, v_2, \ldots, v_{|X|}$
         Number of Iterations $Z$
         Set of Classes $\mathcal{K}$

for each class $k \in \mathcal{K}$ do
    create Weight Vector $\mathbf{w}_k = \mathbf{0}$
end for
for $z = 0$ to $Z$ do
    $errors = 0$
    for each token $x \in X$ do
        $k_{pred} = \operatorname*{argmax}_{i=1}^{|\mathcal{K}|} \mathbf{w}_{k_i} \cdot \mathbf{v}_x$
        if $k_{pred} \neq k_x$ then
            $\mathbf{w}_{k_x} = \mathbf{w}_{k_x} + \mathbf{v}_x$
            $\mathbf{w}_{k_{pred}} = \mathbf{w}_{k_{pred}} - \mathbf{v}_x$
            $errors = errors + 1$
        end if
    end for
    if $errors = 0$ then
        break
    end if
end for
output set of weight vectors $\mathcal{W}$
```

Im Initialschritt des Lernalgorithmus werden alle Komponenten der Gewichtsvektoren auf 0 gesetzt. Anschließend wird Z-mal über der Trainingsmenge iteriert und die Gewichtsvektoren nach jedem Klassifikationsfehler angepasst. Das gesamte Verfahren wird in Algorithmus 2 dargestellt. Die Entscheidung einem Token x mit Feature-Vektor $\mathbf{v}_x$ die Klasse k_{pred} zuzuordnen, wird wie folgt getroffen:

$$k_{pred} = \operatorname*{argmax}_{\mathbf{w}} \mathbf{w} \cdot \mathbf{v}_x \tag{3.9}$$

---

[10] k_x ist die Klasse des Token in der Trainingsmenge.

Entspricht k_{pred} nicht der Klasse ${k_x}^{11}$, so werden die Einträge des Gewichtsvektors $\mathbf{w}_{k_x}$ inkrementiert und die Einträge des Gewichtsvektors $\mathbf{w}_{k_{pred}}$ dekrementiert:

$$\mathbf{w}_{k_x} = \mathbf{w}_{k_x} + \mathbf{v}_x$$
$$\mathbf{w}_{k_{pred}} = \mathbf{w}_{k_{pred}} - \mathbf{v}_x$$

Dieser Rekursionsschritt wird so lange wiederholt, bis die maximale Anzahl der Iterationen Z erreicht ist oder in einem gesamten Durchlauf z kein Fehler mehr begangen wird (Collins, 2002).

3.3.3 Dekodierung

In der vorliegenden Arbeit wird für den Perzeptron-Algorithmus ein simples, lokales Greedy Search-Verfahren verwendet. Obwohl die Ergebnisse für diese Art der Dekodierung bei Ratinov u. Roth (2009) etwas schlechter ausfielen als bei aufwendigeren Verfahren wie Viterbi und Beam Search, erwies es sich doch als erstaunlich performant[12].

Greedy Search durchläuft die Eingangssequenz $X = x_1x_2\ldots x_T$ von links nach rechts und trifft in jedem Schritt determinierte Entscheidungen, die anschließend nicht mehr geändert werden. Es gibt also keinen Trade Off zwischen aktuellen und vorhergehenden Entscheidungen wie bei Viterbi. Es können allerdings die letzten Klassenzuordnungen in den Entscheidungen berücksichtigt werden. Abbildung 3.5 veranschaulicht den Dekodierungsprozess. Die Entscheidung, welcher Klasse ein Token x mit Feature-Vektor $\mathbf{v}_x$ zuzuordnen ist, erfolgt wie im Lernprozess (siehe Gleichung 3.9).

3.3.4 Erweiterungen

Collins (2002) schlägt eine einfache Verfeinerung des Lernalgorithmus vor, die einen großen Einfluss auf die Ergebnisse hat. Es wird dazu geraten, einen Durchschnitt über alle Werte zu errechnen, die jede Komponente des Gewichtsvektors $\mathbf{w}$ während des Lernvorgangs annimmt. Ist also $w_i^{z,t}$ die i-te Komponente des Vektors $\mathbf{w}$ im z-ten Lerndurchgang, nachdem das t-te Token der Trainingsmenge verarbeitet wurde, so ist der Durchschnittsparameter $\overline{w}_i$:

$$\overline{w}_i = \frac{\sum_{z=1}^{Z}\sum_{t=1}^{|X|} w_i^{z,t}}{Z|X|}$$

---

[11] k_x ist die Klasse, der das Token in der Trainingsmenge angehört.

[12] Im finalen System von Ratinov u. Roth (2009) erreichte die Greedy Search-Variante nur 0,1 % weniger F_1-Measure als Beam Search der Größe 100.

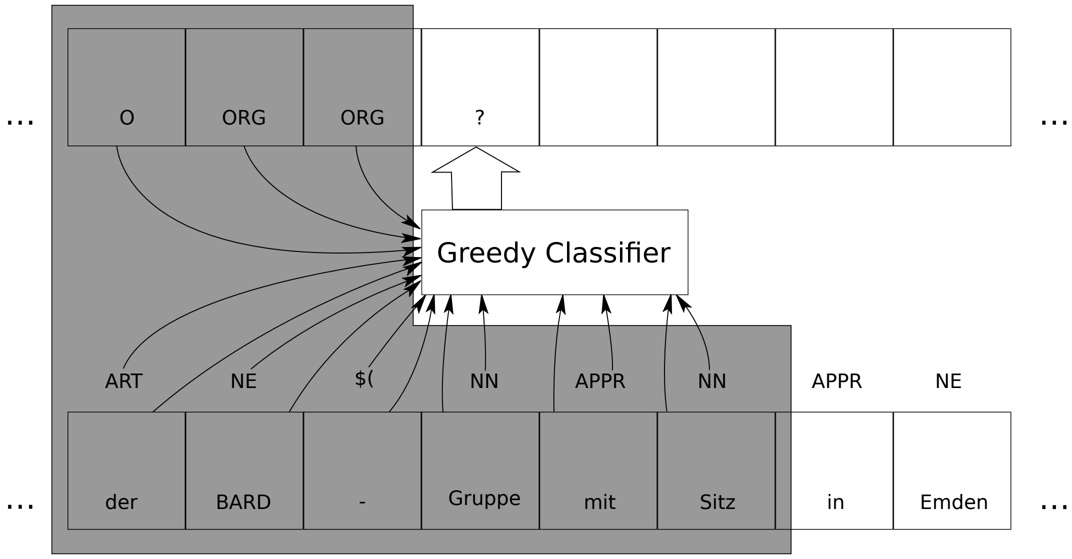


Abbildung 3.5: Ein Greedy Search-Dekodierer mit einer Sequenz aus Beispiel 1.1. Der Klassifizierer durchläuft die Seqenz von links nach rechts und verwendet die Features aus den Templates $word(-3, 2)$, $partofspeech(-3, 2)$ und $entityclass(-3, -1)$. Adaptiert von Jurafsky u. Martin (2008, S. 488).

Das *Voted Perceptron* von Freund u. Schapire (1999) stellt eine noch umfangreichere Erweiterung dar. Dabei wird eine Liste von Gewichtsvektoren gespeichert, wobei der Liste ein Gewichtsvektor nach jedem begangenen Fehler hinzugefügt wird. Für jeden dieser Vektoren wird die Anzahl der Iterationen (Überlebenszeitraum) gezählt, bis der nächste Fehler gemacht wird. Bei der Klassifikation der Testdaten werden dann alle Gewichtsvektoren mit ihrem jeweiligen Überlebenszeitraum gewichtet und daraus eine Entscheidung gefällt.

Der Perzeptron-Algorithmus ermöglicht die Einbeziehung externer Entitätslisten (Gazetteers) über seperate Features, was nach Ratinov u. Roth (2009) eine wichtige Erweiterung darstellt.

Zusätzlich stellen Ratinov u. Roth (2009) Viterbi und Beam Search als Dekodierungsarten für den Perzeptron-Algorithmus vor, deren Performanz sich allerdings gering von der des Greedy Search unterscheiden.

3.4 Gegenüberstellung

Die in dieser Arbeit vorgestellten Verfahren zur Sequenzklassifikation unterscheiden sich fundamental in ihren Grundkonzepten. Im Lernprozess des HMM wird die Wahrscheinlichkeitsverteilung $P(C|X)$ durch die Wahrscheinlichkeitsverteilung $P(X, C)$ modelliert, während in diskriminativen Verfahren, wie dem Perzeptron-Algorithmus, $P(C|X)$ durch eine Funktion $h(X)$ approximiert wird. Der zweite grundlegende Unterschied besteht durch die Dekodierungsverfahren. Im Folgenden wird auf die Besonderheiten und Limitierungen beider Verfahren hingewiesen und anhand dessen in Hinblick auf NER eine erste These für die Perfomanzerwartung formuliert. Diese These wird im weiteren Verlauf der Arbeit durch Experimente mit den implementierten Algorithmen untersucht.

Hidden Markov Model	**Perzeptron-Algorithmus**
1. Wissen	
Beinhalten all ihr im Lernprozess akkumuliertes Wissen in bedingten Wahrscheinlichkeiten. Die denkbar simpelste Form sind die Emissionsahrscheinlichkeiten $P(x_t\|c_t)$ und die Klassenübergangswahrscheinlichkeiten $P(c_t\|c_{t-1})$.	Nutzt eine feature-basierte Wissensquelle für die Klassifikation, die nicht den Umweg über Wahrscheinlichkeiten macht. Jedes Token wird als eine Menge an Features gesehen, die jeweils in einer bestimmten Gewichtung zur Klassifikationsentscheidung beitragen.
2. Flexibilität	
Sind unflexibel in der Auswahl der Features des Textes, da alles durch bedingte Wahrscheinlichkeiten modelliert werden muss.	Sehr flexibel in der Featurewahl. Alle interessierenden Merkmale können in beliebigen Bereichen um die Token ohne Weiteres mit in Betracht gezogen werden.
3. Laufzeit	
Lernprozess: Linear $\mathbf{O}(\|\mathcal{K}\| \times \|X\|)$, Klassifikation: Quadratisch $\mathbf{O}(\|\mathcal{K}\|^2 \times \|X\|)$	Lernprozess (wenn Daten separierbar): Linear $\mathbf{O}(\|X\| \times Z \times \varnothing)$ ($\varnothing$ = durchschnittliche Anzahl an nicht-null Features), Klassifikation: Linear $\mathbf{O}(\|X\|)$
4. Stärken	
Geringe Laufzeit beim Lernen, Trade Off beim Dekodieren.	Sehr flexibel in der Featureauswahl, geringe Laufzeit in der Dekodierungsphase. Einbeziehung nicht-lokaler und externer Features möglich (siehe Ratinov u. Roth, 2009).

5. Limitierungen

Wissensquellen sind sehr begrenzt. Können kein globales Wissen wie aus Ratinov u. Roth (2009) mit einbeziehen.	Die hier verwendete Greedy Search-Dekodierung ermöglicht keinen Trade Off zwischen aktuellen und vorhergehenden Entscheidungen. Sollten die Daten nicht linear separierbar sein, kann keine optimale Lösung gefunden werden.

6. Eignung für Named Entity Recognition

Wahrscheinlich weniger geeignet, da HMMs schwer viel Wissen in Form von lokalen und externen Features mit einbeziehen können. Hinzu kommt, dass HMMs sich besonders dann gut eignen, wenn viel Entscheidungsgehalt in der Sequenz liegt, was bei NER nicht der Fall ist, da es meist sehr wenige oder sogar gar keine Klassenübergänge innerhalb eines Satzes gibt.	Wahrscheinlich gut geeignet, da eine enorme Fülle und Vielfalt an den Daten inhärenten, sowie externen Features mit einbezogen werden können. Das Nicht-Separierbarkeitsproblem entsteht meist durch die hohe Dimensionalität des Featureraumes nicht (siehe Klein u. Taskar, 2005).

Kapitel 4

Implementierung

Im Vorlauf zu dieser theoretischen Abhandlung entstand ein in der Programmiersprache Java umgesetztes Framework zum Thema Named Entity Recognition. Es wurden verschiedene Sequenzklassifikationsalgorithmen umgesetzt und in ihrer Performanz verglichen. Die Algorithmen sind maschinelle Lernverfahren, die ihre Modelle auf dem CoNLL-2003 Shared Task- Datensatz (siehe Abschnitt 4.2) lernen und diese auf die ebenfalls enthaltenen Testdaten anwenden. Die Evaluierung erfolgt durch ein Perl-Script, was mit dem Datensatz mitgeliefert wurde. Dabei werden Precision, Recall und F_1-Measure (siehe Kapitel 2.3) der Testdaten jeweils für jede Klasse berechnet und diese Einzelbewertungen zu einem Gesamtergebnis gemittelt. Der hier dargebotene Exkurs ist vereinfacht. Für detaillierte Implementierungsinformationen wird auf die *JavaDoc* des Frameworks und für theoretische Details auf Kapitel 3 verwiesen.

4.1 Framework

Im Folgenden wird ein Überblick über die Arbeitsweise des Frameworks präsentiert. Der grundsätzliche Programmablauf eines NER-Experiments[1] mit einem vorher gewählten Klassifikationsalgorithmus ist in Abbildung 4.1 dargestellt.

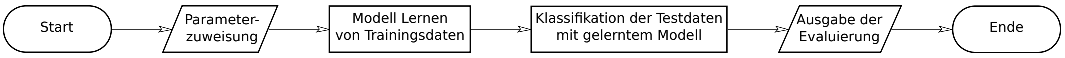


Abbildung 4.1: Der Programmablauf eines Experiments mit einem vorher ausgewählten Klassifikationsalgorithmus.

---

[1] Ein Experiment umfasst das Lernen des Modells, die Klassifikation der Testdaten und die Evaluierung der Ergebnisse.

Die Interaktion mit dem Programm – wie die Zuweisung der Parameterwerte oder der Wahl des Klassifikationsalgorithmus – erfolgt über Konsoleneingaben, die im Abschnitt 4.1.3 beschrieben sind.

4.1.1 Klassenübersicht

Die Paketstruktur des Frameworks ist der *JavaDoc* zu entnehmen. Die ausführbare Klasse ist `NER`. Wird das Programm gestartet, öffnet sich eine Interaktionskonsole, in die man verschiedene Befehle, wie das Starten eines Klassifikationsexperiments mit einem ausgewählten Algorithmus, eingeben kann (siehe Abschnitt 4.1.3). Die Klassen, die die Klassifikationsalgorithmen enthalten, müssen das Interface `Classifier` implementieren und können bei Bedarf von der Superklasse `Parameterizable` erben, damit Parameterwerte vor dem Starten eines Experiments zugewiesen werden können. Die Einzelheiten zu den Klassifikationsalgorithmen folgen in Abschnitt 4.3.

Die zum Lernen und Klassifizieren benutzte Datenstruktur ist `DataSet`, die als Container für Instanzen der Klasse `Token` fungiert (siehe Abschnitt 4.1.2) und von der Klasse `Parser` aus den Datensatzdateien erzeugt wird.

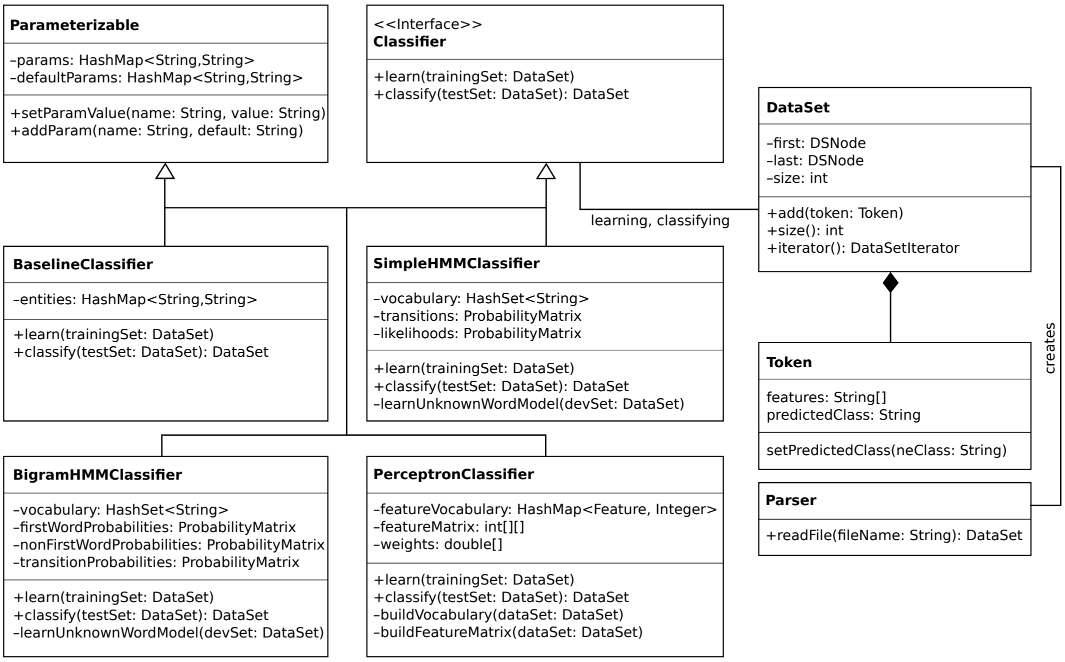


Abbildung 4.2: Das UML-Klassendiagramm mit den wichtigsten Klassen des Frameworks. Für eine Übersicht aller Klassen siehe die JavaDoc.

4.1.2 Datenstrukturen

Die Klasse `DataSet` ist eine selbst implementierte, doppelt verkettete Liste von `Token`-Instanzen. Der Hintergrund dafür ist die Erkenntnis, dass die Iteration über einen Datensatz meist unter Einbeziehung von Satzgrenzinformationen[2] erfolgt, wofür ein eigener Iteratortyp notwendig ist. Dieser Iteratortyp ist in der Klasse `DataSetIterator` implementiert und ist – ähnlich wie der von Java bereitgestellte `ListIterator` – fähig, vorwärts und rückwärts zu iterieren.

Die Klasse `Token` ist die Datenstruktur für ein Token des CoNLL-2003 Shared Task-Datensatzes. Jede Zeile einer Datensatzdatei wird zu einer Instanz dieser Klasse mit allen mitgelieferten Features (siehe Abschnitt 4.2).

4.1.3 Benutzung

Beim Start des Programmes werden der gewünschte Datensatz[3][4], der Pfad zu den Datensatzdateien und optional eine Logdatei, zum Speichern der Ergebnisse, mitgegeben (siehe Anhang A). Ein Beispielprogrammaufruf sähe wie folgt aus:

```
java de.aitools.ner.NER deu ./ eval.log
```

Anschließend öffnet sich eine Interaktionskonsole, in der u.a. Experimente gestartet werden können. Dazu ist die folgende Eingabe nötig:

```
start <Classifier Name> [Optional Flags]
```

Es stehen vier verschiedene Klassifizierer zur Auswahl, die im Abschnitt 4.3 noch näher beschrieben sind. Ein optionales Flag bewirkt z.B. das Loggen der Ergebnisse in der spezifizierten Logdatei. Für eine vollständige Liste der möglichen Klassifizierernamen und der optionalen Flags siehe die JavaDoc. Ein Auszug aus einer Logdatei befindet sich im Anhang A.

Normalerweise arbeiten die Algorithmen nur auf den Development-Mengen. Um die wirkliche Testmenge des jeweiligen Datensatzes mit einzubringen, bedarf es lediglich der Eingabe des Wortes `testset` in die Interaktionskonsole. Des Weiteren kann während der Laufzeit zu einem anderen Datensatz gewechselt werden. Die nötige Eingabe dafür ist `switch`, gefolgt vom Namen des gewünschten Datensatzes[3].

---

[2] Dem Datensatz werden beim Einlesen durch die `Parser`-Klasse `Token`-Instanzen für Satzbeginn und -ende hinzugefügt.

[3] Darunter werden Trainings-, Development- und Testmenge für eine Sprache verstanden.

[4] `deu` – für den deutsche und `eng` – für den englischen Datensatz.

4.2 Datensatz

Der CoNLL-2003 Datensatz beinhaltet zwei Datensätze – einen für die deutsche und einen für die englische Sprache. Jeder Datensatz besteht aus einer Trainingsmenge und zwei Testmengen, die als Development- und echte Testmenge[5] gedacht sind. Die Quelle der enthaltenen Texte sind Zeitungsartikel (Tjong Kim Sang u. De Meulder, 2003).

Alle Datensätze haben die gleiche Struktur. Die Texte wurden tokenisiert und eine Satzsegmentation durchgeführt. Jedem Token wurden Features, die durch Leerzeichen getrennt sind, und die jeweilige Named Entity Klasse von Hand zugeordnet. Jede nichtleere Zeile entspricht einem Token und jede leere Zeile bedeutet das Ende eines Satzes. Der Beginn eines Zeitungsartikels ist durch ein spezielles Token markiert. Es folgt ein kurzer beispielhafter Auszug aus der deutschen Trainingsmenge:

```
-DOCSTART- -X- -X- -X- O

Ereignis Ereignis NN I-NC O
und und KON O O
Erzählung Erzählung NN I-NC O
oder oder KON I-NC O
: : $. O O

Schwierigkeiten Schwierigkeit NN I-NC O
beim beim APPRART I-PC O
```

Die erste Zeile ist das Token für den Artikelbeginn. Im deutschen Datensatz ist das erste Feature der Zeile das Wort, das zweite der Wortstamm, das dritte das Part Of Speech Tag, das vierte das Syntactic Base Phrase Chunk Tag und das letzte die Named Entity Klasse. Die englische Sammlung enthält kein Wortstammfeature. Ansonsten ist die Reihenfolge der Elemente gleich.

Die Chunking-Repräsentation im CoNLL-2003 Datensatz ist IOB_1. Das bedeutet, dass es für jede Entitätsklasse zwei Subklassen gibt, die angeben ob, ein Token den Beginn einer Entität (Präfix ‘B-‘) darstellt oder sich innerhalb einer Entität (Präfix ‘I-‘) bestehend aus mehreren Worten befindet. Bei IOB_1-Repräsentation ist dies jedoch nur der Fall, wenn zwei verschiedene Entitäten direkt aufeinanderfolgen. Anderenfalls werden die Klassen der Token von Entitäten, die aus mehreren Worten bestehen, alle mit Präfix ‘I-‘ begonnen. Die Klasse für Token, die keiner Klasse angehören, ist ‘O‘. Nur $0,26\%$

---

[5] Die Dateinnamen der Datensatzdateien sind *.train (Trainingsmenge), *.testa (Development-Menge) und *.testb (Testmenge). Das * ist durch `eng` bzw. `deu` zu ersetzen.

der Entitäten des deutschen bzw. $0,23\%$ der Entitäten des englischen Datensatzes beinhalten Klassen mit Präfix ‘B-‘. Deswegen wurden alle Präfixe in der Implementierung komplett vernachlässigt und nur mit der Klassenmenge $\mathcal{K} = \{\texttt{PER}, \texttt{LOC}, \texttt{ORG}, \texttt{MISC}, \texttt{O}\}$ gearbeitet.

4.3 Klassifizierer

Im Rahmen der vorliegenden Arbeit wurden vier Sequenzklassifikationsverfahren für NER implementiert. Diese werden im Folgenden beschrieben. Die Ergebnisse, die die jeweiligen Algorithmen hervorgebracht haben, sind in Kapitel 5 dargestellt.

4.3.1 BaselineClassifier

Die Klasse `BaselineClassifier` stellt den wohl einfachsten, vorstellbaren Ansatz eines maschinellen Lernverfahrens für NER dar. Dabei werden einfach alle Named Entities, die in der Trainingsmenge ohne Mehrdeutigkeiten[6] vorkommen, in einer `HashMap` gespeichert. Während des Dekodierens wird anschließend einfach ein simpler *Look Up* in diese `HashMap` gemacht. Der Algorithmus hat keine Parameter.

Lernprozess Der Algorithmus durchläuft sequentiell die gesamte Trainingsmenge und speichert dabei alle eindeutigen Named Entities. Wird ein Token vorgefunden, dass einer Klasse angehört, werden solange die darauffolgenden Token innerhalb des Satzes untersucht, bis ein Token mit einer anderen oder der *outside*-Klasse vorgefunden wird. Alle aufeinanderfolgenden Token der gleichen Klasse formen eine Named Entity, die gespeichert wird.

Dekodierung Da eine Named Entity aus mehreren Token bestehen kann, wird die Look Up-Strategie aus Algorithmus 3 benutzt.

4.3.2 SimpleHMMClassifier

In der Klasse `SimpleHMMClassifier` wird ein sehr einfacher generativer Ansatz benutzt, wobei die Wahrscheinlichkeit einer jeden Klassifikationsentscheidung

---

[6] D.h., dass mehrere Vorkommen der Named Entity in der Trainingsmenge der gleichen Klasse angehören.

Algorithm 3 Dekodierung des `BaselineClassifier`

```
Input:    Test Set $X$
          Hash Map $H$

for each Token $x \in X$ do
    if $x \in H$ then
        $x$.classifyWith($H(x)$)
    else
        for each Token $x'$ following $x$ in the current sentence do
            $x = x + x'$
            if $x \in H$ then
                $x$.classifyWith($H(x)$)
                break
            end if
        end for
    end if
end for
output The classified Test Set $X'$
```

vom Wort und von der vorhergehenden Klasse abhängt. Es wird eine Unabhängigkeit zwischen beiden Bedingungen angenommen, was zum folgenden Modell führt (Malouf, 2002):

$$P(x_t|c_t, c_{t-1}) = P(x_t|c_t)P(c_t|c_{t-1}) \tag{4.1}$$

Die Klassensequenz mit der höchsten Wahrscheinlichkeit C_{max} bezüglich der Eingangssequenz X ist also (Malouf, 2002):

$$C_{max} = \operatorname*{argmax}_{C} \prod_{t=1}^{|X|} P(x_t|c_t)P(c_t|c_{t-1}) \tag{4.2}$$

Um das *Sparse Data*-Problem (siehe Kapitel 3.2.4) zu lösen, wird eine einfache Smoothing-Methode eingeführt. Die Klasse `SimpleHMMClassifier` stellt einen Parameter bereit, der angibt, welche feste Wahrscheinlichkeit Wort-Klassen-Paare haben, die nie in den Trainingsdaten gesehen werden. Des Weiteren wird ein Modell für unbekannte Worte auf den Development-Daten gelernt.

Die Parameter, die für die `SimpleHMMClassifier`-Klasse spezifiziert werden können sind in Tabelle 4.1 zusammengestellt.

Lernprozess Während des Lernens werden die Wahrscheinlichkeiten $P(x_t|c_t)$ und $P(c_t|c_{t-1})$ durch ihre relativen Häufigkeiten in der Trainingsmenge abgeschätzt. Die Berechnungen wurden bereits in den Gleichungen 3.5 und 3.6

Parametername	**Datentyp**	**Anwendung**
`UnknownWordModel`	`boolean`	Wenn `true` wird ein Modell für unbekannte Worte trainiert, sonst nicht
`NullValue`	`double`	Die Wahrscheinlichkeit $P(x_t \| c_t)$ von Wort-Klassen-Paaren, die nie in den Trainingsdaten gesehen werden

Tabelle 4.1: Die Parameter der `SimpleHMMClassifier`-Klasse.

vorgestellt. Zur Speicherung der Wahrscheinlichkeiten wird eine `HashMap` benutzt, die einen schnellen Look Up während des Dekodierens möglich macht.

Zusätzlich kann ein Modell für unbekannte Worte auf den Development-Daten gelernt werden, das in das eigentliche Modell integriert wird. Beim Lernen auf den Trainingsdaten wird dazu ein Vokabular angelegt, das alle bekannten Worte enthält. Zur Abschätzung der Wahrscheinlichkeit für Klassenzuordnungen von unbekannten Worten wird folgende Berechnung durchgeführt:

$$P(-\texttt{UNKNOWN}-|c_t) = \frac{count(-\texttt{UNKNOWN}-, c_t)}{count(c_t)}$$

Die vom `Parser` eingefügten Satzbeginn- und -end-Token werden in zusätzlichen Zuständen modelliert. Die Wahrscheinlichkeit $P(c_t|-\texttt{SENSTART}-)$ sagt aus, wie wahrscheinlich das erste Token eines Satzes der Klasse c_t angehört. Analog dazu sagt die Wahrscheinlichkeit $P(-\texttt{SENEND}-|c_{t-1})$ aus, wie wahrscheinlich das letzte Token eines Satzes der Klasse c_{t-1} entspricht.

Dekodierung Für die Dekodierung wird der Viterbi-Algorithmus eingesetzt, der bereits ausführlich in Kapitel 3.2.3 beschrieben wurde. Das in Abbildung 3.3 dargebotene Beispiel lässt sich exakt auf das Verfahren der Klasse `SimpleHMMClassifier` anwenden. Wie bereits kurz beschrieben sind, die Initialwahrscheinlichkeiten Π aus Kapitel 3.2 durch die Wahrscheinlichkeiten $P(c_t|-\texttt{SENSTART}-)$ und die Wahrscheinlichkeiten $\mathcal{E}$ durch $P(-\texttt{SENEND}-|c_{t-1})$ modelliert.

Wird beim Dekodieren ein Wort gefunden, das nicht im Vokabular der Trainingsdaten vorkommt, wird die Wahrscheinlichkeit des unbekannten Wortes bezogen auf die jeweilige Klasse verwendet. Die Wahrscheinlichkeit von Wort-Klassen-Paaren, die nie in den Trainingsdaten gesehen werden, werden mit der Wahrscheinlichkeit, die im Parameter `NullValue` angegeben ist, bewertet.

4.3.3 BiGramHMMClassifier

In der Klasse `BiGramHMMClassifier` wird die Modellierung des `SimpleHMMClassifier` erweitert, wie bereits in Kapitel 3.2.4 angedeutet. Die Inspiration für den hier implementierten Ansatz kommt von Bikel u. a. (1999), deren Verfahren teilweise umgesetzt ist. Bezüglich der Token wird die Modellierung von der Unigram- $P(x_t|c_t)$ zu einer Bigram-Modellierung $P(x_t|c_t, x_{t-1})$ augmentiert. D.h. die Wahrscheinlichkeit, dass ein Token x_t der Klasse c_t zugeordnet wird, hängt nun auch vom vorhergehenden Token x_{t-1} ab. Des Weiteren wird ein Strukturfeature f des Wortes mit einbezogen. Bikel u. a. (1999) benutzen eine Menge von 14 Features, wie Groß- und Kleinschreibung, Anwesenheit von Zahlen und Sonderzeichen usw.. Die Implementierung in dieser Arbeit nutzt dafür die im Datensatz bereitgestellten Part Of Speech Tags. Außerdem werden Übergänge zwischen Token gleicher Klassen (Gleichung 4.3) mit anderen Wahrscheinlichkeiten modelliert als Übergänge zwischen Token zweier verschiedener Klassen (Gleichung 4.4) (Bikel u. a., 1999).

$$P(\langle x, f\rangle_t | \langle x, f\rangle_{t-1}, c_t) \tag{4.3}$$

$$P(c_t|c_{t-1}, x_{t-1}) P(\langle x_t, f_t\rangle_{first} | c_t, c_{t-1}) \tag{4.4}$$

In Gleichung 4.3 und 4.4 stellen $\langle x, f\rangle_t$ die Wort-Feature-Paare dar. Gleichung 4.4 ist also die Wahrscheinlichkeit eines Token das erste Element einer Tokensequenz gleicher Klassen zu sein. Dem zufolge beinhaltet Gleichung 4.3 die Wahrscheinlichkeit für ein Token ein Element innerhalb einer Tokensequenz gleicher Klassen zu sein. Eine Veranschaulichung des HMM erfolgt in Abbildung 4.3.

Die `BiGramHMMClassifier`-Klasse hat die gleichen Parameter wie die `SimpleHMMClassifier` -Klasse.

Lernprozess Wie beim `SimpleHMMClassifier` werden während des Lernens im `BiGramHMMClassifier` die Transitions- und Emissionswahrscheinlichkeiten durch relative Häufigkeiten abgeschätzt. Durch die erweiterte Modellierung enstehen dabei folgende Gleichungen (Bikel u. a., 1999):

$$P(\langle x, f\rangle_t | \langle x, f\rangle_{t-1}, c_t) = \frac{count(\langle x, f\rangle_t, \langle x, f\rangle_{t-1}, c_t)}{count(\langle x, f\rangle_{t-1}, c_t)}$$

$$P(c_t|c_{t-1}, x_{t-1}) = \frac{count(c_t, c_{t-1}, x_{t-1})}{count(c_{t-1}, x_{t-1})}$$

$$P(\langle x, f\rangle_{first} | c_t, c_{t-1}) = \frac{count(\langle x, f\rangle_{first}, c_t, c_{t-1})}{count(c_t, c_{t-1})}$$

Es kann auch hier zusätzlich ein Modell für unbekannte Worte auf den Development-Daten gelernt werden. Dabei ist zu beachten, dass es nun drei Möglichkeiten gibt, in denen unbekannte Worte in Bigrammen vorkommen können: als aktuelles Wort, als vorhergehendes Wort oder an beiden Positionen. Mit Satzgrenzen wird äquivalent zum `SimpleHMMClassifier`-Lernprozess umgegangen.

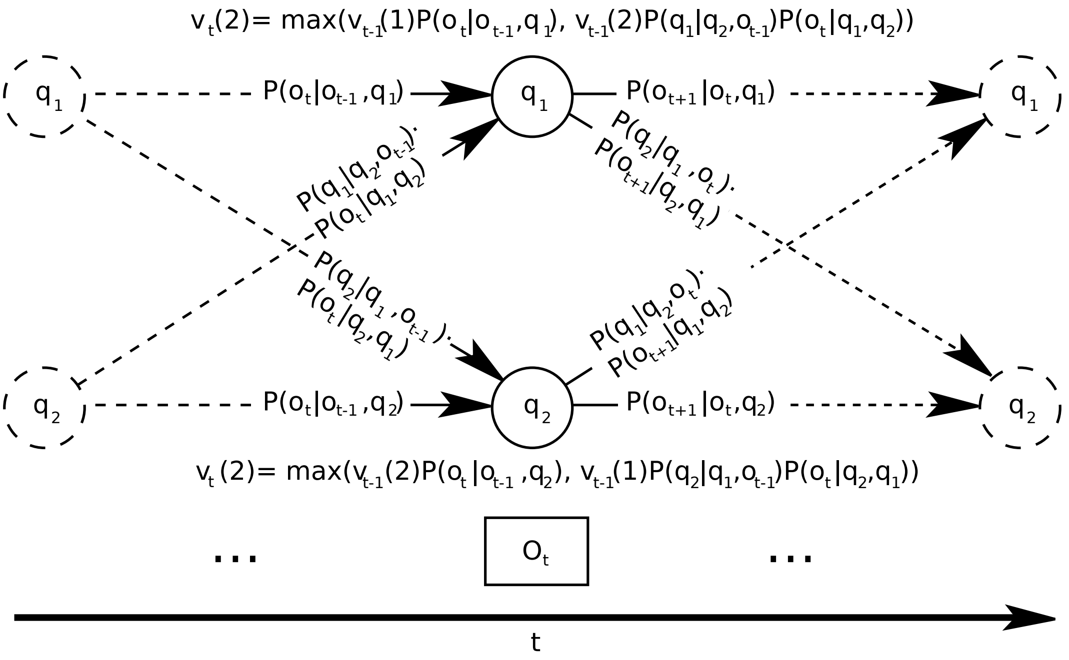


Abbildung 4.3: Die Berechnung einer Spalte des Viterbi-Trellis im Algorithmus der `BiGramHMMClassifier`-Klasse in Anlehnung an Abbildung 3.3.

Dekodierung Wieder kommt der Viterbi-Algorithmus für die Dekodierung zum Einsatz. Die Unterscheidung zwischen Klassenübergängen gleicher und unterschiedlicher Klassen, die Anfang des Abschnitts erwähnt wurde, führt zu etwas veränderten Berechnungen des Trellis-Komponenten während des Viterbi-Algorithmus, die in Abbildung 4.3 dargestellt sind.

Bikel u. a. (1999) schlagen eine intensive Interpolationsmethode vor, um dem Sparse Data-Problem zu begegnen, die im Rahmen dieser Arbeit nicht umgesetzt und durch die gleiche Smoothing-Methode wie im `SimpleHMMClassifier` ersetzt ist. Der Umgang mit unbekannten Worten ist analog zu der `SimpleHMMClassifier`-Klasse.

4.3.4 PerceptronClassifier

Der Algorithmus der `PerceptronClassifier`-Klasse ist angeregt durch Collins (2002), der einen sehr ausgeklügelten Perzeptron-Algorithmus entwickelt hat. Im Rahmen dieser Arbeit ging es v.a. darum, den generativen Lernverfahren der HMMs ein diskriminatives gegenüberzustellen, das mit vielen den Daten inhärenten Features umgehen kann. Es werden bei dieser Implementierung nur lokale Features genutzt. In der folgenden Liste sind alle Feature Templates und ihre Namen in der Implementierung zusammengefasst, die in beliebig großen Fenstern um die Token herum die Features generieren können (siehe Beispiel 3.3):

- Worte (`word`)
- Part Of Speech Tags (`part_of_speech`)
- Wortstämme (nur für deutschen Datensatz) (`stem`)
- Syntactic Base Phrase Chunk Tags (`syntactic`)
- Informationen über Großschreibung (`is_uppercase`, `is_title`)
- Informationen über die Anwesenheit von Sonderzeichen, Satzzeichen und Trennstrichen (`contains_hyphen`, `contains_special_char`)
- Die bereits zugeordneten Klassen der Token $x_{t-1}, x_{t-2} \ldots$ (`ne_class`)
- Präfixe/Suffixe (`suffix1`, `suffix2`, `suffix3`, `prefix1`, `prefix2`, `prefix3`[7])

Vor dem Lernprozess, in dem die Gewichtsvektoren $\mathbf{w}$ ermittelt werden, muss also zuerst ein Feature-Vokabular V aufgebaut werden. Dieses Vokabular ist hier durch eine `HashMap` umgesetzt, in der jedem Feature (z.B. `word(0) = Der`) ein Index zugewiesen ist. Darauf folgt die Erzeugung einer Feature-Matrix für die gesamte Trainingsmenge. In Form eines zweidimensionalen `int`-Arrays enthält die Matrix für jedes Token eine Zeile (hier ein `int`-Array), die nur die Indezes der Features aus dem Vokabular enthalten, die für das Token wahr sind[8]. Diese Arrays sind also in Sparse-Repräsentation (siehe Abbildung 4.4).

Die Parameter der `PerceptronClassifier`-Klasse sind in Tabelle 4.2 dargestellt.

---

[7] Die Zahl in den Template-Namen gibt die Länge des betrachteten Affixes an.

[8] Features sind Indikatorfunktionen (siehe Kapitel 3.3.1).

Parametername	**Datentyp**	**Anwendung**
`NumberOfIterations`	`int`	Die Anzahl der Iterationen über die Trainingsmenge
`AveragingWeights`	`boolean`	Wenn `true` werden die Gewichtsvektoren gemittelt, sonst nicht
`FeatureThreshhold`	`int`	Die Mindestanzahl, die ein Feature während des Aufbaus des Feature-Vokabulars vorkommen muss, um beachtet zu werden
`FeatureTemplateWindows`	`list`	Die Template-Fenster, die die Features generieren sollen in einer Liste, getrennt mit Leerzeichen (z.B. `word(-2,2) stem(-1,1)`)

Tabelle 4.2: Die Parameter der `PerceptronClassifier`-Klasse.

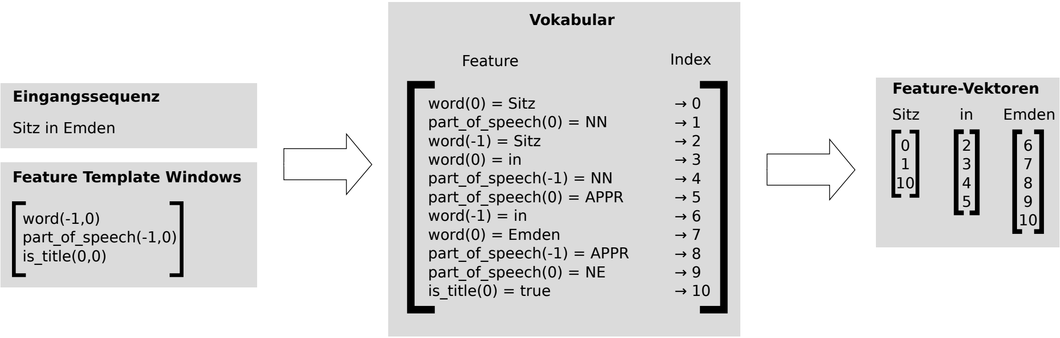
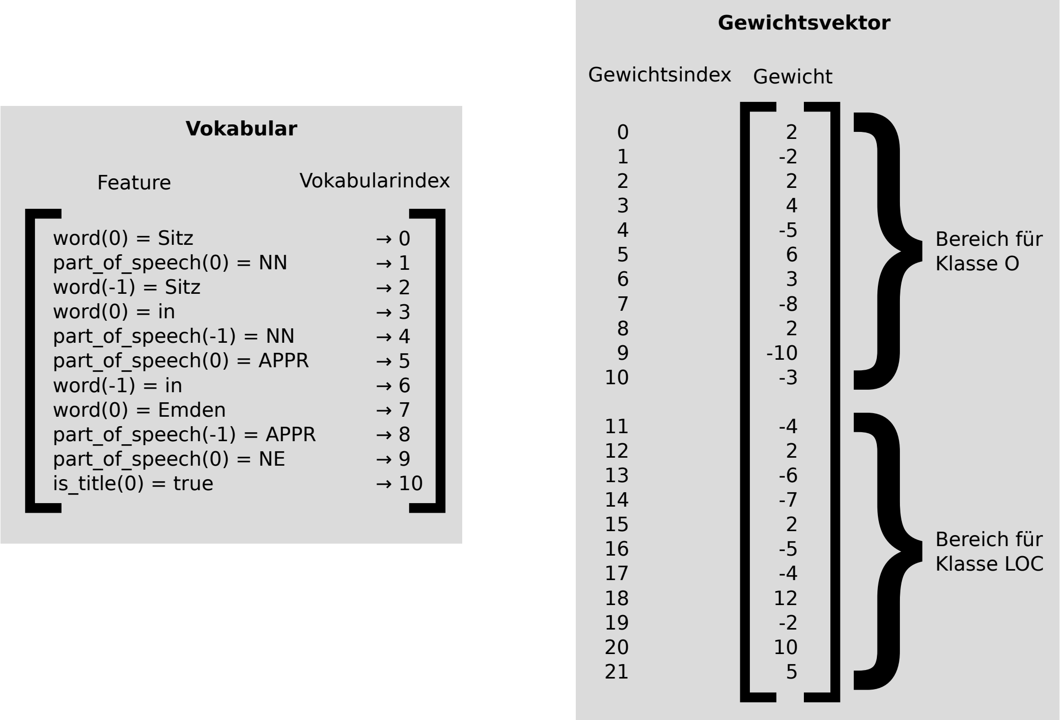


Abbildung 4.4: Aufbau der Daten vor dem Perzeptron-Lernprozess.

Lernprozess Im Perzeptron-Lernalgorithmus gilt es den Gewichtsvektor zu finden, der die Features des Feature-Vokabulars V so gewichtet, dass bei der Klassifikation der Trainingsmenge keine oder lediglich wenige Klassifikationsfehler entstehen. In der hier bearbeiteten Multiclass-Aufgabe gilt es für jede Klasse einen solchen Vektor zu bestimmen. Hier sind diese Gewichtsvektoren alle in einem großen `double`-Array zusammengeführt. Die Gewichtsvektoren beinhalten für jedes Feature des Feature-Vokabulars eine Komponente – sie sind also in Dense-Repräsentation. Das `double`-Array hat also mit allen Gewichtsvektoren eine Gesamtlänge von $|V| \times |\mathcal{K}|$. Um den Bereichsbeginn eines einzelnen Gewichtsvektors zu ermitteln, bedarf es lediglich einer Indezierung der Klassen ($k_1 = 1, k_2 = 2 \ldots$) und der Berechnung $Klassenindex \times |V|$ (siehe Abbildung 4.5).



Vokabular

Feature	Vokabularindex
word(0) = Sitz	→ 0
part_of_speech(0) = NN	→ 1
word(-1) = Sitz	→ 2
word(0) = in	→ 3
part_of_speech(-1) = NN	→ 4
part_of_speech(0) = APPR	→ 5
word(-1) = in	→ 6
word(0) = Emden	→ 7
part_of_speech(-1) = APPR	→ 8
part_of_speech(0) = NE	→ 9
is_title(0) = true	→ 10

Gewichtsvektor

Gewichtsindex	Gewicht	
0	2	Bereich für Klasse O
1	-2	
2	2	
3	4	
4	-5	
5	6	
6	3	
7	-8	
8	2	
9	-10	
10	-3	
11	-4	Bereich für Klasse LOC
12	2	
13	-6	
14	-7	
15	2	
16	-5	
17	-4	
18	12	
19	-2	
20	10	
21	5	

Abbildung 4.5: Beispiel eines Gewichtsvektors für das Vokabular aus Abbildung 4.4 und die Klassen *O* und *LOC*. *O* hat den Klassenindex 0 und *LOC* den Klassenindex 1. Um z.B. den Bereichsbeginn der Klasse *LOC* im Gewichtsvektor zu bestimmen, wird $|V| \times 1 = 11$ berechnet.

Der dafür verwendete Lernalgorithmus entspricht dem in Kapitel 3.3.2 beschriebenen Algorithmus 2. Die einzige Erweiterung ist die von Collins (2002) beschriebene Durchschnittsbildung der Gewichtsvektorkomponenten (siehe Kapitel 3.3.4).

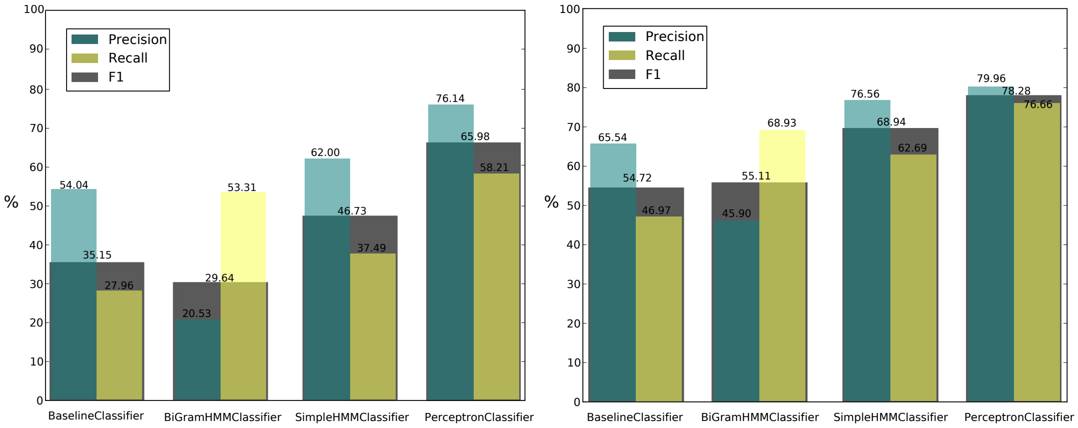
Dekodierung Vor dem eigentlichen Dekodieren wird aus der Testmenge eine Feature-Matrix erstellt (siehe Abbildung 4.4). Die Dekodierung ist durch eine Greedy Search-Strategie implementiert, die die zu klassifizierende Eingangssequenz einfach von links nach rechts durchläuft. Jede Klassifikationsentscheidung wird gegeben eines Token x mit Feature-Vektor $\mathbf{v}_x$ durch die Berechnung aus Gleichung 3.9 getroffen.

Kapitel 5

Experimente

In diesem Kapitel werden die Ergebnisse der implementierten Klassifikationsalgorithmen gegenübergestellt und die Auswirkung von Parametern und datensatzspezifischen Eigenschaften interpretiert.

5.1 Gegenüberstellung der Ergebnisse



(a) Ergebnisse für die deutschen Testdaten (deu.testb).

(b) Ergebnisse für die englischen Testdaten (eng.testb).

Abbildung 5.1: Gegenüberstellung aller Klassifikationsalgorithmen mit den Parameterwerten, die sich als die besten erwiesen (siehe Abschnitt 5.2). Die Ergebnisse werden nur für die eigentlichen Testdaten (*.testb) dargestellt, da die HMMs ihre Modelle unbekannter Worte auf den Development-Daten lernen.

In Abbildung 5.1 sind die Ergebnisse der implementierten Algorithmen für die Testmengen beider Datensätze gegenübergestellt. Wie bereits in Kapitel 3.4 vermutet, erwies sich der Feature-basierte Ansatz des Perzeptron-Algorithmus als performantester. HMMs schnitten sowohl für die deutschen als auch für die englischen Testdaten wesentlich schlechter ab. Diese Beobachtung deckt sich mit der von Florian u. a. (2003). Im CoNLL-2003 Shared Task wählten sie eine Kombination aus mehreren Klassifikationsalgorithmen, worunter sich auch der HMM-Ansatz nach Bikel u. a. (1999) befand, der die geringste Performanz v.a. für die deutschen Daten aufwies.

Im hier implementierten Algorithmus der `BiGramHMMClassifier`-Klasse wurden nicht einmal die Ergebnisse des `BaselineClassifier` für die deutschen Daten übertroffen. Der Algorithmus klassifiziert überdurchschnittlich viele Token und erhält damit einen ziemlich hohen Recall-Wert. Dies geschieht allerdings in Zusammenhang mit einer starken Precision-Einbuße.

Der einfachere Algorithmus des `SimpleHMMClassifier` führte zu besseren Ergebnissen, die jedoch mit $46,73\%$ F_1-Measure bei den deutschen Testdaten immernoch sehr unbefriedigend sind. Für die englischen Daten wäre er allerdings im Ranking des CoNLL-2003 Shared Task noch vor Hammerton (2003) gewesen.

Die Ergebnisse des `PerceptronClassifier` übertreffen die des `SimpleHMMClassifier` jedoch bei weitem. Bezüglich der deutschen Testdaten hätte diese Implementierung Rang 12 von 17 und bezüglich der englischen Rang 14 von 17 im CoNLL-2003 Shared Task erreicht (Tjong Kim Sang u. De Meulder, 2003).

5.2 Parameterauswirkungen

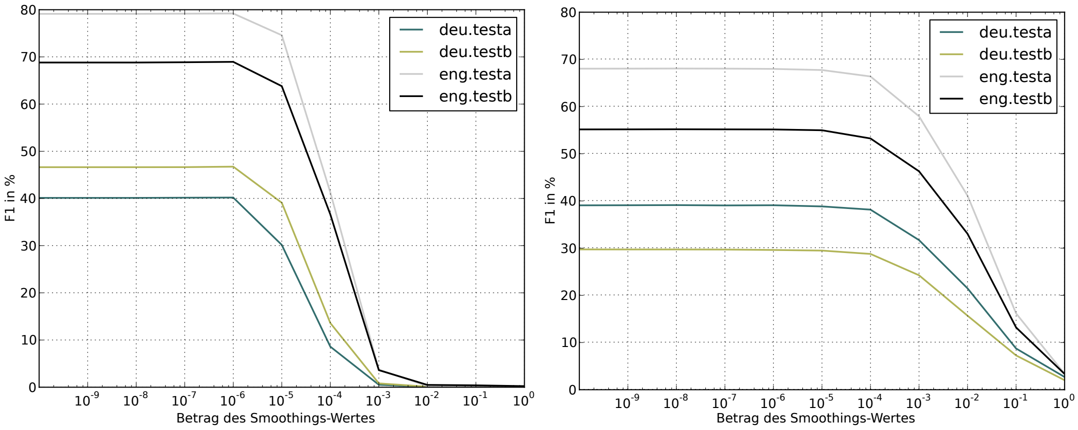
Die Parameterwerte der jeweiligen Algorithmen haben entscheidende Auswirkungen auf die Ergebnisse. Im Folgenden werden diese empirisch anhand von Ergebnisdiagrammen interpretiert.

Hidden Markov Models Das Lernen eines Modells unbekannter Worte bewirkt die folgende Verbesserung der Ergebnisse:

$$\texttt{SimpleHMMClassifier}: \textit{deu.testb} \ +3,5\%, \quad \textit{eng.testb} \ +5,2\%$$

$$\texttt{BiGramHMMClassifier}: \textit{deu.testb} \ +3,3\%, \quad \textit{eng.testb} \ +5,5\%$$

Die optimalen Werte des Smoothing-Parameters `NullValue` sind 10^{-6} für den `SimpleHMMClassifier` und 10^{-8} für den `BiGramHMMClassifier`. Die Beeinflussung der Performanz ist in Abbildung 5.2 dargestellt.



(a) Ergebnisse für den `SimpleHMMClassifier`.

(b) Ergebnisse für den `BiGramHMMClassifier`.

Abbildung 5.2: Die Auswirkung des Smoothing-Parameters.

Perzeptron-Lernalgorithmus Einführend wird die ideale Parameterbelegung des `PerceptronClassifier` für die Datensätze beider Sprachen vorgestellt, die zu den Ergebnissen aus Abbildung 5.1 führen. Wird im weiteren Verlauf des Abschnitts ein Parameter in Isolation betrachtet, so ist den anderen Parametern ihr Idealwert aus der folgenden Liste zugewiesen:

Deutsch:

```
FeatureThreshhold = 0
NumberOfIterations = 100
AveragingWeights = true
FeatureTemplateWindows = word(-2,2) part_of_speech(-1,0)
ne_class(-2,-1) stem(-1,0) contains_hyphen(0,0)
is_uppercase(0,0) suffix3(-1,0) prefix3(0,1) is_title(-1,0)
prefix2(0,1) suffix2(-1,0)
```

Englisch:

```
FeatureThreshhold = 1
NumberOfIterations = 100
AveragingWeights = true
FeatureTemplateWindows = word(-2,2) part_of_speech(-1,0)
ne_class(-2,-1) contains_hyphen(0,0) is_uppercase(0,0)
suffix3(-1,0) prefix3(0,1) is_title(-1,0) syntactic(-1,1)
```

Bezüglich der deutschen Testdaten hat sich der `FeatureThreshhold`-Parameter als unsinnig erwiesen. Wird dieser Parameter mit dem Wert 1 belegt, bewirkt er bereits eine Performanzeinbuße von $-2,0\%$ F_1-Measure. Anders verhält es sich beim Englischen – ein Threshhold von 1 führt zu einer Verbesserung der Ergebnisse von $+0,4\%$ F_1-Measure. Durch einen höheren Threshhold wird kein weiterer Gewinn ermöglicht.

Die Durchschnittsbildung des Gewichtsvektors (siehe Kapitel 3.3.4) wirkt sich sowohl für den deutschen als auch für den englischen Datensatz positiv aus und erhöht die F_1-Measure um ca. $+2,5\%$.

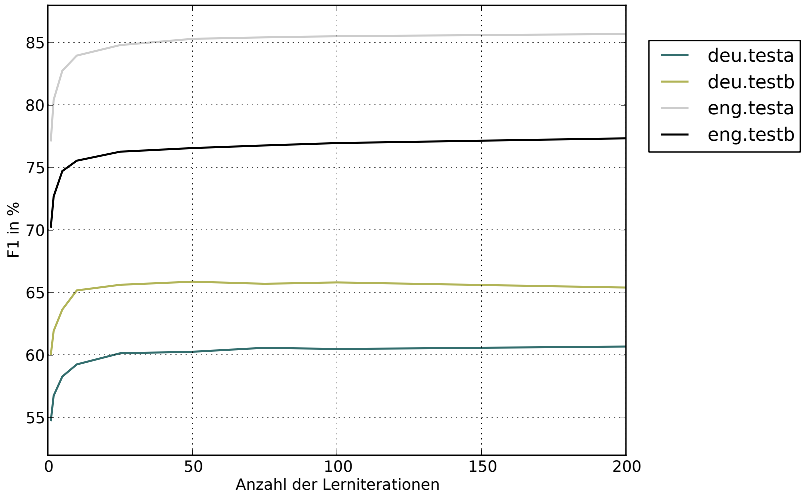
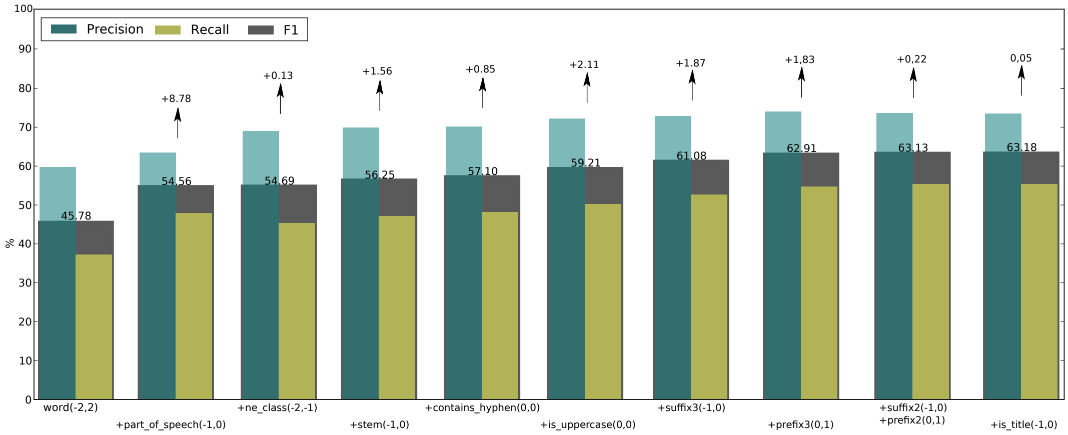
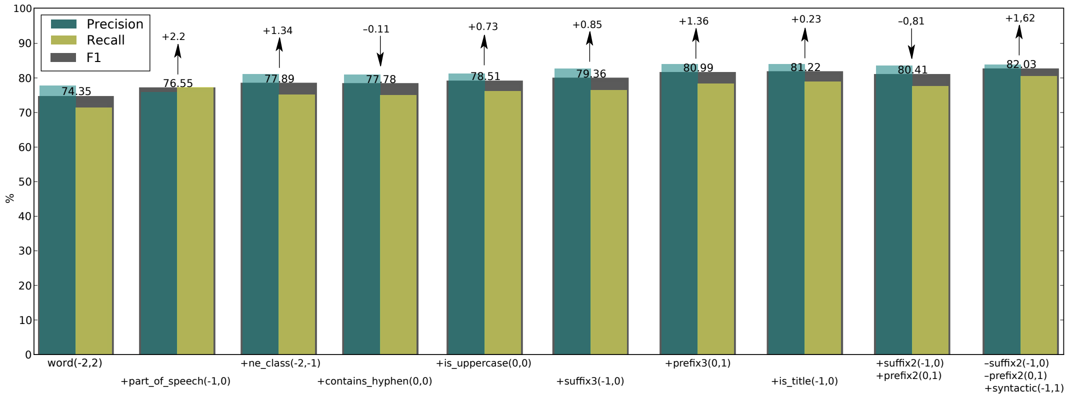


Abbildung 5.3: Die Auswirkung der Anzahl der Lerniterationen.

Erstaunlich ist, dass die Gewichtsvektoren, die nach einem Iterationsschritt gelernt wurden, schon zu relativ guten Ergebnissen führen. Der Performanzgewinn von einer Iteration zu 100 beträgt nur $+5-7\%$ F_1-Measure, wie Abbildung 5.3 zeigt. Zusätzlich dazu zeigt der Graph der deutschen Testdaten ein Phänomen, was von Manning u. Schütze (1999) beschrieben wird: „If the perceptron learning algorithm is run on a problem that is not linearly separable, the linear separator will sometimes move back and forth erratically while the algorithm tries in vain to find a perfectly separating plane.“ (Manning u. Schütze, 1999, S.603). In dieser Feature-Konstellation scheint das Problem für die deutschen Trainingsdaten also nicht linear separierbar zu sein, da die Performanz ab 50 Iterationen um kleine Beträge oszilliert. Der Fakt, dass die Token der CoNLL-2003 Datensätze nicht von Muttersprachlern klassifiziert wurden und somit Inkonsistenzen entstanden sein könnten, stellt eine mögliche Ursache für diese Oszillation dar. Dieses Phänomen lässt sich für den englischen Datensatz und die dafür verwendeten Features nicht beobachten.



(a) Die gemittelten Ergebnisse für deu.testa und deu.testb.



(b) Die gemittelten Ergebnisse für eng.testa und eng.testb.

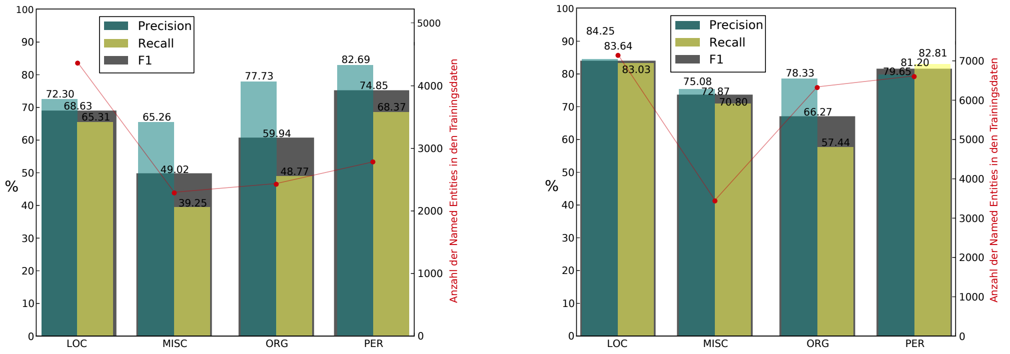
Abbildung 5.4: Die Auswirkung des `FeatureTemplateWindows`-Parameters. Die X-Achse der Diagramme enthält den Aufbau der Template Window-Liste. Angefangen wird mit `word(-2,2)`. Das + vor `part_of_speech(-1,0)` signalisiert, dass dieses Window der Liste hinzugefügt wird. Ein − bedeutet, es wird aus der Liste entfernt.

Die Auswahl der Templates und ihrer Fenstergröße sind die entscheidendsten Komponenten bei der Klassifikation mit dem `PerceptronClassifier`. Abbildung 5.4 zeigt den schrittweisen Aufbau der Template Window-Listen, die die höchste Performanz hervorriefen. Das erste Template, was für beide Datensätze hinzukommt ist `part_of_speech`. Der sprunghafte Performanzgewinn v.a. für den deutschen Datensatz kommt daher, dass dies das erste Template ist, was hinzukommt. Das Template `stem` würde an dieser Stellen den gleichen Effekt hervorrufen. Das Hinzufügen des `ne_class`-Templates bewirkt einen starken Anstieg der Precision bei beiden Datensätzen. Jedoch ist der Gesamtnutzen für den deutschen Datensatz eher gering. Die `prefix`- und `suffix`-Templates bringen v.a. für den deutschen Datensatz einen großen

Performanzgewinn von $+3,92\%$ F_1-Measure. Für den englischen Datensatz genügen Präfix- und Suffixinformationen der ersten beziehungsweise letzten drei Buchstaben. Das Hinzuziehen weiterer Affixinformationen führt zu einer Performanzeinbuße. Des Weiteren bringen die strukturellen Templates `contains_hyphen`, `is_title` und `is_uppercase` für den deutschen Datensatz einen entscheidenden Nutzen von $+3,01\%$ F_1-Measure. Die Performanz für den englischen Datensatz wird dadurch ebenfalls begünstigt. Das `syntactic`-Template erwies sich nur für den englischen Datensatz als gewinnbringend. Da die Wortstamminformationen nur in den deutschen Daten vorhanden war, konnte das dafür vorgesehene Template nur hierbei mit einem Gewinn von $+1,56\%$ F_1-Measure eingesetzt werden.

5.3 Datensatzbezogene Auswirkungen

Die jeweiligen Datensatzeigenschaften und ihre Auswirkungen auf die Ergebnisse werden im Folgenden für den Perzeptron-Algorithmus untersucht, da sich dieser als der performanteste bezüglich NER erwies. Dabei werden die vorgestellten idealen Parameterwertekonstellationen genutzt.



(a) Ergebnisse für deu.testb.

(b) Ergebnisse für eng.testb.

Abbildung 5.5: Die Ergebnisse aufgeschlüsselt für die einzelnen Named Entity Klassen und die Zusammensetzung der Trainingsdaten.

Die Vermutung liegt nahe, dass eine steigende Anzahl der Beispiele von Named Entities einer Klasse in den Trainingsdaten am stärksten mit einer Perfomanzverbesserung bezüglich dieser Klassen einhergeht. Die Diagramme aus Abbildung 5.5 stellen dies jedoch in Frage. Für den deutschen Datensatz wird die Vermutung zwar bestärkt, da die am meist vorkommenden Klassen `LOC` und `PER` die besten und die weniger häufig auftrenden Klassen `MISC` und `ORG` schlechtere Ergebnisse aufweisen, jedoch ist dieses Phänomen für den englischen Datensatz nicht zu beobachten. Dort erhält die Klasse `MISC`, die am seltensten in den Trainingsdaten enthalten ist, bessere Ergebnisse als die Klasse `ORG`, die 2883 mal häufiger in den Trainingsdaten vorkommt. Es scheint also andere Faktoren zu geben, die NER stärker begünstigen.

In Tabelle 5.1 sind einige exemplarische Beispiele für Named Entities aus der deutschen Testmenge zusammengestellt, die die Analyse des datensatzspezifischen Diagramms anhand ihrer Beschaffenheit verfeinern sollen.

`LOC`	`MISC`
in [$_{\text{LOC}}$ Vockenhausen]	[$_{\text{MISC}}$ französischen]
in [$_{\text{LOC}}$ Niederjosbach]	[$_{\text{MISC}}$ europäische]
[$_{\text{LOC}}$ Eppsteiner Straße] in	[$_{\text{MISC}}$ Europacup-Qualifikation]
in [$_{\text{LOC}}$ Niedernhausen]	[$_{\text{MISC}}$ deutschen]
aus [$_{\text{LOC}}$ Tibet]	[$_{\text{MISC}}$ evangelische]
aus [$_{\text{LOC}}$ Afghanistan]	[$_{\text{MISC}}$ Prager]
im [$_{\text{LOC}}$ Pavillon] neben	[$_{\text{MISC}}$ Mendelssohn-Bartholdys]
`PER`	`ORG`
Gegenspieler [$_{\text{PER}}$ Karsten Baumann]	[$_{\text{ORG}}$ Jugendzentrum]
Christdemokratin [$_{\text{PER}}$ Renate Seib]	[$_{\text{ORG}}$ Frankfurter Amt für Multikulturelles]
Vorsteher [$_{\text{PER}}$ Rainer Bergert]	[$_{\text{ORG}}$ Sozialdemokraten]
Pressesprecher [$_{\text{PER}}$ Edmund Volk]	[$_{\text{ORG}}$ SG Anspach]
Trainer [$_{\text{PER}}$ Stepanovic]	[$_{\text{ORG}}$ Fachverband für Hauswirtschaft]
Staatssekretärin [$_{\text{PER}}$ Brigitte Sellach]	[$_{\text{ORG}}$ UVF]
Kollege [$_{\text{PER}}$ Rübenach]	[$_{\text{ORG}}$ Turkish Airlines]

Tabelle 5.1: Einige Named Entity Beispiele aus deu.testb.

Die Beispiele der Klassen `LOC` und `PER` sind mit gewissen Kontextinformationen dargestellt, weil Ortsangaben meist nach oder vor Präpositionen wie ‘in‘, ‘im‘, ‘aus‘, ‘neben‘ u.ä. vorkommen und Personnamen im Datensatz sehr häufig auf eine Funktionsbeschreibung wie ‘Pressesprecher‘, ‘Trainer‘ u.ä. folgen. Für diese Klassen scheinen die umgebenden Worte im Deutschen wichtige Indizien für die Zugehörigkeit zur jeweiligen Klasse zu sein. Die Beispiele der Klasse `MISC` haben einen komplett anderen Aufbau. Sie bestehen meist nur aus einzelnen Worten, die oft nicht einmal Substantive sind. Meist kommen diese Entities mit Attributcharakter bezüglich Orten oder Personen in den Testdaten vor (z.B. ‘französische‘). Das könnte auch ein Indiz für die schlechte Performanz bei dieser Klasse sein. Die Entities aus Klasse `ORG` bestehen oft aus mehreren Token. Oft kommen Token vor, die nur Großbuchstaben enthalten.

`LOC`	`MISC`
in [LOC Lanarkshire] county	[MISC Japanese]
in [LOC Prestwick] in	[MISC Conservative]
neighbouring [LOC Czech Republic] in	[MISC Indian]
of [LOC Poland]	[MISC Jewish]
between [LOC Europe]	[MISC Liberian]
of [LOC Izingolweni] on	[MISC World Grand Prix]
to [LOC Poland] from	[MISC Uzbek]
`PER`	`ORG`
striker [PER Georg Weah]	[ORG South African Breweries Ltd]
coach [PER Shu Kamo]	[ORG Homegate Hospitality Inc]
Defender [PER Hassan Abbas]	[ORG First of Michigan Corp.]
manager [PER Clive Lloyd]	[ORG Commerce Department]
[PER M. Waugh]	[ORG CITIC Pacific]
Mrs. [PER Lara]	[ORG Boatmen] ’s
[PER Steve van Vuuren]	[ORG The National Bank of Romania]

Tabelle 5.2: Einige Named Entity Beispiele aus eng.testb.

Die Beobachtungen, die für die Beispiele des deutschen Datensatzes gemacht wurden, lassen sich zu einem großen Teil auf den englischen Datensatz übertragen. Personennamen folgen oft auf Funktionsangaben wie ‘striker‘ oder ‘manager‘ und Ortsangaben sind oft von Präpositionen umgeben. Auch die Beispiele der `MISC`-Klasse haben wieder hauptsächlich Attributcharakter. Jedoch beginnen diese orts-, personen- und organisationsgebundenen Worte fast immer mit Großbuchstaben, was sie innerhalb eines englischen Satz deutlich von den anderen Worten unterscheidet. Für Organisationsnamen gibt es im Englischen viele weit verbreitete Anfangs- und Endworte (z.B. ‘Ltd‘, ‘Inc‘, ‘Corp.‘ u.a.), die im Deutschen nicht so häufig gebraucht werden. Für das Englische bilden diese Worte jedoch gute Indizien dafür, dass es sich um Organisationsentitäten handelt.

Zur Klasse `MISC` kann allgemein noch gesagt werden, dass sie eine recht unscharf definiert ist. Ratinov u. Roth (2009) vernachlässigen sie in ihren Experimenten gänzlich.

Die Performanz auf den englischen Testdaten ist grundsätzlich besser als die Performanz auf den deutschen. Die gleiche Beobachtung wurde auch während des CoNLL-2003 Shared Task gemacht (Tjong Kim Sang u. De Meulder, 2003). Eine Ursache dafür könnte sein, dass die Inhalte des deutschen Datensatzes nicht von Muttersprachlern klassifiziert wurden[1]. Eine Weitere ist in der Natur der deutschen Sprache selbst verankert: die Wortkomposition. Im Deutschen werden mehrere Worte zu einem zusammengefügt, während im Englischen die Worte voneinander getrennt bleiben. D.h. es entfallen viele Kontextinformationen um ein betrachtetes Wort herum. Z.B. beinhaltet das zusammengesetzte Wort ‘Kegelverein‘ weniger Information als das englische Äquivalent ‘ninepins league‘. Dazu kommt, dass geschriebene englische Sprache nur Worte beginnend mit Großbuchstaben enthält, die am Anfang eines Satzes stehen oder Eigennamen (folglich oft Entitäten) sind.

---

[1] Die inkonsistente Klassifikation könnte auch eine Ursache für die Oszillation aus Abbildung 5.3 sein.

Kapitel 6

Fazit

Im Rahmen dieser Bachelorarbeit wurden erfolgreich zwei grundsätzlich unterschiedliche Sequenzklassifikationsverfahren für das Problem der Named Entity Recognition implementiert und in ihrer Performanz gegenübergestellt. Es stellt sich heraus, dass der diskriminativ-lokale Ansatz des Perzeptron-Lernalgorithmus wesentlich bessere Ergebnisse erreicht als der generativ-globale des Hidden Markov Models. Der für den maschinellen Lernprozess benutzte Datensatz stammt dabei aus dem CoNLL-2003 Shared Task. Die Ergebnisse des hier implementierten Perzeptron-Lernalgorithmus sind in Tabelle 6.1 zusammengestellt und wären im Ranking der teilnehmenden Algorithmen des Shared Task auf Platz 12 von 17 für den deutschen und auf Platz 14 von 17 für den englischen Datensatz gelandet (Tjong Kim Sang u. De Meulder, 2003).

Datensatz	**Precision**	**Recall**	**F_1-Measure**
Deutsch	76,14 %	58,21 %	65,98 %
Englisch	79,96 %	76,66 %	78,28 %

Tabelle 6.1: Die Ergebnisse des implementierten Perzeptron-Lernalgorithmus in seiner besten Parameterkonfiguration. Die Ergebnisse der anderen Verfahren sind Abbildung 5.1 zu entnehmen.

Die These aus Kapitel 3.4, in der behauptet wird, dass sich der Perzeptron-Lernalgorithmus besser für NER eignen würde, kann also gestüzt werden. Wie Ratinov u. Roth (2009) behaupten, handele es sich bei NER um eine wissensintensive Aufgabe. Die Einbeziehung einer großen Anzahl von Features, wozu der Perzeptron-Lernalgorithmus bestens geeignet ist, scheint demzufolge eine Schlüsselqualifikation für ein performantes NER-System zu sein. In HMMs ist es sehr schwierig, viele Features mit einzubeziehen, da alles durch Wahrscheinlichkeiten modelliert werden muss. HMMs erweisen sich eher als performant für Probleme, in denen viel Wissen in der Sequenz, d.h. in der Abfolge der Worte, steckt. Ein Beispielproblem, in dem sich HMMs bewährt haben, ist Part Of Speech Tagging (siehe Jurafsky u. Martin, 2008, S. 173 ff.), wobei fast jedes Wort in einer Sequenz einer anderen Klasse angehört als das vorhergehende und das nachfolgende. HMMs funktionieren also besonders gut für Probleme, in denen viele Zustandsübergänge (siehe Kapitel 3.2.1) stattfinden. Beispiel 2.1 lässt erkennen, dass es bei NER oft nur wenige Übergänge gibt. In vielen Fällen kommt es sogar vor, dass alle Worte innerhalb einer Sequenz gar keiner Klasse angehören[1]. Ein entscheidender Teil des essentiellen Wissens der Klassifikationsentscheidungen für NER befindet sich folglich in anderen Quellen, die bereits in Kapitel 2.2 vorgestellt wurden.

Ausblick

Im Folgenden werden Erweiterungsvorschläge für die implementierten Verfahren in Bezug auf NER vorgestellt.

HMM Eine große Schwäche des HMM sind laut Malouf (2002) die Annahmen aus den Gleichungen 3.2 und 3.1. Diese können umgangen werden, indem ein Maximum Entropy Markov Model (siehe Berger u. a., 1996) anstelle eines HMM benutzt wird. Mit diesem Modell kann auch die gleiche Fülle von Features mit einbezogen werden wie im Perzeptron-Lernalgorithmus. Einen weiteren Performanzgewinn könnte man durch eine ausgeklügelte Smoothing-Strategie erhalten (siehe Katz, 1987; Baum, 1972).

Perzeptron Im Kapitel 3.3.4 wurden bereits Erweiterungen für den Perzeptron-Lernalgorithmus vorgestellt, die sich bereits bewährt haben. Der Voted Perceptron-Ansatz (Freund u. Schapire, 1999) führte in der Implementierung von Collins (2002) zu sehr guten Ergebnissen bezüglich Part Of Speech Tagging und Syntactic Chunking. Des Weiteren ist die Hinzunahme von nicht-lokalen[2] und externen Features wie Gazetteers sinnvoll, wie sie von Ratinov u. Roth (2009) beschrieben werden (siehe Kapitel 3.3.4). Nicht lokale Features verbessern die Ergebnisse auf dem englischen CoNLL-2003-Datensatz um durchschnittlich $+1,5\%$ F_1-Measure. Der Einsatz von Gazetteers brachte sogar einen Gewinn von bis zu $+4\%$ F_1-Measure. Die Nutzung eines komplizierteren Dekodierungsverfahrens brachte jedoch kaum einen Performanzgewinn (Ratinov u. Roth, 2009).

---

[1] D.h. es gäbe für die gesamte Sequenz keinen Zustandsübergang.

[2] D.h. über die Satzgrenzen hinaus.

Anhang A

Auszug aus einer Logdatei

```
#class de.aitools.ner.greedy.PerceptronClassifier:FeatureThreshhold=0;NumberOfIterations=100;
FeatureTemplateWindows=word(-2,2) part_of_speech(-1,0) ne_class(-2,-1) stem(-1,0) contains_hyphen(0,0)
is_uppercase(0,0) suffix3(-1,0) prefix3(0,1) is_title(-1,0) prefix2(0,1) suffix2(-1,0);AveragingWeights=true;

training file:./deu.train
development file:./deu.testa
processed 51645 tokens with 4833 phrases; found: 3584 phrases; correct: 2541.
accuracy:  94.20%; precision:  70.90%; recall:  52.58%; FB1:  60.38
              LOC: precision:  68.78%; recall:  63.42%; FB1:  65.99  1089
             MISC: precision:  68.38%; recall:  39.41%; FB1:  50.00  582
              ORG: precision:  73.09%; recall:  44.64%; FB1:  55.43  758
              PER: precision:  72.73%; recall:  59.96%; FB1:  65.73  1155

test file:./deu.testb
processed 52098 tokens with 3673 phrases; found: 2808 phrases; correct: 2138.
accuracy:  95.77%; precision:  76.14%; recall:  58.21%; FB1:  65.98
              LOC: precision:  72.35%; recall:  64.73%; FB1:  68.33  926
             MISC: precision:  64.85%; recall:  39.10%; FB1:  48.79  404
              ORG: precision:  76.67%; recall:  48.90%; FB1:  59.72  493
              PER: precision:  84.06%; recall:  69.29%; FB1:  75.96  985
```

Der Auszug zeigt den Log-Eintrag eines NER-Experiments. Jeder Eintrag besteht aus 20 Zeilen. Die erste Zeile beinhaltet den Klassennamen des verwendeten Algorithmus und die zugewiesenen Parameterwerte. Nach einer Leerzeile folgen die Namen der Trainings- und Development-Menge. In Zeile fünf beginnt der Evaluierungseintrag für die Klassifikationsergebnisse auf der Development-Menge. Daran schließen sich der Name und die Ergebnisse der Testmenge an.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Baum 1972] Baum, L. E.: An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In: *Inequalities III.: Proceedings of the 3rd Symposium on Inequalities*, Academic Press, 1972, S. 1–8

[Baum u. Pietre 1966] Baum, Leonard E. ; Pietre, Ted: Statistical Inference for Probabilistic Functions of Finite State Markov Chains. In: *The Annals of Mathematical Statistics* 37 (1966), Nr. 6, S. 1554–1563

[Berger u. a. 1996] Berger, A. ; Della Pietra, V. ; Della Pietra, S. A.: A Maximum Entropy Approach to Natural Language Processing. In: *Computional Linguistics* Bd. 22, 1996, S. 39–71

[Bikel u. a. 1999] Bikel, Daniel M. ; Schwartz, Richard ; Weischedel, Ralph M.: An Algorithm that Learns What‘s in a Name. In: *Mach. Learn.* 34 (1999), Nr. 1-3, S. 211–231. `http://dx.doi.org/http://dx.doi.org/10.1023/A:1007558221122`. – DOI http://dx.doi.org/10.1023/A:1007558221122. – ISSN 0885–6125

[Brown u. a. 1992] Brown, Peter F. ; deSouza, Peter V. ; Mercer, Robert L. ; Pietra, Vincent J. D. ; Lai, Jenifer C.: Class-based n-gram models of natural language. In: *Comput. Linguist.* 18 (1992), Nr. 4, S. 467–479. – ISSN 0891–2017

[Carreras u. a. 2003] Carreras, Xavier ; Màrquez, Lluís ; Padró, Lluís: Learning a Perceptron-Based Named Entity Chunker via Online Recognition Feedback. In: Daelemans, Walter (Hrsg.) ; Osborne, Miles (Hrsg.): *Proceedings of CoNLL-2003*, Edmonton, Canada, 2003, S. 156–159

[Chieu u. Ng 2003] Chieu, Hai L. ; Ng, Hwee T.: Named Entity Recognition with a Maximum Entropy Approach. In: Daelemans, Walter (Hrsg.) ; Osborne, Miles (Hrsg.): *Proceedings of CoNLL-2003*, Edmonton, Canada, 2003, S. 160–163

[Collins 2002] Collins, Michael: Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2002

[Cowie u. Wilks 1996] Cowie, Jim ; Wilks, Yorick: Information extraction. In: *Communications of the ACM* (1996), Nr. 39

[DeJong 1982] DeJong, Gerald: An overview of the FRUMP system. In: *In Wendy Lehnert and Martin Ringle, editors, Strategies for Natural Language Processing* (1982)

[Florian u. a. 2003] Florian, Radu ; Ittycheriah, Abe ; Jing, Hongyan ; Zhang, Tong: Named Entity Recognition through Classifier Combination. In: Daelemans, Walter (Hrsg.) ; Osborne, Miles (Hrsg.): *Proceedings of CoNLL-2003*, Edmonton, Canada, 2003, S. 168–171

[Freund u. Schapire 1999] Freund, Yoav ; Schapire, Robert E.: Large Margin Classification Using the Perceptron Algorithm. In: *Machine Learning* 37 (1999), Nr. 3, S. 277–296. – ISSN 0885–6125

[Furcy u. Koenig 2005] Furcy, David ; Koenig, Sven: Limited Discrepancy Beam Search. In: *IJCAI*, 2005, S. 125–131

[Good 1953] Good, I. J.: The Population Frequencies of Species and the Estimation of Population Parameters. In: *Biometrika* 40 (1953), S. 16–264

[Hammerton 2003] Hammerton, James: Named Entity Recognition with Long Short-Term Memory. In: Daelemans, Walter (Hrsg.) ; Osborne, Miles (Hrsg.): *Proceedings of CoNLL-2003*, Edmonton, Canada, 2003, S. 172–175

[Jeffreys 1948] Jeffreys, H.: *Theorie of Probability*. 2. Clarendon Press, 1948

[Jurafsky u. Martin 2008] Jurafsky, Daniel ; Martin, James H.: *Speech and Language Processing*. 2. Prentice Hall, 2008. – ISBN 0131873210

[Katz 1987] Katz, S. M.: Estimation of probabilities from sparse data for the language model component of a speech recogniser. In: *IEEE Transactions on Acoustics, Speech and Signal Processing* Bd. 35, 1987, S. 400–401

[Klein u. Taskar 2005] Klein, Dan ; Taskar, Ben: *Max-Margin Methods for NLP: Estimation, Structure, and Applications*. 2005

[Krishnan u. Manning 2006] Krishnan, Vijay ; Manning, Christopher D.: An effective two-stage model for exploiting non-local dependencies in named entity recognition. In: *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics.* Morristown, NJ, USA : Association for Computational Linguistics, 2006, S. 1121–1128

[Kudo u. Matsumoto 2001] Kudo, Taku ; Matsumoto, Yuji: Chunking with support vector machines. In: *NAACL '01: Second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies 2001.* Morristown, NJ, USA : Association for Computational Linguistics, 2001, S. 1–8

[Lafferty u. a. 2001] Lafferty, John ; McCallum, Andrew ; Pereira, Fernando: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: *ICML*, Morgan Kaufmann, 2001, S. 282–289

[Malouf 2002] Malouf, Robert: Markov Models for language-independent named entity recognition. In: *Proceedings of CoNLL-2002*, Taipei, Taiwan, 2002, S. 187–190

[Manning u. Schütze 1999] Manning, Christopher D. ; Schütze, Hinrich: *Foundations of Statistical Natural Language Processing.* 1. The MIT Press, 1999. – ISBN 0262133601

[McCallum 2005] McCallum, Andrew: Information Extraction: Distilling Structured Data from Unstructured Text. In: *ACM Queue* 3 (2005), December, Nr. 9

[Rabiner 1989] Rabiner, Lawrence R.: A tutorial on hidden markov models and selected applications in speech recognition. In: *Proceedings of the IEEE*, 1989, S. 257–286

[Ratinov u. Roth 2009] Ratinov, Lev ; Roth, Dan: Design challenges and misconceptions in named entity recognition. In: *CoNLL '09: Proceedings of the Thirteenth Conference on Computational Natural Language Learning.* Morristown, NJ, USA : Association for Computational Linguistics, 2009. – ISBN 978–1–932432–29–9, S. 147–155

[Rosenblatt 1958] Rosenblatt, Frank: The perceptron: a probabilistic model for information storage and organization in the brain. In: *Psychological review* 65 (1958), November, Nr. 6, S. 386–408

[Shannon 1948] Shannon, Claude E.: A mathematical theory of communication. In: *Bell Systems Technical Journal* 27 (1948), S. 379–423,623–656

[Sundheim u. Chinchor 1993] SUNDHEIM, Beth M. ; CHINCHOR, Nancy A.: Survey of the Message Understanding Conferences. In: *HLT '93: Proceedings of the workshop on Human Language Technology.* Morristown, NJ, USA : Association for Computational Linguistics, 1993. – ISBN 1–55860–324–7, S. 56–60

[Tjong Kim Sang u. De Meulder 2003] TJONG KIM SANG, Erik F. ; DE MEULDER, Fien: Introduction to the CoNLL-2003 shared task: language-independent named entity recognition. In: *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003.* Morristown, NJ, USA : Association for Computational Linguistics, 2003, S. 142–147

[Viterbi 1967] VITERBI, A.J.: Error bounds for convolutional codes and asymptotically optimal decoding algorithm. In: *IEEE Transactions on Information Theory* Bd. IT-13, 1967, S. 260–269