Universität Hamburg
Fachbereich Informatik

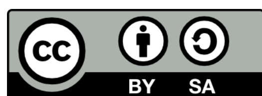
Bachelorarbeit

Inkrementelle Part-of-Speech-Tagger

Arne Köhn

Dezember 2009

Erstbetreuer: Prof. Dr.-Ing. Wolfgang Menzel
Zweitbetreuerin: Dr. Carola Eschenbach



Diese Bachelorarbeit steht unter der Creative-Creative-Commons-Share-Alike-Lizenz.
http://creativecommons.org/licenses/by-sa/3.0/de/

Inhaltsverzeichnis

1. Einführung **5**
1.1. Das Hauptproblem: Umgang mit Ambiguität 5
1.2. Weitere Anforderungen an POS-Tagger 6
1.2.1. Geschwindigkeit 6
1.2.2. Sprachunabhängigkeit 7
1.2.3. Begrenztheit des Trainingskorpus 7
1.2.4. Nachlernbarkeit 7
1.3. Verarbeitungs- und Ausgabemodi 7

2. Part-of-Speech-Tagger: Ein Überblick **9**
2.1. Auf HMM basierende Tagger 9
2.1.1. TnT 11
2.1.2. HunPos 11
2.2. SVMTool 11
2.3. Transformationsbasierte Tagger: Der Brill-Tagger 13
2.3.1. Training 13
2.3.2. Funktionsweise 13

3. Evaluation der Tagger **15**
3.1. Wahl von Evaluationskriterien 15
3.2. POSeval 16
3.2.1. Evaluationsdatensätze 17
3.2.2. Tagger-Wrapper 17
3.2.3. Ergebnisaufbereitung 18
3.3. Traditionelle Evaluation 18
3.3.1. HunPos vs. TnT 18
3.3.2. Optimierung von TnT 19
3.3.3. Verschiedene SVMTool-Versionen 20
3.3.4. Vergleich von HunPos, TnT und SVMTool 21
3.3.5. Der Brill-Tagger 22
3.4. Analyse der Fehler 22
3.4.1. Unbekannte vs. bekannte Worte 22
3.4.2. Fehlerarten 23
3.5. Evaluation auf einem zweiten Korpus 25
3.6. Geschwindigkeit der vorgestellten Tagger 26

4. Inkrementelles Multitagging 29
4.1. Inkrementelles Parsing 30
4.2. Anpassung von HunPos 30
4.2.1. Vorüberlegungen 30
4.2.2. Berechnung der Tag-Wahrscheinlichkeiten 31
4.2.3. Geschwindigkeitsoptimierung 32
4.3. Evaluation 32

5. Ausblick 35

6. Fazit 37

A. Tabellarische Darstellung der Ergebnisse 39

1. Einführung

Part-of-Speech-Tagger sind Programme, die versuchen, für jedes Wort in einem natürlichsprachlichen Satz die Wortart zu ermitteln. Ein Beispiel: In dem Satz „Otto fliegt." hat „Otto" die Wortart *Eigenname*, „fliegt" die Wortart *finites Verb, voll*[1].

Die Wortarten sind die *Tags*, welche an die Wörter *getaggt* (oder *annotiert*) werden. Diese Wortartanalyse ist zum Beispiel für Parser[2] relevant, da diese häufig auf den Ergebnissen von POS-Taggern aufbauen. POS-Tagger gibt es bereits seit über 50 Jahren: Schon 1958 wurde an der University of Pennsylvania ein Parser entwickelt, der einen POS-Tagger enthielt[Joshi and Hopely, 1999].

Diese Arbeit beschäftigt sich mit verschiedenen Ansätzen zum Part-of-Speech-Tagging und versucht diese systematisch zu vergleichen. Hierzu werden in Kapitel 2 verschiedene Ansätze vorgestellt und in Kapitel 3 evaluiert.

Die gängigen POS-Tagger arbeiten satzweise: Sie lesen einen Satz ein und taggen diesen dann. Für die Analyse geschriebener Texte ist dies normalerweise auch unproblematisch, nicht jedoch bei gesprochener Sprache: eine Verarbeitung eines Satzes, bevor der Sprecher diesen zu Ende gesprochen hat, ist damit nicht möglich, wodurch diese Zeit nicht zur Verarbeitung genutzt werden kann.

Deshalb befasst sich das 4. Kapitel dieser Arbeit mit dem inkrementellen Tagging, also dem Taggen der Wörter, sobald diese dem Tagger zur Verfügung stehen. Hierzu wurden verschiedene Tagger so umgebaut, dass sie einen Satz schon taggen, während sie diesen einlesen.

1.1. Das Hauptproblem: Umgang mit Ambiguität

Die offensichtlichste Anforderung an einen POS-Tagger ist, dass er möglichst allen Wörtern den korrekten Tag zuordnen sollte. Was der korrekte Tag ist, kann aber auch für Muttersprachler umstritten sein. Zudem kann die Wahl des korrekten Tags vom Kontext abhängen. Ein Beispiel[3]:

[1] Hier, wie auch in allen Experimenten in dieser Arbeit, wird das Stuttgart-Tübingen-Tagset verwendet. Ein Tagset besteht aus einer Menge von Tags, wobei sich jedes Wort genau einem dieser Tags zuordnen lassen können soll . Für verschiedene Sprachen gibt es unterschiedliche Tagsets, teilweise gibt auch mehr als ein Tagset für eine Sprache. Die Anzahl der Tags kann hierbei stark variieren.

[2] Parser sind im natürlichsprachlichem Kontext Programme, die zu natürlichsprachlichen Sätzen Satzbäume berechnen.

[3] In diesem Beispiel steht hinter jedem Wort mit Schrägstrich getrennt die Wortart. *NE* steht für Eigenname, *VVFIN* für finites Verb, *ART* für unbestimmter Artikel, *CARD* für Kardinalzahl (eins, zwei, drei...) und *NN* für normales Nomen.

Was gibt Hans an Paul?

Hans/NE gibt/VVFIN Paul/NE eine/ART Banane/NN ./.

Hier ist „eine“ ein unbestimmter Artikel. Fragt man jedoch anders:

Wieviele Bananen gibt Hans an Paul?

Hans/NE gibt/VVFIN Paul/NE *eine*/CARD Banane/NN ./.

Durch die geänderte Frage ist der unbestimmte Artikel „eine“ zu einer Kardinalzahl geworden. Da jedoch alle Tagger Satz für Satz vorgehen, ist es für sie unmöglich, diesen Kontext in Betracht zu ziehen.

Der relevante Kontext für die Interpretation eines Satzes kann zudem beliebig groß sein. So kann ein weit vor dem aktuellen Satz geäußerter Sachverhalt relevant sein oder bei gesprochener Sprache Objekte, die in dem Umfeld der Sprecher zu sehen sind.

Dem Tagger unbekannte Wörter - also Wörter, die er nicht beim Training gesehen hat - sind für ihn ebenfalls ambig; der Tagger muss aus den ihm zur Verfügung stehenden Informationen den richtigen Tag raten.

1.2. Weitere Anforderungen an POS-Tagger

Neben der Anforderung, möglichst korrekte Ergebnisse zu liefern, gibt es verschiedene sich zum Teil widersprechende Anforderungen an POS-Tagger.

1.2.1. Geschwindigkeit

Je nach Einsatzgebiet schwankt die Wichtigkeit dieser Anforderung. Sie lässt sich zudem in zwei Teile teilen:

Training Jeder Tagger muss vor dem Einsatz für die Sprache trainiert werden, auf der er später arbeiten soll. Hierfür wird er auf einem Korpus von bereits (normalerweise manuell) annotierten Texten trainiert. Während des Trainings erzeugt der Tagger mit dem Trainingsdatensatz ein Modell für die Zuordnung von POS-Tagfolgen zu Wortformfolgen, welches später beim Taggen benutzt wird.

Die Geschwindigkeit verschiedener Tagger beim Training liegt bei gleichen Trainingsdaten zwischen einigen Sekunden und mehreren Stunden. Normalerweise muss der Tagger jedoch nur einmal trainiert werden, weshalb die Geschwindigkeit beim Tagging für den realen Einsatz wichtiger als die beim Training erachtet werden kann.

Tagging Auch die Geschwindigkeit beim Taggen unterscheidet sich zwischen den Tagging-Ansätzen relativ stark. Zudem muss manchmal bei der Konfiguration eines Taggers zwischen Genauigkeit und Geschwindigkeit abgewogen werden: Manche Optimierungen verbessern zwar das Ergebnis, wirken sich jedoch negativ auf die Geschwindigkeit aus.

1.2.2. Sprachunabhängigkeit

Der Tagger sollte so konzipiert sein, dass er ohne Anpassungen auf verschiedenen Sprachen trainiert werden kann und auf allen annehmbare Ergebnisse liefert. Dies gilt auch für verschiedene Fachsprachen und Sprachvarietäten, wie zum Beispiel Dialekte oder „Chat-Sprache“.

1.2.3. Begrenztheit des Trainingskorpus

Mit POS-Tags annotierte Texte stehen nur sehr begrenzt zur Verfügung. Tagger müssen also in der Lage sein, mit spärlichen Trainingsmengen zu guten Ergebnissen zu kommen.

Zusätzlich wird oft angenommen, dass die Trainingsdaten ein repräsentativer Ausschnitt der zu taggenden Texte sind. Im Deutschen sind jedoch größtenteils auf Zeitungstexten basierende Korpora wie das NEGRA-Korpus verfügbar. Soll der Tagger auch andere Textarten taggen, so ist diese Annahme verletzt. Eine Überanpassung an die Trainingsdaten muss deshalb verhindert werden.

1.2.4. Nachlernbarkeit

„Nachlernbarkeit“ ist die Möglichkeit, einen Tagger auch nach dem initialen Training noch mit weiteren Daten trainieren zu können ohne ein komplett neues Training starten zu müssen. Dies ist insbesondere bei den langsam trainierenden Taggern wichtig. Der einzige mir bekannte Tagger, der diese Möglichkeit bietet, ist der Brill-Tagger (siehe Abschnitt 2.3).

Unterstützt ein Tagger das Nachlernen nicht und soll häufig auf erweiterten Daten nachgelernt werden, so steigt die Wichtigkeit der Trainingsgeschwindigkeit. Wird Nachlernen unterstützt, so ist dann die Nachlerngeschwindigkeit relevant.

1.3. Verarbeitungs- und Ausgabemodi

Zumeist weist ein Tagger jedem Wort genau einen Tag zu. Ich nenne diesen Modus *Unitagging*. Da für die meisten Wörter jedoch mehr als ein Tag in Betracht kommt, können für den Anwender des Taggers auch die nicht ausgewählten Tags eine Relevanz haben, sodass auch diese ausgegeben werden sollen. In diesem Modus (*Multitagging*) werden für jedes Wort die in Frage kommenden Tags, gewichtet nach ihrer Wahrscheinlichkeit, ausgegeben.

Ein Tagger kann unterschiedliche Ausgabemodi haben: Entweder er liest zuerst den gesamten Satz ein und fängt erst danach mit der Ausgabe an oder er arbeitet *inkrementell*, gibt also das Ergebnis für ein Wort aus, sobald er es eingelesen hat. Eine Variation hiervon ist, dass der Tagger eine fixe Anzahl von folgenden Wörtern sehen darf, bevor er die Ausgabe für das aktuelle Wort tätigt. Die Anzahl der miteinbezogenen Wörter nennt man *Lookahead*. Kann der Tagger für das Taggen des aktuellen Wortes die beiden nachfolgenden Worte mit einbeziehen, so sagt man, dass er einen Lookahead von zwei hat.

2. Part-of-Speech-Tagger: Ein Überblick

Die meisten Tagger arbeiten mit verschiedenen stochastischen Modellen des Maschinellen Lernens. Hierzu zählen Hidden-Markov-Modelle [Brants, 2000] und Support Vector Maschines [Giménez and Màrquez, 2004].

Die zweite Klasse von Taggern arbeitet regelbasiert. Neue Ansätze dieser Art wurden in der letzten Zeit meines Wissens jedoch nicht entwickelt.

2.1. Auf HMM basierende Tagger

Ein Markovmodell besteht aus mehreren Zuständen und Übergangswahrscheinlichkeiten zwischen ihnen. Eine Übergangswahrscheinlichkeit von A nach B beschreibt, wie wahrscheinlich es ist, dass der nächste Zustand des Modells B ist, wenn der aktuelle Zustand A ist.

Ein Beispiel: Zu Modellieren ist die Güte des Essens in einer Kantine. Es gibt zwei Zustände: G (das Essen schmeckt gut) und N (es schmeckt nicht gut). Schmeckte das Essen am Vortag gut, so sei die Wahrscheinlichkeit, dass es heute gut schmeckt 70%. Schmeckte es nicht gut, so ist die Wahrscheinlichkeit, dass es heute wieder nicht gut schmeckt 60%.

Diese Modell ist in Abbildung 2.1 grafisch dargestellt.

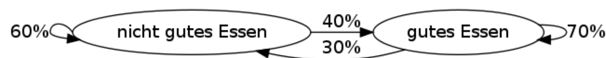


Abbildung 2.1.: Markov-Modell „Kantine“

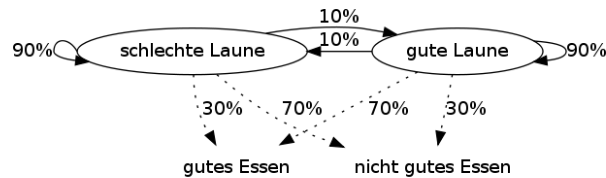
Mit diesem Modell lässt sich zum Beispiel ausrechnen, wie wahrscheinlich die Folge $GGNNG$ ist, wenn der aktuelle Zustand G ist.[1]

Manchmal sind jedoch nicht die Zustände eines Modells beobachtbar, sondern nur Auswirkungen der Zustände. Normale Markov-Modelle können solche Abhängigkeiten nicht darstellen, wohl aber Hidden-Markov-Modelle. Hidden-Markov-Modelle sind Modelle bei denen der Zustand nicht beobachtbar ist, die Beobachtung aber von dem Zustand abhängt.

Ein Beispiel hierzu: Man möchte die Laune des Koches der Kantine modellieren, kann aber nur die Qualität des Essens beobachten. Es ist folgendes bekannt: 1. Hat der Koch schlechte Laune, so beträgt die Wahrscheinlichkeit für nicht gutes Essen

[1] Die Wahrscheinlichkeit hierfür ist $0,7 \cdot 0,7 \cdot 0,3 \cdot 0,6 \cdot 0,4 \approx 0,035$.

90% 2. Ist der Koch gut gelaunt, so beträgt die Wahrscheinlichkeit für gutes Essen 90% 3. Der Koch hat mit 70% Wahrscheinlichkeit die Laune des Vortages. Dieses Beispiel ist in Abbildung 2.2 grafisch dargestellt.



Eingekreist: Die Zustände des Modells; Ohne Umrandung: Die Beobachtungen

Abbildung 2.2.: Hidden-Markov-Modell „Kantinenkoch"

Bei jedem Übergang von Zustand zu Zustand[2] wird genau ein beobachtbares Ereignis emittiert.

Einen weitergehenden Einstig in HMM bietet [Rabiner, 1989].

Bei auf HMM basierenden POS-Taggern sind die Beobachtungen die Wörter und die Zustände die dazugehörigen Tags. Der Tagger versucht also, von den beobachteten Wörtern auf die Tags zurückzuschließen, die diese Wörter emittiert haben.

HMM-Tagger gibt es schon länger, z.B. [Cutting et al., 1992][3]. Der wohl bekannteste auf Hidden-Markov-Modellen basierende Tagger ist der TnT-Tagger[4] von [Brants, 2000].

Die Übergangswahrscheinlichkeit von einem Tag zu einem anderen ergibt sich aus der Folge der Tags im Trainingskorpus. Sie werden daher wie die Emissionswahrscheinlichkeiten $P(w|t)$ direkt aus den Trainingsdaten ermittelt.[5]

So gut wie alle HMM-Tagger arbeiten mit Trigramm-Wahrscheinlichkeiten[6], die Wahrscheinlichkeiten für die Zustandsübergänge hängen also nicht nur von dem aktuellem sondern auch von dem vorherigen Zustand ab.

Für die Berechnung der optimalen Tagfolge beim Taggen nutzen sie eine globale Optimierungsstrategie wie den Viterbi-Algorithmus[7][Viterbi, 1967].

HMM-Tagger haben den Vorteil, dass sie sehr schnell trainierbar sind und schnell taggen. Zu den Geschwindigkeiten der Tagger siehe Abschnitt 3.6.

Kam ein Trigramm in den Trainingsdaten nicht vor, so muss die Wahrscheinlichkeit hierfür aus den Bigramm- ($P(t_{i+1}|t_i)$) oder Unigramm-Wahrscheinlichkeiten ($P(t_{i+1})$)

[2]Hierzu zählt auch der Übergang in denselben Zustand, also von A zu A.

[3]Dieser Tagger ist in Lisp geschrieben und läuft leider nicht mit aktuellen Lisp-Implementationen.

[4]TnT steht für Trigram n' Tags

[5]$P(w|t)$ steht für „Wahrscheinlichkeit von Wort w, gegeben Tag t".

[6]$P(t_{i+1}|t_i, t_{i-1})$, die Wahrscheinlichkeit von dem nächsten Tag (t_{i+1}) hängt von dem aktuellen und dem vorherigen Tag ab. Trigramm bezeichnet hierbei das Dreiertupel aus kommendem, aktuellem und vorherigem Tag.

[7]Der Viterbi-Algorithmus berechnet die wahrscheinlichste Zustandsfolge für eine gegebene Beobachtungsfolge. Betrachtet man die Zustände als Knoten und die Übergänge als Kanten, so ist das Vorgehen identisch zu dem Dijkstra-Algorithmus für kürzeste Pfade. Somit wird nicht mehr der lokal optimale Tag ausgegeben, sondern der, der Teil des globalen Optimums ist.

interpoliert werden. Für eine genauere Beschreibung von Interpolationsverfahren und der Funktionsweise von HMM-Taggern, siehe [Brants, 2000].

2.1.1. TnT

Im Gegensatz zu früheren HMM-Taggern benutzt TnT nicht nur Tag-Trigramme, sondern auch Wortendungen sowie andere Mechanismen, um die Wortart noch nicht gesehener Wörter zu erraten. Hierdurch gelang ein qualitativer Durchbruch bei HMM-basierten Taggern.

Leider ist TnT keine freie Software und der Quellcode ist nicht verfügbar. Aus diesem Grund ist es nicht möglich, die genaue Vorgehensweise des Taggers zu analysieren und ihn zu verändern. [Brants, 2000] beschreibt den Tagger zwar, jedoch nicht detailliert genug, um ihn mit identischem Verhalten nachzubauen.

2.1.2. HunPos

HunPos ist ebenfalls ein HMM-Tagger, der von [Halácsy et al., 2007] in Ocaml implementiert wurde. [Halácsy et al., 2007] haben bei der Implementierung von HunPos versucht, den TnT-Tagger möglichst genau nachzubauen, um ihn dann an besondere Bedürfnisse des Ungarischen anzupassen, wo eine deutlich höhere Anzahl an Tags als im Deutschen oder Englischen benutzt wird.

[Halácsy et al., 2007] haben festgestellt, dass sich der TnT-Tagger zum Beispiel bei der Ratekomponente auf Basis von Suffixen anders verhält als in [Brants, 2000] beschrieben. HunPos ist freie Software und steht unter der GNU Public License.

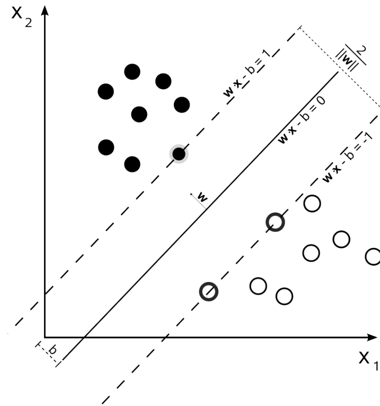
Ich habe diesen Tagger für das inkrementelle Parsing erweitert, sodass er optional keine globale Optimierung mit dem Viterbi-Algorithmus durchführt. Siehe hierzu Abschnitt 4.2.

2.2. SVMTool

SVMTool von [Giménez and Màrquez, 2004] ist ein auf Support Vector Machines basierende Tagger. Er zählt im Moment neben [Toutanova et al., 2003] und [Shen et al., 2007] zu den State-Of-The-Art-Taggern und ist als freie Software unter der LGPL[8] veröffentlicht.

Support Vector Machines ist ein von Vladimir Vapnik und anderen entwickelter Ansatz für das Maschinelle Lernen. Es wird versucht, die zu klassifizierenden Daten (repräsentiert als Vektoren) durch eine Hyperebene zu trennen, wobei beim Training der Abstand der Ebene zu den Datenpunkten maximiert wird (siehe Abbildung 2.3). Da eine solche trennende Hyperebene nicht immer existiert, werden die Vektoren normalerweise mit einer nichtlinearen Transformationsfunktion in einen hochdimensionalen Raum transformiert. Ein Beispiel: Eine Support Vector Machine soll so gebaut werden, dass sie die ganzen Zahlen ($\mathbb{Z}$) in die Kategorien $|a| > 1$ und $|a| \leq 1$ aufteilt. Da die Eingabe ein eindimensionaler Vektor ist, wäre die trennende Hyperebene

[8] Lesser General Public License



Quelle: Wikimedia Commons, Lizenz: Public Domain, abgerufen am 29.10.2009

Abbildung 2.3.: SVM: Maximal separierende Hyperebene

ein Punkt. Mit einem Punkt auf einer Zahlengeraden ist es jedoch offensichtlich nicht möglich, die beiden Mengen voneinander zu trennen; hierfür bräuchte man zwei Punkte ($-1,5$ und $1,5$). Überführt man das Problem jedoch in einen zweidimensionalen Raum, indem der Vektor durch die Funktion $t(x) = \begin{pmatrix} x & x^2 \end{pmatrix}$ in einen zweidimensionalen Vektor umgewandelt und dieser statt x zur Klassifizierung benutzt wird, so ist die Gerade $\vec{v} = \begin{pmatrix} 0 & 2,5 \end{pmatrix} + t \cdot \begin{pmatrix} 1 & 0 \end{pmatrix}$ eine maximal separierende Hyperebene. Sie trennt die Punkte $\begin{pmatrix} -1 & 1 \end{pmatrix}$, $\begin{pmatrix} 0 & 0 \end{pmatrix}$ und $\begin{pmatrix} 1 & 1 \end{pmatrix}$ von denen der anderen Kategorie und ihr Abstand zu $\begin{pmatrix} -1 & 1 \end{pmatrix}$, $\begin{pmatrix} 1 & 1 \end{pmatrix}$, $\begin{pmatrix} 2 & 4 \end{pmatrix}$ und $\begin{pmatrix} -2 & 4 \end{pmatrix}$ ist maximal.

Zudem können Trainingsfehler (also Trainingsdaten, die auf der falschen Seite der Hyperebene liegen) akzeptiert werden. Hierzu wird der sogenannte *c-Wert* angepasst, der die Kosten für diese Verletzungen bestimmt. Support Vector Machines gelten als gut für das Trainieren auf geringen Datenmengen geeignet. Für eine Einführung in Support Vector Machines empfiehlt sich [Vapnik, 1995].

Die zu klassifizierenden Vektoren bestehen bei SVMTool aus binären Features wie „Wort endet auf ung“. Die jeweilige Stelle des Vektors ist 1, wenn das Feature gegeben ist und sonst-1. Da Support Vector Machines nur in zwei Kategorien sortieren können, der Tagger sich aber für eins von vielen Tags entscheiden muss, wird wie folgt vorgegangen: Für jedes Tag wird eine Support Vector Machine trainiert, die darüber entscheidet, ob ein Wort dieses Tag hat. Wird ein Wort getaggt, so bekommt es das Tag der Support Vector Macchine, die den Tag mit der größten Sicherheit vergeben hat, wo also der die Features für das aktuelle Wort beschreibende Vektor den größten Abstand zur trennenden Hyperebene hat.

Da SVMTool eine hohe Anzahl von Features nutzt (und die Vektoren deshalb schon eine hohe Dimension haben), wird auf eine Transformation der Vektoren verzichtet.

SVMTool gestattet es, verschiedene Tagging-Strategien einfach zu implementieren. So können die benutzten Features für das Taggen eines Wortes frei in einer Konfigurationsdatei festgelegt werden. Es ist z.B. möglich, die Tags und Worte der zwei fertig getaggten Worte links sowie die zwei noch ungetaggten Worte rechts des aktuellen

Wortes für die Bestimmung des Tags für das aktuelle Wort zu benutzen.

SVMTool kann mit verschiedenen Strategien benutzt werden. So kann die Richtung des Taggens bestimmt und globale Optimierung verwendet werden.

2.3. Transformationsbasierte Tagger: Der Brill-Tagger

Diese Kategorie von Taggern fängt mit einem initialen Tag für jedes Wort an und wendet daraufhin Transformationsregeln an, um die richtige Tagfolge zu erhalten. In den letzten Jahren wurden keine mir bekannten transformationsbasierten Tagger entwickelt. Aus diesem Grund beschreibe ich hier den Brill-Tagger von 1994 [Brill, 1994].

2.3.1. Training

Das Trainingskorpus wird in zwei gleich große Teile geteilt. Der eine dient dem Erlernen von Regeln für unbekannte Wörter, der andere dem Erlernen der Transformationsregeln.

Die Regeln für unbekannte Worte sind dergestalt, dass sie auf Basis von bestimmten Eigenschaften des Wortes oder nebenstehenden Wörtern den Tag des Wortes verändern können. Die Transformationsregeln ändern den Tag eines Wortes, wenn in dem Kontext des Wortes bestimmte Tags oder Worte auftreten.

Das Training funktioniert folgendermaßen: Aus Regeltemplates werden Regeln erstellt und angewendet. Die Regel, die das Ergebnis am meisten verbessert wird an das Ende der Liste der gelernten Regeln angefügt. Dies wird solange wiederholt, bis es keine oder nur noch eine minimale Verbesserung gibt.

Es werden zuerst die Regeln für die unbekannten Wörter und dann die Transformationsregeln trainiert.

Die Regeln für unbekannte Wörter versuchen, (a) mit Affixen den Tag zu erraten (b) durch Hinzu- oder Wegnahme von Affixen zu einem dem Tagger bereits bekannten Wort zu gelangen (c) durch nebenstehende Wörter den Tag zu erraten (d) dies durch ein Zeichen im Wort zu tun.

Die Transformationsregeln ändern einen Tag von X nach Y, wenn (a) vor oder nach dem jetzigen Wort ein bestimmtes Wort steht oder (b) für ein Wort vor und/oder nach dem jetzigen Wort ein bestimmter Tag vergeben wurde. [9]

2.3.2. Funktionsweise

Bei der Initialisierung erhält jedes bekannte Wort den Tag, den es im Trainingskorpus am häufigsten hatte. Ein unbekanntes kleingeschriebenes Wort erhält „NN“ (für normales Nomen), jedes großgeschriebene „NNP“ (für Eigenname) als Tag. Da im Deutschen alle Nomen groß geschrieben werden, funtkioniert diese Heuristik für unbekannte Wörter nicht. Ich habe den Tagger daher so angepasst, dass er allen unbekannten Wörtern „NN“ zuweist.

[9] Dies ist eine leicht verkürzte Darstellung. Für eine genaue Auflistung, siehe [Brill, 1994].

Nach dieser Initialisierung werden zuerst alle Regeln für unbekannte Wörter in der gelernten Reihenfolge nacheinander angewandt. Daraufhin werden alle gelernten Transformationsregeln nacheinander angewandt.

Der mehrphasige Aufbau des Brill-Taggers ermöglicht es, einen anderen Tagger vor den Brill-Tagger zu schalten. Dieser würde die Initialisierung übernehmen und der Brill-Tagger könnte die Regeln für unbekannte Wörter, die Transformationsregeln oder beide anwenden. Hierfür müsste der Brill-Tagger mit der Ausgabe des anderen Taggers als Initialbelegung trainiert werden.

3. Evaluation der Tagger

Dieses Kapitel beschäftigt sich mit der Evaluation der vorgestellten Tagger. Hierfür werden zuerst die Evaluationskriterien und POSeval, ein Programm zum Evaluieren von Taggern, dargestellt. Anschließend werden sowohl die verschiedenen Tagger als auch verschiedene Versionen des gleichen Taggers miteinander verglichen.

3.1. Wahl von Evaluationskriterien

Oft fällt die Evaluation eines neuen Taggers recht kurz aus. [Cutting et al., 1992] beschreiben z.B. ihre Evaluation der Tag-Leistung wie folgt:

> When using a lexicon and tagset built from the tagged text of the Brown corpus [Francis and Kucera, 1982], training on one half of the corpus (about 500,000 words) and tagging the other, 96% of word instances were assigned the correct tag. Eight iterations of training were used. This level of accuracy is comparable to the best achieved by other taggers [Church, 1988, Merialdo, 1991].

Bei dieser Aussage bleiben verschiedene wichtige Evaluationsaspekte außen vor, die [Megyesi, 2001] erwähnt:

- Auswirkung der Größe der Trainingsdaten auf das Ergebnis
- Robustheit gegenüber unbekannten Worten
- Validierung der Ergebnisse
- Performanz bei nichtenglischen Sprachen

Weder [Brants, 2000] noch [Toutanova et al., 2003] noch [Giménez and Màrquez, 2004] haben ihre Ergebnisse mit mehr als einem anderen Tagger verglichen. Zudem haben die beiden letztgenannten nur ein Testergebnis ohne Validierung angegeben. Durch die meistens nicht standardisierten Test- und Trainingskorpora ist ein nachträglicher direkter Vergleich nicht möglich[1].

Möchte man Tagger nicht nur auf einem Datensatz, sondern auf mehreren verschiedener Größe testen, und diese auch noch jeweils mit mehreren Datensätzen validieren,

[1] [Cutting et al., 1992] ist hier - wie im Zitat ersichtlich - eine Ausnahme. Sie haben in der Dokumentation angegeben, dass sie auf den ungerade nummerierten Sätzen trainiert und auf den anderen getestet haben.

```
 tsets  taggers

No jobs running...

 your taggers
[X] svmt-standard
[ ] tnt-standard
[X] svmt-minimal
[ ] svmt-standard-viterbi
[ ] svmt-minimal-viterbi
[X] hunpos-standard
[ ] svmt-c_values
[ ] svmt-c_values-viterbi

 your tsets
[ ] 1000 - 1
[X] 1000 - 2
[X] 1000 - 12
[X] 1000 - 13
[X] 1000 - 17
[ ] 5000 - 3
[ ] 5000 - 4
[ ] 5000 - 5

train test
```

Abbildung 3.1.: grafische Oberfläche von POSeval

so kommt man schnell zu einer großen Anzahl von Test- und Trainingsdurchläufen. Die Anzahl dieser zur Evaluierung nötigen Trainings- und Testdurchläufe ist linear zur Anzahl der Tagger, der Anzahl der verwendeten Trainingsdatengrößen und der Anzahl der Datensätze zur Validierung[2]. Um diesen Aufwand zu minimieren, wurde POSeval entwickelt.

3.2. POSeval

Um die existierenden Tagger zu evaluieren, habe ich POSeval geschrieben[3], ein Programm zur Verwaltung von Evaluationsdatensätzen, Taggern und Testergebnissen.

Es hat eine grafische Terminaloberfläche (in Abbildung 3.1 zu sehen), weshalb es auch auf Servern ohne grafische Oberfläche laufen kann. Alle Funktionen des Programmes sind über ein Menü erreichbar, wodurch eine benutzerfreundliche Bedienung ermöglicht wird.

Alle Informationen über Evaluationsdatensätze und Tagger werden in einer SQLite3-Datenbank[4] gespeichert. Es wird gespeichert, welche Tagger bereits auf welchen Evaluationsdatensätzen trainiert wurden und welche zudem auf diesen Evaluationsdatensätzen evaluiert wurden. Hierzu wird dann die Auswertung gespeichert. Die Auswer-

[2] Für diese Arbeit wurden über 500 Testkorpora getaggt.

[3] POSeval steht unter der GPLv3 und lässt sich von `http://gitorious.org/poseval` beziehen. Die Dokumentation liegt dem Programm bei.

[4] SQLite (`http://sqlite.org`) ist eine freie Datenbankengine.

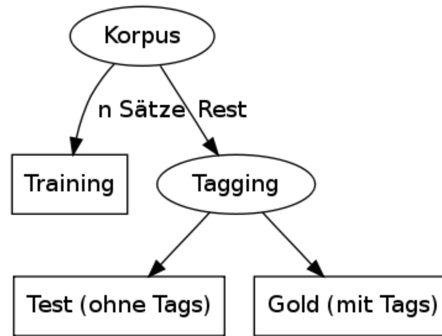


Abbildung 3.2.: Aufteilung des Korpus in Train und Test und Gold

tung übernimmt zur Zeit das Programm SVMTeval, welches Teil von SVMTool ist. Diese Auswertung enthält die Prozentzahl der korrekt zugewiesenen Tags für 1. alle Wörter, 2. die unbekannten Wörter, 3. die bekannten nicht ambigen Wörter und 4. die bekannten ambigen Wörter. Diese Aufteilung ermöglicht eine genauere Analyse der Fehler. POSeval ist in Perl geschrieben.

3.2.1. Evaluationsdatensätze

POSeval ermöglicht das automatische Erstellen von Evaluationsdatensätzen aus einem annotierten Textkorpus. Hierzu wird die für das Training gewünschte Anzahl an Sätzen durch zufälliges Ziehen aus dem Korpus extrahiert. Die nicht für das Training benutzen Sätze werden unannotiert zum Testen und annotiert als Gold-Standard gespeichert (siehe Abbildung 3.2).

Um die in Abschnitt 1.2.3 angesprochene Robustheit gegenüber Sprachvariation zu evaluieren, bietet POSeval die Möglichkeit, die Tagger zusätzlich auf anderen Datensätzen zu testen. Hierfür werden die Tagger mit den oben genannten Trainingsdaten trainiert, aber auf einem von den Trainingsdaten unabhängigen Test-/Gold-Datensatz evaluiert.

3.2.2. Tagger-Wrapper

Für POSeval besteht ein Tagger aus Trainings- und Annotationsprogramm, denen keine Konfigurationsoptionen übergeben werden. Möchte man also verschiedene Konfigurationen z.B. vom SVMTool-Tagger testen, so erstellt man für jede Konfiguration einen eigenen Tagger. Diese sind dann minimale Wrapper um den SVMTool-Tagger. Diese Wrapper sind auch für die Konvertierung der Daten zuständig, insbesondere die Vereinheitlichung der Trennmarker zwischen Wort und Tag. Einige Tagger benutzen hier [TAB], andere [SPACE].

POSeval kann in einer Netzwerkumgebung, in der sich mehrere Rechner ein Dateisystem teilen, Jobs per ssh auf diese Rechner verteilen. Die einzige Voraussetzung hierfür ist, dass bei der ssh-Anmeldung kein Passwort eingegeben werden muss. Auf-

grund der fast vollständigen Unabhängigkeit der zu bearbeitenden Aufgaben skaliert die Leistung linear mit der Anzahl der zur Verfügung stehenden Rechner. Natürlich können auch mehrere Jobs auf einem Rechner parallel laufen.

3.2.3. Ergebnisaufbereitung

Die Ergebnisse können grafisch mit gnuplot aufbereitet werden, wobei Evaluationsdatensätze mit einer gleichen Anzahl von Testsätzen automatisch zusammengefasst und das Maximum, das Minimum und der Durchschnitt angezeigt werden. Hierfür kann man auswählen, welche Tagger in die Aufbereitung einfließen sollen.

3.3. Traditionelle Evaluation

Unter der „traditionellen Evaluation“ verstehe ich die Evaluation der Tagger, wenn sie als Unitagger benutzt werden. Der Tagger emittiert genau einen Tag pro Token (was Wörter, Zahlen, Satzzeichen und ähnliches sein können), die Güte des Taggers wird an der Anzahl der korrekt zugewiesenen Token gemessen:

$$Genauigkeit = \frac{AnzahlKorrektZugewiesenerTags}{AnzahlToken}$$

Zur Evaluation der Trainingsdaten wurde das Negra-Korpus benutzt, welches aus aus 20.602 Sätzen besteht. In diesem Korpus sind die Baumstrukturen der Sätze gespeichert, für das Testen der Tagger werden jedoch nur die POS-Tags benutzt.

Ich habe Evaluationsdatensätze mit unterschiedlich großen Trainingssätzen erstellt, um die Veränderung der Ergebnisse je nach Trainingssatzgröße zu analysieren (Details der Erstellung finden sich in Abschnitt 3.2.1).

Um Ausreißer in den Ergebnissen zu erkennen, wurde mit fünf Datensätzen jeder Größe gearbeitet. Die Grafiken stellen jeweils den Durchschnitt, das beste und das schlechteste Ergebnis zu jeder Trainingssatzgröße dar. Die X-Achse beschreibt die Anzahl der für das Training verwendeten Sätze, die Y-Achse die Prozentzahl der korrekt zugewiesenen Tags. Alle Durchschnittswerte finden sich in tabellarischer Form in Anhang A.

3.3.1. HunPos vs. TnT

Da HunPos wie TnT ein HMM-basierter Tagger ist und versucht, diesen möglichst genau nachzubauen, bietet sich ein Vergleich der beiden Tagger an.

In [Halácsy et al., 2007] vergleichen Péter Halácsy et al. HunPos mit TnT sowohl auf dem WSJ-Korpus (also einem englischen Korpus) als auch auf dem Szeged-Korpus (einem ungarischen Korpus).

Für das WSJ-Korpus geben sie die in Tabelle 3.1 angegeben Werte an. *HunPos 1* ist hierbei der möglichst genaue Nachbau von TnT, *HunPos 2* die Version, in der die Emissionswahrscheinlichkeiten sowohl vom aktuellen als auch vom vorherigen Tag abhängen.

	seen	unseen	overall
TnT	96,77%	85,91%	96,46%
HunPos 1	96,76%	86,90%	96,49%
HunPos 2	96,88%	86,13%	**96,58%**

Tabelle 3.1.: Genauigkeit von HunPos und TnT auf dem WSJ-Korpus nach Halácsy et al.

Wie deutlich zu sehen ist, ist der HunPos-Tagger TnT in beiden Varianten überlegen. Auch auf dem Szeged-Korpus bescheinigen sie HunPos eine ähnliche Güte wie TnT: Während TnT 97,42% der Worte korrekt taggt, sind es bei HunPos 1 97,24% und bei HunPos 2 97,40%.

Bei dem NEGRA-Korpus sieht das Ergebnis anders aus: TnT liefert jederzeit bessere Ergebnisse als HunPos, wobei die TnT-ähnliche Variante genau wie auch auf den anderen Korpora schlechter als die „normale“ Variante von HunPos abschneidet. Wie man in Abbildung 3.3 sieht, nähern sich die Ergebnisse mit steigender Trainingssatzmenge an.

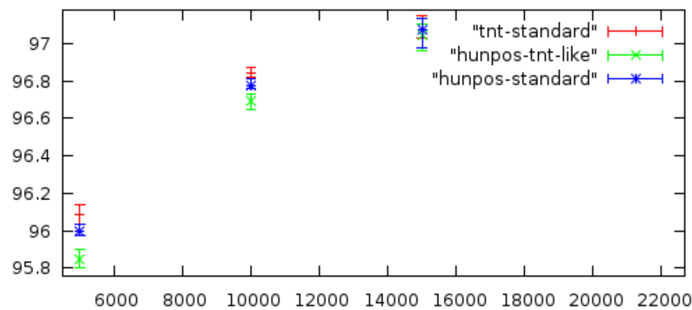


Abbildung 3.3.: HunPos und TnT im Vergleich

3.3.2. Optimierung von TnT

TnT kann neben einem Tag pro Wort auch mehrere Tags gewichtet ausgeben. Um zu überprüfen, wie viel schlechter das Ergebnis ist, wenn von diesen immer der am besten bewertete als einziger Tag genommen wird, habe ich einen kleinen Wrapper geschrieben, der genau dies tut.

Erstaunlicherweise verschlechtert sich das Ergebnis nicht, sondern verbessert sich minimal (durchschnittlich in der zweiten Nachkommastelle).

Auch bei der Nutzung als Multitagger werden die zukünftigen Worte mit in die Berechnung mit einbezogen: Für die gewichtete Ausgabe der Tags benutzt TnT den Backward-Forward-Algorithmus[5]. Sonst wird der Viterbi-Algorithmus benutzt, um

[5]Dieses stammt aus einer E-Mail von Thorsten Brants. In [Brants, 2000] wird es nicht erwähnt.

die beste Tagfolge zu bestimmen. Der Backward-Forward-Algorithmus berechnet die Wahrscheinlichkeit eines Tags T_i, für ein Wort W_s an Stelle s wie folgt:

$$\gamma_s(i) = \frac{\alpha_s(i)\beta_s(i)}{\sum_{i=1}^{N} \alpha_s(i)\beta_s(i)}$$

wobei $\alpha_s(i)$ die Wahrscheinlichkeit für die beobachtete (Wort-)Folge bis zur Stelle s und den Tag T_i (also den Tag für das aktuell bearbeitete Wort) ist. $\beta_s(i)$ ist Wahrscheinlichkeit der beobachteten Wortfolge ab Stelle $s+1$, gegeben den Tag für das aktuelle Wort (was der aktuelle Zustand des Modells ist).

$\alpha_s(i)$ wird also ohne die Festlegung auf bestimmte Tags für andere Worte berechnet. Man muss für alle möglichen Zustände, die das Modell zum Zeitpunkt $s-1$ haben kann, die Wahrscheinlichkeit des Zustandes und die Übergangswahrscheinlichkeit auf den aktuellen Zustand berechnen.

Gleiches, nur in anderer Richtung, gilt für $\beta_s(i)$. Deshalb ist der Berechnungsaufwand höher als bei einer Suche nach dem optimalen Pfad mittels Viterbi. TnT ist deshalb nur ungefähr halb so schnell, wenn mehrere Tags gewichtet ausgegeben werden sollen, als wenn Viterbi eingesetzt wird und nur ein Tag ausgegeben wird, da hier bestimmte unwahrscheinliche Pfade nicht berücksichtigt werden müssen.

3.3.3. Verschiedene SVMTool-Versionen

Wie schon in Abschnitt 2.2 beschrieben, ist SVMTool ein stark konfiguierbarer Tagger. Ich habe drei verschiedene Versionen des Taggers miteinander verglichen: 1. svmt-standard, bei dem die Konfiguration unverändert übernommen wurde, 2. svmt-minimal, bei dem die Menge der benutzten Features reduziert wurde und 3. svmt-c_values, bei dem der „c“-Wert für das Finden der Hyperebene von den Versuchen von [Giménez and Màrquez, 2004] übernommen wurde. So wurden Trainingsfehler zugelassen.

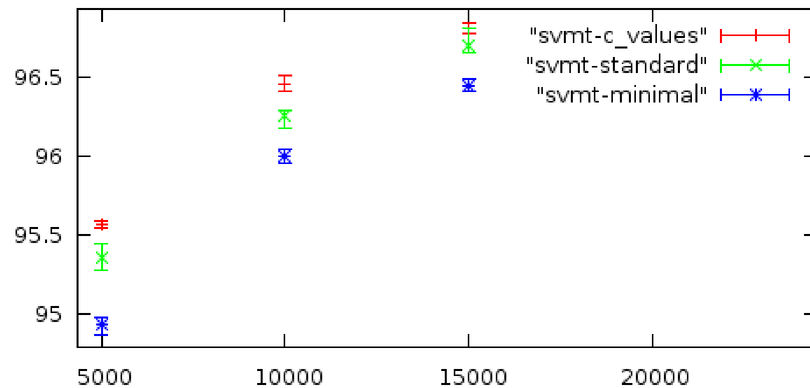


Abbildung 3.4.: Ergebnis der verschiedenen SVMTool-Tagger

Der „c“-Wert wurde mit Absicht nicht von Hand für diese Tests optimiert, um

die Sprachunabhängigkeit des Wertes zu überprüfen. Er wurde auf dem WSJ-Korpus ermittelt.

In Abbildung 3.4 ist deutlich erkennbar, dass sowohl das Benutzen aller zur Verfügung stehenden Features als auch die Nutzung des „c"-Wertes zum Erlauben von Trainingsfehlern eine Verbesserung bringen. Zur globalen Optimierung des Ergebnisses kann der Viterbi-Algorithmus angewandt werden, was zu den in Abbildung 3.5 abgebildeten Ergebnissen führt.

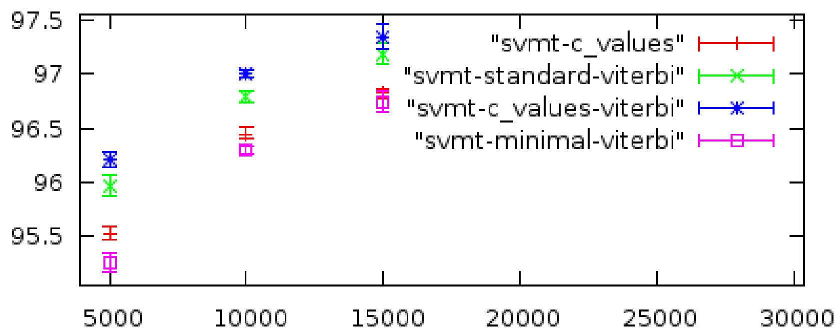


Abbildung 3.5.: SVMTool mit Viterbi, SVMtool ohne Viterbi aber mit c_values zum Vergleich

3.3.4. Vergleich von HunPos, TnT und SVMTool

Nachdem die verschiedenen Versionen der einzelnen Tagger untereinander verglichen wurden, möchte ich hier die jeweils besten Versionen gegenüberstellen.

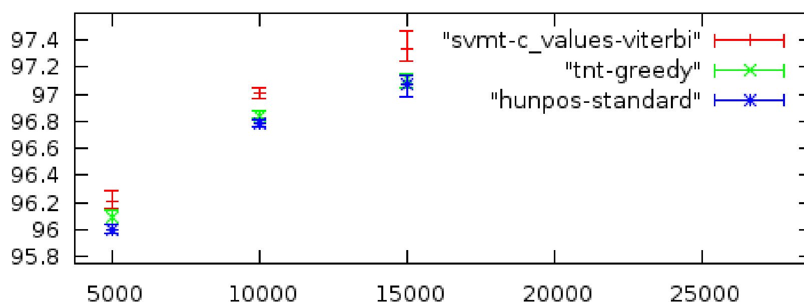


Abbildung 3.6.: Vergleich von HunPos, TnT und SVMTool

In Abbildung 3.6 kann man sehen, dass SVMTool mit Abstand das beste Ergebnis liefert. HunPos und TnT können nicht mithalten.

3.3.5. Der Brill-Tagger

Der Brill-Tagger ist der am Abstand älteste Tagger und zudem strukturell deutlich anders aufgebaut als HunPos, TnT und SVMTool. Aus diesem Grund werden seine Ergebnisse hier gesondert dargestellt. Abbildung 3.7 zeigt die Genauigkeit des Brill-Taggers im Vergleich zu HunPos, dem Tagger mit der niedrigsten Genauigkeit in dem vorherigen Vergleich. Der Brill-Tagger erziehlt deutlich schlechtere Ergebnisse als HunPos.

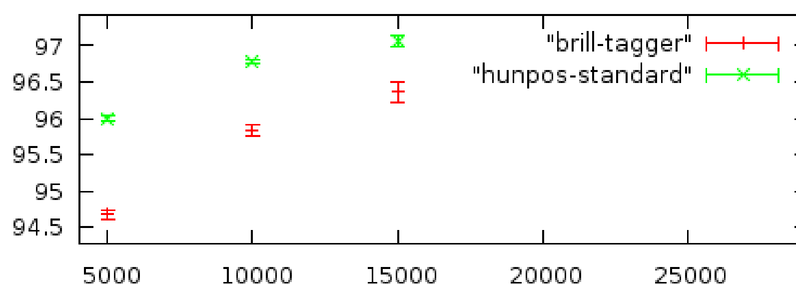


Abbildung 3.7.: Der Brill-Tagger im Vergleich zu HunPos

3.4. Analyse der Fehler

Die vorgestellten Ergebnisse zeigen zwar die Genauigkeit der Tagger, nicht jedoch, wo die einzelnen Tagger Fehler gemacht haben. Für die Entwicklung neuer Tagger und Weiterentwicklung bestehender Tagger ist die Analyse, wo die einzelnen Tagger Fehler gemacht haben, unerlässlich. Aus diesem Grund werde ich hier versuchen, für die verschiedenen Tagger zu analysieren, welcher Art die beobachteten Fehler sind.

Hier stehen mehrere Mittel zur Verfügung: 1. Die Unterteilung des Gesamtergebnisses in ein Ergebnis für dem Tagger bekannte und eins für dem Tagger unbekannte Worte. 2. Die Aufschlüsselung der Fehler nach Fehlerart. Fehlerarten haben die Form „wurde mit X getaggt, hätte aber Y sein sollen oder umgekehrt“.

3.4.1. Unbekannte vs. bekannte Worte

Eine Informationsquelle für die Fehleranalyse ist, wie gut der Tagger unbekannte, also im Trainingsdatensatz nicht vorkommende, Worte taggen kann. Die Güte eines Taggers nur für die unbekannten Worte kann genau wie die allgemeine Güte grafisch dargestellt werden.

Da alle in dieser Arbeit behandelten Tagger Ratekomponenten für unbekannte Worte enthalten, kann ein schlechtes Ergebnis für unbekannte Worte auf Verbesserungspotential in dieser Komponente hinweisen.

Abbildung 3.8 zeigt die Ergebnisse für unbekannte Wörter für SVMTool, TnT und Hunpos. Es ist zu sehen, dass SVMTool und TnT bei kleinen Trainingsdatensätzen

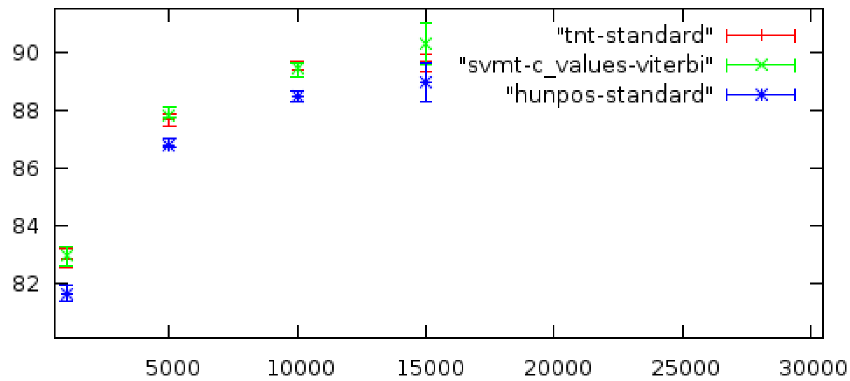


Abbildung 3.8.: Prozentzahl korrekt zugeordneter Tags für unbekannte Wörter

ähnlich gute Ergebnisse liefern, SVMTool jedoch bei einer Trainingsdatengröße von 15000 mehr unbekannten Wörten das richtige Tag zuordnet. Auf dieses Ergebnis werde ich in Abschnitt 3.5 zurückkommen.

3.4.2. Fehlerarten

Die Analyse der Fehlerarten kann ebenfalls helfen, Verbesserungspotentiale aufzudecken. So schreiben [Toutanova et al., 2003], dass ein Namenserkenner deutliche Verbesserungen bringen könnte. Ihre größte Verbesserung hätten sie durch eine Heuristik erlangt, die eine Kette von Worten, die als Normale Nomen getaggt sind und auf die das Wort„Inc." folgt, in Eigennamen umwandelt. Hiermit sollten Firmennamen als Eigennamen erkannt werden.

Die zwei häufigsten Fehlerarten sind in Tabelle 3.2 nach Taggern und Trainingsdatensatzgröße aufgeschlüsselt dargestellt. Die Prozentzahlen sind jeweils der Durchschnitt von fünf Evaluationsdatensätzen. In der Analyse der Fehlerarten sieht man, dass die Unterscheidung zwischen „Normales Nomen" und „Eigenname" das Hauptproblem aller Tagger ist. Ein Namenserkenner kann hier also eventuell das Ergebnis deutlich verbessern.

Wie man sieht, ist der prozentuale Anteil der Fehler für alle getesteten Tagger ähnlich, obwohl sie unterschiedlich arbeiten. Auch der Verlauf der Fehleranteile über die Trainingsdatensatzgrößen ist ähnlich. Die hohe Zahl an „finites⇔infinites Verb"-Fehlern kann darauf zurückzuführen sein, dass im Deutschen der flektierte Anteil im Satz weit von dem infiniten Verb entfernt sein kann. Alle vorgestellten Tagger nutzen für das Taggen eines Wortes Worte (und ihre Tags) mit einem Abstand von maximal drei zu diesem Wort.

	Eigenname ⇔ normales Nomen	infinites Verb ⇔ finites Verb
Trainingsdatensatzgröße 1000		
SVMT-c_values-viterbi	21,91%	7,49%
TnT-standard	23,00%	6,91%
Hunpos-standard	22,30%	6,24%
Brill-Tagger	22,01%	6,47%
Trainingsdatensatzgröße 5000		
SVMT-c_values-viterbi	19,50%	9,94%
TnT-standard	19,96%	9,37%
Hunpos-standard	20,13%	9,09%
Brill-Tagger	20,45%	9,18%
Trainingsdatensatzgröße 10000		
SVMT-c_values-viterbi	17,97%	11,33%
TnT-standard	17,93%	10,27%
Hunpos-standard	18,27%	10,15%
Brill-Tagger	18,57%	9,95%
Trainingsdatensatzgröße 15000		
SVMT-c_values-viterbi	16,81%	11,94%
TnT-standard	17,07%	10,57%
Hunpos-standard	17,04%	10,63%
Brill-Tagger	17,04%	10,43%

Häufigkeit der Fehler Eigenname ⇔ Normales Nomen und Infinites Verb ⇔ Finites Verb in Relation zu der Gesamtfehlermenge

Tabelle 3.2.: Häufigkeit von Fehlerarten in den Ergebnissen der verschiedenen Tagger

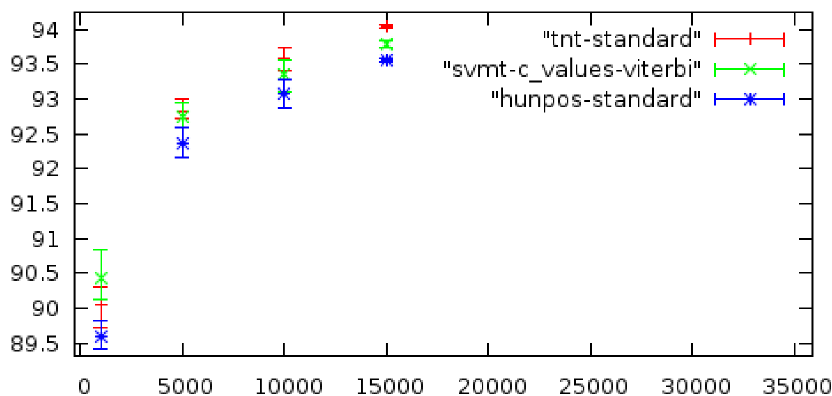


Abbildung 3.9.: Prozentzahl korrekter Tags auf dem Heise-Korpus

3.5. Evaluation auf einem zweiten Korpus

Um das in Abschnitt 1.2.3 angesprochene Problem der Performanz auf Texten zu überprüfen, die andersartig als die zum Training verwendeten sind, habe ich ein zweites Korpus benutzt. Dieses ist eine Sammlung von Texten aus dem Heise-News-Ticker, die im Arbeitsbereich Natürlichsprachliche Systeme am Fachbereich Informatik der Universität Hamburg erstellt wurde. Die Heise-Texte beschäftigen sich mit Themen aus dem Computer-Umfeld und enthalten deshalb eine hohe Anzahl unbekannter Wörter.

Jeder Tagger wurde mit jedem während der in Abschnitt 3.3 beschriebenen Evaluation erstellten Trainingsdatensatz trainiert und dann auf dem gesamten Heise-Korpus (bestehend aus 95267 Sätzen) getestet. Bei dieser Auswertung betrachte ich nicht alle bisher vorgestellten Konfigurationen, sondern zuerst nur je eine pro Tagger.

In Abbildung 3.9 kann man erkennen, dass sich die Ergebnisse deutlich von denen in Abbildung 3.6 unterscheiden: zwar ist bei einer Trainingsdatensatzgröße von 1000 der SVMTool-Tagger noch führend, bei größeren Trainingsdatensätzen sind die Ergebnisse des TnT-Taggers jedoch besser.

Dieses Bild spiegelt sich auch bei den unbekannten Worten wider: in Abbildung 3.10 kann man sehen, dass die prozentuale Anzahl korrekt zugewiesener Tags für unbekannte Wörter bei TnT stärker steigt als bei SVMTool und Hunpos. Dieser Verlauf unterscheidet sich deutlich von dem in Abbildung 3.8 gezeigten. Eventuell hängt das bessere Abschneiden des TnT-Taggers damit zusammen, dass bei der manuellen Annotation des Korpus ein Parser als Hilfsmittel benutzt wurde, welcher TnT benutzt. Der Parser wurde zum initialen Generieren von Satzbäumen benutzt, welche dann von Hand korrigiert wurden. Es kann also vorgekommen sein, dass im Zweifelsfall der von TnT vorgeschlagene Tag übernommen wurde.

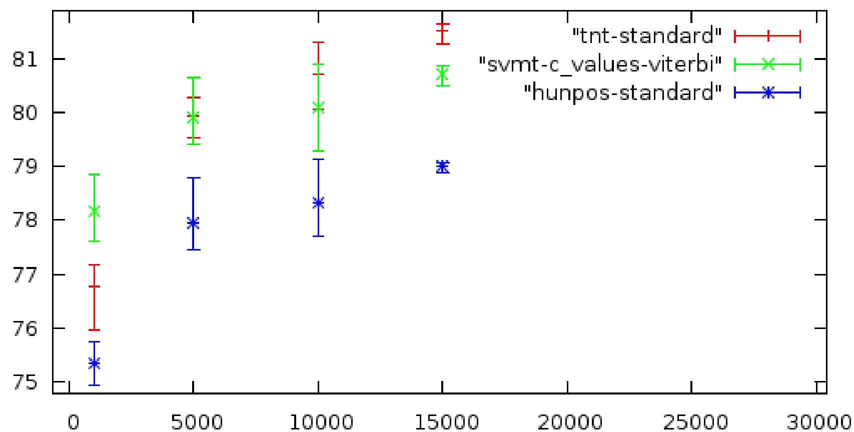


Abbildung 3.10.: Prozentzahl korrekter Tags für unbekannte Wörter auf dem Heise-Korpus

3.6. Geschwindigkeit der vorgestellten Tagger

Zum Schluss der Evaluation sei auf die Geschwindigkeiten der Tagger hingewiesen: Hier gibt es, wie in Abschnitt 1.2.1 angesprochen, zwei zu betrachtende Dimensionen: Die Geschwindigkeit zum Taggen und die zum Trainieren. Tabelle 3.3 enthält eine Aufstellung der benötigten Zeiten. Beim Training müssen sowohl Hunpos als auch TnT nur Vorkommen von Trigrammen und Wortendungen zählen, wohingegen SVMTool und der Brill-Tagger optimieren. SVMTool benutzt Optimierung für das Trainieren von Support Vector Machines und der Brill-Tagger für die Auswahl der besten Transformationsregeln. Diesen Unterschied sieht man bei der benötigten Zeit für das Training sehr deutlich: Der begrenzende Faktor bei Hunpos und TnT beim Training ist die Geschwindigkeit des Laufwerkes, auf dem die Trainingsdaten liegen.

Beim Taggen ist SVMTool deutlich langsamer als die anderen Tagger. Hierzu ist allerdings anzumerken, dass die von mir benutzte Version in Perl geschrieben ist, während die anderen Tagger in kompilierten Sprachen geschrieben sind. Bei dem Brill-Tagger ist das Programm für das Training in Perl geschrieben während das Programm zum Taggen in C geschrieben ist.

SVMT-c_values-viterbi taggt jeden Satz sowohl von Links nach Rechts als auch von Rechts nach Links jeweils mit dem Viterbi-Algorithmus und gibt dann das besser bewertete Ergebnis aus. Würde nur eine Richtung benutzt, wäre er doppelt so schnell.

	Training	Tagging
Hunpos	1s	5s
TnT	1s	1,5s
SVMT-c_values	8min 22s	3min 4s
SVMT-c_values-viterbi	8min 22s	82min
Brill	87min	11s

Jeder Tagger wurde mit 5000 Sätzen trainiert und musste 10000 Sätze taggen. SVMT-c_values-viterbi arbeitet auf dem gleichen Modell wie SVMT-c_values, weshalb die Trainingszeit identisch ist.

Tabelle 3.3.: Geschwindigkeit der vorgestellten Tagger

4. Inkrementelles Multitagging

Dieser Abschnitt befasst sich mit der Frage, wie Tagger angepasst werden müssen, wenn sie als inkrementell arbeitende Multitagger eingesetzt werden. Der *Constraint-Dependency-Grammar Parser* CDG nutzt einen Tagger (namentlich TnT) als Prädiktor. Die Wahrscheinlichkeit, dass ein Wort einen bestimmten Tag hat, wird von dem Tagger ermittelt, an CDG übergeben und dann von diesem in ein POS-Constraint-Gewicht umgewandelt. Eine Satzinterpretation wird also von den POS-Constraints stärker bestraft, wenn die Wortarten der einzelnen Worte vom Tagger für unwahrscheinlich gehalten werden. CDG soll in diesem Kapitel als Beispiel für eine einen Tagger nutzende Anwendung dienen.

Tagger arbeiten wie schon erwähnt meist anderen Komponenten zu. In diesem Abschnitt geht es darum, einen Tagger so anzupassen, dass er inkrementell und als Multitagger arbeitet. Von den behandelten Taggern kann keiner inkrementell (also Wort für Wort) arbeiten. Wird ein Multitagger benötigt, so wird meistens auf TnT zurückgegriffen, da nur er mit Wahrscheinlichkeiten gewichtete Tags ausgeben kann.

Der TnT-Tagger hat allerdings zwei Nachteile:

- TnT ist keine freie Software, weshalb keine Software mit TnT verbreitet werden darf. Stattdessen muss jeder Nutzer beim Autor eine persönliche Nutzungserlaubnis einholen, um TnT zu erhalten. Zudem darf TnT nur für die Forschung eingesetzt werden, was die Nutzbarkeit auch von CDG weiter einschränkt.
- TnT ist nur in Binärform erhältlich, wodurch er nicht an spezielle Bedürfnisse angepasst werden kann.

Ein Multitagger muss in der Lage sein, für jedes Wort die möglichen Tags nach Wahrscheinlichkeit zu gewichten. HMM-Tagger haben hier einen Vorteil, da sie intern mit Wahrscheinlichkeiten rechnen und damit auch Wahrscheinlichkeiten ausgeben können.

Eine Optimierung z.B. mit dem Viterbi-Algorithmus nach dem besten Pfad ist für Multitagger nicht mehr möglich, da dann das Ergebnis genau ein Tag pro Wort wäre.

Der Brill-Tagger hingegen ist aufgrund seiner Konstruktion nicht in der Lage, mehr als einen Tag pro Wort auszugeben. Er ist also für diesen Einsatzbereich nicht geeignet.[1]

SVM-basierte Tagger können mehrere Tags gewichtet pro Wort ausgeben. Jedoch arbeiten sie nicht mit Wahrscheinlichkeiten sondern mit Abständen zu der trennenden Hyperebene, weshalb die Ergebnisse für die Weiterverwendung normiert werden müssen.

[1] [Brill, 1994] beschreibt eine Möglichkeit, einem Wort mehrere mögliche Tags zuzuweisen, allerdings sind diese nicht gewichtet.

4.1. Inkrementelles Parsing

Niels Beuck [Beuck, 2009] hat in seiner Diplomarbeit den CDG-Parser um die Möglichkeit des eingabeinkrementellen Parsings[2] erweitert. Hierbei berechnet der Parser schon während der Satzeingabe mögliche Satzbäume.

Der TnT-Tagger wird für jedes neue Wort mit dem bisher vorhandenen Satzfragment neu aufgerufen. Hierdurch können sich die Bewertungen schon vergebener Tags im Nachhinein ändern, was den Parsing-Aufwand deutlich erhöht.

Wünschenswert wäre also ein Tagger, der sich nicht im Nachhinein bei schon vergebenen Tags umentscheidet. TnT kann dies nicht leisten, weshalb ein anderer Tagger benutzt werden muss.

Bei inkrementellem Parsing hat der Parser (und somit auch der Tagger) keine Möglichkeit, in die Zukunft zu blicken. Der Tagger kann also für seine Analyse nicht auf folgende Wörter zurückgreifen, was die Genauigkeit des Ergebnisses verschlechtern kann. Man kann dieses Problem umgehen, indem die Verarbeitung für ein Wort erst gestartet wird, wenn eine bestimmte Anzahl weiterer Wörter eingelesen wurde. So ist ein konstanter Lookahead für den Tagger realisierbar. Es gilt herauszufinden, inwieweit der Tagger (und der Parser) davon profitieren kann.

4.2. Anpassung von HunPos

Um die Güte eines HMM-basierten Taggers ohne globale Optimierung und damit ohne Lookahead zu testen, habe ich HunPos um die Fähigkeit des inkrementellen Taggings erweitert.

4.2.1. Vorüberlegungen

Definition Eine *Tagfolge* ist eine Liste von Tags. Ist T eine Tagfolge, so ist T[i] der i-te Tag, also das Tag, der in dieser Tagfolge dem i-ten Wort zugeordnet wurde.

Der Tagger arbeitet als Multitagger, gibt also potentiell mehrere Tags pro Wort aus. HMM-Tagger benutzen jedoch für die Berechnung des aktuellen Tags T_i nur einen Tag pro vorherigem Wort. Es gibt zwei Möglichkeiten, mit dieser Diskrepanz umzugehen:

a) Man nimmt für jedes verarbeitete Wort das wahrscheinlichste Tag als das richtigen an.

b) Man entscheidet sich nicht für einen Tag, sondern berechnet für jede mögliche Tagfolge der Länge $i-1$ die Wahrscheinlichkeit der einzelnen Tags und gewichtet diese dann mit der Wahrscheinlichkeit der Tagfolge.

[2] eingabeinkrementelles Parsing bedeutet, dass mit der *Verarbeitung* schon während der Eingabe angefangen wird. Dies steht im Gegensatz zu dem von mir beschriebenem inkrementellem Tagging, wo die *Ausgabe* schon während der Eingabe stattfindet.

Bei Möglichkeit a würde der interne Zustand des Taggers nicht mit seiner Ausgabe übereinstimmen. Da der Tagger als Vorverarbeiter für ein anderes Programm arbeiten soll und dieses gerade nicht möchte, dass ihm der Tagger die Entscheidung abnimmt, habe ich mich dazu entschlossen, dass sich der Tagger auch intern nie für ein Tag für ein Wort entscheiden soll.

Der Tagger taggt deshalb das aktuelle Wort nicht auf Basis einer bestimmten Tagfolge, sondern auf der Basis aller bisher möglichen Tagfolgen, gewichtet nach ihren Wahrscheinlichkeiten.

4.2.2. Berechnung der Tag-Wahrscheinlichkeiten

Der interne Zustand eines Taggers zu einem bestimmmten Zeitpunkt besteht aus den bisher gesehenen Wörtern und den bisher vergebenen Tags. Da die Auswahl eines Tags für ein Wort nicht eindeutig ist, kann es für eine bestimmte Wortfolge verschiedene Tagfolgen geben.

Eine Tagfolge entsteht durch das Anfügen eines neuen Tags an eine bestehende Tagfolge bei der Verarbeitung eines Wortes. Ist eine Tagfolge S auf diese Weise aus der Tagfolge R entstanden, so sei S ein *Nachfolger* von R. Die *Nachfolgerrelation* sei die Relation $\{(P,Q)|Q \text{ ist Nachfolger von } P\}$. Betrachtet man nun die möglichen während der Verarbeitung eines Satzes entstandenen Tagfolgen, so sieht man, dass diese mit der Nachfolgerrelation einen Baum bilden:

Sei T eine Tagfolge. Sei $len(T)$ die Länge der Tagfolge, also die Anzahl der Tags in dieser Folge, dann gilt:

- Von einer Tagfolge T führt ein Übergang nur zu Tagfolgen der Länge $len(T)+1$
  $\rightarrow$ Der Übergangsgraph ist zyklenfrei.

- Es gibt keine Tagfolge T_x, die Nachfolger von zwei Tagfolgen T_1 und T_2, $T_1 \neq T_2$, ist. T_1 und T_2 müssten Präfixe der Länge $len(T_x)-1$ von T_x sein, weswegen sie keine unterschiedlichen Tagfolgen sein können.
  $\rightarrow$ Der Übergangsgraph ist ein Baum, wenn man die leere Tagfolge als Wurzel annimmt.

Es ist also möglich, die Pfadregeln für einen Entscheidungsbaum zu nutzen, um die Wahrscheinlichkeit eines bestimmten Tags für ein Wort zu ermitteln:

$$P(Tag(Wort_i) = t) = \sum_{j}^{Tagfolgen} P(j) \cdot I(j[i] = t)$$

Dabei ist $Wort_i$ das i-te Wort, t das Tag, $P(j)$ die Wahrscheinlichkeit der Tagfolge j und I die Indikatorfunktion.

Mit anderen Worten: Die Wahrscheinlichkeit, dass das i-te Wort den Tag t erhält, ist die Summe der Wahrscheinlichkeiten der Tagfolgen, bei denen an der i-ten Stelle der Tag t steht.

Der	Jäger	sah	den	Mann
[s] [s] ART	[s] ART NE [s] ART NN	ART NE VVFIN ART NN VVFIN	NE VVFIN ART NN VVFIN ART	VVFIN ART NN

Ein Ausschnitt möglicher Tagfolgen für den Satz „Der Jäger sah den Mann“. Durch zwei verschiedene Tags für „Jäger“ entsteht eine zweite Tagfolge, die bei „Mann“ wieder mit der ersten verschmilzt. „[s]“ ist ein Pseudotag für den Bereich vor Satzanfang.

Tabelle 4.1.: Beispiel: Zusammenfassug von Tagfolgen

4.2.3. Geschwindigkeitsoptimierung

Dieses Vorgehen führt bei langen Sätzen zu einer hohen Anzahl von möglichen Tagfolgen, da die Anzahl der Tagfolgen exponentiell mit der Länge des Satzes steigt. In der Praxis führt dies dazu, dass dieser naive Ansatz für das Taggen von 500 Sätzen mehrere Tage benötigt.[3] Die HunPos-Standardversion braucht hierfür hingegen nur wenige Sekunden.

Berücksichtigt man, dass HunPos (wie andere HMM-Tagger auch) für die Bewertung der Wahrscheinlichkeit eines Tags nur die zwei vorhergehenden Tags (und eventuell auch Wörter) berücksichtigt, so kann die Menge der zu berücksichtigenden Tagfolgen stark reduziert werden: Zur Berechnung der Wahrscheinlichkeit eines Tags für das aktuelle Wort werden nur Tagfolgensuffixe konstanter Länge benötigt. Das Ergebnis des Tag-Vorganges ändert sich also nicht, wenn nach jedem Taggen eines Wortes die Tagfolgen auf die Länge zwei gekürzt werden. Danach können alle gleichen Tagfolgen zusammengefasst werden, indem die Wahrscheinlichkeiten der gleichen Tagfolgen addiert werden.

Werden nach jeder Verarbeitung eines Wortes alle möglichen Tagfolgen auf die zwei zuletzt vergebenen Tags gekürzt und dann alle gleichen Tagfolgen vereinigt, indem ihre Wahrscheinlichkeiten addiert werden, so hat man eine asymptotisch konstante Anzahl möglicher Tagfolgen ohne einen Informationsverlust. Beispielhaft ist dies in Tabelle 4.1 dargestellt.

Durch dieses Vorgehen benötigt HunPos-incremental nur noch 10 Sekunden für das Taggen von 10000 Sätzen.

4.3. Evaluation

Dieser Abschnitt beschäftigt sich mit der Frage, welche Genauigkeit von inkrementellen Taggern erzielt werden kann. Hierzu werden verschiedene Tagger ohne Lookahead und mit fixem Lookahead verglichen.

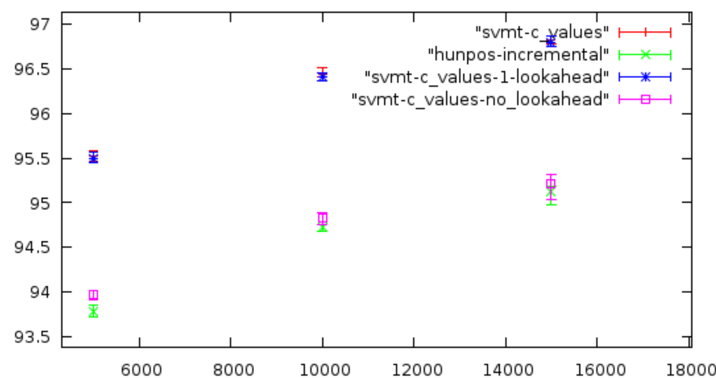
Die Tagger werden wie zuvor auch als Unitagger getestet.

Tagger ohne Lookahead In diese Kategorie fallen zwei Tagger: Der von mir angepasste HunPos-Tagger („hunpos-incremental“) und SVMTool, so konfiguriert, dass

[3] Wie lange der Tagger genau benötigen würde, ist mir nicht bekannt, da der Versuch abgebrochen wurde.

er kein Lookahead verwendet („svmt-c_values-no_lookahead“). Sie entscheiden also alleine auf Basis der vorherigen Worte und Ergebnisse.

Tagger mit fixem Lookahead In diese Kategorie fallen die SVM-Tagger mit fixem Lookahead. Der Tagger mit einem Lookahead von eins heißt „svmt-c_values-1-lookahead“, der Tagger mit einem Lookahead von zwei „svmt-c_values“, da dies die Standardeinstellung von SVMTool ist.



X-Achse: Anzahl der Trainingssätze, Y-Achse: Prozentzahl der korrekten Tags

Abbildung 4.1.: inkrementelle Tagger

Wie man in Abbildung 4.1 sehen kann, gibt es einen deutlichen Unterschied zwischen den Taggern ohne Lookahead und jenen mit. Der Unterschied zwischen einem Lookahead von eins und einem Lookahead von zwei ist jedoch nur marginal.

Der Unterschied zwischen dem HunPos-Tagger und dem SVMTool-Tagger ohne Lookahead ist ebenfalls gering. Nutzt man keinen Lookahead, so kann man also HunPos einsetzen und dadurch von der deutlich höheren Geschwindigkeit profitieren.

5. Ausblick

Im Moment gibt es keinen mir bekannten Tagger, der nichtlokale Abhängigkeiten berücksichtigt. Hierzu zählt zum Beispiel die Satzklammer im Deutschen. Eine Berücksichtigung dieser Effekte einzubauen, könnte die in Abschnitt 3.4.2 angesprochene Fehlerrate bei der Unterscheidung zwischen finiten und infiniten Verben verringern. Zudem ist zu evaluieren, inwiefern sich die Genauigkeit mit Hilfe eines Namenserkenners steigern lässt.

Um Tagger bewerten zu können, benötigt man Evaluationskriterien. Fast immer (auch in dieser Arbeit) werden hierfür Sätze mit genau einem Tag pro Wort getaggt und dann die prozentuale Anzahl der richtigen Tags als Gütemaß genommen, siehe [Giménez and Màrquez, 2004, Toutanova et al., 2003, Halácsy et al., 2007, Cutting et al., 1992].

Benutzt man einen Tagger als Unitagger, so ist dieses Verfahren genau richtig. Wird der Tagger jedoch als Multitagger in eine größere Anwendung eingebettet, so wird eventuell eine zu stark vereinfachende Annahme getroffen. Es kann sein, dass für die Anwendung nicht nur der wahrscheinlichste Tag relevant ist, sondern auch die Reihenfolge und/oder Gewichtung sämtlicher als möglich ausgegebener Tags.

Da die Güte des Ergebnisses der Anwendung, die den Tagger nutzt, eventuell unterschiedlich stark von Reihenfolge und Bewertung der Tags abhängig ist, muss für eine verlässliche Aussage der Güte eines Taggers in diesem Verarbeitungsumfeld das Ergebnis dieser Anwendung als Kriterium für die Güte des Taggers benutzt werden. Bei der Einbettung eines Taggers in einen Parser muss also überprüft werden, wie sich das Parsing-Ergebnis je nach Wahl des Taggers ändert. Für CDG ist hier neben der Güte des Ergebnisses auch die Geschwindigkeit des Parsens interessant: Wird ein inkrementeller Tagger benutzt, sodass sich die einmal vergebenen Tag-Gewichte im Verlauf des Parsens nicht ändern, könnte sich dies deutlich positiv auf die Geschwindigkeit des Parsers auswirken. Die Auswirkungen von inkrementellen Taggern auf eingabeinkrementelles Parsing hat [Beuck, 2009] genauer beschrieben.

6. Fazit

In dieser Bachelorarbeit wurde zuerst ein Überblick über verschiedene Tagger gegeben und ihre Vorgehensweise dargestellt. Alle neuen Tagger arbeiten mit probabilistischen Methoden des maschinellen Lernens. Danach wurde das Programm POSeval zur Evaluation von POS-Taggern vorgestellt, mit dem die vorgestellten Tagger in verschiedenen Konfigurationen evaluiert wurden. Es hat sich gezeigt, dass der SVMTool-Tagger eine höhere Genauigkeit als die anderen Tagger erzielt. HunPos liefert auf deutschen Texten schlechtere Ergebnisse als TnT, obwohl dies auf englischen und ungarischen Korpora umgekehrt ist. Der Brill-Tagger erwies sich als den anderen Taggern deutlich unterlegen.

Bei der Analyse der Fehler hat sich gezeigt, dass der TnT-Tagger im Verhältnis zu der Gesamtgenauigkeit überdurchschnittlich viele unbekannte Wörter korrekt klassifiziert. Die Genauigkeit liegt bei den unbekannten Wörtern teilweise gleichauf mit der des SVMTool-Taggers. Außerdem ist zu beobachten, dass alle Tagger ähnliche Fehler machen: Die meisten Fehler sind Verwechselungen von „Eigenname“ und „Normales Nomen“, gefolgt von der Verwechslung von „Finites Verb“ und „infinites Verb (voll)“. Bei der Geschwindigkeit der Tagger sind die auf Hidden Markov Modellen basierenden Tagger um ein vielfaches schneller als die anderen Tagger. Zudem ist die Genauigkeit von TnT auf dem für die Analyse der Robustheit gegenüber Sprachvariätäten herangezogenen Heise-Korpus höher als die der anderen getesteten Tagger.

Die Anpassung des SVMTool-Taggers und HunPos an inkrementelles Tagging hat gezeigt, dass bei einem Vorgehen ohne Lookahead das Benutzen von Support Vector Machines keinen Vorteil gegenüber Hidden Markov Modellen bringt. Ein Lookahead bringt Verbesserungen bei der Genauigkeit des SVMTool-Taggers, allerdings ist der Unterschied zwischen einem und zwei Wörtern Lookahead vernachlässigbar.

A. Tabellarische Darstellung der Ergebnisse

In den folgenden Tabellen sind die durchschnittlichen Genauigkeiten der Tagger bei verschiedenen Trainingssatzgrößen eingetragen.

Tagger	1000	5000	10000	15000
brill-tagger	89.891	94.717	95.864	96.343
hunpos-incremental	90.386	93.819	94.738	95.092
hunpos-standard	92.958	96.006	96.789	97.038
hunpos-tnt-like	92.703	95.864	96.693	97.014
svmt-c_values	92.471	95.571	96.455	96.812
svmt-c_values-1-lookahead	92.293	95.521	96.418	96.777
svmt-c_values-no_lookahead	90.852	93.980	94.811	95.165
svmt-c_values-viterbi	93.211	96.238	97.012	97.294
svmt-minimal	91.067	94.935	96.000	96.443
svmt-minimal-viterbi	91.433	95.314	96.297	96.719
svmt-standard	92.309	95.361	96.259	96.705
svmt-standard-viterbi	92.736	96.006	96.801	97.141
tnt-greedy	93.288	96.117	96.837	97.093
tnt-standard	93.273	96.114	96.835	97.082

Tabelle A.1.: durchschnittliche Genauigkeit der Tagger auf dem Negra-Korpus

Tagger	1000	5000	10000	15000
brill-tagger	73.255	81.825	84.179	85.049
hunpos-incremental	76.930	82.614	84.581	85.004
hunpos-standard	81.705	86.850	88.549	88.787
hunpos-tnt-like	81.026	86.575	88.570	88.958
svmt-c_values	81.900	87.046	88.562	89.238
svmt-c_values-1-lookahead	81.496	86.756	88.326	88.869
svmt-c_values-no_lookahead	78.718	84.060	85.874	86.016
svmt-c_values-viterbi	82.944	87.916	89.511	90.012
svmt-minimal	76.721	83.149	84.998	85.606
svmt-minimal-viterbi	77.084	83.704	85.542	86.345
svmt-standard	80.795	85.996	87.654	88.602
svmt-standard-viterbi	81.757	87.225	88.847	89.476
tnt-greedy	82.847	87.797	89.338	89.603
tnt-standard	82.849	87.861	89.418	89.590

Tabelle A.2.: durchschnittliche Genauigkeit für unbekannte Worte der Tagger auf dem Negra-Korpus

Tagger	1000	5000	10000	15000
hunpos-standard	89.595	92.359	93.070	93.565
svmt-c_values-viterbi	90.420	92.750	93.350	93.800
tnt-standard	90.058	92.832	93.580	94.057

Tabelle A.3.: durchschnittliche Genauigkeit der Tagger auf dem Heise-Korpus

Tagger	1000	5000	10000	15000
hunpos-standard	75.344	77.952	78.318	79.001
svmt-c_values-viterbi	78.155	79.910	80.096	80.713
tnt-standard	76.769	79.934	80.719	81.524

Tabelle A.4.: durchschnittliche Genauigkeit der Tagger für unbekannte Worte auf dem Heise-Korpus

Literaturverzeichnis

[DBL, 2007] (2007). *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic.* The Association for Computer Linguistics.

[Beuck, 2009] Beuck, N. (2009). Inkrementelle Analyse mit Constraint-Dependency-Grammatiken. Diplomarbeit, Universität Hamburg, Fachbereich Informatik.

[Brants, 2000] Brants, T. (2000). TnT – a statistical part-of-speech tagger. In *ANLP*, pages 224–231.

[Brill, 1994] Brill, E. (1994). Some advances in transformation-based part of speech tagging. In *In Proceedings of the twelfth national conference on artificial intelligence*, pages 722–727.

[Cutting et al., 1992] Cutting, D. R., Kupiec, J., Pedersen, J. O., and Sibun, P. (1992). A practical part-of-speech tagger. In *ANLP*, pages 133–140.

[Giménez and Màrquez, 2004] Giménez, J. and Màrquez, L. (2004). Svmtool: A general pos tagger generator based on support vector machines. In *Proceedings of the 4th LREC*.

[Halácsy et al., 2007] Halácsy, P., Kornai, A., and Oravecz, C. (2007). Poster paper: Hunpos - an open source trigram tagger. In [DBL, 2007].

[Joshi and Hopely, 1999] Joshi, A. K. and Hopely, P. (1999). A parser from antiquity. In Kornai, A., editor, *Extended Finite State Models of Language*, pages 6–15. Cambridge University Press, Cambridge.

[Megyesi, 2001] Megyesi, B. (2001). Comparing data-driven learning algorithms for pos tagging of swedish.

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

[Shen et al., 2007] Shen, L., Satta, G., and Joshi, A. K. (2007). Guided learning for bidirectional sequence classification. In [DBL, 2007].

[Toutanova et al., 2003] Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL*.

[Vapnik, 1995] Vapnik, V. N. (1995). *The nature of statistical learning theory.* Springer-Verlag New York, Inc., New York, NY, USA.

[Viterbi, 1967] Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269.

Ich versichere, dass ich die Arbeit selbst verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Diese Arbeit wurde in keinem anderen Prüfungsverfahren eingereicht. Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereichs Informatik einverstanden. Dies gilt ausdrücklich auch für die digitale Version dieser Arbeit.