Transformation-Based Error-Driven Learning: Eine Fallstudie in Part of Speech Tagging

Malte Helmert

23. Februar 2000

Seminar: Machine Learning, Data Mining and Applications
Abteilung für maschinelles Lernen und natürlichsprachliche Systeme
Institut für Informatik
Universität Freiburg
Wintersemester 1999/2000

1 Übersicht

Diese Seminararbeit befaßt sich mit *transformation-based error-driven learning*, einer allgemeinen Methode im Bereich der korpus-basierten Sprachverarbeitung, und ihrer Anwendung im Bereich des *part of speech tagging*. Die Arbeit beruht im Wesentlichen auf einem Artikel von Eric Brill [1].

In den folgendenen beiden Abschnitten werden zunächst das Problem des part of speech tagging sowie die allgemeine Technik des transformation-based error-driven learning vorgestellt.

Darauf folgen zwei Algorithmen zur Lösung dieses Problems mithilfe der vorgestellten Technik, zunächst eine nicht lexikalisierte Variante, bei der die benutzten Transformationen sich ausschließlich auf die Tags des Textkörpers beziehen, danach eine lexikalisierte Variante, bei der auch Transformationen zugelassen sind, die auf spezielle Wörter des Lexikons Bezug nehmen. Anschließend wird auf das Problem unbekannter Wörter eingegangen.

Im abschließenden Teil werden die vorgestellten Algorithmen evaluiert und zu anderen aktuellen Methoden zur Lösung der vorgestellten Probleme in Beziehung gesetzt. Die Schlußbetrachtungen nehmen eine Bewertung der vorgestellten Technik vor.

2 Part of Speech Tagging

Beim *part of speech tagging* geht es darum, den Wörtern eines Textkörpers algorithmisch die zugehörigen Wortarten zuzuordnen. Je nach Anwendungsgebiet kann die Einteilung in Kategorien verhältnismäßig grob sein (*Verb*, *Substantiv*, *Adjektiv*, ...), oder aber deutlich feinere Unterscheidungen treffen (*Substantiv-Singular*, *Verb-Infinitiv*, ...). Part of speech tagger bilden eine frühe Phase in natürlichsprachlichen Systemen und können dort beispielsweise die Grundlage einer syntaktischen Analyse bilden [3].

Es ist nicht möglich, das part of speech tagging-Problem durch einfache Wörterbücher zu lösen, da die eigentliche Schwierigkeit in einer Vielzahl von Mehrdeutigkeiten liegt, die nur durch den Kontext aufgelöst werden können. Betrachten wir dazu das folgende Beispiel[1]:

1. It was a calm night.

2. He tried to calm the mob.

[1] Da für die vorgestellten Algorithmen nur empirische Daten für englischsprachige Texte vorliegen, werden hier und im folgenden englischsprachige Beispiele verwendet. Die Anwendbarkeit des Ansatzes auf andere natürliche Sprachen wird im abschließenden Abschnitt angesprochen.

3. On the third day, the ship came into a calm.

Im ersten Satz wird *calm* als Adjektiv gebraucht, im zweiten Satz als Verb, im dritten schließlich als Substantiv. Interessanterweise ist es in den meisten Fällen möglich, diese Unterscheidung automatisch zu inferieren, ohne den Sinn der Aussagen zu erfassen, indem grammatikalisches Wissen über die englische Sprache benutzt wird. So steht beim ersten Satz das Wort *calm* zwischen dem Artikel *a* und dem Substantiv *night*, was die Verwendung als Adjektiv nahelegt. Im zweiten Fall geht das Wort *to* voraus und es folgt das Artikel-Substantiv-Paar *the mob*, was auf ein Verb schließen läßt. Im dritten Beispielsatz schließlich steht *calm* zwischen dem Artikel *a* und dem Satzendezeichen und ist daher vermutlich ein Substantiv.

Derartige Überlegungen führten zur Verwendung stochastischer, auf Markovmodellen basierender n-gramm-Tagger. Hierbei werden anhand annotierter (also bereits mit Tags versehener) Beispieltexte kontextuelle bedingte Wahrscheinlichkeiten gelernt, die Aussagen darüber machen, wie wahrscheinlich das Auftreten eines bestimmten Tags ist, wenn ihm n bestimmte Tags vorhergehen.

Der Tagging-Algorithmus ordnet dann einem Satz $w_1 \dots w_m$ eine Tag-Folge $t_1 \dots t_n$ zu, die für jedes Wort w_i das Produkt $P(w_i|t_i) \cdot P(t_i|t_{i-1} \dots t_{i-n})$ maximiert.

Im folgenden soll eine andere Methode vorgestellt werden, die ebenfalls auf der Verwendung annotierter Textkörper beruht, allerdings im Gegensatz zu den stochastischen Modellen eine relativ explizite Darstellung des gelernten linguistischen Wissens benutzt und wesentlich weniger speicherintensiv ist.

3 Transformation-Based Error-Driven Learning

Beim Ansatz des *transformation-based error-driven learning* wird anhand eines annotierten Trainingstextes eine Liste von *Transformationen* gelernt, die einen nicht (oder falsch) annotierten Text in einen weitgehend korrekt annotierten Text überführen sollen. Dies geschieht wie folgt:

1. Versehe den Trainingstext durch den *Startannotator* (*initial state annotator*) mit einer vorläufigen Annotierung. Dies könnte z.B. eine zufällige Annotierung sein, oder aber eine erste Näherung, etwa unter Verwendung eines Lexikons, das jedem Wort sein häufigstes Tag zuordnet, ohne den Kontext zu berücksichtigen.

2. Wähle aus der Menge der möglichen Transformationen ein „bestes Element“ aus. Füge dieses an die Liste der gelernten Transformationen an und wende es auf den Text an.

3. Wiederhole Schritt 2., bis ein Abbruchkriterium erfüllt ist.

Durch diesen Algorithmus wird also eine (geordnete) Liste von Transformationsregeln gelernt. Wenn später ein neuer Text annotiert werden soll, geschieht dies einfach dadurch, daß zunächst der Startannotator darauf angewandt wird und anschließend die gelernten Regeln der Reihe nach benutzt werden, um diese anfängliche Annotation zu transformieren. Das Verfahren ähnelt also den stochastischen n-gramm-Taggern in der Hinsicht, daß zunächst anhand eines Beispieltextes linguistisches Wissen gesammelt wird, das dann zur Annotation anderer Texte verwendet wird.

Der obige Algorithmus beinhaltet im Wesentlichen vier Freiheitsgrade:

- Die Wahl des Startannotators: Näheres hierzu bei der Besprechung der verschiedenen Varianten in den nächsten Abschnitten.

- Die Menge der zulässigen Transformationen: Dies ist das wesentliche unterscheidende Kriterium der verschiedenen Varianten, daher folgt Näheres dazu in den entsprechenden Abschnitten.

- Die Methode zur Auswahl der besten Transformation: Hier verfolgt Brill einen einfachen *Greedy*-Ansatz, wählt also jeweils eine Transformation, die die Zahl der falsch annotierten Wörter am stärksten reduziert.

- Die Abbruchbedingung: Bei Brill wird abgebrochen, sobald keine Transformation mehr gefunden werden kann, die eine wesentliche Reduktion der Fehlerzahl bewirkt.

3.1 Transformationen

Eine Transformation besteht aus zwei Teilen, einer *Modifikationsregel* (*rewrite rule*) und einer auslösenden Umgebung (*triggering environment*). Ein Beispiel für eine Modifikationsregel ist etwa: „Ändere das Tag von Verb zu Substantiv“, eine auslösende Umgebung dafür „Das vorhergehende Wort ist ein Artikel“. Diese Transformation würde die folgende falsche Annotation korrigieren:

John/Eigenname caught/Verb a/Artikel fly/Verb ./Satzende

Nach Anwendung der Transformation ergäbe sich dann:

John/Eigenname caught/Verb a/Artikel fly/Substantiv ./Satzende

Die Anwendung einer Transformation auf den Text geschieht derart, daß der Text Wort für Wort durchgegangen wird, und an jeder Stelle, die eine auslösende Umgebung markiert, die Modifikationsregel angewandt wird.

Die Durchlaufreihenfolge ist dabei von links nach rechts — diese Angabe ist wichtig, da die Anwendung der Modifikationsregel die Umgebung der nachfolgenden Wörter beeinflussen kann. So führt etwa die Regel „Ändere das Tag von A nach B, falls ein Wort mit Tag A vorangeht“ angewandt auf die Tag-Folge $A\ A\ A\ A\ A\ A$ bei einer Abarbeitung von links nach rechts zu der Tag-Folge: $A\ B\ A\ B\ A\ B$, bei einer Abarbeitung von rechts nach links hingegen zu der Tag-Folge $A\ B\ B\ B\ B\ B$.

3.2 Transformationslisten vs. Entscheidungsbäume

Um ein Gefühl für die Mächtigkeit von Listen von Transformationen zu geben, soll hier ein kurzer Vergleich zwischen Transformationslisten und den wohlbekannten Entscheidungsbäumen (*decision trees*) vorgenommen werden. Brill hat gezeigt [1], daß die Menge der Klassifikationsprobleme, die durch Entscheidungsbäume gelöst werden können, eine echte Teilmenge derjenigen Probleme ist, die durch Transformationslisten gelöst werden können, wenn beiden dieselbe Menge an primitiven Anfragen (*queries*) zur Verfügung steht.

An dieser Stelle soll eine Hälfte des Beweises geführt werden: Jede Klassifikation mithilfe von Entscheidungsbäumen kann auch mit Transformationslisten durchgeführt werden. Der einfacheren Darstellung wegen beschränken wir uns auf binäre Entscheidungsbäume. Knoten höheren Grades können analog behandelt werden. Der Beweis wird durch Induktion über dem Aufbau des Entscheidungsbaumes geführt. $\top$ bezeichnet dabei die triviale Anfrage, die immer wahr ist.

Basisfall: Ein primitiver Entscheidungsbaum liefert die Klassifikation A, falls die Antwort auf die Anfrage X? wahr ist, die Klassifikation B sonst.

Dies entspricht der folgenden Transformationsliste:

1. Anfangstag: Beschrifte mit S.
2. Wenn X, dann $S \rightarrow A$.
3. Wenn $\top$, dann $S \rightarrow B$.

Induktionsschritt: Sei jetzt T ein Baum, der an der Wurzel die Anfrage X? beinhaltet und nach T_1 bzw. T_2 verzweigt, je nach dem, ob die Antwort ja oder nein lautet.

Nach Induktionsvoraussetzung existieren zu T_1 und T_2 entsprechende Transformationslisten L_1 und L_2. Die erste Transformation von L_1 setze das Anfangstag S_1, die erste Transformation von L_2 setze das Anfangstag S_2. Ohne Einschränkung trete S_1 in L_2 nicht auf, ebenso S_2 nicht in L_1.

Die Transformationsliste zu T beginnt dann wie folgt:

1. Anfangstag: Beschrifte mit S.
2. Wenn X, dann $S \rightarrow S_1$.
3. Wenn $\top$, dann $S \rightarrow S_2$.

Es folgen die Regeln aus L_1 und schließlich die Regeln aus L_2. Offensichtlich führt die Anwendung dieser Transformationsliste zu derselben Klassifikation wie die Auswertung des Entscheidungsbaums.

4 Die Algorithmen

Da beide im folgenden vorgestellten Algorithmen das Lexikon und den Startannotator gemeinsam haben, sollen diese zunächst erklärt werden.

In einem Preprocessing-Schritt werden zunächst alle im Trainingstext auftretenden Wörter mit allen zugehörigen Tags in ein Lexikon eingetragen. Für jedes Wort wird dabei das am häufigsten auftretende Tag markiert. Das Lexikon bildet also Wörter auf Mengen von Tags ab, die immer jeweils ein ausgezeichnetes Element enthalten, das wir als *Standard-Tag* bezeichnen wollen.

Der Startannotator geht nun einfach den zu annotierenden Text durch und versieht jedes Wort mit seinem Standard-Tag. Wir gehen zunächst davon aus, daß der zu annotierende Text keine unbekannten Wörter enthält; diese Annahme wird später im Abschnitt *Unbekannte Wörter* aufgegeben.

4.1 Der nicht lexikalisierte Algorithmus

Der nicht lexikalisierte Algorithmus verwendet ausschließlich auf Tags basierende Transformationen. Dies bedeutet jedoch nicht, daß er keinerlei lexikalische Information verwendet: allen Transformationen ist gemeinsam, daß ihre Modifikationsregel „Ändere Tag A nach Tag B“ bei einem Wort w nur dann feuert, wenn entweder w nicht im Lexikon auftritt oder B laut Lexikon eines der möglichen Tags für w ist. Eine Transformation wird also niemals einem Wort, über das lexikalische Informationen vorliegen, ein Tag zuordnen, das nicht im zugehörigen Lexikoneintrag vermerkt ist.

Die Transformationen des nicht-lexikalisierten Algorithmus entsprechen den folgenden Schemata:

Ändere beim aktuellen Wort das Tag A nach B, falls

1. Das vorhergehende (folgende) Wort das Tag Z besitzt.
2. Das zweite Wort davor (dahinter) das Tag Z besitzt.
3. Eines der beiden vorhergehenden (folgenden) Wörter das Tag Z besitzt.
4. Eines der drei vorhergehenden (folgenden) Wörter das Tag Z besitzt.
5. Das vorhergehende Wort das Tag Z und das folgende Wort das Tag W besitzt.
6. Das vorhergehende (folgende) Wort das Tag Z und das zweite Wort davor (dahinter) das Tag W besitzt.

Die Menge der erlaubten Transformationen umfaßt nun alle Instantiierungen der obigen Schemata mit Tags A, B, W und Z. Geht man von ca. 30 verschiedenen Tags aus, ergeben sich so mehrere Millionen potentielle Transformationen — eine Zahl, die zwar sehr groß erscheint, aber durch entsprechende Programmierung in den Griff zu bekommen ist. In [1] wird dazu erwähnt, daß durch Verwendung eines datengetriebenen (*data-driven*) Ansatzes nur solche Transformationen betrachtet werden, die auch tatsächlich für eine Verbesserung in Frage kommen.

4.2 Der lexikalisierte Algorithmus

Der lexikalisierte Algorithmus unterscheidet sich von der eben vorgestellten Variante nur dadurch, daß die Menge der möglichen Transformationen noch durch weitere Schemata erweitert wird, in denen auch auf spezifische Wörter Bezug genommen werden kann. Diese Schemata sind im Einzelnen:

Ändere beim aktuellen Wort das Tag A nach B, falls

1. Das vorhergehende (folgende) Wort w ist.
2. Das zweite Wort davor (dahinter) w ist.
3. Eines der beiden vorhergehenden (folgenden) Wörter w ist.
4. Das aktuelle Wort w und das vorhergehende (folgende) Wort x ist.
5. Das aktuelle Wort w ist und das vorhergehende (folgende) Wort das Tag Z besitzt.
6. Das aktuelle Wort w ist.

7. Das vorhergende (folgende) Wort w ist und das vorhergehende (folgende) Wort das Tag Z besitzt.

8. Das aktuelle Wort w ist, das vorhergehende (folgende) Wort x ist und das vorhergehene (folgende) Wort das Tag Z besitzt.

Wiederum stehen A, B und Z für beliebige Tags, w und x für beliebige Wörter.

4.3 Unbekannte Wörter

Bisher wurde das Problem unbekannter Wörter im Startannotator nicht behandelt. Dies soll jetzt nachgeholt werden. Um unbekannte Wörter mit Tags zu versehen, wird ebenfalls eine Variante des transformation-based error-driven learning benutzt, um Regeln zur Markierung unbekannter Wörter durch Training mit einem bereits annotierten Text zu lernen.

Der Startannotator markiert dabei alle Wörter des Textes entweder mit *Substantiv* oder *Substantiv-Eigenname*, je nach dem, ob diese klein- oder großgeschrieben sind. Danach werden dann wie oben erklärt Transformationen gelernt, die den folgenden Schemata entsprechen:

Ändere beim aktuellen Wort das Tag A nach B, falls

1. Das Entfernen des Präfix (Suffix) x ein Wort ergibt.

2. Die Zeichenkette x ein Präfix (Suffix) des Wortes ist.

3. Das Hinzufügen des Präfixes (Suffixes) x ein Wort ergibt.

4. Das Wort w jemals direkt davor (dahinter) auftritt.

5. Das Zeichen z in dem Wort auftritt.

Dabei steht w für ein beliebiges Wort, x für eine beliebige Zeichenkette mit $1 \leq |x| \leq 4$ und z für ein beliebiges Zeichen.

5 Empirische Ergebnisse

In diesem Abschnitt sollen die Ergebnisse von Brills Experimenten mit den vorgestellten Algorithmen dargestellt werden. Insbesondere sollen dabei der Einfluß der Größe des Trainingstextes, der Einfluß der Lexikalisierung und die Effizienz der Behandlung unbekannter Wörter betrachtet werden. Bei sämtlichen Experimenten bildete der *Penn Treebank Wall Street Journal Corpus* [2] die Grundlage des Lernverfahrens und der Evaluation.

Zum Vergleich mit stochastischen n-gramm-Taggern wurde [4] herangezogen. Leider liegen für diesen Algorithmus nur Daten unter der Annahme eines vollständigen Lexikons (*closed vocabulary assumption*) vor, d.h. es wurde davon ausgegangen, daß keine unbekannten Wörter auftreten und alle möglichen Tags aller auftretenden Wörter bekannt sind. Um vergleichbare Daten zu erhalten, wurde daher zusätzlich auch eine Reihe von Experimenten durchgeführt, bei denen auch die Wörter des *Evaluationstextes* mit zugehörigen Tags in das Lexikon eingetragen wurden, bevor der Algorithmus gestartet wurde. Zum Lernen von Regeln wurde der Evaluationstext selbstverständlich *nicht* herangezogen. Diese Einträge sind in Tabelle 1 mit (CVA) markiert.

Methode	# Wörter im Trainingstext	# Regeln/ Wahrscheinlichkeiten	Trefferrate
stochastisch (CVA)	64.000	6.170	96,3%
stochastisch (CVA)	1.000.000	10.000	96,7%
Transformationen, lex. (CVA)	64.000	215	96,7%
Transformationen, lex. (CVA)	600.000	447	97,2%
Transformationen, nicht lex. (CVA)	600.000	378	97,0%
Transformationen, lex.	950.000	690	96,6%
Transformationen, nicht lex.	950.000	621	96,2%

Tabelle 1: Trefferrate unter verschiedenen Voraussetzungen

Die Daten lassen folgende Schlüsse zu:

- Transformation-based error-driven learning kann mit stochastischen Algorithmen auf Grundlage von n-grammen konkurrieren. Die erzielte Genauigkeit übertrifft die des stochastischen Algorithmus deutlich; schon mit einer kleinen Trainingsmenge von nur 64.000 Wörtern wird die Trefferrate erreicht, für die der stochastische Ansatz eine Trainingsmenge von 1.000.000 Wörtern benötigt.

- Auch ohne vollständiges Lexikon ergibt sich noch eine sehr gute Performance. Leider gibt es hierzu keine direkt vergleichbaren Daten anderer Systeme. Ergänzend ist anzumerken, daß die Zahl der korrekt annotierten *unbekannten* Wörter bei der lexikalisierten Version des Algorithmus bei 82,2% liegt. Andere (stochastische) Systeme zum Tagging unbekannter Wörter [4] erreichen hier Werte bis 85%, wobei das Modell allerdings sehr speicherintensiv ist und bis zu 10^8 Wahrscheinlichkeiten erfassen muß, während der vorgestellte Ansatz mit wenigen Regeln auskommt.

- Die Lexikalisierung bringt nicht so viel zusätzlichen Nutzen, wie man das vielleicht erwarten könnte. Dies könnte evtl. daran liegen, daß die Datengrundlage nicht groß genug ist, um ausreichend viele wesentliche wortabhängige Transformationen lernen zu können (*sparse data*).

Einen interessanten Zusammenhang sieht man in Abbildung 1, das für die lexikalisierte Version des Algorithmus unter Annahme eines vollständigen Lexikons bei einer Trainingsmenge von 600.000 Wörtern zeigt, wie groß der Einfluß der einzelnen Transformationen auf das Endergebnis ist. Man kann erkennen, daß die ersten Transformationen die Trefferrate rasch ansteigen lassen, während der Graph mit steigender Transformationszahl immer weiter abflacht.

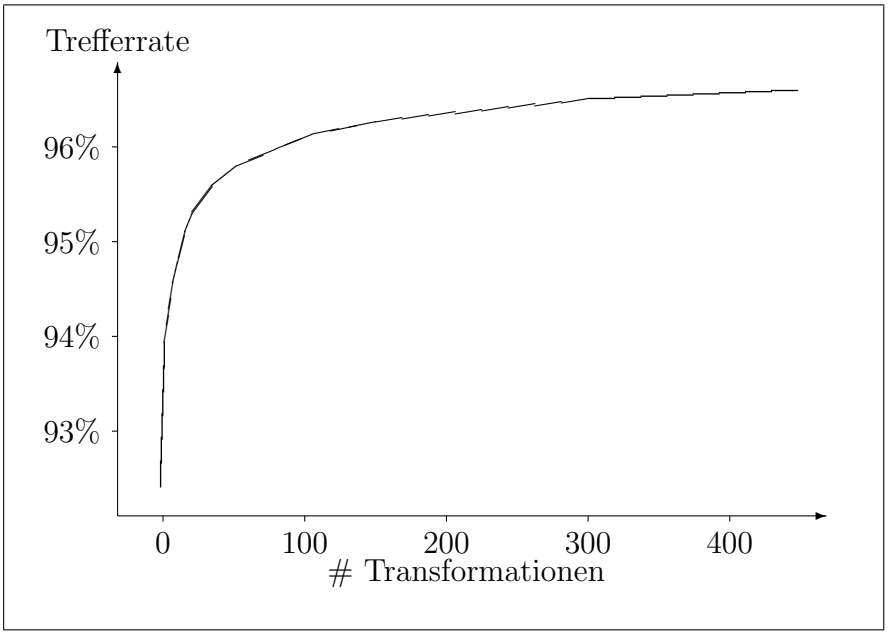

Abbildung 1: Trefferrate in Abhängigkeit von der Transformationszahl

6 Bewertung

Mit diesen Daten in der Hand kann man nun versuchen, den vorliegenden Ansatz zu bewerten und gegenüber stochastischen Taggern abzugrenzen. Was sind also die Vor- und Nachteile der vorgestellten Technik?

- Hinsichtlich der Effektivität nehmen sich die beiden Ansätze nicht viel. Leichten Vorteilen der transformations-basierten Methode insgesamt

steht ein Vorteil des stochastischen Ansatzes bei unbekannten Wörtern gegenüber.

- Die transformations-basierte Methode liefert als Ausgabe eine kleine Anzahl von Regeln. Dies ist verglichen mit den umfangreichen Wahrscheinlichkeitstabellen eines n-gramm-Taggers eine sehr kompakte Darstellung, und noch dazu eine, die Menschen gut zugänglich ist. So könnte man etwa einen vollwertigen Tagger als Startannotierer der vorgestellten Methode verwenden und anhand der gelernten Regeln eventuelle Mängel dieses Taggers identifizieren (und hoffentlich beheben).

- Laufzeitbetrachtungen sehen den transformationsbasierten Algorithmus klar im Vorteil. Die erzeugten Transformationslisten lassen sich in einen deterministischen endlichen Automaten umwandeln, der nach Angaben der Autoren einen Tagging-Algorithmus ermöglicht, der zehnmal so performant ist wie die schnellsten stochastischen Tagger. Allerdings geht bei der Umwandlung in einen Automaten naturgemäß die angesprochene kompakte Repräsentation verloren.

- Interessant wären auch Informationen über die Laufzeit des *Lernalgorithmus*. Zwar ist dies weniger kritisch, da nur einmal gelernt werden muß, aber die Autoren bleiben die Information schuldig, ob das Lernverfahren auch bei umfangreicheren Trainingstexten praktikabel bleibt.

- Beide Algorithmen sind weitgehend sprachunabhängig, werden aber bei Sprachen mit einer komplett anderen Morphologie als Englisch Schwierigkeiten haben. Sie betrachten ausschließlich *lokale* Informationen, was bei manchen sprachlichen Konstrukten etwa der deutschen Sprache problematisch ist (z.B. erweiterter Infinitiv mit *zu* oder Trennung von Modalverb und Partizip). Dies kann man jedoch nicht den Algorithmen anlasten. Festzuhalten ist auch, daß der präsentierte Lernalgorithmus für unbekannte Wörter im Gegensatz zu dem zum Vergleich herangezogenen Algorithmus [4] sprachunabhängig ist.

Literatur

[1] Brill, Eric: *Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging.* Computational Linguistics, 21(4), 1995.

[2] Marcus, Mitchell, Beatrice Santorini und Maryann Marcinkiewicz: *Building a Large Annotated Corpus of English: The Penn Treebank*. Computational Linguistics, 19(2), 1993.

[3] Russell, Stuart J. und Peter Norvig: *Artificial Intelligence. A Modern Approach*. Prentice-Hall, 1995.

[4] Weischedel, Ralph, Marie Meteer, Richard Schwartz, Lance Ramshaw und Jeff Palmucci: *Coping with Ambiguity and Unknown Words Through Probabilistic Models*. Computational Linguistics, 19(2), 1993.