LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
CENTRUM FÜR INFORMATIONS- UND SPRACHVERARBEITUNG
STUDIENGANG COMPUTERLINGUISTIK

Bachelorarbeit

im Studiengang Computerlinguistik

an der Ludwig-Maximilians-Universität München

Fakultät für Sprach- und Literaturwissenschaften

Evaluation des Satzendeerkenners EOSV3 und Erweiterung um die Sprache Russisch

vorgelegt von
Daria Glazkova

Betreuer:	Dr. Maximilian Hadersbeck
Prüfer:	Dr. Maximilian Hadersbeck
Bearbeitungszeitraum:	3. April 2017 - 12. Juni 2017

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, den

.....
Daria Glazkova

Abstract

Das Ziel der vorliegenden Bachelorarbeit ist die Evaluierung des Satzendeerkenners *End-of-Sentence Version 3* (EOSV3). Satzendeerkennung ist eine zentrale Aufgabe in der Bearbeitung von natürlichen Sprachen. Insbesondere ist es von grosser Bedeutung für das Maschinelle Lernen. Die Analyse basiert auf Korpora in der englischen, deutschen sowie in der russischen Sprache. Als Resultat wurde eine Analyse der Wirkung der internen Algorithmen im EOSV3 erstellt. Des Weiteren wurden die Resultate mit den Ergebnissen von den meistbenutzten Satzendeerkennungsprogramme verglichen. Für die Evaluierung wurden die weitverbreiteten Messkriterien *Precision, Recall, F1-Measure, Error-Rate* verwendet. Der zweite Schwerpunkt dieser Arbeit lag in der Erweiterung des EOSV3 auf die Sprache Russisch. Dazu wurden Frequenzlisten aus dem Korpora MultiUn extrahiert und im EOSV3 eingebaut.

The main task of this thesis is to provide an evaluation analysis of the tool *End-of-Sentence Version 3* (EOSV3). End of sentence detection is an important task in the natural language processing especially for machine learning systems. The basis for the analysis are corpora in the English, German and Russian lunguage. As a result we have achieved an overview of the most influencial internal algorithm in the EOSV3 system. Further we have compared the results with popular tools in this field. For the analysis we have used the evaluation criteria *Precision, Recall, F1-Measure and Error-Rate*. The second task was to incorporate the Russian language into EOSV3. This was accomplished by obtaining frequency lists extracted from the Russian corpora MultiUn.

Keywords: *Sentence boundary detection, EOSV3, natural language processing, machine learning, corpora preprocessing, end of sentence detection, Russian, frequency list.*

В данной бакалаврской работе оценивается качество работы алгоритма программы EOSV3, направленной на автоматическую идентификацию окончаний предложений в больших многоязычных тектовых объемах информации с помощью машинного обучения. Автоматизация процессов обработки теста является основной задачей в сфере компью-терной лингвистики. На основе обработанных текстовых корпусов для английского, немецкого и русского языка было проведено исследование, которое выявило, какие из встроенных способов для распознавания границ предложений являются самыми эффективными в рамках данной программы. Непосредственно для русского языка была расширена внутренняя библиотека системы, содержащая частотные списки, на базе которых функционирует алгоритм.

Inhaltsverzeichnis

Einleitung

In den letzten Jahren ist die Automation des Prozesses der Textbearbeitung im Bereich Computerlinguistik verstärkt in den Fokus der Forschung gerückt. Denn die Anzahl an Texten, dessen linguistische Inhalte analysiert werden müssen, wächst täglich. Aufgrund deren Menge können die Texte nicht manuell bearbeitet werden. Dieser Prozess wird unter dem Begriff *Natural Language Processing* (NLP) zusammengefasst.

Die vorliegende Arbeit befasst sich mit dem Thema "Satzendeerkennung"im NLP. Im Mittelpunkt dieser Bachelorarbeit steht das Programm EOSV3 (*End-of-Sentence Version 3*). Der Algorithmus wurde von einer Gruppe von Studenten in der Ludwig-Maximilians-Universität Münchens, unter der Leitung von Dr. Maximilian Hadersbeck entwickelt.

Beim Aufbau des Systems wurde die Erfahrung aus der vorangehenden Forschung in dem Bereich Satzendeerkennung berücksichtigt. Insbesondere basiert EOSV3 auf den Ideen aus dem Programm *SATZ* von D.Palmer und M. Hearst (1997), sowie unter Berücksichtigung der Ergänzungen durch A. Mikheev in seinen Werken (Tagging Sentence Boundaries, 2000; Periods, Capitalized Words, 2002).

EOSV3 besteht aus 9 Algorithmen (*Agenten*), die das Satzende in einem großen Textkorpus erkennen können. Die Effizienz des Systems wurde manuell auf dem *Brown* und *Europarl* Korpora in der Bachelorarbeit von Stefan Schweter (2013) überprüft, der auch für den Unterhalt des Programmes zuständig ist.

Im Rahmen der vorliegenden Arbeit wird der Schwerpunkt auf die Anpassung des Systems auf die russische Sprache gesetzt. Besonderen Wert wird auch auf die weitere Evaluierung des Satzendeerkenners EOSV3 mit den russischen, englischen und deutschen Korpora gelegt. Die Arbeit gliedert sich in sechs Kapitel. Zu Beginn wird eine Übersicht über den aktuellen Forschungsstand im Bereich Satzenderkennung geliefert. Besonders ausführlich werden die Verfahren von Palmer und Hearst (1997) und Mikheev (2000) erläutert.

Kapitel zwei ist dem EOSV3 und dessen Struktur gewidmet. Dem Leser wird eine ausführliche Erläuterung über das Programm gegeben. Insbesondere werden die Funktion der *Agenten* und deren Frequenzlisten erklärt.

Danach wird in Kapitel drei der Prozess zur Erstellung von den Frequenzlisten und einer Abkürzungsliste ausführlich dargestellt. Zudem wird dem Leser gezeigt, wie die russische Sprache, im Rahmen dieser Arbeit, in das EOSV3 eingebaut wurde.

Anschließend werden im Kapitel vier die sieben verwendeten Korpora vorgestellt.

Kapitel fünf dient der Evaluierung der Programme *EOSV3, OpenNLP*. Zudem werden die vier benutzten Evaluierungsmethoden erklärt: *Precision*, *Recall*, *F1-Measure* und *Error-Rate*. Zwecks Evaluierung der Ergebnisse von EOSV3 wurde ein Skript, programmiert von Stefan Schweter auf Basis der Python Programmiersprache, verwendet. Im Rahmen dieser Arbeit wurde dieses Evaluierungsskript um die Evaluierungsmethode *Error-Rate* ergänzt und im Kapitel fünf wird dieser Vorgang erläutert.

Kapitel sechs dient der Schlussfolgerung, der im Rahmen dieser Arbeit gefundenen, Ergebnisse.

1 Stand der SBD-Forschung

Im Rahmen des NLP wird die SBD (*Sentence Boundary Detection*) in den meisten Prozessen als wichtiger Aspekt der Forschung angesehen, insbesondere in der maschinellen Übersetzung, dem syntaktisches *Parsing* sowie in der *Information Extraktion* (Mikheev, 2000).

Um ein effizientes SBD-System zu entwickeln, werden zuerst die syntaktischen Besonderheiten des Punktes in einem Satz erläutert und im folgenden Abschnitt kurz erklärt.

Im Duden[1] wird ein Satz wie folgt definiert: "Ein Satz ist eine abgeschlossene Einheit, die nach den Regeln der Syntax gebildet worden ist."

In der geschriebenen Sprache werden Sätze untereinander mit Punkten, Ausrufezeichen und Fragezeichen abgetrennt. Nach dem Punktuationszeichen wird das erste Wort des nächsten Satzes mit einem Großbuchstaben geschrieben.

Allerdings musste bei der Entwicklung eines SBD-Programmes auch die syntaktische Disambiguierung des Punktes beachtet werden. Harald Zimmermann formuliert in seiner Arbeit "Zur Leistung der Satzzeichen"drei weitere Funktionen, die ein Punkt in einem Satz übernehmen kann, wie folgt (Zimmermann, 1968):

- Zur Bezeichnung einer Ordinalzahl: am 29. August
- Zur Bezeichnung einer Abkürzung: z.B.
- Als Trennzeichen in der Zeitangabe: 7.45 Uhr

Dr. D. D. Palmer und M. A. Hearst (1997) zeigen zudem weitere übersichtliche Beispiele für Situationen in denen Punkte nicht am Satzende stehen:

1. The group included Dr. J. M. Freeman and T. Boone Pickens Jr.
2. "This issue crosses party lines and crosses philosophical lines!"said Rep. John Rowland (R., Conn.).
3. Somit entsprach ein ECU am 17. 9. 1984 0.73016 US Dollar (vgl. Tab. 1).
4. Crdd au ddbut des ann es 60 . . . . . par un gouvernement conservateur : ... cet Office s' tait vu accorder six ans . . .

Um dieses Problem der Disambiguierung des Punktes zu lösen, gibt es verschiedene Lösungsansätze in der SBD-Forschung.

Nach Jonathan Read und Rebecca Dridan (2012) können SBD-Systeme in drei Gruppen eingeteilt werden, welche unterschiedliche Lösungsansätze verfolgen, um diese Mehrdeutigkeit zu beheben:

1. Regelbasierte SBD-Systeme
2. Supervises Machine Learning
3. Unsupervised Machine Learning [2]

In den folgenden Unterkapiteln werden diese Gruppierungen kurz vorgestellt.

[1] http://www.duden.de/rechtschreibung/Satz
[2] Read and Dridan, 2012

1.1 Regelbasierte SBD-Systeme

Die syntaktische Mehrdeutigkeit des Punktes in einem Satz kann mit der Hilfe von handgeschriebenen grammatischen Regeln sowie einer Auflistung von Ausnahmen (Reguläre Ausdrücke, Abkürzungslisten, Eigennamenlisten usw.) für ein SBD-System minimiert werden. Gemäß Palmer und Hearst (1997), basiert diese Methode auf der regulären Grammatik und kann sehr leicht implementiert werden. Die grammatischen Regeln des Systems sind auf der Suche nach einem typischen Satzendmuster:

„Ein Punkt gefolgt von einem Leerzeichen und schließlich ein Großbuchstabe".

In diversen Arbeiten (siehe zum Beispiel Palmer und Hearst (1997), A.Mikheev(2000) und Dan Gillick(2009)) wird das *Alembic Information Extraktion System*(AIES) erwähnt, welchem in dieser Kategorie von SBD-Systemen große Bedeutung zukommt und an dieser Stelle kurz erläutert wird. Nach Aberdeen et al. (1995) hat das *Alembic Information Extraktion System*ein Satzteilungsmodul, welches auf *Flex*(Lexical Analyzer Generator) basiert. *Flex* ist ein Programm welches 1993 von Nicol entwickelt wurde (Dale et al. 2000) Im Rahmen von AIES wird dieses in der Prozessvorbereitung verwendet um die Wörter zu segmentieren *The Punctoker* und dabei die Sätze zu segmentieren *The Parasenter*.

Das *Alembic Information Extraktion System* enthält eine Liste von 75 Abkürzungen und 100 handgeschriebenen grammatischen Regeln. Infolgedessen können nicht nur die Satzendepunktuation erkannt werden, sondern auch die in den Abkürzungen, Zeitangaben und Ordinalzahlen enthaltende Punkte. *Alembic Information Extraktion System* wurde auf dem Korpus Wall Street Journal (WSJ) getestet und als Resultat wurde 0.9 % als Fehlerquote geliefert. Trotz der erfolgreichen Ergebnisse haben die Forscher erwähnt, dass die eingebauten Regeln und Listen nur auf einem bestimmten sprachspezifischen Korpus wirksam funktionieren (Mikheev, 2000).

In ihrer Arbeit kritisieren Jeffrey C. Reynar und Adwait Ratnaparkhi (1997) die regelbasierten SBD-Systeme dahingehend, dass nicht alle Situationen überwacht und in Regeln eingebaut werden können. Als Argument verweisen die Forscher auf die Arbeit von Michael White, "Presenting punctuation"(1995), in welchem die These entwickelt wurde, dass in Situationen, wo mehrere Punkte stehen sollten, nur ein Punkt geschrieben wird. Beispielsweise, wenn eine Abkürzung am Ende des Satzes steht und einen Punkt enthält, dann wird höchstwahrscheinlich am Satzschluss kein zusätzlicher Punkt stehen.

„Er lebt in Washington D.C." [3]

Deshalb kritisieren Jeffrey C. Reynar und Adwait Ratnaparkhi die regelbasierten Systeme und sie erstellen ein neues innovatives Verfahren, welches in die Kategorie der nächsten Gruppierung fällt und im folgenden Unterkapitel vorgestellt wird.

1.2 Supervised Machine Learning Verfahren

Laut Mehryar Mohri et al., kann *Machine Learning* (ML) als Bereich der maschinellen Methoden bezeichnet, die aus vorherigen Erfahrung Neues generieren können, um Vorhersagen *Predictions* zu treffen (Mehryar Mohri et al., 2012).

Alle ML Methoden werden in 3 große Gruppen unterteilt: *Supervised*, *Semi-Supervised* und *Unsupervised*. Allerdings werden die SBD-Verfahren nach Read und Dridan (2012) nur in *Supervised* ML und *Unsupervised* ML angeordnet.

Unter *Supervised* ML wird ein überwachtes Lernen verstanden. Ein Programm braucht markierte Trainings-Beispiele *Wort-Label-Set* um Prädiktionen auf neuen unbearbeiteten Korpora zu machen.

In dem Werk von Reynar und Ratnaparkhi (1997) wird ein *Supervised* ML Modell *MxTerminator* dargestellt, welches wiederum aus zwei Subsystemen besteht.

Das Ziel der Methode ist es, alle Vorkommnisse von ".", "?", und "!"gefolgt von einem Leerzeichen aus einem Trainingskorpus zu sammeln und damit zu lernen und danach auf

[3] übersetzt aus dem Englischen aus Michael White 1995:"Presenting Punctuation"

unbearbeiteten Texten die potenzielle Satzgrenzen automatisch zu erkennen.

Der Unterschied der beiden Subsysteme besteht darin, dass ein System auf Finanztexte trainiert wurde, wodurch es eine hohe Leistung in dieser Domäne aufweist. Dieser Algorithmus kann jedoch nur unzureichend Vorhersagen über domänenfremde Korpora machen.

Das zweite System benutzt keine fachspezifischen Informationen und kann gute Ergebnisse auf unterschiedlichen Korpora liefern. Die einzige Begrenzung besteht darin, dass die Texte mit dem lateinischen Alphabet erfasst werden müssen. Im Rahmen ihrer Arbeit haben die Forscher alle potenziellen EOS (*End-of-Sentence*) analysiert, die Wörter, die vor und nach dem Satzzeichen stehen (Präfixe und Suffixe) und in einem *Maximum Entropy* Model eingebaut.

Die Algorithmen wurden dann auf den *WSJ* und *Brown* Korpus getestet und folgende Resultate wurden geliefert: das erste System hat eine Fehlerquote von 1.2% für den Korpus *WSJ* und eine Fehlerquote von 2.1% für den *Brown* Korpus, das zweite System – weist die Fehlerquoten von 2% respektive 2.5% aus.

Der größte Vorteil des Systems *MxTerminator* besteht darin, dass es keine handgeschriebenen Regeln und POS-Tags braucht.

Ein weiteres SBD-Verfahren wurde von Michael Riley (1989) entwickelt. Seine Forschung basiert auf der Idee von Leo Breiman (1984), der die *CART* (Classification und Regression Tree) Algorithmen entwickelt hat. Die folgenden Features werden dabei für die Klassifikation des Punktes in einem Satz verwendet:

- Prob[word with "."occurs at end of sentence]
- Prob[word after "."occurs at beginning of sentence] 351
- Length of word with "."
- Length of word after "."
- Case of word with ".": Upper, Lower, Cap, Numbers
- Case of word after ".": Upper, Lower, Cap, Numbers
- Punctuation after "."(if any)
- Abbreviation class of word with ".": –e.g., month name, unit-of-measure, title, address name, etc. [4]

Die Klassifikationbäume *Classification Tree* werden mit den oben genannten Features erstellt. Anschließend werden die entsprechenden Wörter gefunden und deren Wahrscheinlichkeiten berechnet. Das System wurde auf einem *AP Nachrichten* Korpus (25 Mio Wörter) trainiert und auf dem *Brown* Corpus getestet. Die Fehlerquote beträgt nur 0.2 %, was ein sehr beeindruckendes Resultat in dem Bereich Satzendeerkennung ist.

Dr. Palmer und Marti Hearst entwickeln 1997 ein neues verbessertes SBD-System *SATZ*, welches Merkmale aufweist, welche wiederum im EOSV3 verwendet wurden. Daher wird im folgenden Unterkapitel das *SATZ* System vorgestellt.

1.2.1 SATZ System

Adaptive Sentence Boundary Disambiguation System SATZ, wird von Dr. Palmer und Marti Hearst (1997) als ein innovatives und grundlegendes SBD-System charakterisiert. Jeweils drei Kontextwörter, die um einen potenziellen Satzendepunkt im Satz stehen (jeweils 3 vor dem Punkt und nach dem Punkt), werden als Vektoren dargestellt. Demzufolge besteht jeder Vektor aus den berechneten POS-Tags, die das System aus dem eingebauten

[4] Riley, 1989

POS-Lexikon erstellen kann. Diese Vektoren werden als *Descriptor Arrays* bezeichnet und sind die Eingabedaten für den Algorithmus.

Das ganze System wird in 2 Modi dargestellt: Trainieren und Disambiguieren. Die Aufgabe von dem ersten Modus ist es, aus den annotierten Korpus zu lernen und im zweiten Modus mittels gelernten Inhalten die SBD zu erkennen. Während des Trainings werden die Parameter des Systems zuerst auf den Korpus angepasst, danach bleiben sie konstant. In der Abbildung 1.1. ist die Struktur von *SATZ* abgebildet nach Dr. Palmer und Marti Hearst (1997):

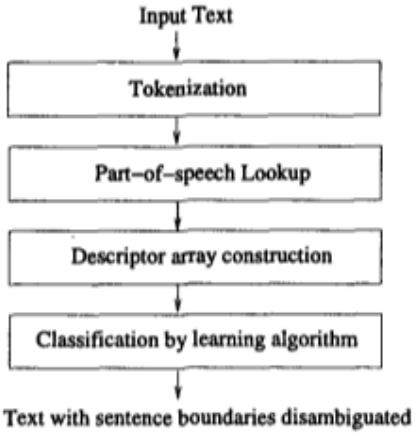

Abbildung 1.1: Satz Struktur von Dr. Palmer und M.Hearst (1997)

Zuerst wird der Input-Text eingelesen und mit dem *LEX Tokenizer* in *Tokens* segmentiert. Als Ergebnis wird sowohl eine Gruppe von Buchstaben, Ziffern als auch eine Reihe von Punktuationszeichen geliefert.

Im zweiten Schritt werden die aufgeteilten *Tokens* mit möglichen POS markiert. Das Programm berechnet entweder die A-priori-Wahrscheinlichkeit von jedem möglichen POS-Tag bei den Kontextwörtern oder gibt ihm einen binären Wert. Die Art der Berechnung hängt davon ab, ob das in dem System eingebaute Lexikon die POS-Frequenzen für das Wort enthält. Jedes NLP-Tool, das zum POS-Tagging benutzt wird, muss ein entsprechendes Lexikon haben.

Im dritten Schritt haben Palmer und Hearst die *Penn Treebank* POS-Data verwendet, die sie in 18 Kategorien verteilt und deren POS-Werte berechnet haben. Um einen Wahrscheinlichkeitsvektor zu erzeugen wird jede Kategorie-Häufigkeit für das Kontextwort in die gesamte Häufigkeit für dieses Wort geteilt. Bei der Berechnung des binären Vektors bekommen alle Kategorien, deren Häufigkeit nicht gleich 0 ist, den Wert 1, und die anderen Wörter – den Wert 0. Zusätzlich haben die beiden Forscher die zwei nachfolgenden Kategorien gebildet: *Der erste Buchstabe des Wortes ist groß* und *Dieses Wort steht nach einem Punkt.*

Schließlich werden im vierten Schritt die Vektoren in einen ML Algorithmus eingebaut. Für *SATZ* haben die beiden Forscher zwei ML Methoden implementiert: Neuronale Netzwerke (*Neural Networks*) und Entscheidungsbäume (*Decision Trees*), welche im folgenden Abschnitt kurz erläutert werden.

Künstliche neuronale Netzwerke wiederspiegeln das Prinzip der Arbeit von einem menschlichen Gehirn (Lippman, 1987). Die computergestützten neuronalen Modelle werden in der letzten Zeit erfolgreich in vielen NLP Prozessen implementiert. Auf dem Bild 1.2 wird die Architektur des *feed-forward* Netzwerks von Palmer und Hearst (1997) dargestellt.

Die Inputknoten sind aus miteinander multiplizierten Kontextgröße k und 20 Kategorien der *Discriptor Arrays* gebildet. Die multiplizierte Summe von allen vorherigen Knotenoutputs erstellt dann den *Hidden Layer*. Am Schluss liefert der Algorithmus einen einzigen Output, dessen Wert entweder 0 oder 1 ist, wobei 0 bezeichnet, dass der Punkt

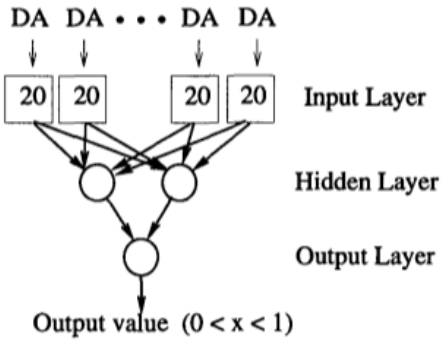

Abbildung 1.2: Architektur der neuronalen Netzwerken von Dr. Palmer und M. Hearst (1997)

im Kontext ein Satzende ist.

Die zweite ML Methode, die Entscheidungsbäume, sind gemäß Breimann et al. (1984) eine von den effektivsten Klassifikationsmethoden. Palmer und Hearst benutzen in Ihrem Model den von Ross Quinlann (1993) entwickelten *c4.5 Induktion* Algorithmus für die Generierung der Entscheidungsbäume aus den *Discriptor Arrays*. Als Input nimmt das System k Vektoren mit einem einzigen Zielattribut, welches bestimmt, ob der Punkt das Ende des Satzes ist oder nicht.

Beide ML Algorithmen werden auf den *WSJ* Korpus evaluiert und als Resultat erhalten die Forscher eine Fehlerquote von 1.1% mit der ML Methode der neuronalen Netzwerke und eine Fehlerquote von 1.0% mit der ML Methode der Entscheidungsbäume. Weil *SATZ* völlig sprachunabhängig ist, haben Palmer und Hearst das Programm auf die französische und deutsche Sprache erweitert. Das System kann auch leicht auf jeden Korpus angepasst werden. Für die deutsche Sprache wurden die Korpora der *Süddeutsche Zeitung* und *The German News* aufbereitet. Die besten Ergebnisse auf dem *Süddeutsche Zeitung* Korpus wurden mittels der ML Methode der neuronalen Netzwerke mit einer Fehlerquote von 1.3% ermittelt. Auf dem Korpus *The German News* wurden die besten Resultate mittels der ML Methode Entscheidungsbäume mit einer Fehlerquote von 0.7% ermittelt.

Die beiden Forscher kommen zu der Schlussfolgerung, dass das *SATZ* System sehr stabil und schnell ist. Es bedarf keiner manuell eingebauten Regeln und liefert gute Resultat bei der SB-Erkennung.

1.3 Unsupervised Machine Learning Verfahren

Unter *Unsupervised* ML ist ein nicht überwachter Lernprozess gemeint, wobei das Training des Systems keinen annotierten Korpus verwendet. Eine wichtige Arbeit im Bereich *unsupervised* ML ist die Arbeit "Tagging Sentence Boundries"von Andrei Mikheev (2000). Darin ergänzt er die Idee des POS-Tagging. Mittels einem auf dem *Hidden Markov Model* (HMM) und *Maximum Entropy* (ME) basierten *tri-gram POS-Tagger* werden die Kontextwörter auf ihre POS-Klassen zugewiesen. Dabei wurde das Lexikon, welches die POS-Daten enthält, aus der *Penn Treebank* extrahiert und modifiziert. Im Unterschied zum *SATZ* System hat Mikheev insgesamt 40 POS Kategorien verwendet.

Um der Disambiguierung des Punktes entgegenzuwirken, verwendet das Programm ein zusätzliches Modul, welches automatisch Abkürzungen erkennen kann. Aufgrund grammatischer Regeln markiert das Modul eine potenzielle Abkürzung in den Kontextwörtern. Mikheev basiert sein System auf einer Eigenentwicklung - dem *Document-centered approach* (DCA), welches als Teil der *Positional Guessing Strategy* (PGS) angesehen wird (Mikheev, 2002). Das Prinzip der DCA reduziert die Disambiguierung von Großgeschriebenen Wörter am Anfang des Satzes. Anders formuliert, es bestimmt, ob das Wort ein Eigenname ist oder nicht.

Das Modul zur Erkennung der Abkürzungen wurde auf dem unannotierten *New York Times* Korpus aus dem Jahr 1996 (300.000 Wörter) durchgeführt und es wurden 270 Abkürzungen extrahiert. Diese Liste hat Mikheev in sein Hauptsystem implementiert und anschließend auf dem *Brown Korpus* ohne Labels trainiert. Dann wurde das Modell auf dem *WSJ* getestet und umgekehrt. Die Fehlerquote auf dem *WSJ* beträgt 0.39% und auf dem *Brown* Korpus 0.25%. Dabei handelt es sich um das zweitbeste Resultat in der SBD-Erkennungsforschung. Der Autor beweist, dass seine Methode stabil und unabhängig vom gewählten Korpus ist. Es kann auf kleinen unannotierten Daten trainiert und dann erfolgreich auf großen Korpora evaluiert werden.

Das nächste *unsupervised* SBD-Verfahren entwickelten Tibor Kiss und Jan Strunk (2006) in ihrem System mit den Namen *Punkt*. Es braucht sowohl keine zusätzlichen Listen und Regeln, als auch kein POS-Tagging, die ganze benötigte Information wird aus einem unannotierten Korpus abgelesen und analysiert. Die Idee basiert auf dem Gedanken, dass wenn alle Abkürzungen im Text identifiziert werden, dann kann das System die EOS leicht erkennen. Das *Punkt* System hat die folgende Architektur:

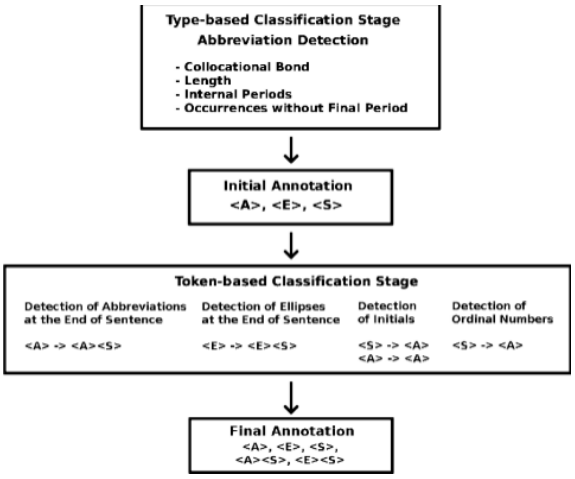

Abbildung 1.3: Punkt Architektur von Kiss und Strunk (2006)

Im ersten Schritt werden alle potenziellen Abkürzungen markiert. Die drei folgenden Charakteristiken von einer Abkürzung werden in dem System nach Kiss und Strunk (2006) berücksichtigt:

1. Strong collocational dependency: Abbreviations always occur with the final period.
2. Brevity: Abbreviations tend to be short
3. Internal periods: Many abbreviations contain an additional internal periods [5]

Der Hauptalgorithmus stützt sich auf dem *Likelihood-Ratio* (LR) von Dunning (1993), welcher in der *type-basierten* und *token-basierten* Phase berechnet wird. Dieser statistische Test überprüft, ob eine Verbindung zwischen den potenziellen Abkürzungen und den Punkten existiert und berechnet deren Wahrscheinlichkeit. In der *type-basierten* Klassifikation wird durch LR die Abhängigkeit des Wortes von dem Typ des vorherstehenden Wortes analysiert und in der *token-basierten* Klassifikation wird mittels LR und erweiterten heuristischen Regeln überprüft, ob zwischen 2 Wörtern, vor- und nach dem Punkt, eine Kollokation besteht.

Im Zwischenschritt gibt das System jedem Wort eine interne Markierung. Schließlich werden alle *Tokens* als Abkürzungen, Initialen (Single Buchstabe mit Punkt) oder Ordinalzahlen klassifiziert und die Punkte als EOS markiert.

[5] Kiss und Strunk, Unsupervised multilingual sentence boundary detection, 2006

Die Texte aus dem *WSJ* Korpus haben Kiss und Strunk (2006) in eine Entwicklungs- und in eine Testphase aufgeteilt. Aus 10 MB Daten haben sie Abkürzungen gesammelt und damit experimentiert. Der so trainierte Algorithmus wurde auf mehreren multilingualen Korpora getestet. Zur Evaluierung des Systems für die deutsche Sprache haben die Forscher den Korpus der *Neue Zürcher Zeitung* verwendet. Für die englische Sprache haben die Forscher den Korpus *WSJ* sowie den *Brown* Korpus genommen. Dabei haben die Forscher folgende Fehlerquoten erhalten: 0.35% Fehlerquote für Englisch und 1.65% für Deutsch.

1.3.1 Apache OpenNLP

Aktuell wächst die Anzahl der Programme, deren Funktionen für NLP Zwecke genutzt werden können. Ein solches System ist *Apache Software Foundation (ASF)* [6] eine 1999 gegründete Wohltätigkeitsorganisation, deren Ziel es ist eine *Open Source Software* unter den involvierten freiwilligen Forscher zu entwickeln und diese frei zur Verfügung zu stellen. ASF enthält zahlreiche Projekte, zu denen auch *OpenNLP*[7] gehört. *OpenNLP* Bibliothek ist ein ML Tool zur Berabeitung von natürlichen Sprachen. Es beinhaltet mehrere Subroutinen, unter denen sich auch ein Satzendeerkenner befindet. Um die letzte Version von *OpenNLP* (1.8.0) zu installieren, müssen vorweg folgende zwei Pakete installiert werden:

1. Vorinstallation der JDK *(Java Development Kit)* 8 - eine Entwicklungsumgebung für die Java Programmiersprache, die auf der offizielle Seite von Oracle Technology Network [8] heruntergeladen werden kann.

2. Vorinstallation der *Apache Maven* Version 3.3.9[9] - ein ASF Tool zum Aufbau von *Apache* Projekten, welches auf POM (*Project Object Model*) basiert.

Nachdem beide Pakete installiert wurden, kann *OpenNLP* mit dem Befehl "*mvn install* "kompiliert werden. Mit dem Aufruf von "*opennlp*"werden alle Subroutinen des Tools angezeigt:

```
opennlp
OpenNLP 1.8.1-SNAPSHOT. Usage: opennlp TOOL
where TOOL is one of:
  Doccat                          learned document categorizer
  DoccatTrainer                   trainer for the learnable document
      categorizer
  DoccatEvaluator                 Measures the performance of the Doccat
      model with the reference data
  DoccatCrossValidator            K-fold cross validator for the
      learnable Document Categorizer
  DoccatConverter                 converts leipzig data format to native
      OpenNLP format
  DictionaryBuilder               builds a new dictionary
  SimpleTokenizer                 character class tokenizer
  TokenizerME                     learnable tokenizer
  TokenizerTrainer                trainer for the learnable tokenizer
  TokenizerMEEvaluator            evaluator for the learnable tokenizer
  TokenizerCrossValidator         K-fold cross validator for the
      learnable tokenizer
  TokenizerConverter              converts foreign data formats (
      irishsentencebank,ad,pos,conllx,namefinder,parse,conllu) to native
      OpenNLP format
  DictionaryDetokenizer
  SentenceDetector                learnable sentence detector
  SentenceDetectorTrainer         trainer for the learnable sentence
      detector
```

[6] https://www.apache.org/
[7] http://opennlp.apache.org/
[8] //www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html
[9] https://maven.apache.org/

```
  SentenceDetectorEvaluator          evaluator for the learnable sentence
      detector
...
token(s) of a sequence of tokens in a language model
All tools print help when invoked with help parameter
Example: opennlp SimpleTokenizer help
```

Weil diese Software auf CoNLL *(Conference on Computational Natural Language Learning)* Korpora pretrainierte NLP Modelle in diversen Spachen enthält, gehört dieses zu den *Unsupervised Learning* Algorithmen. Es ist zudem möglich ein eigenes Modell mittels eigenem Korpus zu erstellen und anschliessend damit zu arbeiten.

Im Rahmen dieser Arbeit wird der *SentenceDetector* Algorithmus von *OpenNLP* auf diversen vorbereiteten Korpora evaluiert.

Mit dem folgenden Befehl wird der *SentenceDetector* aufgerufen:

```
opennlp SentenceDetector model.bin < ~/Path/to/file.txt | opennlp_output.
    txt
```

Der *SentenceDetector* untersucht den Eingabetext nach potenziellen Satzenden und speichert diese in Form "Satz per Zeile"ohne externe Markierungen in einer Ausgabedatei.

Im Rahmen dieser Arbeit werden die Ausgabewerte des *SentenceDetector* berechnet und anschließend mit den Ergebnissen des Programmes EOSV3 verglichen und graphisch dargestellt.

2 End-of-Sentence Version 3

Der EOS-Satzendeerkenner EOSV3 wurde von einer Gruppe von Studenten der Ludwig-Maximilians-Universität Münchens unter der Leitung von Dr. Maximilian Hadersbeck in 2009 entwickelt. Die aktuelle Version 3 wird von Stefan Schweter unterstützt und regelmäßig optimiert. In dem folgenden Kapitel werden der Aufbau und die Struktur des Programms dargestellt.

2.1 Kompilieren

Das komplette EOS Kernverzeichnis kann von der *Git* Seite der Zentrum für Informations- und Sprachverarbeitung der LMU mit einer fachspezifischen Berechtigung heruntergeladen werden. Das Programm wurde für Unix-basierte Systeme entwickelt, deswegen bedarf es einer Kompilierung. Zuerst muss mit dem *git clone* Befehl das EOSV3 Verzeichnis lokal gespeichert werden. Danach wird mittels *mkdir built && cd$* ein Ordner für die Kompilierung des Tools erstellt, wobei mittels *cmake* und *make fast* EOSV3 gebildet wird. *CMake* ist ein *cross-platform open-source* System zum Erstellen, Testen und Speichern der Software Projekte. Das Tool kreiert dann die Binary Dateien für jeden *Agenten* und EOSV3 selbst. Mit dem nachfolgenden Standardbefehl kann ein Text auf Satzenden untersucht werden:

```
.build/bin/eosv3 -f input.txt -o output.txt
```

Die *-f* Option bezeichnet die Inputdatei und die *-o* Option die Outputdatei. Die weiteren Funktionen können in der Dokumentation[1] von EOSV3 nachgelesen werden. Eine weitere Möglichkeit die Funktionen des EOSV3 zu testen ist die folgende Demonstrationsseite: http://eos.cis.uni-muenchen.de/

2.2 Struktur des EOSV3

Der Programmcode wurde auf der Programmiersprache C++ geschrieben. Das Fundament des Programms bilden die sogenannten *Agenten*: *Audhumbla*, *Abbreviation*, *Cross*, *Minuscule*, *Primus*, *Regex*, *Score*, *Print* und *Config*. Jeder Algorithmus löst eine spezifische Aufgabe und liefert die Resultate dem Hauptsystem. In der folgenden Abbildung wird die Struktur von EOSV3 aufgezeigt.

Zuerst wird der Textkorpus in den Systempuffer des Programmes eingelesen. Der *Config Agent* besteht aus einer JSON (*JavaSkript Object Notation*) Konfigurationsdatei und kontrolliert alle Variablen in den anderen *Agenten*. Diese Objektdatei wird dann jedem *Agenten* übergeben.

Danach wird der erste *Agent Audhumbla* aktiviert, der alle Vorkommnisse von potenziellen Satzendezeichen - Ausrufezeichen, Fragezeichen, Satzpunkte und Doppelpunkte - aufsammelt.

Im nächsten Schritt analysieren die weiteren *Agenten* die markierten Satzschlusszeichen und berechnen Werte auf einer Skala von -127 und +127. Wenn die Werte größer als Null sind, dann klassifiziert der Algorithmus diese als Satzende, wenn die Werte kleiner als Null sind, dann werden diesen keine Satzendezeichen zugewiesen. Die Werte von allen *Agenten* werden dann zum *Score Agent* geliefert, der diese dann summiert. Danach wird die entscheidende Bewertung vorgenommen.

[1] https://gitlab.cis.uni-muenchen.de/cis/eos

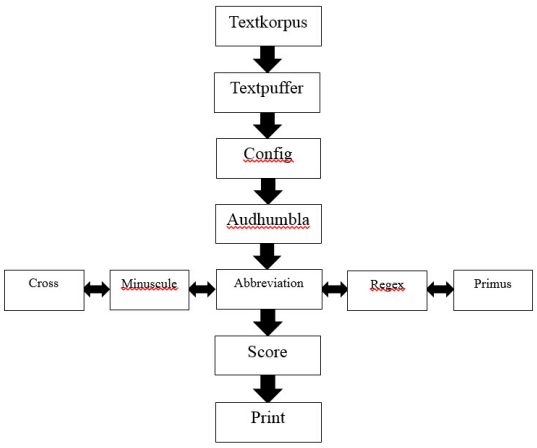

Abbildung 2.1: EOSV3 Struktur

Weil EOSV3 eine große Menge an Information untersuchen kann, kann der Bearbeitungsprozess relativ lang dauern. Infolgedessen wurde eine innovative Methode zur Optimierung des Systems von Stefan Schweter (2013) implementiert *array hash map*. Das ist eine Art der Datenstrukturierung, die zur Speicherung der *Key-Value* Paare den *Cache*-Speicher verwendet. Demzufolge hat das Programm schnelleren Zugriff zu den im Puffer eingeladenen Angaben und die Geschwindigkeit des Bearbeitungsprozesses wird beschleunigt.

2.3 Die Funktionen der Agenten

Wie bereits erwähnt, besteht das EOSV3 aus den mehreren *Agenten*. Die Beschreibung der *Agenten* basiert auf den Informationen der internen *Gitlab Projects* an der LMU, den Quellcodes des EOSV3 und der Bachelorarbeit von Stefan Schweter (2013).

2.3.1 Config

Dieser *Agent* basiert auf einer *JSON* Konfigurationsdatei, worin alle EOSV3 Systemeinstellungen enthaltend sind. In der vorangehenden Version EOSV2 wurden die Gewichtungen für alle *Agenten* in einer XML-Datei gespeichert. Im Rahmen seiner Bachelorarbeit hat Stefan Schweter (2013) einen neuen *Config Agenten* erstellt, der auf der *JSON* Objektklasse basiert. *JSON* ist ein Textformat für die Serialisierung von strukturierten Daten, welches aus primitiven Variablentypen (*Strings, Zahlen, Booleans* und Nullen) und zwei strukturierten Variablentypen (Objekte und Listen) besteht (T.Bray, 2014).

Im folgenden Codeabschnitt werden die Standardgewichtungen für die Agentenvariablen vorgestellt.

```
1  void PConfig::set_config_json_default_values() {
2    {...
3    std::string default_values = R"({
4      "agents": {
5        "regex": {
6          "weighting": 1.0,
7          "scores": {
8            "eos": {
9              "min": 40,
10             "max": 50
11           },
12           "non-eos": {
13             "min": -40,
14             "max": -50
15           },
```

```
16            "default": {
17              "min": 0,
18              "max": 0
19            ...}
```

2.3.2 Audhumbla

Der erste *Agent Audhumbla* wurde von David Kaumanns im Jahr 2009 erstellt. Des Weiteren haben Benno Weck und Eamonn Lawlor im Jahr 2013 diesen weiterentwickelt und schließlich wurde er im Jahr 2016 von Stefan Schweter ergänzt.

Der Algorithmus findet alle möglichen EOS Markierungen ("!", ".", "?", ":") in dem im Puffer eingeladenen Text und speichert diese in einen *EOS_Property* Vektor[2]. Die Zeichen, die vor und nach dem potenziellen EOS stehen, werden in einem Tupel (*eos_from_pos*, *eos_punct_pos*, *eos_to_pos*) gespeichert. Diese Positionangaben werden wiederum in den anderen *Agenten* weiterverwendet.

2.3.3 Primus

Der zweite *Agent* wurde von Simon Thum und Carsten Pfuhl im Jahr 2010 geschrieben und später von Martin Röhrs (2013) und Stefan Schweter (2014, 2016) überarbeitet.

Der *Primus Agent* überprüft, ob das Wort nach einer potenziellen EOS Markierung aus dem *EOS_Property* Vektor in einer im *Agenten* eingebauten Frequenzliste existiert. Die sortierten Frequenzlisten unterscheiden sich je nach Sprache und enthalten die Wörter, die statistisch am häufigsten zu Beginn des Satzes stehen. Der *Primus Agent* berechnet die Werte anhand der Frequenz in der Liste mit der folgenden Formel:

$$(117.0/max\_frequency) * current\_frequency + 10$$

Max_frequency = Höchste Wortfrequenz aus der Liste
Current_frequency = Wortfrequenz aus dem aktuell untersuchten Text
Somit ist der maximale Wert +127 und der Minimalwert +10.

2.3.4 Cross

Der dritte *Agent* wurde von Carsten Pfuhl und Simon Thum im Jahr 2010 programmiert, von Matthias Lindinger (2013) und Stefan Schweter (2013, 2014, 2016) weiterentwickelt. Die Hauptfunktion des *Agenten* besteht darin, das nach einer potentiellen EOS Markierung stehende Wort zu untersuchen, ob dieses in dem Textkorpus sonst kleingeschrieben wird oder ob dieses in der eingebundenen Frequenzliste auftaucht. Wenn eine von den beiden Bedingungen erfüllt wird, dann wird ein Wert mit der folgenden Formel berechnet:

$$(40.0/max\_frequency) * current\_frequency + 10$$

Der maximale Wert ist dabei +40 und der minimale Wert +10

2.3.5 Abbreviation

Der vierte *Agent* wurde von Dr. Maximilian Hadersbeck im Jahr 2010 erstellt und später von Stefan Schweter (2013, 2016) erweitert.

Der *Abbreviation Agent* untersucht die Einträge im *EOS_Property* Vektor nach Abkürzungen, die in dem Programm enthaltenden multilingualen Abkürzungslisten gespeichert sind. Wenn ein Eintrag im Text gefunden wird, dann überprüft der Algorithmus, ob die Abkürzung von einem Fragezeichen oder Ausrufezeichen gefolgt wird. Falls dies zutrifft wird dieser als Satzende bezeichnet und der Wert wird positiv. Wenn auf eine Abkürzung ein Punkt oder ein Doppelpunkt folgt, dann wird der Wert negativ. Wenn das Programm

[2] siehe Quellcode audhumbla.cpp

das markierte Wort als eine Abkürzung vermutet, dann übernimmt die Funktion ein zweiter Algorithmus.

Der zweite Algorithmus mit dem Namen *Morphochecker* den Matthias Lindinger und Jasmin Chebib im 2013 programmiert haben, ist für unbekannte Abkürzungen geeignet. Dabei werden die folgenden morphologischen Methoden implementiert[3]

- Wenn das Wort nur aus Konsonanten besteht oder aus zwei Zeichen: Punkt und Buchstabe, wird der *Score* -40 zurückgeliefert.

- Wenn das Wort ein Akronym ist ("aus den Anfangsbuchstaben mehrerer Wörter gebildetes Kurzwort"(Duden[4])), wird der *Score* +5 zurückgeliefert.

2.3.6 Minuscule

Der fünfte *Agent* wurde von Dino Azzano und Estelle Perez im 2010 entwickelt und später von Angela Krey (2012) und Stefan Schweter (2013, 2016) ergänzt. Das Programm betrachtet nur die Wörter in der "Nach EOS"Position. Basierend auf den grammatikalischen Regeln sucht der Algorithmus nach folgendem Muster:

- Eine Großbuchstabe oder eine Zahl;

- Zwei Zeilenumbrüche;

- Ein Zeilenumbruch, anschließend ein Kleinbuchstabe gefolgt von einem Leerzeichen, oder ein Zeilenumbruch gefolgt von einer geöffneten Klammer, einer

- Zahl und einem Punkt;

- Ende der Datei;

In den obengenannten Fällen wird ein Satzeende markiert. Wenn nach dem potenziellen EOS ein Kleinbuchstabe kommt, dann wird kein Satzende markiert.

2.3.7 Regex

Der sechste *Agent* wurde von Dino Azzano, Ekaterina Peters und Estelle Perez im Jahr 2010 entwickelt und von Stefan Schweter (2013, 2016) erweitert.

Der *Agent* öffnet die interne Listen mit Regulären Ausdrücken und vergleicht diese mit den *Tokens* aus dem *EOS_Property Vektor*. Falls diese Werte gleich sind, dann berechnet der *Agent* die entsprechenden *Scores*.

Das folgende Muster wird zur Erstellung solcher Listen benutzt:

$$[] + | - /Ausdruck/[i]$$

Ein Plus in *Regex* bedeutet, dass der Ausdruck ein Satzendemuster findet und ihm einen positiven Wert übergibt. Ein Minus bezeichnet, dass es sich um kein Satzende handelt und das Wort einen negativen Wert bekommt. Der Wert bei einer negativen Bewertung beträgt -50, und bei der positiven Bewertung +50.

[3] Quellcode morhpo_checker.cpp
[4] http://www.duden.de/suchen/dudenonline/akronym

2.3.8 Score

Der *Score Agent* hat eine entscheidende Rolle in der Markierung von Satzgrenzen. In der Funktion *add_value_to_agent_scores* werden die in den vorangehenden *Agenten* berechneten Werte aufsummiert. Wenn die Summe der Werte mehr als +127 beträgt, dann wird die Summe auf den oberen Grenzwert von +127 reduziert. Falls die Summe kleiner als -127 ist, dann wird die Summe auf den unteren Grenzwert erhöht. Die *is_eos* Methode bestimmt, ob die gefundene Markierung eine echte Satzgrenze ist, indem er die unter *add_value_to_agent_scores* Summe untersucht und vergleicht ob diese grösser/kleiner als 0 ist.

2.3.9 Print

Der *Print Agent* wurde von Davin Kaumanns im Jahr 2010 erstellt und später von Stefan Schweter (2013) komplett neu geschrieben. Dabei wurde der Code von 300 Zeilen auf 100 Zeilen reduziert. Der Print-Algorithmus druckt den Text mit den am Satzende hinzugefügten </eos> Markierungen aus.

2.4 Array Hash Map

Die aktuelle Version von EOSV3 wurde von Stefan Schweter im Jahr 2013 optimiert. Der Speicherverbrauch und die Laufzeit des Programms wurden mit sogenannten *Array Hash Map* wesentlich verbessert. Gemäß Stefan Schweter, basiert die *Array Hash Map* Datenstruktur auf der Array_Hash Methode von Chris Vaszauskas und Tyler Richard und wurde für zweispaltige *key-value* Frequenzlisten modifiziert. "Darüberhinaus unterstützt die implementierte Datenstruktur auch das Serialisieren und Deserialisieren von Frequenzlisten; das bedeutet, dass die Datenstruktur binär auf die Festplatte gespeichert werden kann – und, dass man dieses Format sehr schnell wieder einlesen und somit in den Arbeitsspeicher laden kann."(Stefan Schweter, 2013, Seite 28)

3 Erweiterung um die Sprache Russisch

In diesem Kapitel wird der Prozess zur Bearbeitung eines russischen Trainingskorpus beschrieben, sowie die Methoden, die für die Erstellung der Abkürzungslisten und Frequenzlisten für die EOS *Agenten Primus, Cross* und *Regex* verwendet wurden. Der Programmcode wurde auf der Python Programmiersprache in *PyCharm IDE* Version 216.3.3 entwickelt.

3.1 MultiUn russicher Trainingskorpus

Der *MultiUn* Korpus (A Multilingual Corpus from United Nation Documents) wurde von Andreas Eisele und Yu Chen (2010) aus *The United Nations Webpage* extrahiert, gesäubert und in ein XML File mithilfe der *Language Technology Lab* in DFKI GmbH konvertiert. Der Korpus enthält Textdokumente aus den Jahren 2000 bis 2009, welche auf diverse Sprachen übersetz wurden.

Für die vorliegende Arbeit wurde der russische Anteil des Korpus von der Hauptpage[1] heruntergeladen und entpackt. Die Texte aus den Jahren 2008 und 2009 wurden in das Programm *PyCharm IDE* Version 216.3.3 importiert und mithilfe eines im Rahmen dieser Arbeit geschriebenen Skript wie folgt extrahiert:

```
1  from io import StringIO
2  import xml.etree.ElementTree as ET
3  import os
4  def create_train_text(infile, outfile):
5      tree = ET.parse(StringIO(infile))
6      doc = tree.getroot()
7      global txt
8      for sent in doc.findall('text/body/p'):
9          txt = sent.find ('s').text
10         txt = txt[1:-1]
11         outfile.write(txt + "\n")
12     return txt
13 if __name__=="__main__":
14     folder = os.listdir("MultiUn_ru")
15     for f in folder:
16         file = os.path.join("MultiUn_ru", f)
17         print (file)
18         with open (file, 'r') as text, open ("MultiUn_ru_train.txt", 'a')
19         as work_file:
20             xml = text.read()
21             corpus = create_train_text(xml, work_file)
```

Das Modul *xml.etree.ElementTree* ermöglicht dem Anwender mit XML-Dateien zu arbeiten. Der *ElementTree.parse* zergliedert dabei das Dokument in einen XML-Baum mit den einzelnen Elementen, die *root* Funktion übernimmt die Tags aus dem originalen XML-File und erstellt dann ein *Dictionary* mit den Attributen. Dieser Dictionary wird im Rahmen dieser Arbeit jedoch nicht benötigt, sondern es wird mit den Tags gearbeitet. Um ebendiese Tags zu finden, wird die Funktion *finall* verwendet, welche eine Liste mit allen Tags <p> bildet. Die Funktion find sucht die einzelnen <s> Tags und extrahiert aus diesen die entsprechenden Textteile. Da jeder Satz im Korpus ein Leerzeichen am Satzanfang und am Satzende enthält, werden diese im nächsten Schritt ausgelassen. Der Trainingskorpus

[1] http://www.euromatrixplus.net/multu-un/

wird mit einem Zeilenumbruch nach jedem Satz, in eine neue Datei mit dem Namen *MultiUn_ru_train*, gespeichert. Dies bezweckt die Erstellung der Frequenzlisten effizienterer zu gestalten.

3.2 Erstellung der Cross und Primus Listen

Wie im zweiten Kapitel erwähnt verwenden die beiden *Agenten Cross* und *Primus* multilinguale Frequenzlisten.

Mittels folgendem Python Skript wurden die Listen für die russische Sprache aus dem *MultiUn_ru_train* erstellt, um diese dann in die beiden *Agenten* einzubauen:

```
1  def find_primus(in_file, out_file):
2      primus = []
3      for line in in_file.split("\n"):
4          new_line = regex.findall(r"^\b[А-Яа-я]+\b",line)
5          if (len(new_line) >=1):
6              new_line = [w for w in new_line[0].split() if w[0].isupper()]
7              if (len(new_line) >= 1):
8                  primus.append(new_line[0])
9      counter_primus = collections.Counter(primus)
10     for k, v in sorted(counter_primus.items(), key=lambda k:k[1],
11     reverse=True):
12         out_file.write("%s␣%s\n" % (v, k))
13     return counter_primus
14
15 def find_cross(in_file, out_file):
16     wordlist=[]
17     for word in in_file.split("␣"):
18         if word.isalpha() and word[0].islower():
19             wordlist.append(word)
20     counter_cross = collections.Counter(wordlist)
21     for k, v in sorted(counter_cross.items(), key=lambda k:k[1],
22     reverse=True):
23         out_file.write("%s␣%s\n" % (v, k))
24     return counter_cross
25
26 if __name__=="__main__":
27     with open ("MultiUn_ru_train.txt", "r") as file:
28         text = file.read()
29         text = text[:-1]
30         with open("ru_lower_words.txt", "w") as cross_file:
31             list_of_words = find_cross(text, cross_file)
32         with open ("primus_frequency_list_ru.txt", "w") as primus_file:
33             #primus_list=find_primus(text, primus_file)
```

Für beide Frequenzlisten wurden zwei eigens entwickelte Funktionen erstellt.

Die erste Funktion *find_primus* läuft jede Zeile im Text durch, findet die Anfangswörter, die keine Zahlen und keine Punktionszeichen sind (*regex.find*) und erstellt eine Liste, die mehrere Unterlisten (*Subarrays*) aus einzelnen Wörtern, enthält.

Wenn die Unterliste nicht leer ist, dann sucht der Algorithmus nach einem großgeschriebenen Wort und falls er das findet, wird die Liste mit diesem Wort überschrieben. Falls die Liste keine Elemente enthält, dann wird die Liste ignoriert, und alle anderen Wörter werden dann in eine vorher deklarierte *Primus List* gespeichert.

Mit der Funktion *collections.counter*, die die Verdoppelungen von Wörtern vermeidet, wird ein *Dictionary* aus der Liste gebildet, wobei alle Wörter nach ihren Häufigkeiten sortiert werden und in eine neue *Primus_frequency_list_ru.txt* Datei geschrieben werden.

Die zweite Funktion, *find_cross*, geht jedes Wort in dem *MultuUn_ru_train.txt* File durch. Wenn ein Wort aus Buchstaben besteht (kein Punktuationzeichen oder keine Zahl) und dabei kleingeschrieben ist, dann wird eine *Cross* Liste erstellt. Diese *Cross* Liste wird dann mit der Funktion *collections.dictionary* wieder in ein *Dictionary* umgewandelt, nach Wortfrequenzen sortiert und in eine Datei namens *ru_lower_words.txt* gespeichert.

3.3 Erstellung der Abkürzungsliste

In der russischen Sprache wird zwischen fünf Arten von Abkürzungen nach ihrer Bildungsart unterschieden:

1. Graphische Abkürzungen
   - mit Punkt, z.B., рус. für русский.
   - mit Bindestrich, z.B., пр-во für правительство.
   - mit Schrägstrich, z.B., п/п für по порядку.
2. Initiale Akronyme
   - Ausbuchstabierte, z.B., СНГ für Сотружество Независимых Государств.
   - Phonetische, z.B., ЦСКА für Центральный Спортивный Клуб Армии.
   - Gemischte, z.B., ИТАР für Информационное Телеграфное Агентство России
3. Kurzwörter, z.B., колхоз für коллективное хозяйство.
4. Silbenkurzwörter, z.B., млрд für м[ил]л[иа]рд.
5. Gemischte Abkürzungen, z.B., стб. für столбец. (Мильчин А.Э., Чельцова Л.К. , 2003)

Abkürzungen können je nach Verwendungskontext in folgende drei Klassen eingeteilt werden:

1. Allgemeingültige Abkürzungen
2. Fachbezogene Abkürzungen
3. Individuelle Abkürzungen

Für die Erstellung der für EOSV3 relevanten Abkürzungsliste wurden die graphischen (mit Punkten) und gemischten Abkürzungen aus der Liste der russischen allgemeingültigen Abkürzungen entnommen und alphabetisch sortiert.

Die letzten Punkte bei jedem Element in der Liste werden ausgelassen, weil der *Abbreviation Agent*, falls er eine solche Abkürzung in dem *EOS_ Property Vektor* findet, einen Punkt diesem Element übergibt. In diesem Fall überprüft der Algorithmus das nächste Zeichen, wenn es sich um ein „?“ oder „!“ handelt, dann wird ein Satzende festgestellt.

3.4 Erstellung der Regex Liste

Der *Regex Agent* verwendet das folgende Muster für die Erstellung der sprachbezogenen Regulären Ausdrücke für EOSV3.

```
[#]+|-/Ausdruck/[i]
```

Das bedeutet, dass jede Zeile in der Datei nach den Regeln der BNF (Backus Normal Form) geschrieben werden muss. Deswegen wurden die folgenden zwei Regulären Ausdrücke für die russische Sprache erstellt:

- Ausdruck zur Erkennung der Straßennamen
  `-/(ул\.)$/i`
- Ausdruck zur Erkennung der russischen Namen und Initialen.
  `-/^([А-Я]\.\-[А-Я]\.)$/`

Allen so gefundenen *Token* werden negative Werte zugewiesen, was bedeutet, dass sie keine Satzenden bezeichnen.

4 Testkorpora

In diesem Kapitel wird der Prozess der Bearbeitung von dem englischen, deutschen und russischen Korpora beschrieben.

4.1 OpenCorpora

Für die Evaluierung des EOSV3 für die russische Sprache wurde ein Korpus namens *OpenCorpora* ausgesucht.

OpenCorpora ist ein frei verfügbares Projekt, welches dem Aufbau von einem großen semantisch, syntaktisch und morphologisch annotierten russischen Korpus gewidmet wurde. Die ersten Texte stammen aus dem Jahr 2009 und werden auf der Hauptseite[1] für Interessenten publiziert. Der Korpus steht frei zur Verfügung und kann ohne Lizenz verbreitet werden. Der Korpus enthält 108.960 Sätze, 1.966.903 *Tokens* und 1.522.271 Wörter[2] und wird ständig erneuert. Zum Herunterladen ist der *OpenCorpora* sowohl in einer kompletten XML Datei, wie auch als Sammlung von XML Dateien in einem Ordner verfügbar.

Im Rahmen dieser Arbeit wurden die XML-Dateien aus dem *OpenCorpora* Ordner mit der Hilfe von einem Python Skript bearbeitet und in 2 Textdateien gespeichert: *RU_gold.txt* und *RU_eos_input.txt*.

Das Python Skript für den Trainingskorpus wurde modifiziert und auf dem *OpenCorpora* ausgeführt.

```
1      for sent in doc.findall('paragraphs/paragraph/sentence'):
2          txt = sent.find ('source').text
3          if regex.match (r"[A-Яа-я]", txt[-1]):
4              outfile.write(txt + ".</eos>\n")
5          else:
6              outfile.write(txt + "</eos>\n")
7      return txt
```

Der Pfad zum <source> Tag wurde geändert.

Der Korpus besteht aus mehreren Texten, welche wiederum durch Titel markiert sind. Die Titel haben jedoch keine Satzendzeichen. Dabei kann die Funktion Regulärer Ausdruck r"[A-Яа-я]"zu Hilfe genommen werden, diese Funktion prüft das letzte Satzzeichen und wenn es ein kyrillischer Buchstabe ist, dann schreibt eine weitere Funktion einen Punkt nach diesem Buchstaben.

Der extrahierte Satz wird in einen RU_gold.txt dann zeilenweise mit dem </eos> Tag ergänzt. Wenn der Satz ein Satzendezeichen enthält, dann wird er nur mit einem </eos> Tag und einem Zeilenumbruch in den RU_gold.txt gespeichert.

Die zweite Variante des Korpus, welche als Input für den EOSV3 geeignet ist, wurde aus der vorherigen Textdatei entwickelt:

```
1      with open("RU_gold.txt", 'r') as readfile,
2      open("RU_eos_input.txt", 'w') as outfile:
3          text = readfile.readlines()
4          for line in text:
5              line = line.strip("</eos>\n")
6              outfile.write(line + "␣")
```

Der Text wird zeilenweise eingelesen, von Zeilenumbrüchen und </eos> Tags gesäubert und mit einem Leerzeichen zwischen den Sätzen in einer *RU_eos_input.txt* gespeichert.

[1] www.opencorpora.org
[2] Stand 4.06.2017

4.2 DeReKo

Der *DeReKo*, der deutsche Referenz Korpus der geschriebenen Gegenwartssprache, wurde im Institut für deutsche Sprache (IDS) in Mannheim im Jahr 1964 von Paul Grebe und Ulrich Engel zusammengefasst. Heute enthält der Korpus ca. 31,68 Milliarden Wörter und gilt als der Hauptquelle für die Entwicklung der deutschen Sprache (Marc Kupietz at al., 2010). Die Nutzung des Korpus ist nur unter Lizenz erlaubt. Deswegen ist das Kopieren und die Weitergabe von diesem Korpus verboten.

Der Korpus wurde als einzelne XML-Datei heruntergeladen. Der Textteil aus dem Tag <s> wurde mit dem Befehl xml_grep in der Kommandozeile in einer Textdatei extrahiert.

```
xml_grep 's' bzl.i5.xml text_only | tee Dereko_raw.txt
```

Die Anfangsbuchstaben von den ersten Wörtern im Text sind kleingeschrieben, deswegen musste der Text weiter bearbeitet werden. Mit dem folgenden Python Skript wurden alle Sätze in 2 neue Textdateien gespeichert: *Dereko_gold* und *Dereko_eos_input*.

```
1  with open ("Dereko_raw.txt", "r") as file,
2  open ("Dereko_eos_input.txt", "w") as outfile:
3      text = file.readlines()
4      list = []
5      for line in text:
6          new_line = line[0].upper() + line[1:]
7          if len(new_line) >= 3:
8              if not new_line[0].isalpha():
9                  not_letter_line = line[0] + line[1].upper() + line[2:]
10                 list.append(not_letter_line)
11             else:
12                 list.append(new_line)
13     for i in list:
14         temp = i.strip("\n")
15         if temp[-1].isalpha():
16             outfile.write(temp + ". ")
17         elif temp[-1].isspace():
18             outfile.write(temp)
19         else:
20             outfile.write(temp + " ")
```

In dem vorliegenden Code werden zuerst alle kleingeschrieben Satzanfangsbuchstaben mit großgeschriebenen Buchstaben ersetzt und danach werden </eos> Markierungen für den *Dereko_gold* Korpus hinzugefügt. Für den *Dereko_eos_input* werden die Zeilenumbrüche entfernt und mit Leerzeichen ersetzt. Weil einige Sätze in dem *Dereko_raw* Texttitel sind, haben diese keine Satzendemarkierung. Infolgedessen werden Punkte hinzugefügt, damit das Satzendeerkennungsprogramm keine Fehler ausgibt.

4.3 Leipzig Corpora Collection

Die Leipzig *Corpora Collection* ist als Teil des „Projektes Deutscher Wortschatz“ entstanden und besteht aus einer Sammlung von Korpora aus diversen Sprachen, die für NLP Zwecke verwendet werden können. Seit Juni 2006 stehen auf der offiziellen Seite von der Leipziger Universität diverse Archive in verschiedenen Sprachen zur Verfügung[3]. Dabei teilen sich die Korpora in zwei Gruppen: Nachrichten und Wikipedia-Einträge.

Aufgrund der im Rahmen dieser Bachelorarbeit besser geeigneten Satzstruktur wurden für die Evaluierungphase die Korpora basierend auf den Wikipedia-Einträgen gewählt. Die Korpora wurden dabei sowohl in der deutschen, sowie englischen und auch in der russischen Sprache gewählt. Jeder Korpus beinhaltet 300.000 Sätze.

Im Original hat jeder Korpus den folgenden Aufbau:

[3] www.wortschatz.uni-leipzip.de

- 24 *tabulator* 10. Januar 2009 Während der Nacht wurden aus dem Gazastreifen keine Qassamraketen abgeschossen, am Vormittag trafen drei Geschosse Aschkelon und weitere sechs schlugen in offenem Gelände ein.

- 25 *tabulator* 10. Juli bis 10. September (5 Tote) Ab dem Abend des 10. Juli betrug die Gesamtzahl der Todesfälle laut der auf den 10. Juli datierten TTB-Verletztenstatistik fünf.

Mit folgendem *Shell* Befehl wurden die Sätze aus dem deutschen, russischen und englischen Korpora bereinigt:

```
cat wiki_de.txt | sed 's/^[0-9]*\t//g' | tee wiki_de_corpus.txt
```

Die *sed* Funktion sucht anhand des *Regulären Ausdrucks (^[0-9]*)* alle Zahlen, die am Anfang der Zeile stehen, gefolgt von einem Tabulator und verwirft diese.

Anschliessend wurde der Korpus in zwei Formen gespeichert: *wiki_language_gold* und *wiki_language_eos_input*.

Mittels einem kleinen Python Skript wurde zu jedem Satz im Korpora je nach Verwendungszweck entweder die </eos> (GOLD) Endungen hinzugefügt oder nur ein Leerzeichen (EOS Input):

```
1 import sys
2
3 with open (sys.argv[1], "r") as file, open(sys.argv[2], "w") as outfile:
4     text = file.readlines()
5     for line in text:
6         words = line.strip ("\n")
7         outfile.write (words + "</eos>" + "\n") # for GOLD
8         # outfile.write (words + " ") # for EOS_input
```

4.4 Europarl

Der *Europarl* (*European Parliament*) parallel Korpus wurde im Jahr 2005 von Philipp Koehn veröffentlicht (Philipp Kehn, 2005). Der Korpus gehört zu *OpenSource* Quellen und steht im Internet[4] frei zur Verfügung. *Europarl* besteht aus Prokollen des europäischen Parlaments, die seit dem Jahr 1996 auf 11 Sprachen parallel erfasst wurden. Für *Machine Translation* Zwecke wurde der Korpus paarweise für je zwei Sprachen gespeichert.

Im Rahmen dieser Arbeit wurden 300.000 Sätze aus dem *Europarl* Archiv für das deutsch-englische Paar entnommen. Die jeweiligen Korpora wurden in jeweils zwei Varianten gespeichert: *Europarl_language_gold.txt* und *Europarl_language_eos_input.txt*.

Da der Korpus mehrere leere Zeilen enthalten hat, wurden diese mit dem folgenden Befehl bereinigt.

```
cat europarl-v7.de-en.de | grep . | tee europarl_de.txt
```

Mit dem "head -300000 > eos_input.txt"wurden 300.000 Sätze aus dem ganzen Korpus (1.920.209 Sätze) extrahiert. Mittels den oben erwähnten Python Funktionen wurde der Korpus in die Form der Gold- und Inputdatei gebracht. Bei der Bearbeitung des *Europarls* wurden auch die Titel der Artikel berücksichtigt und entsprechend behandelt:

```
if line[-1].isalpha():
            outfile.write(line + ".␣") # for Title in EOS-Input
            # outfile.write (line + ".</eos>\n") # for Title in Gold
        else:
            outfile.write(line + "␣")
            # outfile.write(line + "</eos>")
```

[4] http://www.statmt.org/europarl/

5 Evaluierung

In diesem Kapitel werden zuerst die vier meist verwendeten Kennwerte für die Evaluation von einem ML Programm erläutert. Danach werden die Ergebnisse von der Evaluierung des EOSV3 mit unterschiedlichen Einstellungen der *Agenten* analysiert und diese mit den Resultaten der Evaluierung des Apache *OpenNLP* auf denselben Korpora verglichen.

5.1 Precision, Recall, F1-Measure und Error-Rate

Die sieben ausgesuchten Korpora werden im vorliegenden Kapitel anhand der folgenden vier Messkriterien evaluiert:

Precision : tp/(tp + fp)
Recall : tp / (tp + fn)
F1 - Measure : 2*(P*R)/ (P + R)
Error-Rate : fp + fn / (tp + fn + fp + tn)

Bei den *True Positives* (tp) handelt es sich um die EOS Markierungen, die sowohl im Gold-Standard wie auch in der Outputdatei richtigerweise als EOS gekennzeichnet wurden.

Unter *False Positives* (fp) sind diejenige EOS Markierungen aufgelistet, die von dem Satzendeerkenner als Satzende markiert wurden, aber die gemäß Goldstandard keine EOS darstellen.

False Negatives (fn) sind die EOS Markierungen, die im Goldstandard als Satzende gelten, aber nicht von dem Algorithmus gefunden werden.

True Negatives (tn) sind keine Satzenden und wurden vom System auch nicht als solche markiert.

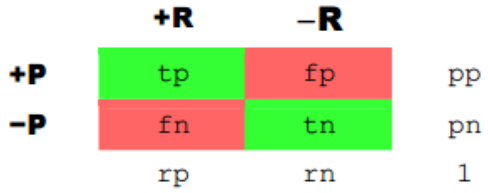

Abbildung 5.1: Kontingenztabelle von David M.W. Powers (2011)

In Anlehnung an die Kontingenztabelle von David M.W.Powers (2011) ist *Precision* (Genauigkeit) der Anteil der True Positives (korrekte Satzendmarkierungen) aus allen vom System erstellten Satzendmarkierungen.*Precision* wird folgenderweise berechnet: tp/(tp + fp).

Recall (Trefferquote) ist der Anteil der korrekten Kennzeichnungen im Outputtext *True Positives* (tp) im Verhältnis zu allen tatsächlichen Satzendungen im Goldstandard. *Recall* wird mit der folgenden Formel berechnet: tp/(tp + fn).

F1-Measure ist ein harmonischer Mittelwert, welcher die *Precision* und *Recall* ausgleicht. *F1-Measure* wird mit der folgenden Formel berechnet: 2*(P*R)/ (P + R).

Zum Vergleich der Ergebnisse mit den Resultaten von den oben erwähnten SBD-Systemen wurde ein weiteres Messkriterium, die Fehlerquote, zu den Evaluierungsresultaten hinzugefügt. Die *Error-Rate* (Fehlerquote), in der ML Forschung auch *Accuracy* genannt (David M.W.Powers, 2011), ist eine Methode zur Berechnung der Systemfehler (fp und fn) bezüglich der Fälle (tp, tn, fn, fp), welche das System lieferte. Die Formel lautet dann: fp + fn / (tp + fn + fp + tn).

Mittels untenstehender Python Funktion, welche durch Stefan Schweter für die Evaluierung von EOSV3 im Jahr 2017 programmiert wurde, werden die Ausgabewerte *Precision, Recall, F1-Measure* automatisch kalkuliert:

```
recall    = float(tp_counter / (tp_counter + fn_counter))
recall    = float(tp_counter / (tp_counter + fn_counter))
f_score   = float(2 * ((precision * recall) / (precision + recall)))
```

Weil in vorherigen Forschungen neben obenstehenden Ausgabewerten die Fehlerquote von SBD Systemen rapportiert wurde, ist im Rahmen dieser Bachelorabeit, zudem eine Funktion zur Berechnung eben dieser Fehlerquote im Skript eingebaut worden.

```
errorrate = float(((fp_counter + fn_counter)/len(system_tokens)))
```

Die Ausgabedateien von beiden Systemen, EOSV3 und *OpenNLP*, wurden im Evaluierungsskript mit dem folgenden Befehl verglichen:

```
python3 eval.py -g gold.txt -s systemoutput.txt
```

Das Programm vergleicht zeilenweise die Anzahl der EOS Markierungen in der Golddatei und sowie in der EOS-Outputdatei. Dabei klassifiziert das Evaluationsprogramm die EOS Markierung in der EOS-Outputdatei in *True Positive, True Negative, False Positive, False Negative*. Zudem besteht die Möglichkeit mittels der Funktion *-verbose* im Terminalfenster die Satzendepaare anzuzeigen.

Schlussendlich werden die berechneten Werte für *Precision, Recall, F1-Measure, Error-Rate* dargestellt:

```
...
period.</eos>  period.</eos>  tp
Although ,   Although ,
...
death.   death.</eos>   fp
We We
...
silence)</eos> silence) fn
Madam Madam
...
i.e.  i.e.  tn
on on
...
Precision: 0.9538111676455672
Recall: 0.9716633333333333
Errorrate: 0.0029530427425419905
F1-Score: 0.9626544917150335
```

EOS	**Gold**	**Output**
tp	</eos>	</eos>
tn	x	x
fp	</eos>	x
fn	x	</eos>

Abbildung 5.2: Skript Ausgabe EOS

5.2 Resultate der Evaluierung

In den folgenden Tabellen sind die Resultate der Evaluierung getrennt für die beiden überprüften Programme aufgeführt. Dabei sind die beiden ersten Tabellen gleich aufgebaut. In der ersten Spalte ist der Name des getesteten Korpus aufgeführt. Spalte zwei repräsentiert die Anzahl Satzenden im Goldstandard. In der letzten Spalte sind die Anzahl vom Programm evaluierten Satzenden aufgeführt.

Korpus	GOLD/EOS input	EOSV3 output
Wiki_en	300.000	295.534
Wiki_de	300.000	311.699
Wiki_ru_with_list	300.000	304.410
Wiki_ru_without_list	300.000	298.213
DeReKo	204.637	199.238
OpenCorpora_with_lists	108.960	102.690
OpenCorpora_without_lists	108.960	106.685
Europarl_en	300.000	305.650
Europarl_de	300.000	325.793

Abbildung 5.3: Größe der Korpora im Verhätnis zur EOSV3 Ausgabe

Korpus	GOLD/EOS input	OpenNLP output
Wiki_en	300.000	309.054
Wiki_de	300.000	322.028
DeReKo	204.637	218.684
Europarl_en	300.000	304.836
Europarl_de	300.000	317.476

Abbildung 5.4: Größe der Korpora im Verhältnis zur OpenNLP Ausgabe

Tabelle 5.3 repräsentiert die Anzahl der Sätze im Goldstandard von jedem Korpus im Vergleich zu der EOSV3 Ausgabe, während die Tabelle 5.4 eine Übersicht der *OpenNLPs* Ausgabe darstellt.

Ein wichtiger Unterschied der beiden Programme besteht dahingehend, dass die Sprachmodellenbibliothek im *Apache OpenNLP* keine Angaben für die Sprache Russisch enthält. Diesbezüglich wird aber ermöglicht mit dem Befehl "*SentenceDetectorTrainer*"ein eigenes Modells zu trainieren und später zu evaluieren. Allerdings kann diese Funktion nur dann aufgerufen werden, wenn ein heruntergeladener Textkorpus eine zusätzliche Modelldatei beinhaltet. Für die beiden Korpora *Wiki_ru* und *OpenCorpora* gibt es keine solchen Angaben, daher konnten keine Resultate für die Sprache Russisch mit *OpenNLP* evaluiert werden. Damit verkürzen sich die Einträge der zweiten Tabelle auf fünf Zeilen mit den zwei Englischen sowie den drei Deutschen Korpora.

Die *Wiki* Korpora beinhalten in allen Sprachen 300.000 Sätze als Input sowie im Goldstandard. Die Resultate bei der EOSV3 Outputdatei sind zwischen 295.534 evaluierten Sätze für die englische Sprache bis zu 311.699 Sätze bei der deutschen Sprache ausgegeben.

Im Vergleich dazu liefert das Programm *OpenNLP* 309.054 für den englischen *Wiki* Korpora und 322.028 für den deutschen *Wiki* Korpora. In beiden Fällen findet das Programm *OpenNLP* mehr Satzenden als der EOSV3, jedoch auch in beiden Fällen mehr als im Goldstandard zu finden sind.

Um die Resultate zu interpretieren und die Programme zu vergleichen, müssen die vier obengenannten Evaluierungswerte zu Rate gezogen werden. Die Tabellen 5.5 und 5.6 representieren die Evaluierungswertde für die beiden SBD-Systeme.

Dabei fällt auf, dass der Wert für *Recall* beim *OpenNLP* Output für beide Sprachen Deutsch sowie Englisch in den *Wikipedia* Korpora mit 99.58/99.55% markant höhere Werte ausgibt als der EOSV3 mit 93.34% für Englisch sowie 96.09% für Deutsch. Bei der *Precision* sind die Resultate näher 96.67/92.74% für *OpenNLP* gegenüber 94.80/92.51% beim EOSV3, jedoch gilt auch hier, dass das Programm *OpenNLP* beim englischen Korpus

Korpus	P%	R%	F1%	Err%
Wiki_en	94.80	93.34	94.06	0.61
Wiki_de	92.51	96.09	94.26	0.68
Wiki_ru_with_lists	95.40	96.71	96.05	0.53
Wiki_ru_without_lists	94.78	94.14	94.46	0.74
DeReKo	92.67	90.16	91.40	1.14
OpenCorpora_with_lists	96.79	90.96	93.78	0.81
OpenCorpora_without_lists	**97.13**	90.90	93.91	0.79
Europarl_en	95.38	97.17	**96.26**	**0.26**
Europarl_de	90.12	**97.86**	93.83	0.55

Abbildung 5.5: Ergebnisse des EOSV3

Korpus	P%	R%	F1%	Err%
Wiki_en	**96.67**	**99.58**	**98.10**	**0.20**
Wiki_de	92.74	99.55	96.03	0.48
DeReKo	84.65	90.46	87.46	1.75
Europarl_en	95.95	97.50	96.72	0.26
Europarl_de	92.61	98.00	95.23	0.42

Abbildung 5.6: Ergebnisse des OpenNLP

leicht besser abschneidet. Die niedrigste Fehlerquote beträgt 0.2% auf dem von *OpenNLP* segmentierten *Wiki_en* Korpus. Diese Ergebnisse sind nicht weiter erstaunlich, da der *OpenNLP* laut der Dokumentation [1] auf der Leipziger Korpora Collection trainiert wurde.

Daher ist der DeReKo Korpus besser geeignet um die beiden Programme miteinander zu vergleichen, da beide Programme nicht mittels DeReKo Korpus trainiert wurden. Der DeReKo Korpus beinhaltet 204.637 Sätze in der deutschen Sprache. Die Evaluierung bringt folgende Ergebnisse zu tage: Das Programm *OpenNLP* extrahiert 14.047 Sätze mehr als im Goldstandard vorhanden sind. Währenddessen das Programm EOSV3 nur 5.399 Sätze weniger erkennt als im Goldstandard. Die Evaluierung zeigt die entsprechenden Resultate: Der Wert für *Precision* beträgt für das Programm EOSV3 92.67% und 84.65% für *OpenNLP*. Infolgedessen ist die Fehlerquote bei der EOSV3 niedriger (1.14% bei EOSV3 gegen 1.75% bei *OpenNLP*). Beim Messkriterium *Recall* erzielen beide Programme vergleichbare Werte. Aufgrund dessen, ist das *F1-Measure* bei EOSV3 (91.4%) markant höher, als beim Programm *OpenNLP* (87.46%).

Im *Europarl* Korpora segmentiert EOSV3 die Sätze unterschiedlich für die Sprachen Englisch sowie Deutsch. Der Algorithmus markiert für beide Sprachen mehr Satzenden, als die 300.000, welche im Goldstandard vorhanden sind. In der deutschen Sprache sind in der Outputdatei 325.793 Satzendmarkierungen und 305.650 in der Englischen. Relativ ähnliche Resultate liefert das Programm *OpenNLP* auf dem englischen *Europarl* Korpus. Die Differenz zwischen den Ergebnissen beträgt nur 814 Sätze. Entsprechend erzielen beide Systeme ähnliche Werte beim Evaluierungsskript.

Beim deutschen *Europarl* Korpus ist der Unterschied zwischen *OpenNLP* und EOSV3 markanter. Hier markiert *OpenNLP* insgesamt 317.476 Satzenden. Dies wird auch in den Resultaten des Evaluierungsskript wiedergespiegelt. Auf dem deutschen *Europarl* erzielt das Programm *OpenNLP* bessere Werte als EOSV3, für *Recall* liegt der Wert bei *OpenNLP* bei 98.00% gegen 97.86%, für *Precision* 92.61% gegen 90.12%, für *F1-Measure* 95.23% gegen

[1] https://opennlp.apache.org/docs/1.8.0/manual/opennlp.html

93.83%. Das wiederspiegelt sich auch auf die Fehlerquote, die 0.55% bei EOSV3 und 0.42% bei *OpenNLP* beträgt.

5.3 Zeitaufwand

Das nächste Kriterium, welches bei der Evaluierung eines jeden Programmes beurteilt wird, ist der Zeitaufwand. In der folgenden Tabelle werden die Resultate für beide Systeme *EOSV3* und *OpenNLP* dargestellt.

Korpus	**EOSV3 (sek)**	**OpenNLP (sek)**
Wiki_de	10.70	6.329
Wiki_en	9.63	6.305
Wiki_ru	12.46	x
DeReKo	7.08	**4.163**
OpenCorpora	**4.79**	x
Europarl_de	11.83	7.901
Europarl_en	10.42	7.184

Abbildung 5.7: Zeitaufwand von *EOSV3* und *OpenNLP*

Aus der Abbildung 5.7 lässt sich entnehmen, dass *EOSV3* im Vergleich zu *OpenNLP* länger für die Kalkulation der Resultate benötigt. Das beste Resultat bringt *OpenNLP* im Korpus *DeReKo*, der aus 204.637 Sätze besteht, während *EOSV3* das beste Resultat bei *OpenCorpora* mit 4.79 Sekunden bei 108.960 Sätzen erreicht.

5.4 Vergleich diverser SBD Systeme

Wie weiter oben erwähnt, wurde im Rahmen dieser Arbeit die *Error-Rate* der evaluierten Systemen mit vorangehenden Forschungen im Bereich SBD verglichen.

Systemname	**Korpus**	**Beste Err%**
Alembic	WSJ	0.9
MxTerminator	WSJ	1.2
Riley	Brown	0.2
SATZ	German News	0.7
Mikheev	Brown	0.35
Punkt	WSJ	0.35
EOSV3	Europarl_en	0.26
OpenNLP	Wiki_en	0.20

Abbildung 5.8: Error-Rate Vergleich

Gemessen an der *Error-Rate* liefert das Programm *EOSV3* die drittniedrigste Fehlerquote in der Abbildung. Dabei wurden alle Systeme anhand eines englischen Korpora gemessen, so dass die Resultate eine gewisse Vergleichbarkeit zulassen. Dennoch muss an dieser Stelle erwähnt werden, dass die Resultate je nach Korpus unterschiedliche Werte aufweisen können.

5.5 Experimente

EOSV3 kann von den Nutzern mittels JSON *Config Agent* gesteuert werden. Um herauszufinden, welche *Agenten* die Resultaten beeinflussen, wurde mit dem Parameterset

"deactivated-agents" experimentiert. Als Ausgangslage für die russischen Korpora ist die EOSV3 Version mit den russischen Frequenzlisten gemeint.

In den Tabellen 5.9 - 5.15 werden die Ergebnisse der Experimentevaluierung auf den sieben Korpora dargestellt. In der ersten Zeile in der Farbe rot werden die Evaluierungswerte für die Basiskonfigurationeinstellungen angezeigt, welche die Funktion von allen *Agenten* miteinschließt. Mit der grünen Farbe werden die Angaben für die getrennt ausgeschalteten *Agenten* veranschaulicht, in gelb werden die Resultate für die Evaluierung mittels einem einzelnen *Agenten* dargestellt. Dabei steht in der ersten Spalte der Name des ausgeschalteten *Agenten* und in den nächsten 4 Spalten die Evaluierungswerte für *Precision, Recall, F1-Measure* und *Error-Rate*.

Agents:	P%	R%	F1%	Err%
default Config	92.51	96.09	**94.26**	**0.68**
without Abbr	90.49	96.32	93.31	0.80
without Cross	**92.55**	95.77	94.13	0.69
without Primus	80.84	82.89	81.85	2.14
without Regex	73.76	86.99	79.83	2.55
only Abbr	84.60	99.50	91.45	1.08
only Cross	80.57	**99.98**	89.23	1.40
only Primus	80.57	**99.98**	89.23	1.40
only Regex	90.58	95.05	92.76	0.86

Abbildung 5.9: Wiki_de Evaluation

Agents:	P%	R%	F1%	Err%
default Config	96.79	90.96	93.78	0.81
without Abbr	96.30	91.80	**94.00**	**0.79**
without Cross	96.80	90.90	93.76	0.82
without Primus	**96.89**	90.79	93.74	0.82
without Regex	94.87	91.84	93.33	0.88
only Abbr	94.87	91.71	93.26	0.89
only Cross	92.37	**92.97**	92.67	0.99
only Primus	92.37	**92.97**	92.67	0.99
only Regex	96.41	91.71	**94.00**	**0.79**

Abbildung 5.10: OpenCorpora Evaluation

Die Tabelle 5.9 enthält die Werte für den deutschen *Wikipedia* Korpus. Die besten Resultate für *Precision* erzielt das System ohne *Cross Agent*. Für *Recall* sind die Werte bei *Cross/Primus* am höchsten. Für *F1-Measure* sowie für die *Error-Rate* erzielt das System in den Basiseinstellungen die besten Resultate.

In der Tabelle 5.10 sind die Resultate der Evaluierungsexperimente für den russischen *OpenCorpora* erörtert. Die besten Werte für *Precision* erzielt das System ohne den *Primus Agenten*. Bei der Evaluierung des *Recall* bringt das System welches nur auf den *Agenten Cross/Primus* zurückgreift die höchsten Werte. Das *F1-Measure*, wie auch die *Error-Rate*, ist bei den Einstellungen ohne *Abbreviation Agent* am besten.

Agents:	P%	R%	F1%	Err%
default Config	95.40	96.71	96.05	0.53
without Abbr	94.50	97.07	95.76	0.57
without Cross	95.41	96.62	96.01	0.54
without Primus	**95.47**	95.98	95.73	0.57
without Regex	95.02	99.58	97.24	**0.38**
only Abbr	95.11	99.49	**97.25**	**0.38**
only Cross	90.51	**99.94**	94.99	0.71
only Primus	90.51	**99.94**	94.99	0.71
only Regex	94.53	96.39	95.45	0.61

Abbildung 5.11: Wiki_ru Evaluation

Agents:	P%	R%	F1%	Err%
default Config	**94.80**	93.34	94.06	0.61
without Abbr	92.62	94.08	93.34	0.70
without Cross	94.96	92.40	93.66	0.65
without Primus	6.92	6.73	6.82	9.59
without Regex	6.85	7.21	7.03	9.95
only Abbr	94.04	98.41	**96.17**	**0.41**
only Cross	88.47	**99.69**	93.75	0.69
only Primus	88.47	**99.69**	93.75	0.69
only Regex	92.91	91.91	92.41	0.79

Abbildung 5.12: Wiki_en Evaluation

Die Tabelle 5.11 wiederspiegelt die Resultate für den russischen *Wikipedia* Korpus. Das System ohne *Primus Agent* liefert dabei die beste *Precision*. Bei *Recall* gilt wiederum, dass die Systeme nur mit *Cross/Primus* die besten Resultate liefern. Das *F1-Measure* schneidet mittels *Abbreviation Agent* am besten ab. Die *Error-Rate* ist bei den Modellen ohne *Regex Agent* resp. nur *Abbreviation Agent* am niedrigsten.

In der Tabelle 5.12 werden die Evaluierungswerte für den englischen *Wikipedia*Korpus angezeigt. Die besten Resultaten für *Precision* zeigt das Modell mittels Basiskonfiguration. Die besten Resultate bei der *Recall* bildet das Modell mittels *Cross/Primus Agent*. Das höchste *F1-Measure* erzielt das Modell mittels *Abbrevation Agent*, dieses Modell erzielt zudem die niedrigste *Error-Rate*.

Agents:	P%	R%	F1%	Err%
default Config	**92.67**	90.16	**91.40**	**1.14**
without Abbr	84.38	91.37	87.74	1.72
without Cross	92.66	89.95	91.28	1.16
without Primus	37.96	36.74	37.34	8.31
without Regex	8.96	10.05	9.48	12.95
only Abbr	84.25	94.19	88.94	1.58
only Cross	74.94	**96.70**	84.44	2.40
only Primus	74.94	**96.70**	84.44	2.40
only Regex	84.32	90.90	87.45	1.75

Abbildung 5.13: DeReKo Evaluation

Agents:	P%	R%	F1%	Err%
default Config	**95.38**	97.17	96.26	0.26
without Abbr	95.34	97.79	96.54	0.27
without Cross	95.40	96.74	96.06	0.31
without Primus	12.43	12.57	12.50	6.89
without Regex	12.43	12.76	12.60	6.94
only Abbr	95.36	97.50	96.42	0.28
only Cross	95.29	**98.53**	**96.88**	**0.25**
only Primus	95.29	**98.53**	**96.88**	**0.25**
only Regex	95.37	97.17	96.26	0.29

Abbildung 5.14: Europarl_en Evaluation

In der Tabelle 5.13 werden die Evaluierungswerte für den *DeReKo* Korpus angezeigt. Das System mittels Basiskonfiguration bietet für alle Messkriterien die besten Resultate. Nur bei dem *Recall* Kriterium schneidet das Modell mittels *Cross/Primus Agent* besser ab.

Tabelle 5.14 zeigt die Evaluierungswerte für den *Europarl_en* Korpus. Das System mittels Basiskonfiguration weist die höchste *Precision* aller Systeme aus. In allen anderen Messkriterien hat das Modell mittels *Cross/Primus* die besten Resultate.

Agents:	P%	R%	F1%	Err%
default Config	90.12	97.86	93.83	**0.55**
without Abbr	89.22	98.03	**94.43**	0.59
without Cross	90.14	97.79	93.81	**0.55**
without Primus	**90.16**	97.68	93.77	**0.55**
without Regex	88.78	98.53	93.40	0.60
only Abbr	88.80	98.46	93.38	0.59
only Cross	87.61	**98.71**	92.83	0.65
only Primus	87.61	**98.71**	92.83	0.65
only Regex	89.25	97.88	93.36	0.59

Abbildung 5.15: Europarl_de Evaluation

Tabelle 5.15 zeigt die Evaluierungswerte für den *Europarl_de* Korpus. Die besten Werte für *Precision* erzielt das System ohne den *Primus Agenten*. Bei der Evaluierung des *Recall* bringt das System welches nur auf den *Agenten Cross/Primus* zurückgreift die höchsten Werte. Das *F1-Measure*, ist bei den Einstellungen ohne *Abbreviation Agent* am höchsten. Die niedrigste *Error-Rate* liefert das System mittels Basiseinstellungen.

5.6 Beobachtungen aus dem Experiment

Aus dem Unterkapitel 5.5 Experimente ist hervorgegangen, dass EOSV3 ein Programm ist, dessen Algorithmen effektiv kombiniert wurden. Die Verknüpfung von allen *Agenten* bringt die ausgewogensten Resultate bei der Evaluierung des Systems. Das bedeutet, dass die Werte von *Precision, Recall, F1-Measure* und *Error-Rate* vor allem in der Basiskonfiguration gut ausgeglichen sind.

Zwar können einige *Agenten* die Werte für den *Recall* erhöhen, jedoch wird dabei der Wert für *Precision* verringert. Insbesondere können die *Agenten Primus* und *Cross* viele zusätzliche korrekte Satzendezeichen erkennen (*True Positives*), aber auch viele fehlerhafte Satzendezeichen (False Positives). Dies lässt sich aus den Tabellen 5.9, 5.12, 5.13 schlussfolgern. Betrachtet man die Formel im Unterkapitel 2.3.3 beziehungsweise 2.3.4 so ist klar, dass ein Modell, welches nur auf die *Agenten Primus/Cross* zurückgreift jedes potentielle Satzende als solches markiert. Damit erreicht dieses System hohe Werte im Messkriterium

Recall, jedoch generiert es auch mehr Fehler (*False Positives*) was sich in Form einer geringeren *Precision* auswirkt. Zudem erklärt dieser Umstand warum die beiden Modelle mittels den *Agenten Primus* beziehungsweise Cross deckungsgleiche Resultate liefern. Dies liegt wohl an der obenerwähnten Formel, welche für beide Modelle immer positive Resultate liefert und somit setzen beide Modelle die gleichen EOS Markierungen.

Im Allgemeinen gilt für die beiden englischen Korpora *Wiki,Europarl*, dass das Modell ohne die *Agenten Primus* sowie *Regex* sehr tiefe Werte für *Precision*, *Recall* sowie *F1-Measure* erzielt. Dies lässt sich damit erklären, dass im Englischen nur die Eigennamen, Akronyme sowie Satzanfangswörter grossgeschrieben werden. Deswegen ist der *Primus Agent* im Englischen zentral für die Findung von Satzendezeichen. Der *Regex Agent* ist zentral für die Auffindung der obenbeschriebenen Eigennamen und Akronyme.

Wie in der englischen Sprache gilt auch im Russischen, dass nur Eigennamen, Akronyme sowie Satzanfänge grossgeschrieben werden. Aufgrund der offenen Satzstellung ist die Bildung von Frequenzlisten in der russischen Sprache weniger relevant. Im russischen kann derselbe Satz auf verschiedene Arten umgestellt werden, ohne dass der Sinn im Satz geändert wird. Dadurch können unterschiedliche Wörter am Satzanfang stehen, was sich wiederum negativ auf die Bildung von Frequenzlisten auswirkt. Dies lässt sich in den Abbildungen 5.10 sowie 5.11 daran erkennen, dass das weglassen einzelner *Agenten* fast keinen Einfluss auf die Resultate ausübt. Bei den deutschen Korpora gilt, dass die beiden *Agenten Primus* und *Regex* den grössten Einfluss auf die Resultate ausüben wie sich den Abbildungen 5.9 sowie 5.13 entnehmen lässt. Dies gilt jedoch nicht beim Korpus *Europarl_ de*. Bei der Analyse der *false positive* im *Europarl* Korpora konnte im Rahmen dieser Arbeit jedoch festgestellt werden, dass der Goldstandard viele fehlerhafte Satzendmarkierungen aufweist. Somit werden diese Resultate nicht weiter kommentiert.

6 Fazit

Im Rahmen dieser Arbeit wurde eine Übersicht über den aktuellen Forschungsstand im Bereich SBD geliefert. Desweiteren wurde der Leser in das Programm EOSV3 eingeführt. Ein Hauptziel dieser Arbeit war es EOSV3 um die Sprache Russisch zu erweitern. Dieses Unterfangen konnte erfolgreich abgeschlossen werden. Die Resultate aus der Evaluierung der russsischen Korpora sind alle zufriedenstellend. Die Werte aus der Evaluierung sind vergleichbar mit den bereits etablierten Sprachen Deutsch und Englisch. Es sollte an dieser Stelle jedoch erwähnt werden, dass die Frequenzlisten für die russische Sprache einen weniger starken Einfluss haben als in den anderen Sprachen. Zukünftige Arbeiten sollten daher weitere Korpora testen um, die im Rahmen dieser Bachelorarbeit erhaltenen Resultate, zu verifizieren.

Zudem wurde im Rahmen dieser Arbeit das Programm EOSV3 evaluiert. EOSV3 hat sich als ein akurates Satzendeerkunngsprogramm erwiesen. Es kann flexibel auf diverse Sprachen erweitert werden. Die Evaluationskriterien weisen bei allen Korpora über 90% auf und liefern somit gute Resultate. In der manuellen Analyse der Fehlerdaten *False Positive* ist der Autorin jedoch aufgefallen, dass EOSV3 bei Zahlen gefolgt von einem Grossbuchstaben fälschlicherweise ein Satzende marktiert. So gilt zum Beispiel der *10. April* als Satzende wobei es sich jedoch um einen *False Positive* handelt. Dies könnte im Rahmen einer zukünftigen Arbeit verbessert werden. Ein weiterer Fehler der so aufgedeckt wurde, handelt um die Behandlung der Doppelpunkte. Diese werden von EOSV3 immer als Satzende markiert, wobei die vorliegenden Korpora diese nicht als Satzende taxiert haben. Die Behebung dieser beiden Aspekte könnte die guten Evaluierungsresultate des EOSV3 weiter verbessern.

Im Rahmen des Unterkapitel Experimente konnte zudem aufgezeigt werden, dass einige *Agenten* stärkeren Einfluss nehmen auf die Resultate als andere *Agenten*. Gleichzeitig haben jedoch alle *Agenten* eine feste Gewichtigung, die durch die Konfigurationsdatei bestimmt wird. Es wäre interessant zu sehen, ob eine Parametrisierung, dieser Gewichtung dazu führen könnte, dass EOSV3 noch bessere Resultate erzielt. Dabei könnte beispielsweise das *F1-Measure* als Maximierungsvariable verwendet werden und so die Agentengewichtung optimiert werden.

References

Aberdeen, J., J Burger, D. Day, L. Hirschman, P. Robinson, and M. Vilain. 1995. Mitre: Description of the alembic system used for muc-6. In The Proceedings of the Sixth Message Understanding Conference (MUC-6),

Bray T. The javascript object notation (json) data interchange format.

Breiman L, Friedman J, Stone CJ, Olshen RA. Classification and regression trees. CRC press; 1984.

Dale R, Moisl H, Somers H, editors. Handbook of natural language processing. CRC Press; 2000 Jul 25.

Dunning T. Accurate methods for the statistics of surprise and coincidence. Computational linguistics. 1993 Mar 1;19(1):61-74.

Gillick, D. (2009, May). Sentence boundary detection and the problem with the US. *In Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers* (pp. 241-244). Association for Computational Linguistics.

Kiss T, Strunk J. Unsupervised multilingual sentence boundary detection. Computational Linguistics. 2006 Dec;32(4):485-525.

Koehn, P. (2005, September). Europarl: A parallel corpus for statistical machine translation. In MT summit (Vol. 5, pp. 79-86).

Kupietz M, Belica C, Keibel H, Witt A. The German Reference Corpus DeReKo: A Primordial Sample for Linguistic Research. InLREC 2010 May.

Lippmann R. An introduction to computing with neural nets. IEEE Assp magazine. 1987 Apr;4(2):4-22.

Mikheev, A. (2000, April). Tagging sentence boundaries. *In Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference* (pp. 264-271). Association for Computational Linguistics.

Mikheev, A. (2002). Periods, capitalized words, etc. Computational Linguistics, 28(3), 289-318.

Milchin, A. (2015). Справочник издателя и автора. *Студия Артемия Лебедева.*

Mohri M, Rostamizadeh A, Talwalkar A. Foundations of machine learning. MIT press; 2012 Aug 17.

Palmer, D. D., Hearst, M. A. (1997). Adaptive multilingual sentence boundary disambiguation. Computational Linguistics, 23(2), 241-267.

Powers, D. M. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.

Read, J., Dridan, R., Oepen, S., Solberg, L. J. (2012). Sentence boundary detection: A long solved problem?. COLING (Posters), 12, 985-994.

Reynar, J. C., Ratnaparkhi, A. (1997, March). A maximum entropy approach to identifying sentence boundaries. *In Proceedings of the fifth conference on Applied natural language processing* (pp. 16-19). Association for Computational Linguistics.

Riley, M. D. (1989, October). Some applications of tree-based modelling to speech and language. *In Proceedings of the workshop on Speech and Natural Language* (pp. 339-352). Association for Computational Linguistics.

Schweter S. (2013). Performanzoptimierung, Einsatz und Evaluation des C++ Programms EOS_V2 zur Satzendeerkennung.

White, M. (1995). Presenting punctuation. arXiv preprint cmp-lg/9506012.

Zimmermann, H. H. (2006). Zur Leistung der Satzzeichen.

Abbildungsverzeichnis

CD Contents

Korpora

1. europarl_de_gold
2. europarl_en_gold
3. wiki_de_gold
4. wiki_en_gold
5. wiki_ru_gold
6. OpenCorpora_gold

Code

1. eos_marks.py
2. evaluierungsskript.py
3. make_eos_input.py

Bachelor Latex

Papers

1. Gillick
2. Kiss, Strunk
3. Mikheev, Mikheev 2
4. Palmer, Hearst
5. Reynar, Ratnaparkhi
6. Riley
7. Stefan Schweter Bachelor

Russische Listen

1. Primus_frequency_list_ru
2. regex_list_ru
3. ru_abk
4. rilower_words