INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Entwicklung eines Systems zur Stilanalyse wissenschaftlicher Veröffentlichungen

Christoph Richard Danell

Bachelorarbeit

Beginn der Arbeit:	03. Februar 2015
Abgabe der Arbeit:	08. Mai 2015
Gutachter:	Prof. Dr. Stefan Conrad
	Prof. Dr. Martin Mauve

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 08. Mai 2015

Christoph Richard Danell

Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde ein System zur sprachlichen und stilistischen Analyse englischsprachiger Texte aus akademischen Veröffentlichungen in der Programmiersprache Java entwickelt.

Hierzu wurde als Grundlage eine Natural Language Processing Pipeline, die Stanford CoreNLP, verwendet, die verschiedene Funktionen zur Analyse einer natürlichen Sprache bereitstellt. Anhand der so ermittelten Daten wurden verschiedene Methoden entwickelt, um englischsprachige Texte auf verschiedene stilistische oder sprachliche Mängel zu untersuchen und die Ergebnisse für den Anwender zu visualisieren.

Im Verlauf der Arbeit wird sowohl auf die Funktionen der Stanford CoreNLP, als auch auf die entwickelten Methoden eingegangen. Hierbei werden eventuelle Probleme und Lösungsansätze bei der Implementierung beschrieben, sowie Beispiele für die verschiedenen Analyseziele aufgezeigt.

In einer Evaluation wird das entwickelte System in einem realistischen Anwendungsszenario getestet und die daraus gewonnenen Ergebnisse präsentiert.

Inhaltsverzeichnis

1 Einleitung

Dieses Kapitel dient als Einleitung in die Arbeit. Es werden die Motivation, Aufgabenstellung, Zielsetzung und der Aufbau der Arbeit beschrieben.

1.1 Motivation

Da in der heutigen Zeit immer öfter von Akademikern weltweit erwartet wird die englische Sprache zu beherrschen, um eine geeignete Kommunikationsbasis zu schaffen, werden viele akademische Publikationen in Englisch verfasst. Dabei handelt es sich bei einer Vielzahl von Akademikern nicht um Muttersprachler, weswegen Schwierigkeiten auftreten können, einen geeigneten Sprachstil einzuhalten. Doch selbst bei Muttersprachlern der englischen Sprache kann es, besonders bei Anfängern im Bereich des akademischen Schreibens, zu Unsicherheiten kommen.

Aufgrund der Fähigkeiten moderner Computersysteme, besteht die Möglichkeit Unsicherheiten im akademischen Schreibstil vorzubeugen, indem die Überprüfung des Sprachstils in Texten automatisiert wird.

1.2 Aufgabenstellung und Zielsetzung

Im Rahmen dieser Bachelorarbeit soll eine Anwendung zur Analyse englischsprachiger Texte entwickelt werden, die den Autor auf stilistische sowie sprachliche Mängel hinweisen kann.

Hierzu wird ein vom Anwender gewählter Text zunächst mit einer Natural Language Processing Pipeline (NLP) analysiert und die so erhaltenen Daten in einer für die Analyse geeigneten Datenstruktur organisiert.

Daraufhin wird eine Abfolge von Funktionen (Tools) aufgerufen, welche den Text über die Datenstruktur gezielt auf verschiedene sprachliche Mängel untersucht.

Die Ausgabe der Anwendung erfolgt über eine HTML Datei, in der die Analyseergebnisse der einzelnen Tools aufgelistet und Fundstellen im Text farblich markiert werden.

1.3 Aufbau der Arbeit

Das erste Kapitel dieser Bachelorarbeit dient als Einleitung in die Arbeit und das behandelte Thema. In Kapitel 2 wird auf die zum Verständnis der Arbeit nötigen Grundlagen eingegangen. Es werden sowohl das Konzept der maschinellen Sprachverarbeitung, als auch eine Einführung in die Stanford CoreNLP behandelt. Kapitel 3 und 4 beschäftigen sich mit dem für die Bachelorarbeit entwickelten System und bieten einen Überblick über das Konzept und die Implementierung. Dabei wird insbesondere auf die einzelnen Tools eingegangen und deren Funktionsweise erklärt. In Kapitel 5 wird in einer Evaluation auf die Möglichkeiten der Anwendung und Probleme des Systems eingegangen. Zum Schluss wird in Kapitel 6 eine Zusammenfassung der Ergebnisse sowie ein Ausblick auf mögliche weiterführende Arbeiten gegeben.

2 Grundlagen

Dieses Kapitel gibt einen kurzen Einblick in die grundlegende Herangehensweise bei der maschinellen Sprachverarbeitung und beschreibt die für diese Arbeit benutzte NLP, die *Stanford CoreNLP* [MSB+14, SCN15]. Die für das entwickelte Programm relevanten Funktionen der Stanford CoreNLP werden einzeln beschrieben

2.1 Maschinelle Sprachverarbeitung

Maschinelle Verarbeitung natürlicher Sprache ist ein Bereich der Computerlinguistik, welche im Überschneidungsbereich von Informatik und Linguistik angesiedelt ist. Eine menschliche Sprache wird in der Computerlinguistik oft als natürliche Sprache bezeichnet, um eine deutliche Unterscheidung von einer maschinellen Sprache zu treffen.

Nachdem bereits im Jahr 1950 erste Gedankenexperimente zur maschinellen Sprachanalyse existierten, haben sich über die Jahre verschiedene Ansätze herausgebildet. Ein Ansatz ist dabei das korpusstatistische Verfahren, bei dem die Grundlage ein speziell aufgearbeiteter Korpus von Sprachdaten ist. Die Ermittlung geeigneter Modellparameter aus diesem Korpus wird oftmals als Training bezeichnet [CEE+09].

2.2 Stanford CoreNLP

Die in dieser Arbeit benutzte NLP, die *Stanford CoreNLP*, ist eine Java Anwendung, die verschiedene Funktionen für die Analyse geschriebener menschlicher Sprache bereitstellt. Die Stanford CoreNLP kann in der Konsole ausgeführt werden, stellt aber zusätzlich eine Programmierschnittstelle (API[1]) für eigene, in der Programmiersprache Java geschriebene, Anwendunen bereit. Die verschiedenen Funktionen werden nach Bedarf vom Anwender gewählt und nacheinander in Form einer Pipeline ausgeführt.

Die Stanford CoreNLP kann jeweils nur eine Sprache (beispielsweise Englisch oder Deutsch) gleichzeitig analysieren. Dafür greift sie auf zuvor erstellte Dateien zu, die *Models* genannt werden. Jedes *Model* entspricht dabei einer bestimmten Sprache und muss vorher über die Stanford CoreNLP erstellt werden, indem die Pipeline automatisch, über einen lernfähigen Algorithmus, trainiert wird.

In den folgenden Abschnitten wird genauer auf die, in dieser Arbeit verwendeten, Komponenten der Stanford CoreNLP eingegangen.

2.2.1 Tokenizer

Der *Tokenizer* hat die Aufgabe einen Text in kleinere Einzelteile zu zerlegen, die *Tokens* genannt werden. Die Tokens entsprechen, bis auf wenige Ausnahmen, den Wörtern und Satzzeichen einer menschlichen Sprache. Ein Text wird dabei hauptsächlich an Stellen zerteilt, an denen Leerzeichen auftreten. Dabei werden die Leerzeichen selbst ignoriert und nicht in eigenständige Token geladen.

[1] nlp.stanford.edu/nlp/javadoc/javanlp/

Dieses Kriterium reicht nicht aus, um einen englischen Text zu zerteilen, da beispielsweise Satzzeichen meistens direkt an einem Wort anliegen, jedoch in eigenen Token repräsentiert werden. Werden die Satzzeichen hingegen dazu genutzt eine Abkürzung (z. B. *Mrs.*) oder eine Zahl (z. B. *42.1*) darzustellen, wird der Text an dieser Stelle nicht getrennt, wie das folgende Beispiel demonstriert.

Originalsatz:
Mrs. Peterson paid $245.50 for a used car.

Nach Anwendung des Tokenizers:
| Mrs. | Peterson | paid | $ | 245.50 | for | a | used | car | . |

Der hier verwendete Tokenizer wendet zudem spezielle Regeln an, welche für die Englische Sprache optimiert sind. Wörter, die ein Apostroph enthalten, werden in zwei unabhängige Tokens gespalten, selbst wenn es sich dabei um ein einzelnes Wort handelt (*Peter's* ist die Deklination eines einzigen Wortes, *doesn't* besteht hingegen aus den zwei Wörtern *does* und *not*). Enthält ein Wort jedoch einen Bindestrich, werden die Teile vor und nach dem Bindestrich nicht getrennt (wie bei *father-in-law* oder *six-year-old*). Wörter, die zwar eine logische Einheit bilden, aber aus zwei getrennten Wörtern bestehen (siehe *ice cream*), werden vom Tokenizer erwartungsgemäß in zwei Token gespalten.

Originalsatz:
Peter's six-year-old son doesn't like ice cream.

Nach Anwendung des Tokenizers:
| Peter | 's | six-year-old | son | does | n't | like | ice | cream | . |

Manche Satzzeichen werden vom Tokenizer, der in der Stanford CoreNLP benutzt wird, umgeschrieben. Klammern bekommen beispielsweise ein spezielles Kürzel zugewiesen (runde Klammer ⇒ **-LRB-** bzw. **-RRB-**, geschweifte Klammer ⇒ **-LCB-** bzw. **-RCB-**). Dieses Verhalten lässt sich jedoch bei Aufruf der Stanford CoreNLP über diverse Parameter deaktivieren.

2.2.2 Sentence Splitter

Der *Sentence Splitter* wird nach dem *Tokenizer* aufgerufen und hat die Aufgabe eine Folge von Tokens in Sätze aufzuteilen. Dabei sucht der *Sentence Splitter* nach Tokens, die einem Satzende entsprechen, wie beispielsweise einem Punkt, Ausrufezeichen oder Fragezeichen (., *!*, *?*), sowie einer Kombination aus diesen (z. B. *!?!*). Eine Ausnahme bildet eine Folge von drei Punkten (...), die nicht als Satzende erkannt wird. Bei einer Folge von mehr als drei Punkten werden die restlichen Punkte ignoriert und genau so verfahren, wie bei einer Folge von drei Punkten.

Befinden sich hinter dem Token, welches als Satzende erkannt wurde, weitere Satzzeichen, wie beispielsweise eine geschlossene Klammer oder Anführungszeichen, werden diese ebenfalls dem Satz zugeteilt.

Wurde beim Aufruf der Stanford CoreNLP zusätzlich die Option `tokenizeNLs=true` für den *Tokenizer* gesetzt, werden Sätze ebenfalls nach dem Aufkommen von zwei aufeinander folgenden Zeilenumbrüchen getrennt. Diese Funktion ist nützlich, falls ein Text in betitelte Abschnitte zerteilt ist, da Titel meistens nicht mit einem Satzzeichen enden und ohne die genannte Option, an den darauffolgenden Satz gebunden werden.

2.2.3 POS Tagger

Ein *Part-Of-Speech Tagger* (POS Tagger) analysiert jedes Token und weist diesem ein *Part-Of-Speech Tag* (POS Tag) zu. Dieses enthält Informationen über die Wortart eines Tokens und klassifiziert ein Wort beispielsweise als Adjektiv, Verb oder Nomen. Zudem enthält es, je nach Wortart, zusätzliche Informationen über den verwendeten Tempus, den Numerus oder sonstige Eigenschaften des Wortes.

Der in der Stanford CoreNLP verwendete POS Tagger nutzt die im Penn Treebank Tagset definierten POS Tags (siehe Tabelle 1 und [MMS93]). Manche POS Tags werden jedoch gar nicht genutzt, beispielsweise das Tag **FW** für Fremdwörter oder **SYM** für Symbole.

Das folgende Beispiel zeigt einen Satz, bei dem den einzelnen Tokens jeweils ein POS Tag zugewiesen wird. Zur besseren Übersicht werden diese farblich markiert.

This /DT is/VBZ a/DT sample/NN sentence/NN ./.

Dabei ist zu beachten, dass ein regelbasierter Tagger eine Genauigkeit von ungefähr 90% erreicht [CEE+09]. Beim POS Tagger des Stanford CoreNLP beträgt die Genauigkeit 97,24% [TKMS03].

Tag	Beschreibung
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
IN	Preposition or subordinating conjunction
JJ	Adjective
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
POS	Possessive ending
PRP	Personal pronoun
VB	Verb, base form
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present

Tabelle 1: Beispiele für POS Tags aus dem Penn Treebank Tagset [MMS93]

2.2.4 Lemmatizer

Der *Lemmatizer* hat die Aufgabe jedem Token ein *Lemma* zuzuweisen. Ein *Lemma* entspricht der Grundform eines Wortes. Die Grundform des entsprechenden Wortes richtet sich dabei nach der Wortart: Bei Verben wird der Infinitiv gebildet, bei Nomen hingegen die nicht deklinierte Form im Singular. Im Folgenden sind einige Beispiele genannt.

variables → *variable*
saved → *save*
has → *have*
are → *be*
an → *a*

2.2.5 Named Entity Recognizer

Der *Named Entity Recognizer* dient dazu einem Wort ein *Named Entity Tag* zuzuweisen. Ein *Named Entity Tag* entspricht jeweils einer von mehreren Gruppen, die verschiedenen Entitäten entsprechen, beispielsweise einer Organisation oder einem Ort. Wörter, die keiner Gruppe zugewiesen werden können, erhalten das Tag **0**.

Folgende Gruppen stehen dabei zur Auswahl:

- Person
- Location
- Organization
- Misc
- Money
- Number
- Ordinal
- Percent
- Date
- Time
- Duration
- Set

Über die Erweiterung *RegexNER* werden zusätzliche Gruppen eingeführt.

- Religion
- Ideology
- Title
- Nationality

Hierbei wird das Wort mit einer Liste von fest definierten Einträgen abgeglichen, über welche die Zugehörigkeit ermittelt wird. Der Anwender kann eine eigene Liste erstellen oder die vorhandene erweitern, um dadurch zusätzliche Gruppen zu definieren.

Die Einteilung der Wörter in Gruppen ist nicht fehlerfrei. Beispielsweise wird die Religion *Judaism* als Organisation erkannt, *Scientology* hingegen wird weder als Religion noch als Organisation klassifiziert. Bei den Wörtern *left* und *right* wird immer das Tag *Ideology* gesetzt, selbst wenn es sich um Richtungsangaben handelt. Um hierbei eine höhere

Genauigkeit zu erreichen, müssen die Listen angepasst werden. Allerdings steigt dabei auch die Wahrscheinlichkeit, dass Wörter, die eigentlich keiner Kategorie angehören, eine, in dem Fall falsche, Gruppe zugewiesen bekommen. Beispielsweise würde das Wort *Apple* nach einem Eintrag in die Liste immer als Organisation klassifiziert werden, selbst wenn es sich um das englische Wort für *Apfel* handelt.

3 Konzept

In diesem Kapitel wird das generelle Konzept der entwickelten Anwendung beschrieben. Die Anwendung wird im weiterem Verlauf der Arbeit als *Academic Style Analyser* (ASA) bezeichnet. Alle relevanten Schritte des ASA werden im Folgenden aufgelistet und genauer beschrieben.

3.1 Genereller Ablauf

Der Academic Style Analyser wurde in der Programmiersprache Java geschrieben und ist dadurch plattformunabhängig. Das Hauptprogramm hat einen linearen Aufbau. Die verschiedenen Schritte, die bei einem Aufruf des Hauptprogramms des ASA ausgeführt werden, sind in Abbildung 1 anschaulich dargestellt und werden in diesem Abschnitt genauer beschrieben.

3.2 Übergabe der Parameter und Vorbereitung der Quelldatei

Beim Aufruf des ASA wird zunächst überprüft, ob das Programm mit einer Liste von Parametern gestartet wurde. Bei einem Aufruf ohne Parameter wird die reduzierte grafische Oberfläche des ASA gestartet, in der jedes Tool des ASA ausgewählt werden kann. Möchte der Anwender auf die einzelnen Optionen der Tools zugreifen, ist dies nur über die Konsole möglich. Wird die Anwendung mit Parametern aufgerufen, werden diese direkt an den Hauptteil des Programms weitergeleitet und die Anwendung als Konsolenanwendung gestartet.

Im Hauptprogramm werden zuerst die eingegebenen Parameter extrahiert und überprüft, ob eine zulässige Eingabe vorliegt. Eine Tabelle mit den zulässigen Parametern befindet sich im Abschnitt 4.1.1 auf Seite 11.

Wurde ein Dateipfad mit dem akzeptierten Dateityp (`.txt`) gefunden, wird der Pfad an den *File Extractor* übergeben. Diese Klasse hat die Aufgabe die Quelldatei am angegebenen Pfad zu öffnen, den Inhalt einzulesen und in Form eines Strings zu speichern.

3.3 Der Model Builder und das Text Data Model

Sobald der Inhalt der Quelldatei als String vorliegt, wird dieser an Stanford CoreNLP weitergeleitet, der Text in einzelne Tokens zerteilt und in Sätzen gruppiert. Alle für die Analyse notwendigen Daten werden danach in eine eigene Datenstruktur, das *Text Data Model* (TDM), übertragen.

Im TDM werden grundlegende Daten zur absoluten Anzahl aller Tokens, sowie die absolute Anzahl der Sätze gespeichert, aus denen der zuvor mit dem Tokenizer zerteilte Originaltext besteht. Einzelne Tokens werden durch das Objekt *Token* repräsentiert. Ein Token beinhaltet Informationen zum enthaltenen Wort aus der Stanford CoreNLP, beispielsweise die Grundform des Wortes, ein entsprechendes POS Tag, sowie einige nachträglich ermittelte Informationen, wie die Angabe, ob es sich bei einem Wort um ein Sonderzei-

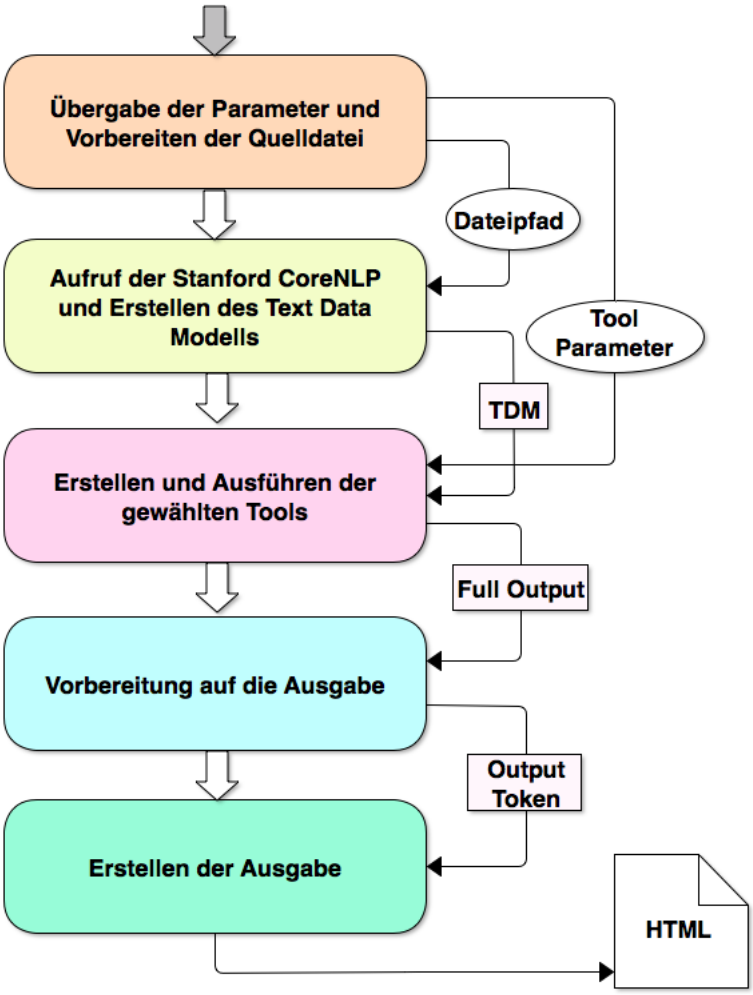

Abbildung 1: Ablauf des Academic Style Analyser

chen oder eine Zahl handelt. Ebenfalls wird die Originalposition der einzelnen Zeichen, aus denen das jeweilige Wort besteht, erfasst, da diese für das korrekte Zusammensetzen der Originaltextes in der Ausgabe erforderlich ist.

Das TDM enthält eine Liste aller Sätze (*Sentence*), die wiederum Referenzen auf die zugehörigen *Tokens* enthalten.

3.4 Erstellen der Tools durch den Tool Creator und Aufruf über den Main Analyser

Der *Tool Creator* hat die Aufgabe alle vom Anwender gewünschten Analysetools in einer Liste bereitzustellen und eventuelle Parameter an die Tools zu übergeben. Eine genaue Beschreibung der einzelnen Tools befindet sich in Abschnitt 4.2.

Sobald die Liste der Tools erstellt ist, wird diese vom Hauptprogramm an den *Main Analyser* übergeben. Da alle Analysetools das Interface *Tool* implementieren müssen, lassen sich diese zentral im *Main Analyser* aufrufen. Hierbei wird das TDM an die Tools übergeben.

3.5 Speichern der Ergebnisse in Full Output

Jeder sprachliche Fehler, der von einem Tool als solcher erkannt wird, wird in einem Objekt der Klasse *Tool Output Line Marker* (Line Marker) gespeichert. Ein *Line Marker* enthält Daten über die Position des Satzes in dem der Fehler auftritt, die Positionen des ersten und letzten Tokens die als Fehler markiert werden sollen, sowie eine Beschreibung und Klassifizierung des Fehlers. Die Klassifizierung erfolgt in drei Stufen (0, 1, 2) und soll eine Information darüber geben, wie schwer ein Fehler gewichtet wird. Da die verschiedenen Fehlerklassen in der Ausgabe verschiedene Arten von Markierungen erhalten, kann diese Information auch dazu genutzt werden, bei der Ausgabe eine bessere Übersichtlichkeit zu erreichen.

Die einzelnen *Line Marker* werden von jedem Tool zu einem *Tool Output* zusammengefasst und dort in einer Liste gespeichert. Jedes Tool produziert eine Instanz dieser Klasse und übergibt optional zusätzliche Daten, beispielsweise verwendete Parameter, sowie generelle Informationen zum Tool oder allgemeine Analyseergebnisse.

Schließlich werden die von den Tools erstellten *Tool Output* Objekte in einer Liste im *Full Output* zusammengefasst, sodass die vollständigen Analyseergebnisse jedes Tools eingesehen werden können.

3.6 Vorbereitung auf die Ausgabe über den Output Formatter

Sobald die Analyseergebnisse der Tools im *Full Output* gruppiert vorliegen, müssen diese für die Visualisierung vorbereitet werden. Diese Aufgabe übernimmt der *Output Formatter*.

Dazu werden *Output Tokens* eingeführt, die reinen Text, einen Verweis auf ein *Token* oder stilistische Angaben, wie beispielsweise Farbwerte und Beginn sowie Ende einer Markierung, enthalten. Die *Output Tokens* enthalten zusätzliche Verweise auf ihre Vorgänger und Nachfolger, sodass die Information eines einzelnen *Output Tokens* ausreicht, um die komplette visuelle Ausgabe zu erstellen, da durch die Verkettung jedes *Output Token* erreicht werden kann.

Im ersten Schritt generiert der *Output Formatter* jeweils ein *Output Token*, das die Start- sowie Endposition der Liste markiert und auf das jeweils andere *Output Token* verweist. Danach lassen sich weitere *Output Tokens* in die verkettete Liste einfügen.

Als nächstes wird aus dem TDM der Originaltext hergestellt und dabei Informationen über den Abstand der Wörter zueinander benutzt, um die Tokens richtig zu platzieren. Da die *Output Tokens*, welche auf jeweils ein Token aus dem TDM verweisen, zusätzlich in einer eigenen Liste gruppiert sind, können stilistische Angaben mit geringem Aufwand aus den Daten der *Line Marker* erstellt und richtig positioniert werden.

Da Fehlermarkierungen im Text für jedes Tool durch eine andere Farbe repräsentiert werden, wird jedem Tool eine eigene Farbe zugewiesen. Anschließend werden die einzelnen *Line Marker* abgearbeitet. Die Informationen aus den *Line Markern* werden dazu genutzt, ein entsprechendes *Output Token* mit den Farbwerten vor oder hinter das gewünschte Token zu setzen. Sind alle *Line Marker* analysiert, fügt der *Output Formatter* Informationen zu den verwendeten Tools, den genutzten Parametern und eine detaillierte Auflistung aller gefundenen Fehler hinzu. Zusätzlich werden verschiedene Angaben zum Stil, beispielsweise Tabellen, Überschriften oder Listen in Form von *Output Tokens* generiert.

3.7 Ausgabe über den HTML Output Formatter

Wird vom Anwender beim Start des Programms eine Ausgabe im HTML-Format durch das Setzen des Parameters `-out :` erwünscht, generiert der *HTML Output Formatter* eine HTML-Datei an dem gewählten Pfad (siehe Abbildung 2).

Durch spezielle *Output Tokens* vom Typ **INFO** können alle für den Head-Bereich des HTML Dokuments relevanten Informationen erstellt werden. Dazu gehören die Angabe eines Titels, sowie die einzelnen CSS-Befehle für die Markierungen. Die einzelnen HTML-/CSS-Befehle werden nun nacheinander zu einer Zeichenkette zusammengefügt, die dem finalen HTML Quelltext der Ausgabe entspricht.

Um den Body-Bereich des Dokuments zu erstellen, werden die *Output Tokens* nacheinander abgearbeitet und die entsprechenden HTML-Befehle gesetzt, bis die Endposition erreicht ist.

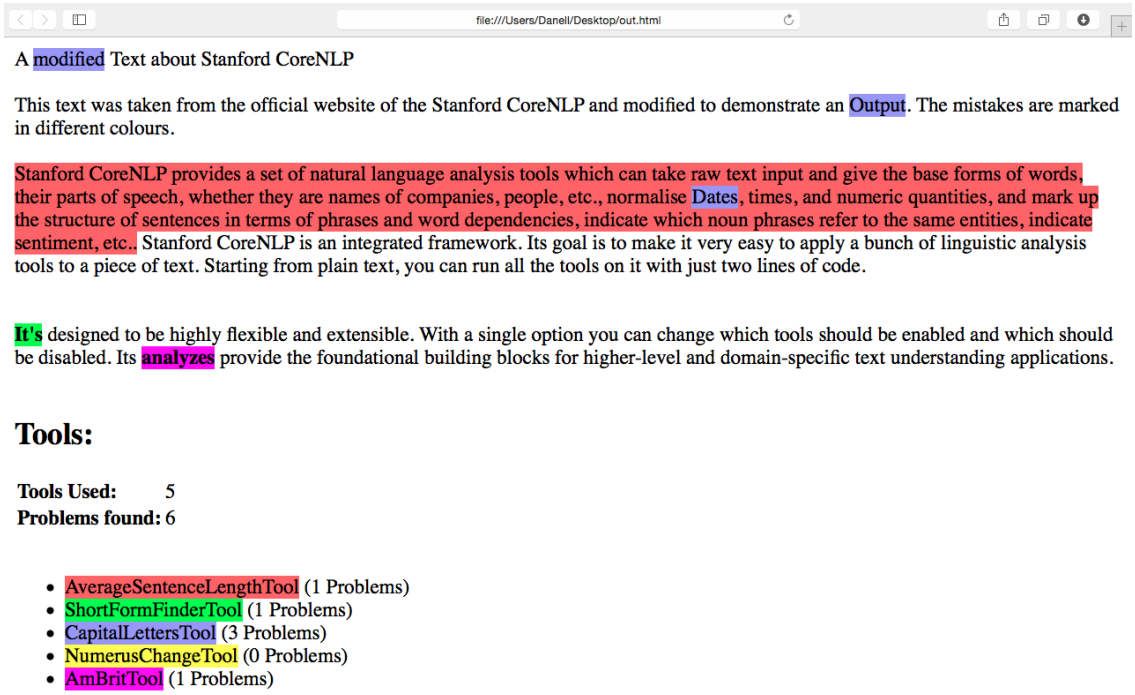

Abbildung 2: Beispiel einer Ausgabe

4 Implementierung

In diesem Kapitel wird auf die Bedienung des entwickelten Systems, sowie auf die genaue Funktionsweise der einzelnen Tools eingegangen. Darüber hinaus werden diverse Entscheidungen bei der Implementierung begründet und eventuelle Schwierigkeiten genannt.

4.1 Bedienung des Programms

In diesem Abschnitt wird auf die Bedienung des *Academic Style Analysers* über die Konsole eingegangen. Es wird erklärt, wie ein potentieller Anwender eine Quelldatei definiert, verschiedene Tools anwendet und diese individualisiert. Weiterhin wird beschrieben, wie die Bedienung durch das Auslagern von Parametern, vereinfacht werden kann.

4.1.1 Bedienung der Anwendung über Parameter

Der Academic Style Analyser kann direkt aus der Kommandozeile gestartet werden. Über diverse Parameter (siehe Tabelle 2) kann der Anwender die zu analysierende Quelldatei auswählen, einen Pfad für die HTML Ausgabe bestimmen, sowie einzelne Tools auswählen oder diesen eigene Parameter zuweisen, um Optionen für die Tools zu setzen.

Die einzelnen Parameter werden nach dem eigentlichen Programmaufruf über die Kommandozeile mit Leerzeichen getrennt. Falls ein Parameter einen Doppelpunkt enthält, werden weitere Angaben, beispielsweise eine Pfadangabe, direkt im Anschluss notiert. Das folgende Beispiel demonstriert einen einfachen Programmaufruf, der alle Tools auf die Quelldatei mit dem Pfad `PATH` anwendet und eine Ausgabe in der Konsole anzeigt, ohne eine HTML Datei zu generieren.

```
java AcademicStyleAnalyser -file:PATH -full
```

Für den Fall, dass der Anwender keine komplette Analyse durchführen möchte, lassen sich einzelne Tools gezielt auswählen. Alle Tools können optional mit eigenen Parametern versehen werden. Die zusätzlichen Parameter für die Tools bestehen immer aus einem Attribut, sowie einem dazugehörigen Wert und müssen dabei hinter dem Doppelpunkt platziert werden. Falls mehrere Parameter gesetzt werden, müssen diese durch das Zeichen + getrennt werden.

Das folgende Beispiel demonstriert einen Aufruf des Tools *Average Sentence Length* (siehe Abschnitt 4.2.1), bei dem einerseits die maximale Anzahl an Wörtern gesetzt wird und andererseits ebenfalls Sätze ohne Satzzeichen am Ende überprüft werden sollen. Zudem soll eine HTML Ausgabe am Ort `OUT` erstellt werden.

```
java AcademicStyleAnalyser -file:PATH -out:OUT
-avg:maxWords=20+checkTitles=true
```

Möchte der Anwender eine volle Analyse über den Parameter `-full` durchführen, ist es ebenfalls möglich einzelnen Tools Parameter zuzuweisen.

```
java AcademicStyleAnalyser -file:PATH -out:OUT -full
-avg:maxWords=20 -ambrit:americanWanted=true
```

Parameter	Funktion
`-help`	Zeigt Hinweise zur Nutzung des Programms und listet alle anwendbaren Parameter.
`-config:`	Gibt den Pfad zur Konfigurationsdatei an.
`-file:`	Gibt den Pfad zur Quelldatei an, welche analysiert werden soll (Der Parameter darf nur ein Mal vorkommen).
`-out:`	Gibt den gewünschten Pfad für die HTML Datei der Ausgabe an.
`-full`	Wendet alle vorhandenen Tools auf die Quelldatei an. Das Entwickler Tool *Dump* und das experimentelle Tool *Tempus Change* werden nicht aufgerufen.
`-avg(:)`	Wendet das Tool *Average Sentence Length* mit optionalen Parametern an.
`-phrase(:)`	Wendet das Tool *Equal Phrases* mit optionalen Parametern an.
`-short(:)`	Wendet das Tool *Short Form Finder* mit optionalen Parametern an.
`-capital(:)`	Wendet das Tool *Capital Letters* mit optionalen Parametern an.
`-signs(:)`	Wendet das Tool *Missing Signs* mit optionalen Parametern an.
`-num(:)`	Wendet das Tool *Numerus Change* mit optionalen Parametern an.
`-ambrit(:)`	Wendet das Tool *American British English* mit optionalen Parametern an.
`-temp(:)`	Wendet das experimentelle Tool *Tempus Change* mit optionalen Parametern an.
`-dump(:)`	Wendet das Entwickler Tool *Dump* mit optionalen Parametern an. Dieses gibt alle Informationen die im TDM gespeichert sind aus und wird nicht durch den Parameter `-full` aufgerufen.

Tabelle 2: Parameter zur Steuerung des ASA

4.1.2 Die Konfigurationsdatei

Beim Academic Style Analyser besteht die Möglichkeit, die Parameter zur Steuerung der Anwendung in einer Textdatei auszulagern. Dies kann sinnvoll sein, falls der Nutzer eine bestimmte Kombination an Parametern oft verwenden möchte. Über den Parameter `-config:` wird der Pfad zur angegebenen Konfigurationsdatei geöffnet und die dort ausgelesenen Parameter an der Position von `-config:` platziert.

Der Parameter `-config:` kann sowohl mehrfach in einem Aufruf, als auch innerhalb einer Konfigurationsdatei verwendet werden. Um potentiell mögliche Endlosschleifen zu vermeiden, wurde die Anzahl dieses Parameters auf einen Maximalwert begrenzt.

Das folgende Beispiel demonstriert die Anwendung des Parameters `-config:`.

Textdatei configuration.txt:

```
-full
-avg:maxWords=30+useAvg=false+checkTitles=true
-ambrit:britishWanted=true
-short:markGenitive=true
```

Aufruf in der Kommandozeile:

```
java AcademicStyleAnalyser -file:PATH -out:OUT
-config:configuration.txt
```

4.2 Tools

In diesem Abscnitt werden die einzelnen Tools beschrieben, die im ASA für die Sprachanalyse zuständig sind. Hierbei wird die Funktionsweise der Tools beschrieben und in Beispielen demonstriert, welche Art von sprachlichen Fehlern erkannt wird.

Zusätzlich wird zu jedem Tool eine Liste an Parametern angegeben, über die individuelle Anpassungen für die Analyse vorgenommen werden können.

Die hier gezeigten Beispiele und die dazugehörigen Ausgaben befinden sich ebenfalls auf der CD, die zu dieser Arbeit gehört.

4.2.1 Auffinden langer Sätze durch Average Sentence Length

Das Tool *Average Sentence Length* hat die Aufgabe den Eingabetext auf zu lange Sätze zu überprüfen. Dabei soll es einerseits anzeigen, ob ein Satz länger als eine totale Anzahl an Wörtern oder um einen Faktor länger, als der Durchschnitt der Satzlänge aller Sätze im Text, ist.

Dazu wird zunächst jeder Satz aus dem TDM auf die Anzahl der Wörter analysiert, die weder Satzzeichen noch Zahlen in numerischer Form enthalten. Dabei werden die Sätze, die nicht mit einem Satzzeichen enden, als Titel interpretiert und nicht mitgezählt, da viele kurze Titel einen relevanten Einfluss auf den Durchschnittswert haben und diesen signifikant senken. Aus der Anzahl der Sätze und der summierten Anzahl aller relevanten Wörter lässt sich die durchschnittliche Länge aller Sätze ermitteln.

Bei einem zweiten Durchlauf aller Sätze des TDM wird die Satzlänge sowohl mit einem Maximalwert an Wörtern abgeglichen, als auch einem Maximalwert, der aus der durchschnittlichen Satzlänge und einem optional wählbaren Faktor generiert wird. Die Maximalwerte können optional vom Anwender über Parameter bestimmt werden (siehe Tabelle 3).

Überschreitet ein Satz einen der beiden Maximalwerte, wird der gesamte Satz als Fehler markiert. Wenn beide Maximalwerte überschritten sind, wird der gesamte Satz als Fehler mit einer höheren Stufe markiert, falls der von der durchschnittlichen Satzlänge abhängige Maximalwert größer ist als der Maximalwert, der durch eine feste Anzahl an Wörtern

definiert wird. Im folgenden Beispiel wird gezeigt, dass der letzte Satz, der mehr als 30 Wörter enthält, durch *Average Sentence Length* als Fehler markiert wird (Begründung für den Standartmaximalwert siehe [Hä06]).

> `The Academic Style Analyser is an application written in Java. The main component of the application is the Stanford CoreNLP.` **`The Stanford CoreNLP is a so called natural language processing pipeline, which is used to analyse natural languages and provides information about words, like for example part of speech tags and other data.`**

In einem ersten Prototypen wurde bei diesem Tool lediglich die Anzahl der Tokens gezählt, was dazu führte, dass viele Sätze als überlang gekennzeichnet wurden, wenn zu viele Satzzeichen, wie zum Beispiel Klammern oder Anführungszeichen, in einem Text vorkamen. Ebenfalls wurden in einer früheren Version des ASA Sätze markiert, die kürzer als die durchschnittliche Satzlänge sind. Aus jener Version stammt die Option Sätze, die nicht mit einem Satzzeichen enden und daher als Titel interpretiert werden, nicht auf ihre Länge zu prüfen und ist nun optional auswählbar.

Parameter	Standartwert	Funktion
`maxWords`	**30**	Bestimmt die maximale Anzahl an Wörtern in einem Satz.
`upperLimit`	**1.0**	Bestimmt die maximale relative Abweichung von der Durchschnittlichen Satzlänge.
`useMax`	**true**	Prüft die Sätze auf ihre maximale Anzahl an Wörtern (vergleiche `maxWords`).
`useAvg`	**true**	Prüft die Sätze auf ihre maximale Abweichung von der durchschnittlichen Satzlänge (vergleiche `upperLimit`).
`checkTitles`	**false**	Sollen Titel (Sätze, die nicht mit einem Satzzeichen enden) überprüft werden?
`printData`	**false**	Sollen Informationen über die Durchschnittliche Satzlänge ausgegeben werden, falls das Tool keinen Fehler entdeckt?

Tabelle 3: Parameter für das Tool Average Sentence Length

4.2.2 Wechsel zwischen Amerikanischen und Britischen Englisch mit American British

Das Tool *American British* erkennt Wechsel zwischen der amerikanischen und der britischen Schreibweise. Dabei wird jedes Wort, sowie seine Grundform mit zwei verschiedenen Listen abgeglichen, die jeweils eine Auswahl an Wörtern enthalten, die der amerikanischen oder britischen Schreibweise entsprechen. Entspricht ein Wort oder seine Grund-

form einem Wort in der Liste, wird diesem, je nach Liste, die amerikanische oder die britische Schreibweise zugeteilt. Die Einträge der beiden Listen stammen zum größten Teil aus dem Quellcode der Klasse *Americanize*[2] der Stanford CoreNLP. Diese Klasse hat die Aufgabe Wörter in einem Text, die der britischen Sprachweise entsprechen, in ihre amerikanisch geschriebenen Äquivalente umzuwandeln. Diese Option wurde beim Aufruf der Stanford CoreNLP deaktiviert, da sonst keine Erkennung britisch geschriebener Wörter durchgeführt werden kann und ein vom ASA eingelesener Text stark verfälscht wird.

Sobald alle Wörter des zu analysierenden Textes geprüft wurden, werden alle unmarkierten Wörter entfernt und die Häufigkeit amerikanisch und britisch geschriebener Wörter ermittelt. Anhand dieser Häufigkeit wird die Sprachvariante bestimmt, die als korrekt angesehen wird. Enthält ein Text beispielsweise drei britische und vier amerikanische Wörter, sollen nur die britischen Wörter als Fehler markiert werden. Für den speziellen Fall, dass beide Häufigkeiten gleich sind, wird automatisch angenommen, dass die korrekte Sprachvariante britisches Englisch ist. Alternativ lässt sich die gewünschte Sprachvariante über Parameter einstellen (siehe Tabelle 4) und die Ermittlung der korrekten Sprachvariante über die Häufigkeit entfällt.

Bei einem zweiten Durchgang der Wortliste werden alle Wörter, die nicht der korrekten Sprachvariante entsprechen als fehlerhaft markiert. Folgendes Beispiel demonstriert einen Fehler, der durch das Tool *American British* gefunden wird. Dabei wurde der Text aufgrund der höheren Anzahl an britischen Wörtern (blau) als Britisch erkannt, somit werden nur die amerikanischen Wörter (rot) markiert.

```
The ASA is used to analyse text.  Every tool has its own
colour to mark possible mistakes.  The color used in the
output file is defined in the head section of the HTML
document.  The ASA can recognise a variety of mistakes,
like wrong capitalization.
```

Sobald der Nutzer den Parameter `americanWanted=true` setzt, werden hingegen die blau markierten Wörter als Fehler markiert.

Bei den ersten Versuchen Wörter in der britischen von Wörtern in der amerikanischen Schreibweise zu unterscheiden, wurde versucht, diese Unterscheidung über verschiedene Regeln zu ermöglichen. Es wurden beispielsweise diverse Suffixe von Verben untersucht, die sich in der amerikanischen und britischen Schreibweise unterscheiden [Tot02].

Leider stellte sich diese Methode als unzuverlässig heraus, da sie oft zu fehlerhaften Markierungen führte. Daher wurde der Ansatz verworfen und die Überprüfung wird lediglich über die beiden Listen vorgenommen. Dabei entsteht das Problem, dass Wörter, die zwar amerikanisch oder britisch geschrieben sind, jedoch nicht in einer der Listen vorkommen, nicht erkannt werden. Dieses Problem kann mit genügend Zeit und einer manuellen Überprüfung verschiedener Texte reduziert werden, indem die Listen erweitert werden.

[2] http://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/process/Americanize.html

Parameter	Standartwert	Funktion
`americanWanted`	**false**	Stellt die gewünschte Sprachvariante auf Amerikanisches Englisch ein.
`britishWanted`	**false**	Stellt die gewünschte Sprachvariante auf Britisches Englisch ein (sind beide Parameter auf **true** gesetzt, werden sie als **false** interpretiert und die als richtig angesehene Sprachvariante wird automatisch ermittelt).

Tabelle 4: Parameter für das Tool American British English

4.2.3 Gleiche Phrasen und Satzanfänge mit Equal Phrases

Das Tool *Equal Phrases* hat die Aufgabe, häufig verwendete Phrasen bei Satzanfängen und innerhalb eines Textes aufzuspüren. Dabei sollen Phrasen mit einer beliebigen Anzahl an Wörtern, somit auch mehrfache identische Sätze, erkannt werden.

```
The ASA was developed to search for a variety of
mistakes in academic publications.    The ASA was written
in the programming language Java.     Although the ASA is
not perfect, it can still help to improve your writing
skills.
```

Hierzu wird jeder Satz aus dem TDM betrachtet und in alle möglichen Gruppierungen von aufeinander folgenden Worten zerteilt. Auf technischer Ebene handelt es sich bei den Gruppierungen, die *Token Groups* genannt werden, um erweiterte Sentence Objekte. Hat der Nutzer keine Parameter gesetzt, werden Token Groups, die ein Sonderzeichen enthalten, nicht in die Analyse einbezogen. Dieses Verhalten lässt sich mit dem Parameter `checkSym=true` ändern (siehe Tabelle 5), führt aber dazu, dass Phrasen, wie beispielsweise **).**, oder ähnliche häufige Abfolgen von Satzzeichen angezeigt werden und so die Übersicht bei der Ausgabe stören. Die Token Groups werden zu zwei verschiedenen Listen hinzugefügt. Eine Liste beinhaltet alle möglichen Token Groups, die andere ausschließlich solche, die Satzanfänge enthalten.

Als nächstes werden die Listen aufsteigend nach der Anzahl der Tokens in den Token Groups sortiert. Dadurch ist es möglich, die Anzahl der nötigen Vergleiche zwischen den Gruppen im weiteren Verlauf, signifikant zu senken. Gruppen gleicher Länge werden jeweils untereinander verglichen. Dabei werde die Grundformen der einzelnen Wörter betrachtet. Somit werden beispielsweise die Phrasen *I think* und *I thought* als gleichwertig angesehen. Sind alle Grundformen einer Gruppe identisch mit den Grundformen einer zweiten Gruppe, wird die betrachtete Gruppe gespeichert und ein Zähler erhöht. Optional lässt sich mit *Equal Phrases* auf volle Gleichheit prüfen (siehe Tabelle 5). Wurde der entsprechende Parameter gewählt, werden, anstatt der Grundformen, die Wörter selbst verglichen.

Sobald alle Gruppen in beiden Listen untereinander verglichen wurden, befinden sich alle Token Groups, die öfter als ein mal vorkommen, in separaten Listen. Im nächsten

Schritt werden die neuen Listen abgearbeitet und die Distanz in Tokens zwischen den Gruppen ermittelt, um eine Analyse der Frequenz, mit der eine Phrase im Text genutzt wird, durchführen zu können.

Als nächstes werden die Kriterien ermittelt, nach denen bewertet wird, ob ein sprachlicher Fehler vorliegt. *Equal Phrases* erkennt drei verschiedene Arten von Fehlern. Die absolute Häufigkeit von Phrasen am Anfang eines Satzes wird anders bewertet, als die absolute Häufigkeit innerhalb der Sätze. Die erlaubte Häufigkeit lässt sich über die Parameter `wordLimit` und `startLimit` einstellen (siehe Tabelle 5). Dabei entspricht die Zahl einem Wert zwischen 0 und 1. Dieser Wert gibt an, welcher Prozentsatz an Worten im Text identisch sein darf. Übergibt der Anwender den Wert `wordLimit=0` oder `startLimit=0`, wird jede Phrase aus mindestens zwei Tokens, die mehr als ein mal vorkommt, als Fehler markiert. Bei dem Anderen Extremfall `wordLimit=1` oder `startLimit=1` darf jede Phrase unbegrenzt wiederholt werden und es wird kein Fehler markiert.

Da zur Bestimmung der erlaubten Häufigkeit von Phrasen, die relative Häufigkeit gleicher Wörter als Grundlage herangezogen wird, hängt die zugelassene Häufigkeit einer Phrase mit der Anzahl der in ihr enthaltenen Wörtern zusammen. Kürzere Phrasen dürfen somit häufiger im Text vorkommen, als längere Phrasen.

Die dritte Art von Fehler bezieht sich auf den Abstand gleicher Phrasen zueinander. Dabei werden Phrasen als Fehler markiert, wenn der Abstand zwischen gleichen Phrasen geringer ist, als ein Minimalwert, der optional über den Parameter `minDistance` gesetzt werden kann. Da in mehreren Tests zu beobachten war, dass *Equal Phrases* bei dieser Fehlerart eine große Anzahl an Markierungen setzt, muss ein entsprechender Parameter gesetzt werden, um diese Funktion zu aktivieren.

Nachdem die Kriterien bestimmt wurden, werden die Token Groups in den Listen überprüft. Sobald ein Fehler gefunden wird, wird dieser zunächst in einer Fehlerliste gespeichert. Diese Liste wird aufsteigend nach der Position des ersten Tokens im Text sortiert.

4.2.4 Falsche Groß- oder Kleinschreibung mit Capital Letters

Das Tool *Capital Letters* hat die Aufgabe fehlerhafte Groß- und Kleinschreibung zu erkennen und den Anwender darauf hinzuweisen.

Bei jedem Token im TDM wird überprüft, ob es sich um ein Satzzeichen oder eine Zahl handelt. Tritt einer der beiden genannten Fälle ein, wird sofort das jeweils nächste Token überprüft, bis ein Wort gefunden wird, dass die zuvor genannten Ausschlusskriterien nicht erfüllt.

Im nächsten Schritt wird der erste Buchstabe des Wortes angeschaut und es wird ermittelt, ob das Wort mit einem großen oder kleinen Buchstaben beginnt. Das Token wird nun an eine Funktion weitergeleitet, deren Aufgabe es ist zu entscheiden, ob das Wort mit einem großen oder kleinen Buchstaben beginnen sollte.

Dabei werden verschiedene Faktoren nacheinander betrachtet. Handelt es sich bei dem Wort um das erste Wort in einem Satz, welches weder einem Zeichen noch einer Zahl in numerischer Form entspricht, muss dieses immer groß geschrieben werden. Ebenfalls

Parameter	Standartwert	Funktion
`useFreq`	**false**	Soll der Text auf zu kurze Abstände zwischen Phrasen überprüft werden?
`useAbs`	**true**	Soll der Text auf häufige Phrasen überprüft werden?
`useBegin`	**true**	Soll der Text auf häufige Satzanfänge überprüft werden?
`minDistance`	**30**	Bestimmt den minimalen Abstand zwischen Phrasen in Wörtern.
`wordLimit`	**0.02**	Bestimmt die relative Häufigkeit gleicher Wörter im Text.
`startLimit`	**0.005**	Bestimmt die relative Häufigkeit gleicher Wörter bei Satzanfängen.
`checkSym`	**false**	Sollen Sonderzeichen überprüft werden?
`fullEqual`	**false**	Sollen Phrasen auf Gleichheit überprüft werden? (bei `false`) wird die Grundform der Wörter betrachtet)

Tabelle 5: Parameter für das Tool Equal Phrases

muss das englische Wort *I* unabhängig von seiner Position immer groß geschrieben werden.

Trifft keiner, der beiden zuerst genannten Fälle auf das Wort zu, wird der POS Tag des Tokens betrachtet (siehe Abschnitt 2.2.3 auf Seite 4). Normalerweise werden die meisten Wörter in der englischen Sprache kleingeschrieben. Sofern Ausnahmen von dieser Regel auftreten, geschieht dies in der Regel bei bestimmten Nomen oder Adjektiven [BB10, SKS14].

Zu den Ausnahmen gehören Nomen, sowie daraus gebildete Adjektive, die eine Organisation, eine Ideologie, einen geographischen Ort, wie beispielsweise einen Nationalstaat, oder den Namen beziehungsweise den Titel einer Person beschreiben. Diese Art von Nomen wird im Englischen als *Proper Noun* bezeichnet und muss groß geschrieben werden. Die Tokens, welche *Proper Nouns* entsprechen, bekommen im optimalen Fall das POS Tag **NNP** zugewiesen. Da sich in mehreren Tests herausgestellt hat, dass der POS Tagger eine Großzahl dieser Nomen nicht korrekt klassifiziert und sogar andere Worte als *Proper Nouns* erkennt, die dies nicht sind, ist eine Überprüfung des POS Tags nicht ausreichend und führt in manchen Fällen zu gravierenden Fehlern. Diese Fehlerhäufigkeit steigt, sobald das Wort mit einem falschen Groß- oder Kleinbuchstaben beginnt.

Um dieses Problem zu umgehen, prüft *Capital Letters* lediglich, ob es sich um ein Nomen oder Adjektiv handelt und zieht zur weiteren Analyse die *Named Entity Recognition* hinzu (siehe Abschnitt 2.2.5 auf Seite 5). Diese Methode erweist sich als zuverlässiger, als das bloße Prüfen des POS Tags, ist jedoch ebenfalls fehleranfällig. Beispielsweise werden die Wörter *left* und *right* als Ideologien erkannt, was bei einem falsch gesetzten POS Tag zu einer fehlerhaften Markierung durch das Tool führen kann. Da die Erkennung möglicher Fehler mit dieser Methode nicht ganz zuverlässig funktioniert, werden gefundene Treffer mit einer niedrigeren Fehlerstufe bewertet, als in den ersten beiden Fällen.

Sofern ein Token aus einem Satz ohne Satzzeichen am Ende stammt und somit wahrscheinlich Teil eines Titels ist, gelten andere Regeln für die Großschreibung [Chi06]. Laut dem *Chicago Manual of Style* muss das erste und letzte Wort in einem Titel groß geschrieben werden. Ansonsten werden bis auf einige Ausnahmen alle Wörter groß geschrieben. Zu den Ausnahmen gehören unter anderem die Artikel *a*, *an* und *the*, sowie nebenordnende Konjunktionen (*and*, *but*, *or*, *nor*, *for*, *yet*, *so*). Da diese Wortarten ebenfalls nicht immer korrekt vom POS Tagger erkannt werden, liegen die Wörter, welche in Titeln klein geschrieben werden sollen, in Listen vor.

Aufgrund der oft fehlerhaften Zuweisung der POS Tags, sollten die markierten Fehler bei der Nutzung von *Capital Letters* eher als Empfehlungen zur nochmaligen Überprüfung des Textes gesehen werden. Der Anwender hat zudem die Möglichkeit über diverse Parameter (siehe Tabelle 6) bestimmte Funktionen des Tools auszuschalten. Beispielsweise können die speziellen Regeln für Titel oder die Überprüfung von Titeln ausgeschaltet werden. Es ist ebenfalls möglich nur auf falsche Großschreibung sowie Kleinschreibung zu testen und den jeweils anderen Fall zu ignorieren.

Das folgende Beispiel zeigt, welche Wörter im fehlerhaften Text markiert werden. Bei dem ersten Satz werden die alternativen Regeln für Titel angewendet. Hierbei wird ebenfalls demonstriert, dass die Nomen *Germany* und *Rome*, wie erwartet, nicht als Fehler markiert werden.

```
Capital letters with the ASA

The ASA can find Words written with a Capital Letter
although they should be written with a small one.    the
tool can recognise nationalities like Germany or cities
like Rome and does not mark these as mistakes.
```

Parameter	Standartwert	Funktion
`checkQuotes`	**true**	Soll der Text, der sich zwischen Anführungszeichen befindet geprüft werden?
`wrongSmall`	**true**	Soll der Text auf fehlerhafte Kleinschreibung geprüft werden?
`wrongCapital`	**true**	Soll der Text auf fehlerhafte Großschreibung geprüft werden?
`checkTitles`	**true**	Sollen Titel (Sätze, die nicht mit einem Satzzeichen enden) überprüft werden?
`titleRules`	**true**	Sollen spezielle Regeln für Titel bei der Überprüfung angewendet werden?

Tabelle 6: Parameter für das Tool Capital Letters

4.2.5 Unbewusster Wechsel des Numerus mit Numerus Change

Das Tool *Numerus Change* überprüft den Wechsel zwischen verschiedenen Numeri. Schreibt der Autor beispielsweise häufig in der ersten Person Singular und wechselt im Verlauf des Textes zur ersten Person Plural, wird angenommen, dass es sich hierbei um einen Fehler handelt. Das folgende Beispiel demonstriert eine mögliche Ausgabe des Tools *Numerus Change*. Die Pronomen, über die der als korrekt angesehene Numerus bestimmt wird, sind dabei blau markiert und können über das Setzen des Parameters `markCounterpart=true` in der Ausgabe angezeigt werden.

```
The program I have developed is called the ASA.    It is
very important to me that it behaves as expected.    Our
main goal is to improve the writing style in academic
publications.
```

Als Basis für dieses Tool dienen zwei Listen mit entsprechenden Personalpronomina. Kommt ein Personalpronomen aus einer der beiden Listen im Text vor, merkt sich *Numerus Change* die Position des Pronomens im Text und der entsprechende Zähler wird erhöht. Wenn beide Zähler einen Wert anzeigen, der größer als Null ist, liegt wahrscheinlich ein Fehler im Text vor.

Dabei wird der Numerus, der durch den Zähler mit dem höheren Wert repräsentiert wird als korrekt angenommen und Personalpronomen des jeweils anderen Numerus als Fehler markiert. Beinhalten beide Zähler den gleichen Wert, werden beide Arten von Personalpronomen markiert. Sobald ein Wechsel des Numerus in dem gleichen Satz entdeckt wird, handelt es sich um einen Fehler einer höheren Stufe.

Um die Entscheidung des Tools für den als korrekt angesehenen Numerus als Anwender besser nachzuvollziehen, können über den Parameter `markCounterpart` (siehe Tabelle 7) ebenfalls die korrekten Personalpronomina mit einem Fehler der geringsten Stufe markiert werden.

Parameter	Standartwert	Funktion
`markCounterpart`	**false**	Sollen als korrekt erkannte Präpositionen ebenfalls mit geringerer Fehlerstufe markiert werden?

Tabelle 7: Parameter für das Tool Numerus Change

4.2.6 Fehlende Klammern mit Missing Signs

Die Aufgabe des Tools *Missing Signs* besteht hauptsächlich darin fehlerhafte Klammersetzung in einem Satz zu erkennen. Wird beispielsweise eine Klammer geöffnet und nicht geschlossen, da der Verfasser des Textes dies vergessen hat, zeigt das Tool diesen Umstand an. Das Tool ist ebenfalls in der Lage zu erkennen, wenn verschiedene Klammerarten in der falschen Reihenfolge geschlossen wurden.

```
The Academic Style Analyser (ASA can be used to find
missing brackets.   It is also possible to find brackets
closed in the wrong order (like this [for example)].
```

Um fehlende Klammern erkennen zu können, wird für jede der drei Hauptklammerarten (rund, eckig und geschweift) ein eigener Zähler, sowie zwei zusätzliche Werte angelegt, welche die Position der zuletzt betrachteten geöffneten und geschlossenen Klammer der jeweiligen Art speichern. Für jede geöffnete Klammer wird der jeweilige Zähler um Eins erhöht, für jede geschlossene wieder gesenkt. Beinhaltet der Zähler nach dem Durchlauf eines Satzes einen von Null verschiedenen Wert, geht das Tool davon aus, dass ein Fehler vorliegt und markiert die letzte Klammer der Art, die betrachtet wurde. Über das Vorzeichen des Zählers lässt sich zudem ermitteln, ob eine schließende oder eine öffnende Klammer fehlt.

Die Reihenfolge der Klammern wird ermittelt, indem jeder Klammerart ein bestimmter Wert (0, 1, 2) zugewiesen wird. Sobald eine Klammer einer Art geöffnet wird, wird der entsprechende Wert in einer Liste gespeichert. Wird eine Klammer geschlossen, wird der letzte Wert aus der Liste überprüft. Gleicht dieser Wert der Art der geschlossenen Klammer, wurde die richtige Reihenfolge eingehalten. Weicht der Wert hingegen ab, liegt ein Fehler vor und die Stelle wird markiert.

Das Tool *Missing Signs* verfügt ebenfalls über die Funktion fehlende Anführungszeichen innerhalb eines Satzes aufzuspüren, allerdings wurde diese Option ausgeschaltet und kann ebenfalls über einen Parameter genutzt werden (siehe Tabelle 8). Die Entscheidung diese Funktion auszuschalten basiert wesentlich auf zwei Faktoren. Zum Einen lassen sich nur fehlende Anführungszeichen innerhalb eines Satzes markieren, was dazu führt, dass Anführungszeichen in Zitaten, die aus mehreren Sätzen bestehen, immer als Fehler angesehen werden. Zum Anderen neigt der Sentence Splitter (siehe Abschnitt 2.2.2 auf Seite 3) dazu, Anführungszeichen zu Beginn eines Satzes, dem vorherigen Satz zuzuordnen, was ebenfalls zu falschen Ergebnissen führt.

Über den Parameter `skipEnum` ist es möglich, runde Klammern an zweiter Stelle in einem Satz zu ignorieren, da es sein kann, dass es sich hierbei um eine Aufzählung handelt. Da dieser Fall selten auftritt ist die Option bei einem Aufruf ohne Parameter deaktiviert. Das folgende Beispiel zeigt das Verhalten von *Missing Signs* in diesem Fall.

```
1) This is an enumeration
2) The brackets for the enumeration will not be marked
3) The recognition of other brackets does still work as
expected)
```

4.2.7 Kurzformen mit dem Short Form Finder

Der *Short Form Finder* dient dazu Kurzformen (*I'm, We'll, ...*) aufzufinden und zu markieren. Kurzformen sollten in akademischen Veröffentlichungen vermieden werden [Hä06].

Da der Tokenizer (siehe Abschnitt 2.2.1 auf Seite 2) Kurzformen mit einem Apostroph in zwei Tokens teilt, werden zum Auffinden einer Kurzform immer beide Tokens benötigt.

Parameter	Standartwert	Funktion
`skipEnum`	**false**	Sollen geschlossene runde Klammern an zweiter Stelle im Satz ignoriert werden (mögliche Aufzählung)?
`checkBrackets`	**true**	Soll der Text auf fehlerhafte Klammersetzung geprüft werden?
`checkQuotes`	**false**	Sollen Sätze auf fehlende Anführungszeichen geprüft werden (sobald mehrere Sätze zwischen Anführungszeichen stehen, ist diese Option ungenau)?

Tabelle 8: Parameter für das Tool Missing Signs

Enthält ein Token ein Apostroph, wird zunächst überprüft ob ebenfalls der Buchstabe *s* hinter dem Apostroph steht. In diesem Fall könnte es sich bei dem Wort um einen zulässigen Genitiv handeln, der nicht markiert werden soll. Um diesen Fall zu ermitteln, wird das Token, welches dem Vorgänger entspricht, mit einer Liste verglichen, die Wörter enthält, welche in Kombination mit einem *'s* keinen Genitiv repräsentieren und bei denen es sich definitiv um eine Kurzform handelt. Das Folgende Beispiel demonstriert den Unterschied zwischen einer Kurzform und einem möglichen Genitiv.

Kurzformen:
He's → *He is*
Let's → *Let us*

Möglicher Genitiv:
Peter's → *Peter is* **oder** Genitiv

Das Beispiel demonstriert ein zusätzliches Problem. Bei einem Genitiv mit der Endung *'s* kann es sich ebenfalls um eine Kurzform handeln. Um diesen Fall zu unterscheiden, reichen die begrenzten Mittel der Stanford CoreNLP nicht aus, weshalb die Option, mögliche Genitive zu markieren, vom Anwender über einen Parameter eingeschaltet werden kann (siehe Tabelle 9). Der POS Tagger markiert ein Token mit dem Wort *'s* zwar als *Possesive Ending* über das POS Tag **POS** (siehe Tabelle 1 auf Seite 4), unterscheidet aber nicht, ob es sich möglicherweise um die Kurzform des Wortes *is* handelt, wodurch eine Überprüfung des POS Tags überflüssig ist.

Das folgende Beispiel demonstriert eine mögliche Ausgabe des Tools *Missing Signs*. Bei einer Ausführung ohne zusätzliche Parameter werden die Genitive (blau) nicht markiert.

```
The ASA's tool named Missing Signs can detect short
forms.    It's distinguishing between genitive forms and
short forms, so that genitive forms won't be marked
as mistakes.     But some possible genitive forms aren't
```

```
genitive and will not be marked without the right
argument set.
```

Parameter	Standartwert	Funktion
`markGenitive`	**false**	Sollen mögliche Kurzformen, bei denen es sich ebenfalls um einen Genitiv handeln könnte, markiert werden?

Tabelle 9: Parameter für das Tool Short Form Finder

4.2.8 Unbewusster Wechsel des Tempus mit Tempus Change

Das Tool *Tempus Change* hat die Aufgabe unbewusste Wechsel zwischen der sprachlichen Vergangenheit und Gegenwart aufzufinden. Um dieses Tool zu nutzen, muss der Anwender bei Aufruf des ASA explizit den Parameter `-temp` setzen, da dieses Tool nicht über den Parameter `-full` gestartet wird.

Bei *Tempus Change* wird zu Beginn jeder Satz einzeln analysiert. Anhand der POS Tags der einzelnen Tokens, wird die am häufigsten genutzte Zeit im Satz ermittelt. Über die POS Tags lässt sich ermitteln, ob ein Wort einer Form der Vergangenheit (Bsp) oder einer Form der Gegenwart (Bsp) entspricht [MMS93].

VBD → Verb, past tense
VBG → Verb, gerund or present participle
VBN → Verb, past participle
VBP → Verb, non-3rd person singular present
VBZ → Verb, 3rd person singular present

Wird ein Verb mit einem der gesuchten POS Tags gefunden, wird die Information über die Zeit zu einem String hinzugefügt. Enthält ein Satz mehrere Verben, wird die Häufigkeit der einzelnen Zeitformen ermittelt und darüber die am häufigsten verwendete Zeitform ermittelt. Somit wird jedem Satz genau eine Zeitform zugeteilt.

Im nächsten Schritt wird die, als korrekt angesehen, Zeitform des gesamten Textes ermittelt. Diese lässt sich, analog zum Tool *American British*, über Parameter bestimmen (siehe Tabelle 10). Wird bei einem Satz eine Abweichung von der hauptsächlich verwendeten Zeitform festgestellt, werden die Verben, die im falschen Tempus vorliegen, als Fehler markiert markiert.

In einer ersten Version des Systems, wurden ganze Sätze als Fehler markiert. Die aktuelle Form der Darstellung wurde gewählt, um eine höhere Übersichtlichkeit zu erreichen, da durch einige Tests festgestellt werden konnte, dass das Tool häufig ausschlägt. Die Mehrzahl aller Sätze bekommt dabei den korrekten Tempus zugewiesen, allerdings handelt es sich bei vielen Tempuswechseln um korrekten Schreibstil, da beispielsweise eine Vergangenheitsform benötigt wird. Deshalb sollen die durch *Tempus Change* gefundenen Probleme als Empfehlungen angesehen werden, den Text nochmal genau zu überprüfen.

Auch ist die Methode, eine Zeit über das Zählen der einzelnen Zeitformen zu bestimmen, ungenau, da die POS Tags von der Stanford CoreNLP oft falsch gesetzt werden. In diversen Tests hat sich jedoch gezeigt, dass die Methode ausreicht, um in den meisten Fällen Wechsel in der Zeitform eines Satzes zuverlässig zu erkennen.

Parameter	Standartwert	Funktion
`pastWanted`	**false**	Stellt die gewünschte Zeitform auf Vergangenheit ein.
`presentWanted`	**false**	Stellt die gewünschte Zeitform auf Gegenwart ein (sind beide Parameter auf **true** gesetzt, werden sie als **false** interpretiert und die als richtig angesehene Zeitform wird automatisch ermittelt).

Tabelle 10: Parameter für das Tool Tempus Change

5 Evaluation

In diesem Abschnitt werden reale Anwendungsszenarien des ASA beschrieben. Das System wurde mit verschiedenen Parameterkonfigurationen auf frei verfügbare wissenschaftliche Veröffentlichungen aus dem Internet angewendet. Die daraus resultierenden Beobachtungen und Schlüsse werden in den nachfolgenden Abschnitten behandelt. Alle Beispiele, die resultierenden Ausgaben, sowie die angewendeten Konfigurationsdateien befinden sich auf der beilegenden CD.

5.1 Erster Durchlauf: Grundkonfiguration

Im folgenden Versuch wird der *Academic Style Analyser* auf eine wissenschaftliche Veröffentlichung[3] angewendet und somit ein realistischer Anwendungsfall demonstriert. Dazu wird der Inhalt der PDF Datei kopiert und in eine Textdatei (.txt) eingefügt. Der ASA wird mit der folgenden Grundkonfiguration gestartet:

```
java AcademicStyleAnalyser -file:PATH -out:OUT -full
```

Durch diese Konfiguration, werden alle vorhandenen Tools auf die Datei am Ort `PATH` angewendet und die Ausgabe in einer HTML Datei am Ort `OUT` gespeichert. Bei einer Überprüfung der Beispiele, müssen `PATH` und `OUT` durch die gewünschten Pfade ersetzt werden.

Der Text von Beispiel **P1** wurde vom ASA in 367 Sätze und 7111 Token zerteilt. Nach einem Durchlauf mit der zuvor genannten Konfiguration markiert die Anwendung 174 Stellen als Fehler. Die Anzahl der einzelnen Markierungen für jedes Tool können der folgenden Liste entnommen werden.

- AverageSentenceLengthTool (22 Problems)
- EqualPhrasesTool (0 Problems)
- ShortFormFinderTool (7 Problems)
- CapitalLettersTool (134 Problems)
- MissignSignsTool (1 Problems)
- NumerusChangeTool (10 Problems)
- AmBritTool (0 Problems)

Capital Letters:
Auffällig ist hierbei die hohe Fehlerzahl des Tools *Capital Letters* mit 134 Markierungen. Ein großer Teil der Fehler bezieht sich auf fehlerhafte Großschreibung und resultiert daraus, dass *Capital Letters* diverse *Proper Nouns* nicht als solche klassifiziert, da die Daten

[3] http://www.cs.cornell.edu/ danco/research/papers/suggestbot-iui07.pdf

aus den POS Tags, die durch den POS Tagger der Stanford CoreNLP generiert werden, ungenau sind. Beispielsweise wird das Wort *Dfrankow* in Satz 11 nicht als *Proper Noun* erkannt und vom Tool als Fehler markiert. In Satz 14 wird das gleiche Wort nicht markiert, da es in dieser Konstellation als *Proper Noun* erkannt wurde. Der Folgende Ausschnitt aus dem Beispiel demonstriert diese fehlerhafte Markierung.

Satz 11:
`Why can't Dfrankow find articles to edit in Wikipedia?`
Satz 14:
`New members might not be aware that the portal exists, but Dfrankow has made over 100 edits and is aware of it.`

Eine ebenfalls häufige fehlerhafte Markierung bezieht sich auf die Angabe von Monaten (siehe Satz 95: *May* und Satz 233: *June*). Diese Art von Fehler, hätte durch das Hinzufügen einer zusätzlichen Liste von Monatsnamen in *Capital Letters* vermieden werden können. Dadurch wäre jedoch das Problem aufgetreten, dass das englische Wort *may* (*können, dürfen*) jedes Mal im Text markiert werden würde, wenn es kleingeschrieben wird.

Ein ähnliches Problem lässt sich bei der Nutzung von einzelnen großgeschriebenen Buchstaben feststellen, die dem Autor beispielsweise als Variablen dienen (siehe Satz 237). Würde *Capital Letters* die Großschreibung einzelner Buchstaben akzeptieren, müssten einzelne Buchstaben immer großgeschrieben werden, um nicht als Fehler markiert zu werden. Dieser Fall führt dazu, dass kleingeschriebene Buchstaben, die ebenfalls häufig zur Benennung von Variablen genutzt werden, als Fehler markiert werden. Zudem müsste es eine zusätzliche Ausnahmeregelung für den Artikel *a* geben.

Capital Letters entscheidet bei jedem Wort, ob es groß- oder kleingeschrieben werden soll. Eine mögliche Lösung für das zuvor genannte Problem der einzelnen Buchstaben, wäre das Hinzufügen einer Option, die es *Capital Letters* erlaubt Fälle zu definieren, bei denen nicht entschieden wird, ob ein Wort groß- oder kleingeschrieben werden soll. Der momentane Programmaufbau lässt dieses jedoch nicht ohne größere Modifizierung zu.

Bei einem erneuten Aufruf des ASA, kann sich der Nutzer dafür entscheiden den Parameter `wrongCapital=false` zu setzen, um den Text lediglich auf falsche Kleinschreibung zu überprüfen. Somit werden die zuvor genannten fehlerhaften Markierungen nicht angezeigt.

Ein weiteres Problem entsteht häufig bei Zitaten, die durch Anführungszeichen eingeleitet werden, jedoch kein Zeichen enthalten, welches als Satzende interpretiert wird (siehe Satz 135).

Satz 135:
`"I've reviewed the things on my watchlist, and I'm not so sure that they are representative of my interests (at least some things are listed because they were things I thought that should be deleted, some I thought should be merged, and some were things that I can no longer recall why they were originally watched, etc.)"`

In diesem Fall wird der Satz durch *Capital Letters* als Titel klassifiziert, da in dem Satz kein Zeichen vorkommt, welches auf ein Satzende schließen lässt. Der Punkt hinter *etc* wird dabei als Teil der Abkürzung interpretiert. Dadurch werden bei diesem Satz die speziellen Regeln für Titel nach dem *Chicago Manual of Style* angewendet. Um fehlerhafte Markierungen innerhalb von Zitaten auszuschließen, kann der Anwender bei einem erneuten Aufruf den Parameter `checkQuotes=false` setzen, damit *Capital Letters* alle Wörter zwischen Anführungszeichen ignoriert (siehe Abschnitt 5.2). Alternativ lassen sich ebenfalls die Parameter `checkTitles=false` oder `titleRules=false` setzen, allerdings entfällt dabei die Überprüfung der Titel.

Capital Letters erkennt bei den Titeln der Abschnitte im Text, dass diverse Wörter laut dem *Chicago Manual of Style* großgeschrieben werden sollten. Die Markierungen, die hierbei gesetzt wurden, hätten den Autor der vorliegenden Publikation dazu motivieren können, sich nach der korrekten Schreibweise von Titeln in der englischen Sprache zu informieren.

Average Sentence Length:
Das Tool *Average Sentence Length* zeigt in diesem Aufbau 22 Probleme an. Die durchschnittliche Satzlänge beträgt 17,18 Wörter, somit liegt der obere Maximalwert, der aus der durchschnittlichen Satzlänge bestimmt wird, bei 34 Wörtern. Der andere Maximalwert beträgt, ohne Modifikation durch geeignete Parameter, 30 Wörter.

Das Tool erkennt zuverlässig alle Sätze, die einen oder beide Maximalwerte überschreiten. Diese Information hätte der Autor, als potentieller Anwender des ASA, nutzen können, um die langen Sätze in mehrere Einheiten zu zerteilen. Bei manchen Sätzen wurde der Maximalwert lediglich leicht überschritten (siehe Satz 55 oder Satz 74 mit jeweils 31 Wörtern). Diese Sätze werden vom Tool *Average Sentence Length* mit einer geringen Stufe gekennzeichnet. Weicht ein Satz jedoch deutlich von der Durchschnittlichen Satzlänge ab (siehe Satz 65 mit 46 Wörtern), benutzt der ASA eine Markierung, die einer höheren Fehlerstufe entspricht. So kann sich der Anwender zunächst auf die längsten Sätze konzentrieren, da diese direkt ersichtlich sind.

Short Form Finder:
Der *Short Form Finder* erkennt in der analysierten Veröffentlichung sieben Kurzformen. Die Erkennung der Kurzformen funktioniert wie erwartet, allerdings werden zwei der Markierungen im Text nicht angezeigt, und können lediglich der Liste am Ende des HTML Dokumentes entnommen werden. Das liegt daran, dass andere Tools, in dem Fall die Tools *Capital Letters* und *Numerus Change*, ebenfalls Markierungen an diese Stellen gesetzt haben. Wird eine Stelle durch eine hohe Anzahl an Tools markiert, kann es zu Fehlern in der Anzeige, in Form von fehlenden Markierungen, kommen. Dieses Problem kann dadurch umgangen werden, dass die Tools einzeln aufgerufen werden. In einem weiteren Aufruf (siehe Abschnitt 5.2) mit angepassten Parametern, lassen sich die Markierungen erkennen, da das Tool *Capital Letters* keine Markierungen an dieser Stelle setzt.

In einer zukünftigen Version des Programms ließe sich die Ausgabe optimieren, sodass der ASA besser mit mehrfachen Markierungen an der selben Stelle umgehen kann. Dafür muss beim *Output Formatter* (siehe Abschnitt 3.6 auf Seite 9) eine zusätzliche Methode implementiert werden, die vor dem Setzen einer Markierung prüft, ob ein Abschnitt bereits von einem anderen Tool markiert wurde. Sobald dies der Fall ist, muss eine andere Art

von Markierung gesetzt werden, beispielsweise indem ein Fehler der Stufe 1 mit der Markierung der Stufe 0 versehen wird. Da bei einer Markierung der Stufe 0 ein Rahmen mit CSS gesetzt wird, bei Stufe 1 jedoch eine Hintergrundfarbe definiert wird, können beide Markierungen simultan verwendet werden. Allerdings ist ein Algorithmus, der dieses Problem löst, zeitaufwändig und müsste daher nachträglich implementiert werden.

Numerus Change:
Das Tool *Numerus Change* findet im vorliegenden Text zehn Probleme. Bei genauerer Betrachtung lässt sich feststellen, dass die Markierungen innerhalb von Zitaten gesetzt wurden. Obwohl der Wechsel des Numerus an diesen Stellen wahrscheinlich absichtlich vom Autor gewählt wurde, erkennt das Tool, wie beabsichtigt, den Wechsel des Numerus. Würde an dieser Stelle ein Fehler vorliegen, könnte der Autor eingreifen und die fehlerhafte Formulierung anpassen. Bei einem erneuten Durchlauf kann der Anwender das Tool *Numerus Change* ignorieren und sich auf einen Durchlauf mit den restlichen Tools beschränken.

American British:
Obwohl *American British* keine Fehlermeldung generiert, liefert das Tool Informationen über die Sprachvariante des analysierten Textes. Diese Information befindet sich in der Liste, die genaue Angaben für alle Tools enthält.

> Total American words: 13
> Total British words: 0
> Correct language style is AMERICAN ENGLISH

Würde der Anwender lieber britisches Englisch verwenden, wäre er in diesem Fall informiert, dass er amerikanisches Englisch im Text nutzt. In diesem Fall kann der Parameter `britishWanted=true` im Tool *American British* gesetzt werden, um die, dann als Fehler erkannten, Stellen markieren zu lassen.

5.2 Zweiter Durchlauf: Angepasste Konfiguration

Bei einem zweiten Aufruf des Academic Style Anslysers wurde die Konfiguration beim Start der Anwendung angepasst. Dabei wird das System wieder auf das Beispiel **P1** angewandt. Die neue Konfiguration befindet sich in der Datei `configP1b.txt` auf der CD und entspricht dem folgenden Aufruf:

```
java AcademicStyleAnalyser -file:PATH
-out:OUT -avg:maxWords=35 -phrase -short
-capital:checkQuotes=false+wrongCapital=false
-ambrit:britishWanted=true
```

In dieser Konfiguration wird eine Auswahl an Tools gezielt angepasst und zur Analyse eingesetzt. Hat der Anwender den ASA wie in Abschnitt 5.1 eingesetzt und eine volle Analyse durchgeführt, kann er, durch das Setzen verschiedener Parameter, eine erneute Analyse individualisieren.

In diesem Beispiel wurde das Tool *Average Sentence Length* so modifiziert, dass Sätze erst ab einer Länge von 35 Wörtern als Fehler markiert werden. Zusätzlich ignoriert *Capital Letters* bei diesem Aufruf Zitate und prüft den Text nur auf falsche Kleinschreibung, sodass die Anzahl der Fehler durch das Tool, im Vergleich zu den in Abschnitt 5.1 genutzten Einstellungen, deutlich sinkt. Ebenfalls wird beim Tool *American British* die gewünschte Sprachvariante auf britisches Englisch gesetzt. Eine erneute Analyse ergibt die folgende Fehlerliste:

- AverageSentenceLengthTool (9 Problems)
- EqualPhrasesTool (0 Problems)
- ShortFormFinderTool (7 Problems)
- CapitalLettersTool (21 Problems)
- AmBritTool (13 Problems)

Das Tool *American British* setzt bei dieser Konfiguration Markierungen, sobald ein Wort als amerikanisch klassifiziert wird. So kann der Nutzer die Worte gezielt aufspüren und diese gezielt durch Worte der britischen Sprachvariante ersetzen, sofern er britisches Englisch verwenden möchte. Die Ausgabe von *Capital Letters* ist übersichtlicher und weist auf fehlende Großschreibung in den Titeln hin. Ebenfalls lassen sich die Kurzformen besser im Text erkennen.

5.3 Dritter Durchlauf: Sonstige Funktionen

In einem dritten Durchlauf des ASA auf Beispiel **P1** wird demonstriert, welche Ergebnisse von zwei Funktionen produziert werden, die in einer frühen Version des Programms bei der Grundkonfiguration enthalten waren. In diesem Abschnitt wird begründet, weshalb diese Funktionen bei einem einfachen Aufruf ausgeschaltet wurden.

Die erste Funktion ist Teil des Tool *Equal Phrases* und hat die Aufgabe Phrasen, die aus Wörtern mit der gleichen Grundform bestehen, zu markieren, sobald diese einen bestimmten Abstand zueinander unterschreiten. Die zweite Funktion ist das eigenständige Tool *Tempus Change*, welches nachträglich als *experimentell* eingestuft wurde, jedoch über den Parameter `-temp` gezielt ausgewählt werden kann. Bei diesem Beispiel wird der folgende Programmaufruf verwendet:

```
java AcademicStyleAnalyser -file:PATH -out:OUT
-phrase:useFreq=true -temp
```

Equal Phrases markiert dabei 139, *Tempus Change* 162 Stellen im Text. Da diese beiden Funktionen häufig eine große Anzahl an Markierungen setzen, wird die Ausgabe unübersichtlich, sobald die Funktionen aktiviert sind.

Der Grundgedanke bei der Frequenzanalyse von *Equal Phrases* ist es Wiederholungen von Phrasen zu finden, die nah beieinander stehen. Obwohl die Erkennung solcher Phrasen

funktioniert, hat sich in mehreren Tests herausgestellt, dass viele Phrasen in akademischen Publikationen gruppiert auftreten. Wird beispielsweise ein Wort zum ersten Mal in einer Publikation definiert, ist die Wahrscheinlichkeit hoch, dass es in den folgenden Sätzen mehrfach auftritt, wie das folgende Beispiel demonstriert.

> Sätze 228 - 230:
> **The links recommender** exploits links users have created between articles. This is similar to using citation networks in recommending research papers [8, 28]. Pseudocode for **the links recommender** is shown in Algorithm 1.

Trotz der hohen Anzahl an Markierungen, kann die Frequenzanalyse sinnvoll sein, sofern diese, isoliert von anderen Tools, durchgeführt wird. Die folgenden Beispiele demonstrieren Stellen im Text, an denen das mehrfache Aufkommen der Phrasen den Lesefluss eines potentiellen Lesers stören kann.

> Satz 32:
> One way **to do** so is to make it easy to find work **to do**.
> Satz 89:
> Wikipedia makes much **of its** content available for offline analysis through occasional XML dumps **of its** database.

Da es sich bei der Einschätzung, ob in den zuletzt genannten Beispielsätzen ein Fehler vorliegt, hauptsächlich um eine subjektive Empfindung handelt, muss der Anwender des ASA selbst entscheiden, ob er die Funktion der Frequenzanalyse von *Equal Phrases* nutzen möchte.

Bei der zweiten ausgeschalteten Funktion, deren Anwendung in diesem Abschnitt beschrieben ist, handelt es sich um das Tool *Tempus Change*. Dieses Tool wurde entworfen, um Wechsel des Tempus innerhalb eines Textes aufzuspüren. Der hauptsächliche Grund für die hohe Anzahl an Fehlern ist die gewählte Art der Darstellung. Sobald *Tempus Change* einen Satz mit einem, als fehlerhaft definierten, Tempus registriert, werden die einzelnen Verben markiert, die zu dieser Markierung beitragen. In Abschnitt 4.2.8 wird begründet, wieso diese Art der Darstellung verwendet wird. Würde das Tool, anstatt der Verben, die gesamten Sätze markieren, würde die Anzahl der Markierungen von 162 auf etwa 70 sinken.

Die Erkennung des hauptsächlich verwendeten Tempus eines Satzes verläuft, bis auf wenige Ausnahmen, zuverlässig. Im Beispiel **P1** wird nahezu allen Sätzen der korrekte Tempus zugewiesen, allerdings scheint der Wechsel in die Vergangenheitsform vom Autor gewünscht zu sein. Läge tatsächlich ein sprachlicher Fehler vor, wäre dieser bei der hohen Anzahl an Markierungen schwer zu finden, selbst wenn die kompletten Sätze markiert werden. In einem Text, bei dem der Autor beispielsweise ausschließlich Verben im Präsens verwenden möchte, kann *Tempus Change* hingegen hilfreich sein. Da sich in mehreren Tests herausgestellt hat, dass ein Wechsel des Tempus meistens nicht als

sprachlicher Fehler zu bewerten ist, wird dieses Tool nicht mehr durch den Aufruf des Parameters `-full` aktiviert, sondern muss vom Anwender explizit durch Setzen des Parameters `-temp` aufgerufen werden.

6 Zusammenfassung

Dieses Kapitel dient als Zusammenfassung der bei der Entwicklung des ASA gewonnenen Erkenntnisse. Es werden die grundlegenden Probleme bei der Implementierung beschrieben und ein Ausblick auf mögliche zukünftige Arbeiten gegeben.

6.1 Fazit

Ziel der vorliegenden Arbeit war es ein System auf Basis einer *Natural Language Processing Pipeline* zu entwickeln, welches sprachliche und stilistische Mängel in englischsprachigen akademischen Veröffentlichungen auffindet. Zu diesem Zweck wurde die *Stanford CoreNLP* genutzt, um eine Reihe an Informationen aus geschriebenen englischsprachigen Texten zu extrahieren.

Dabei konnte bereits zu Beginn der Programmierung festgestellt werden, dass die aus der *Stanford CoreNLP* gewonnenen Daten Ungenauigkeiten aufweisen. Diese Ungenauigkeiten, die bei den meisten Funktionen der NLP auftreten, erschweren eine Analyse und müssen durch eine vorausschauende Programmierung umgangen werden.

Beispielsweise funktioniert das bestimmen einer Entität durch den *Named Entity Recognizer* und der Erweiterung *Regexner* nicht fehlerfrei, was sich besonders bei dem Tool *Capital Letters* durch eine hohe Anzahl an Fehlern bemerkbar macht. Ebenfalls sind die durch den *POS Tagger* generierten POS Tags häufig fehlerhaft oder unvollständig. Häufig kommt es vor, dass ein *Proper Noun* lediglich als Nomen erkannt wird, oder ein Wort einer falschen Wortart zugewiesen wird.

Durch diese Ungenauigkeiten der *Stanford CoreNLP* und das gelegentliche Fehlen von Daten, war es notwendig alternative Vorgehensweisen zu implementieren und das Programm so abzusichern, dass es mit fehlerhaften Daten nahezu korrekte Ergebnisse produziert.

Durch die Komplexität menschlicher Sprache, steigt die Schwierigkeit ein nahezu fehlerfreies System zu entwickeln. Bei der Programmierung der verschiedenen Tools, wurde zunehmend klar, dass eine menschliche Sprache zwar diversen Regeln folgt, diese jedoch so vielfältig sind, dass es schwierig ist einen geschriebenen Text maschinell zu analysieren. Ein Mensch nutzt eine zuvor über Jahre erlernte Sprache intuitiv und hat in der Regel keine Schwierigkeiten Informationen aus der Sprache zu extrahieren und zu verstehen. Bei der maschinellen Analyse einer Sprache durch ein computerbasiertes System müssen diese intuitiv genutzten Regeln und Ausnahmen beachtet werden, wobei ein Programm die übermittelten Informationen nicht versteht.

Schwierig gestaltet sich auch die Definition eines *schlechten* Schreibstils. Zu diesem Thema existiert eine Vielzahl subjektiver Einschätzungen, die sich schwer in festen Regeln definieren lassen. Dieser Umstand führte dazu, dass die Empfindlichkeit verschiedener Tools des *Academic Style Analysers* vom Anwender über verschiedene Parameter angepasst werden kann.

In einer Evaluation wurde das fertige Programm unter realen Bedingungen getestet. Es stellte sich heraus, dass die Tools des ASA in vielen Fällen korrekte Ergebnisse liefern,

jedoch aufgrund der zuvor beschriebenen Probleme, ungenau sind.

6.2 Ausblick

In der Planungsphase des Programms wurde die Möglichkeit in Betracht gezogen, Texte aus anderen Dateiformaten zu extrahieren. Unter anderem wurde geplant Texte direkt aus LaTeX-Dateien auszulesen und an die Analysefunktion im ASA weiterzuleiten. Dieses Vorhaben scheiterte jedoch daran, dass keine geeignete API für die Programmiersprache Java gefunden werden konnte. Da der ASA plattformunabhängig gestaltet werden sollte, wurde auf die Abhängigkeit von externen Anwendungen verzichtet und das Vorhaben nicht umgesetzt. Eine Möglichkeit den ASA zu erweitern, wäre es einen eigenen LaTeX-Parser zu entwickeln, der zuverlässig Text aus LaTeX-Dokumenten ausliest. Da dieses Vorhaben zeitaufwändig ist, müsste es in einer zukünftigen Version des Academic Style Analysers umgesetzt werden. Auch die Extraktion aus PDF-Dokumenten, sowie anderen häufig genutzten Dateiformaten, wäre eine praktische Erweiterung.

Zudem ist es nötig die Tools weiter zu optimieren. Mit steigender Anzahl an Tests mit der vorhandenen Version des ASA, lässt sich dieser durch Auffinden verschiedener Schwächen in den Tools weiter optimieren. Ebenfalls lassen sich die Basiswerte der verwendeten Parameter weiter anpassen, sodass eine möglichst effiziente Konfiguration gelingt. Da es möglich ist auf weitere Daten der *Stanford CoreNLP* zuzugreifen können neue Tools erstellt oder vorhandene verbessert werden. Das Hinzufügen neuer Tools lässt sich in der aktuellen Programmstruktur leicht bewältigen, da die Anwendung an wenigen Stellen angepasst werden muss. Denkbar wäre zum Beispiel ein Tool, welches Sätze auf eine korrekte Satzstellung prüft. Hierbei können Daten über die Beziehung zwischen grammatikalischen Objekten innerhalb von Sätzen nützlich sein.

Mit den zuvor genannten Optimierungen lässt sich der Academic Style Analyser zu einer vollwertigen Analysesoftware ausbauen, die sowohl benutzerfreundlich als auch effizient arbeitet.

Literatur

[BB10] BÖRJARS, Kersti ; BURRIDGE, Kate: *Introducing English Grammar second edition*. Taylor and Francis, 2010

[CEE+09] CARSTENSEN, Kai-Uwe (Hrsg.) ; EBERT, Christian (Hrsg.) ; EBERT, Cornelia (Hrsg.) ; JEKAT, Susanne (Hrsg.) ; KLABUNDE, Ralf (Hrsg.) ; LANGER, Hagen (Hrsg.): *Computerlinguistik und Sprachtechnologie: Eine Einführung*. Spektrum, 2009

[Chi06] CHICAGO, University of: *The Chicago Manual of Style*. 1906

[Hä06] HÄMÄLÄINEN, Wilhelmiina: *Scientific Writing for Computer Science Students*. `http://www.cs.joensuu.fi/pages/whamalai/sciwri/sciwri.pdf`. Version: September 2006

[MMS93] MARCUS, Mitchell P. ; MARCINKIEWICZ, Mary A. ; SANTORINI, Beatrice: Building a Large Annotated Corpus of English: The Penn Treebank. In: *Comput. Linguist.* 19 (1993), Juni, Nr. 2, S. 313–330

[MSB+14] MANNING, Christopher D. ; SURDEANU, Mihai ; BAUER, John ; FINKEL, Jenny ; BETHARD, Steven J. ; MCCLOSKY, David: The Stanford CoreNLP Natural Language Processing Toolkit. In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014, S. 55–60

[SCN15] *Stanford CoreNLP*. `http://nlp.stanford.edu/software/corenlp.shtml`. Version: Mai 2015

[SKS14] STRAUS, Jane ; KAUFMAN, Lester ; STERN, Tom: *The Blue Book of Grammar and Punctuation: An Easy-to-Use Guide with Clear Rules, Real-World Examples, and Reproducible Quizzes*. John Wiley and Sons, 2014

[TKMS03] TOUTANOVA, Kristina ; KLEIN, Dan ; MANNING, Christopher ; SINGER, Yoram: Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In: *Proceedings of HLT-NAACL 2003*, 2003, S. 252–259

[Tot02] TOTTIE, Gunnel: *An Introduction to American English*. Blackwell Publishing, 2002

Abbildungsverzeichnis

Tabellenverzeichnis