Originalveröffentlichung in: Journal for Language Technology and Computational Linguistics Jg. 35 (2023) Nr. 1, S. 151-170
DOI: *https://doi.org/10.21248/jlcl.36.2023.241*

Gertrud Faaß / Helmut Schmid

Segmentierungs- und Annotationsverfahren für die Texte Udo Lindenbergs: Apostrophe und andere Herausforderungen

Kurzfassung

In der Computerlinguistik ist eine kaskadische Prozessierung von Texten üblich. Dabei werden diese zuerst segmentiert (tokenisiert), d.h. Tokens und ggf. Satzgrenzen werden erkannt. Dabei entsteht meist eine Liste bzw. eine einspaltige Tabelle, die sukzessive durch weitere Prozessierungschritte um zusätzliche Spalten – also positionale Annotationen wie z.B. Wortarten und Lemmata für die Tokens in der ersten Spalte – ergänzt wird. Bei der Tokenisierung werden alle Spatien (Leerzeichen) gelöscht. Schon immer problematisch waren dabei Interpunktionszeichen, da diese äußerst ambig sein können, aber auch mehrteilige Namen, die Leerzeichen enthalten und eigentlich zusammengehören. Dieser Beitrag fokussiert auf den Apostroph, der in vielfältiger Weise in den Texten Udo Lindenbergs eingesetzt wird sowie auf mehrteilige Namen, die wir als Tokens erhalten möchten. Wir nutzen dafür das komplette Lindenberg-Archiv des *songkorpus.de*-Repositoriums, kategorisieren die auftretenden Phänomene, erstellen einen Goldstandard und entwickeln ein teils regel-, teils auf maschinellem Lernen basierendes Segmentierungswerkzeug, das insbesondere die auftretenden Apostrophe, aber auch - lexikonbasiert - mehrteilige Namen nach unseren Vorstellungen erkennt und tokenisiert. Im Anschluss trainieren wir den RNN-Tagger (Schmid, 2019) und zeigen auf, dass ein spezifisch für diese Texte angepasstes Training zu Genauigkeiten $\geq 96\%$ führt.
Dabei entsteht nicht nur ein Goldstandard des annotierten Korpus, das dem Songkorpus-Repositorium zur Verfügung gestellt wird, sondern auch eine angepasste Version des RNN-Taggers (verfügbar auf github), die für ähnliche Texte verwendet werden kann.

1 Einführung

In der Computerlinguistik ist die kaskadische Textverarbeitung seit Jahrzehnten eine übliche Methodik:[1] Der Text wird zuerst segmentiert (auch: tokenisiert), d.h. in ein Format gebracht, das ein Token pro Zeile zeigt, wobei mit dem Begriff Token entweder ein orthographisches Wort, eine Zahl oder ein Interpunktionszeichen gemeint ist. Alle Spatien werden üblicherweise dabei gelöscht; danach folgen weitere Verarbeitungsschritte, welche die resultierende Liste bzw. einspaltige Tabelle um weitere Spalten bzw. Annotationen auf Token-Ebene wie Wortart und Lemma ergänzt. Dadurch ergibt sich

[1] Zwar ist auch eine End-to-End-Verarbeitung möglich, die Tokenisierung, Tagging und Lemmatisierung in einem Schritt durchführt, bspw. mit einem Encoder-Decoder-Ansatz. Dies ist jedoch noch keine gängige Praxis.

eine dreispaltige Tabelle, die wiederum als Eingabedatei für weitere (z.B. syntaktische) Analysen oder auch direkt für korpuslinguistische Untersuchungen verwendet wird. Von manchen Segmentierungswerkzeugen werden auch Satzgrenzen erkannt und als strukturelle Merkmale des ursprünglichen Textes automatisiert eingefügt.

Diese kaskadische Verarbeitung ist sinnvoll, hat allerdings den Nachteil, dass sich Fehler in der Tokenisierung auf die weiteren Verarbeitungsschritte auswirken, denn was falsch tokenisiert ist, kann kaum korrekt mit einer Wortart annotiert werden. Hier zeigt sich, dass auftretende Interpunktion problematisch sein kann, denn sie ist in Teilen ambig, kann also mehrere Funktionen haben.

In diesem Artikel fokussieren wir uns auf den Apostroph, welcher in der uns vorliegenden Fassung der Texte einheitlich als gerader Apostroph (') auftritt. Wir vernachlässigen aber nicht ein weiteres Problem der Tokenisierung: Mehrwortlexeme und mehrteilige Namen, deren Bestandteile durch Bindestriche, aber auch durch Apostroph und/oder Leerzeichen getrennt werden.

Das Lindenberg-Korpus beinhaltet 2.827 Apostrophe (von 448.996 Zeichen insgesamt), die im Original teilweise als eigene Tokens, teilweise als Teil von fälschlicherweise als Tokens identifizierten Einheiten auftreten (z.B. *ich's*). Da wir eine möglichst hohe Korrektheit bei der späteren Wortartenannotation erzielen möchten, halten wir ein angepasstes Segmentierungsverfahren für notwendig.

Apostrophe haben in schriftlich konzipierten deutschen Standardtexten mehrere Einsatzmöglichkeiten, wie in (a) bis (c) geschildert:

(a) Zitate eingrenzen (*Sie bezeichnete den Artikel als 'groben Unfug'.*)
(b) Den Genitiv kennzeichnen bei Wörtern, die auf s enden (*Sokrates' Äußerung, ...*)
(c) Das Abkürzen von Jahreszahlen (*Im Frühjahr '21 kam der Schnee.*)

In eher mündlich konzipierten Texten finden sich darüber hinaus öfter Verwendungen wie in (d) bis (h) beschrieben. Hierbei sind dialektale Einsatzmöglichkeiten nicht berücksichtigt.

(d) Ersetzen eines Schwa am Ende eines Wortes (*Ich hab' das auch gelesen, Lass' mich in Ruh'*)
(e) Ersetzen eines Schwa innerhalb eines Verbs (*Wir sind verlor'n*)
(f) Verkürzen eines Artikels nach einer Präposition, wobei die Wörter miteinander verbunden werden (*Gekauft für'n Apfel ...*)
(g) Verkürzen eines Artikels, abgetrennt vom vorherigen Wort (*... und 'n Ei.*)
(h) Verkürzen eines expletiven *es* und Verbindung mit dem vorstehenden Wort (*Ja, gibt's denn so was?*)

Kaum eine der aktuellen Publikationen über Tagsets und ihre Erweiterungen beschreibt, wie die konkrete Tokenisierung in solchen Fällen zu erfolgen hat. Im Artikel "What is a word, What is a sentence? Problems of Tokenization" hatten Grefenstette und Tapanainen (1994) zwar bereits mögliche Herausforderungen und verschiedene Lösungen zu den hier im Fokus stehenden Apostrophen beschrieben, aber kein generelles

Handhabungskonzept vorgelegt. Bartz et al. (2013) erwähnen zwar teilweise die o.g. Fälle, gehen aber in ihrer Beschreibung aufkommender Segmentierungsprobleme von Tokenisierern für internetbasierte Kommunikation nicht weiter auf diese speziellen Fälle ein. Sie schlagen allerdings auch Lösungsmöglichkeiten für die dadurch aufkommende Problematik der kaskadischen Verarbeitung vor, denen wir in diesem Beitrag teilweise folgen.

Besonders viele Vorkommen von Apostrophen in verschiedenen Anwendungsfällen finden wir in den Texten des Songwriters und Interpreten Udo Lindenberg, die wir als Teil aus dem Songkorpus (*songkorpus.de*) zur Verfügung gestellt bekamen und auf die wir uns als typisches Beispiel eines Nicht-Standardtextes fokussieren. Trotz vieler Angebote an Tokenisierungswerkzeugen fand sich keines, das diese Texte adäquat und einheitlich tokenisiert. Daher beschreiben wir hier, Bartz et al. (2013) folgend, unseren eigenen zum Teil regel- und wissensbasierten korrigierenden Ansatz der Vereinheitlichung und optimalen Vorbereitung von Lindenberg-Texten für das Tagging, das auch ein Post-Editing umfasst, wie auch in Zinsmeister et al. (2014, S. 4101) angedacht.

In den Abschnitten 3 und 4 beschreiben wir unsere Anpassungen der bereits im Songkorpus mit Wortarten annotierten Fassung der Texte, die sich unter anderem aus der neuen Tokenisierung ergeben. Ziel ist die Erstellung eines neuen Goldstandard-Korpus, mit dessen Hilfe ein Tagger trainiert werden kann. In Abschnitt 5 evaluieren wir drei verschiedene Versionen des RNNTaggers (Schmid, 2019):

1. Die online verfügbare Originalversion des deutschen RNNTaggers, die nur auf dem Tiger-Korpus (Brants et al., 2004) trainiert wurde;
2. eine Version, die auf dem Tiger-Korpus und dem Trainingsteil des ursprünglichen Lindenberg-Korpus trainiert wurde;
3. eine dritte Version, die auf dem Tiger-Korpus und dem Trainingsteil des von uns überarbeiteten Lindenberg-Korpus trainiert wurde.

Die Evaluationsergebnisse zeigen, dass sich die Qualität der Wortartannotation deutlich verbessert, wenn der Text optimal tokenisiert wird und der Tagger auf einem Trainingskorpus aus der Zieldomäne trainiert wird. Wir zeigen auf, dass Tokenisierung immer noch ein nicht-triviales Problem darstellt, das bei der computerlinguistischen Verarbeitung eine wichtige Rolle spielt.

Während das Originalkorpus nach unserer Zählung 91.223 Tokens, davon 1.991 als Eigennamen erkannte Tokens (918 Types) umfasst, beinhaltet der neue Goldstandard 89.791 Tokens, davon 1.901 als Eigennamen erkannte, z.T. mehrteilige Tokens (866 Types). Die geringere Anzahl an Tokens insgesamt im Goldstandard erklärt sich aus der Zusammenschreibung von Tokens mit Apostroph durch den neuen Tokenisierungsansatz, erläutert in Tabelle 2.

2 Tokenisierung: Besonderheiten

2.1 Interpunktion

Liedtexte werden oft zumindest teilweise ohne Interpunktion verfasst. In ihrer üblicherweise vorliegenden Form zeigen Zeilenumbrüche stattdessen an, dass ein (Teil-)Satz endet. Darauf begründet beschreibt Schneider (2020, S. 843) eine TEI P5-konforme Annotation der gegebenen Liedzeilen, die es ermöglicht, Satzgrenzen zu erkennen. Die Lindenberg-Texte wurden entsprechend aufbereitet und liegen im songkorpus-Repositorium mit z.T. ergänzter Interpunktion vor.

Punkt-Disambiguierung ist, wie bereits in Grefenstette und Tapanainen (1994) geschildert, immer noch ein Problem: So erkennen einige aktuell verfügbare Tokenisierer wie z.B. der ASV Segmentizer[2] Punkte, die eigentlich Teil von Abkürzungen sind (wie in *Dr.* oder *o.k.*) nicht als solche, trennen diese ab und verhindern so eine korrekte Annotation.

Dazu treten in den vorhandenen Texten Sequenzen von zwei oder drei Punkten (oder auch das Tripledot-Zeichen …) auf, manchmal gefolgt von einem Satzendepunkt, manchmal nicht. Falls kein Satzendepunkt folgt, sollte man davon ausgehen, dass das Tripledot-Zeichen die Satzgrenze darstellt, es muss also vom vorgehenden Wort getrennt werden, auch wenn es diesem ohne Leerzeichen folgt. Auch für diese spezifische Verwendung empfiehlt sich ein regelbasiertes Post-Editing der tokenisierten Fassung, weil die Anzahl der Vorkommen im Korpus zu gering ist, um eine Maschine das Phänomen mit ausreichender Qualität lernen zu lassen.

2.2 Apostrophe als Herausforderung für die Tokenisierung

Die nachfolgenden Beispiele (1) bis (11) aus dem Lindenberg-Korpus verdeutlichen die dortige Verwendung von Apostrophen, die wir in Tabelle 1 tiefer analysieren bzw. kategorisieren werden. Zur Fokussierung wurden die Beispiele teilweise auf das Wesentliche verkürzt. Wo immer möglich erfolgt eine Zuordnung zu den oben genannten bekannten Phänomenen (a) bis (h).

1. *Die kämpfen für's ewige Gestern.* (f)
2. *Für 'ne Woche nach Wien.* (g)
3. *Die Frau schreibt'n läppischen Abschiedsbrief.*
4. *Ich baute 'ne Mauer um mein Herz.* (g)
5. *Auf der Erde gibt's sowas noch.* (h)
6. *Ich denk' immer nur an dich.* (d)
7. *Was sonst so um mich 'rum passiert.*
8. *Da kommt man nicht so ohne weit'res rein.* (e)
9. *Es sind des Haifisch's Flossen rot.*
10. *Im Sommer '84 kommen wir vorbei.* (c)
11. *Einen Groupie hab'n die auch.*

[2] verfügbar auf `https://weblicht.sfs.uni-tuebingen.de`.

Wir können die aufgeführten Beispiele in Tabelle 1 klar in 11 Kategorien verorten. Spalte 2 in Tabelle 1 zeigt auf, dass Segmentierung nicht unabhängig von der anschließenden Annotation mit Wortarten betrachtet werden kann. Schließlich können wir keine Tokens definieren, für die das Tagset im nächsten Verarbeitungsschritt keine Annotierungsmöglichkeit bietet. Wir müssen uns auch wie im Fall 9 daran halten, dass wir, den Grundsätzen der Korpuslinguistik folgend, auch leicht abweichende Varianten von der Standardschreibweise prozessieren sollten, wollen also auch Fälle wie *Haifisch's* verarbeiten, auch wenn eigentlich *Haifisches* bzw. *Haifischs* erwartet werden würde.

Wir verwenden bei unserem Vorgehen das ursprüngliche STTS-Tagset 1.0 (Schiller, Teufel & Stöckert, 1999) vor allem, weil wir in den vorliegenden Texten keine Fälle sehen, die eine Erweiterung benötigen (siehe auch Abschnitt 3.1). Dazu gibt es bis heute wenige Tagger, die für spätere Fassungen dieses Tagsets trainiert wurden. Auch das zugrundeliegende Korpus verwendet diese Annotation.

Nr.	Kategorie	Beschreibung	Beispiel
1	APPR'ART	Ein verkürzter Artikel folgt APPR ohne Leerzeichen	... *für's ewige* ...
2	APPR_'ART	Ein verkürzter Artikel folgt APPR nach einem Leerzeichen	... *für 'ne* ...
3	^APPR'ART	Ein gekürzter Artikel folgt einer anderen Wortart als APPR ohne Leerzeichen (wobei das Zeichen ^ negiert)	... *schreibt'n* ...
4	^APPR_'ART	Ein gekürzter Artikel folgt einer anderen Wortart als APPR nach einem Leerzeichen	... *baute 'ne* ...
5	POS'PPER	Ein gekürztes Pronomen (*es*) folgt einer beliebigen Wortart ohne Leerzeichen	... *gibt's* ...
6	POS'	Verkürztes Wort (Schwa in Endstellung oder Endung gelöscht)	... *denk'* ..., ... *is'* ...
7	'POS	Verkürztes Wort (meist Silbe *he-* aus ADV gelöscht)	... *mich 'rum* ...
8	^VTeil1'Teil2	Verkürztes Wort (kein Verb, Schwa im Wort gelöscht)	... *weit'res* ...
9	GEN's	Verkürzte Genitiv-Form	... *Haifisch's* ...
10	'YEAR	Verkürzte Jahreszahlen	*'84*
11	V'n	Verkürzte Verbform (Schwa im Wort gelöscht)	... *hab'n* ...

Tabelle 1: Kategorisierung von Apostrophenvorkommen in Lindenberg-Texten

2.3 Mehrwortlexeme, die Apostrophe, Bindestriche und/oder Leerzeichen enthalten

Beim Auftreten von Komposita, deren Bestandteile durch Bindestriche getrennt werden (aus den Lindenberg-Texten: *Schlager-Fuzzi, Gerade-aus-Maus, Schicki-Micki-Mode-Huhn, 08/15-Casanovas*), können sich Tokenisierungs- bzw. Taggingprobleme ergeben, wie der ASV-Tokenisierer zeigt, der im Wort *U-Boot* drei Tokens erkennt. Bekannt ist das außerdem bereits für Komposita, deren Bestandteile durch Leerzeichen getrennt sind (*Apollo 13, Rolling Stones, St. Tropez*) bzw. Kombinationen aus beiden (*Never Come Back-Airline, D-471 81 61*). Zuletzt – und hier gibt es bei den Songtexten besonders viele Fälle – wird die Bezeichnung *Rock'n'Roll* (auch *Rock'n Roll* oder *Rock 'n' Roll*) in den Texten Lindenbergs nicht nur auf verschiedenste Weisen geschrieben, sondern auch mit weiteren Wörtern ergänzt, wodurch neue Komposita gebildet werden (*Rock'n'Roll-Gespenster, Rock 'n' Roll Zigeuner*).

Man kann von einem Tokenisierer nicht erwarten, dass er diese Schreibweisen alle voneinander unterscheiden lernt und Namen oder englische Abkürzungen zweifelsfrei identifizieren kann. Daher werden Leerzeichen üblicherweise beim Tokenisierungsvorgang gelöscht, die Bestandteile der Namen voneinander getrennt und erst in einem weiteren Verarbeitungsschritt (der sog. *Named Entity Recognition*) wieder zusammengefügt. Ähnlich gehen wir vor, im Einzelnen beheben wir das Problem wie folgt:

1. Wir tokenisieren den Text zuerst regulär mit dem Tokenisierer ohne externe Wissensbasis.
2. Wir annotieren die Tokenliste mithilfe eines passenden Taggers bzw. lassen mit einem entsprechenden Werkzeug Namen erkennen.
3. Die so erkannten Namen werden von einem Skript gesammelt und in eine Liste geschrieben.
4. Die Liste der mehrteiligen Namen wird von Hand überprüft und korrigiert.
5. Anschließend wird der Text unter Berücksichtigung der erstellten Liste vom Tokenisierer erneut tokenisiert und steht dem Tagger korrigiert zur Verfügung.

Der von uns verwendete RNN-Tagger (Schmid, 2019) konnte auch ohne besonderes Training diese mehrteiligen Tokens zuverlässig als Eigennamen erkennen. Die Erweiterung der Eigennamen-Liste des Tokenizers um die neuen Eigennamen ist derzeit noch nicht automatisiert und muss ggf. für andere Korpora erneut durchgeführt werden.

3 Tokenisierung und anschließende Annotation mit Wortarten (Tagging)

3.1 Analyse der Fälle 1 bis 5 nach Tabelle 1

Bartz et al. (2013) schlagen Wortartannotationen von umgangssprachlich kontrahierten Formen (dort: Phänomentyp III.2) vor. Sie berücksichtigen allerdings nicht die hier vorliegende vielfältige Verwendung von Apostrophen. Wenn wir z.B. “APPR'ART” (wie in “für'n”) als ein Token erkennen und dieses mit KTRAPPRART annotieren würden, wäre keine Gleichbehandlung mit “APPR 'ART” (mit trennendem Leerzeichen) möglich.

Wir gehen nicht davon aus, dass die Verwendung des Apostrophs bzw. des Leerzeichens in den Liedtexten andere als phonetische Gründe hat und entscheiden uns dafür, bei der Tokenisierung alle mit einem Apostroph verkürzten Artikel einheitlich zu behandeln, sie also als eigenständige Tokens zu segmentieren und mit dem Tag "ART" zu annotieren. Im Test mit verschiedenen Tokenizern, die in WebLicht (Hinrichs, Zastrow & Hinrichs, 2010) angeboten werden, zeigt sich, dass auch diese eine Abtrennung des Artikels in fast allen Fällen korrekt durchführen (die Ergebnisse finden sich in Tabelle 3).

Verkürzte Pronomina (Fall 5), die auf andere Wörter folgen, werden von uns genauso behandelt wie verkürzte Artikel: sie sind eigenständige Tokens und werden ggf. von dem vorhergehenden Token abgetrennt.

3.2 Analyse der Fälle 6-10 nach Tabelle 1

Die Fälle 6 bis 11 beschreiben verkürzte Wortformen. Hier darf der Wortteil ab dem Apostroph keinesfalls abgetrennt werden und auch der Apostroph muss am bzw. im Wort verbleiben. Die meisten Tokenizer aus WebLicht (siehe Abschnitt 4.2) erkennen übrigens Tokens, die mithilfe von Apostrophen intern verkürzt werden, korrekt.

3.3 Analyse des Falls 11 nach Tabelle 1

Fall 11 ist problematisch. In der ambig erscheinenden Form *hab'n* könnten zwei Verkürzungen beinhaltet sein: *hab 'n* (*habe ein/einen*), es könnte sich aber auch um ein einfaches, intern verkürztes Verb handeln: *haben*. Eine Einzelfallentscheidung muss der Tokenisierer treffen, eventuell muss diese Entscheidung jedoch kontextabhängig im Rahmen eines manuellen Post-Editing korrigiert werden.

3.4 Zusammenfassung

Tabelle 2 zeigt unsere Wunsch-Tokenisierung der mit Apostroph versehenen Tokens, die sich aus der bisherigen Argumentation ergibt. Fälle 11A und 11B werden im Rahmen des Posteditings manuell unterschieden, bevor die Wortartannotation durchgeführt wird.

4 Vorgehen bei der Tokenisierung

4.1 Regelbasierte Anpassung der bestehenden Tokenisierung

Für die weitere Verarbeitung wurden uns die von Schneider (2020, S. 843-844) beschriebenen Liedtexte von Udo Lindenberg mit z.T. eingefügten Satzzeichen (Kommata, Punkte und Fragezeichen) untokenisiert sowie tokenisiert und mit Wortarten sowie Lemmata annotiert zur Verfügung gestellt. So hatten wir die Möglichkeit, weitere Tokenizer anzuwenden und deren Ergebnis mit der bereits bestehenden Fassung zu vergleichen. Allerdings fanden sich in den bereitgestellten Dateien auch für uns nicht akzeptable Tokenisierungen, die wir regelbasiert mithilfe von Skripten sowie - in ambigen Fällen

Nr.	Kategorie	Token1	Token2
1	APPR'ART	*für*	*'s*
2	APPR 'ART	*mit*	*'ner*
3	VVFIN'ART	*schreibt*	*'n*
4	VVFIN'ART	*baute*	*'ne*
5	PWAV'PPER	*wo*	*'s*
6	VVFIN'	*denk'*	
7	POSS'ADV	*mich*	*'rum*
8	gekürztes ADJ	*weit'res*	
9	Genitiv's	*Haifisch's*	
10	'Jahr	*'84*	
11A	V'n	*hab'n*	
11B	V'n	*hab*	*'n*

Tabelle 2: Default-Tokenisierung der Apostrophvorkommen in Lindenberg-Texten

- manuell angepasst haben. Dadurch entstand ein Goldstandard, der für die weiteren Verarbeitungsschritte verwendet wurde.

Um die Evaluation der in WebLicht vorhandenen Tokenisierer zu vereinfachen, wurden Sätze mit den als problematisch bekannten Phänomenen aus dem Goldstandard extrahiert und diese Testsuite (siehe Tabelle 5 im Anhang 1) in WebLicht geladen. Diese Testsuite umfasst alle oben beschriebenen Phänomene. Unsere Default-Tokenisierung umfasste 457 Tokens mit insgesamt 68 Fällen, überwiegend Apostroph-Einsätze, aber auch Komposita und Tripledots-Verwendungen. Alle Tokenizer in WebLicht (siehe Tabelle 6 in Anlage 2) erkennen jedoch eine höhere Anzahl, wobei der ASV Tokenizer mit den meisten (555) Tokens alle Apostrophe abtrennt. Dies ist sicherlich der Tatsache geschuldet, dass das Werkzeug eigentlich als Satz-Segmentierer eingesetzt werden sollte und nur eine einfache Tokenisierungkomponente enthält[3]. Allerdings wird er augenscheinlich gleichwertig zu den weiteren Tokenizern in WebLicht zur Verwendung angeboten. Auch die anderen Phänomene werden von diesem Tokenizer nicht identisch zu unserem Default prozessiert. Sehr problematisch sind Segmentierungen wie z.B. die Sequenz *geseh* | ' | *n* (die senkrechten Striche symbolisieren die erkannten Tokengrenzen). Damit werden auch nicht mit eine Wortart annotierbare Tokens (Einzelbuchstaben) erkannt. Der ASV-Segmentierer wird daher als nicht geeignet angesehen, diese Art von Texten zu prozessieren.

Die beiden SFS-Tokenizer haben alle für uns relevante Phänomene gleich gehandhabt und zum Beispiel verkürzte Verben sehr gut erkannt, dafür jedoch Artikel nicht von vorstehenden Präpositionen abgetrennt. Sie trennen außerdem abkürzende Apostrophe

[3]Siehe Erläuterungen auf `https://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/Tools_in_Detail#Tokenizers`.

Ergebnis	SoJaMo	sfs-OpenNLP	sfs-Tübingen	BBAW	Blingfire	ASV-Segmentation
wie Default	42	47	47	56	11	0
nicht wie Default	26	21	21	12	57	68

Tabelle 3: Ergebnisse der WebLicht-Tokenizer

am Anfang von Sätzen wenn Großschreibung folgt, wie in *'Ne dunkle Wand*. Fast dieselben Abweichungen zeigen sich auch bei SoJaMo. Blingfire erkannte außerdem Komposita wie *U-Boot* nicht. Tabelle 3 zeigt die Zahl der aufgetretenen Abweichungen aufsummiert[4].

4.2 Tokenisierung mit statistischer Desambiguierung

Da die vorhandenen Tokenisierer die Songtexte noch nicht zufriedenstellend tokenisieren konnten, haben wir einen neuen Tokenisierer mit einer statistischen Desambiguierungskomponente entwickelt, der auf Ideen in (Schmid, 2000) basiert. Der Tokenisierer verarbeitet seine Eingabe in folgenden Schritten:

- Ersetzung von (Folgen von) Leerzeichen, Tabulatoren und anderen "Whitespace"-Symbolen durch Tokengrenzen
- Abtrennung von Satzzeichen, Klammern und Anführungszeichen am Beginn und Ende jedes Tokens als separate Tokens
- statistische Desambiguierung zwischen Satzpunkten, Abkürzungspunkten und Ordinalzahlpunkten wie in (Schmid, 2000) beschrieben
- statistische Desambiguierung von Apostrophen wie unten beschrieben
- Erkennung von Mehrwort-Einheiten durch eine einfache Longest-Match-Suche mit einer gegebenen Liste von Mehrwortausdrücken.

4.2.1 Statistische Desambiguierung von Apostrophen

Wenn ein Apostroph innerhalb eines Tokens auftritt, betrachten wir folgende Möglichkeiten (wobei der Unterstrich für ein Leerzeichen steht):

- Der Apostroph ersetzt "e" wie in *ander'n*.
- Der Apostroph ersetzt "e_" wie in *möcht'so*.
- Der Apostroph ersetzt "_e" wie in *gibt's*.
- Der Apostroph ersetzt "ei" wie in *irgend'nem*.
- Der Apostroph ersetzt "_ei" wie in *für'n Ei*.
- Der Apostroph ersetzt "_eine" wie in *für'n Apfel*.

[4] Dies liegt darin begründet, dass eine komplette Übersicht der Einzelergebnisse den Rahmen dieses Beitrags sprengen würde (gedruckt umfasst sie 18 Seiten). Die Einzelergebnisse sind jedoch bei den AutorInnen dieses Beitrags verfügbar

- Der Apostroph ersetzt "_de" wie in *Er ist über'n Berg.*
- Der Apostroph ersetzt "_da" wie in *über's Ziel.*
- Der Apostroph ersetzt "e_e" wie in *hab's.*
- Der Apostroph ersetzt nichts wie in *Smog'n'Roll.*[5]

Wir desambiguieren zwischen diesen Ersetzungsoperationen, indem wir den Apostroph im Token t nacheinander durch jede der obigen Zeichenfolgen – inklusive den Apostroph selbst – ersetzen und die Zeichenfolge r mit der höchsten Aposteriori-Wahrscheinlichkeit $p(r|t)$ laut folgender Gleichung auswählen:

$$p(r|t) = \frac{p_{lm}(t/r)\; p_{op}(r \rightarrow \text{'})}{\sum_s p_{lm}(t/s)\; p_{op}(s \rightarrow \text{'})} \tag{1}$$

Dabei bezeichnet t/r das Ergebnis der Ersetzung des Apostrophs im Token t durch r. Bspw. ergibt *hab's/e_e* den String *habe_es*. Das Sprachmodell $p_{lm}(t/r)$ liefert die Wahrscheinlichkeit des Wortes (bzw. Wortpaares) t/r, und $p_{op}(r \rightarrow \text{'})$ liefert die Wahrscheinlichkeit dafür, dass der Teilstring r in t/r durch den Apostroph ersetzt wird (statt unverändert zu bleiben).[6]

Für das Token *schreib'n* bekommen wir unter anderem die möglichen Ersetzungen *"schreiben"* und *"schreib ein"* mit den Bewertungen $p_{lm}(\text{schreiben})\; p_{op}(\text{e} \rightarrow \text{'})$ bzw. $p_{lm}(\text{schreib ein})\; p_{op}(\text{\_ei} \rightarrow \text{'})$.

Wenn die wahrscheinlichste Ersetzungsoperation $\hat{r} = \arg\max_r p(r|t)$ mit einem Leerzeichen beginnt (bspw. _ei), fügen wir vor dem Apostroph eine Tokengrenze hinzu (bspw. *für 'n*). Wenn die wahrscheinlichste Ersetzungsoperation mit einem Leerzeichen endet (e_), fügen wir nach dem Apostroph eine Tokengrenze hinzu (*möcht' so*). Im Fall der Ersetzung e_e verdoppeln wir den Apostroph und fügen dazwischen eine Tokengrenze ein (*hab' 's*). In allen anderen Fällen bleibt das Token unverändert.

4.2.2 Sprachmodell

Für das Sprachmodell $p_{lm}(.)$ verwenden wir ein wortbasiertes Markowmodell erster Ordnung (Bigramm-Modell), welches nicht zwischen groß- und kleingeschriebenen Buchstaben unterscheidet. Das Bigramm-Modell wird mit interpolierter Kneser-Ney-Glättung "on-the-fly" auf den zu tokenisierenden Daten (ohne Verwendung der Goldstandard-Annotation) trainiert. Daher ist dieses Verfahren weniger geeignet, um kurze Texte zu tokenisieren. Kurze Texte können jedoch mit längeren Textstücken gleicher Art zusammengefasst werden, um bessere Ergebnisse zu erzielen.

[5] Man kann hier natürlich argumentieren, dass es sich im Englischen durchaus um eine Verkürzung von *and* zu *'n'* handelt. Mit fremdsprachlichen Verkürzungen befassen wir uns jedoch im Rahmen dieser Arbeit nicht.

[6] $p_{op}(r \rightarrow \text{'})$ ist keine Wahrscheinlichkeitsverteilung über die verschiedenen Ersetzungen r. Es gilt also in der Regel: $\sum_r p_{op}(r \rightarrow \text{'}) \neq 1$. Unser statistisches Modell ist ein Noisy-Channel-Modell, welches erst mit dem Sprachmodell p_{lm} eine Phrase t/r generiert (bspw. **habe_es**) und dann zufällig mit Wahrscheinlichkeit $p_{op}(r \rightarrow \text{'})$ entscheidet, ob der Teilstring r in t (hier: e_e durch einen Apostroph ersetzt (und damit "verrauscht") wird oder nicht. Bei der Desambiguierung versuchen wir zu rekonstruieren, wie die unverrauschte Zeichenfolge wahrscheinlich aussah.

4.2.3 Training

Unser statistisches Modell umfasst die Parameter $p_{lm}(.)$ und $p_{op}(.)$ und wird mit dem Expectation-Maximization-Algorithmus (EM-Algorithmus) trainiert. Dazu werden zunächst die zu verarbeitenden Texte mit dem Tokenisierer von Schmid (2000) vorläufig tokenisiert. Auf den tokenisierten Texten wird dann das Sprachmodell $p_{lm}(.)$ trainiert, um es zu initialisieren. Die Wahrscheinlichkeiten der verschiedenen Ersetzungsoperationen $p_{op}(.)$ werden mit 0,01 initialisiert mit der Ausnahme $p_{op}(\text{'} \rightarrow \text{'}) = 0,0001$. Diese Ausnahme dient dazu, (echte) Ersetzungen zunächst zu präferieren.[7] Dann führen wir zwei[8] EM-Iterationen durch. In jeder EM-Iteration berechnen wir zunächst nach Formel 1 für jedes apostrophierte Token t die Aposteriori-Wahrscheinlichkeit jeder möglichen Ersetzungs-Operation r.

Für jede Operation r summieren wir dann die Aposteriori-Wahrscheinlichkeiten $p(r|t)$ über alle apostrophierten Tokens t des Textes wie folgt:

$$f_{r,t} = F_t \, p(r|t)$$

F_t ist hier t's Häufigkeit im Text. Mit den so erhaltenen erwarteten Häufigkeiten schätzen wir die Ersetzungswahrscheinlichkeiten neu:

$$p_{op}(r \rightarrow \text{'}) = \frac{\sum_t f_{r,t}}{\sum_t f_{r,t} + F_{t/r}}$$

$F_{t/r}$ ist hier die Häufigkeit des Tokens (oder Token-Paares) t/r (bspw. *habe_es*) im Text.

Analog werden die Parameter des Sprachmodelles neugeschätzt. Für jedes Wortpaar (s,t) mit $t = t_1\text{'}t_2$, subtrahieren wir 1 von seiner Texthäufigkeit $h_{s,t}$ und addieren für jede mögliche Ersetzung r ohne Leerzeichen den Wert $p(r|t)$ zur Häufigkeit $h_{s,t_1 r t_2}$. Für jede Ersetzung $r = r_1\_r_2$ mit Leerzeichen addieren wir den Wert $p(r|t)$ zur Häufigkeit $h_{s,t_1 r_1}$ und zur Häufigkeit $h_{t_1 r_1, r_2 t_2}$. Dann schätzen wir die Parameter des Sprachmodelles mit der Kneser-Ney-Methode (Kneser & Ney, 1995) neu:

$$\begin{aligned} h_t &= \sum_s h_{s,t} \\ p(t) &= \frac{h_t - \delta_1}{\sum_{t'} h_{t'}} \\ p(t|s) &= \frac{h_{s,t} - \delta_2}{\sum_{t'} h_{s,t'}} \\ \alpha(s) &= 1 - \sum_t p(t|s) \end{aligned}$$

[7] Noch bessere Ergebnisse werden erzielt, wenn die Wahrscheinlichkeiten der Ersetzungsoperationen " da" und " eine" mit 1 initialisiert werden.

[8] Mehr EM-Iterationen haben zu etwas schlechteren Ergebnissen geführt.

$$
\begin{aligned}
k_t &= |\{(s,t)|h_{s,t} > 0\}| \quad \text{// Kneser-Ney-Methode} \\
p_{bo}(t) &= \frac{k_t}{\sum_{t'} k_{t'}} \\
p(s,t) &= p(s)\ (p(t|s) + \alpha(s)\ p_{bo}(t))
\end{aligned}
$$

δ_1 und δ_2 sind hier Discounts, die nach der Formel $N_1/(N_1 + 2N_2)$ berechnet werden, wobei N_i die Zahl der Wörter/Wortpaare mit Häufigkeit i ist. Discounting bei den Unigramm-Wahrscheinlichkeiten $p(t)$ hat sich in Experimenten als vorteilhaft erwiesen.

4.2.4 Evaluation

Wir evaluierten den statistischen Tokenisierer auf dem Goldstandard-Korpus, das 89.791 Tokens umfasst. Der Tokenisierer machte dabei insgesamt 52 Fehler, die sich in folgende Klassen einteilen lassen:

- In 11 Fällen wurde ein Satzpunkt nicht abgetrennt, weil der nachfolgende Satz mit einem Kleinbuchstaben begann.

  Beispiel: *... du bist verloren und ganz alleine hier oben. so 'n Gefühl, das hab' ich manchmal ...*
- In 8 Fällen wurde *'s* nicht abgetrennt, weil das vorausgehende Wort ausschließlich mit nachfolgendem *'s* auftrat.

  Beispiele: *solln's klappt's versuchen's*
- In 6 Fällen wurde *'n* nicht abgetrennt, weil der Apostroph auch für *e* hätte stehen können.

  Beispiele: *mal'n war'n wär'n echt'n*
- In 5 Fällen wurde *'s* nicht von dem Wort *krieg* abgetrennt, weil das Wort *Krieges* relativ häufig auftrat und Groß-/Kleinschreibung nicht unterschieden wird.
- In 5 Fällen wurde nach dem Ausdruck *D-471 81 61* der nachfolgende Satzpunkt nicht abgetrennt. Diese Fehler waren durch eine Lücke im Programmcode verursacht.
- In 4 Fällen wurden Tokenisierungsfehler durch fehlende Leerzeichen in der Eingabe verursacht.

  Beispiele: *... oder -denn wissen-*
- 3 Tokenisierungsfehler lassen sich auf fremdsprachliche Einsprengsel zurückführen: Der Ausdruck *Harry's* wurde fälschlich zerlegt und die Ausdrücke *C'est* und *That's* wurden fälschlich nicht zerlegt.
- In 3 Fällen wurde vor dem Apostroph abgetrennt, weil die ungekürzte Wortform nicht auftrat.

  Beispiele: *zwei'n seid'nen*
- Die restlichen 7 Fehler lassen sich keiner größeren Klasse zuordnen.

5 Tagging

Für die Wortart-Annotation und Lemmatisierung verwenden wir den RNNTagger (Schmid, 2019), der auf rekurrenten neuronalen Netzen basiert.

Der RNNTagger wurde für Deutsch und viele andere Sprachen trainiert und ist für nicht-kommerzielle Zwecke frei verfügbar[9]. Für unsere Evaluationen haben wir den Tagger auf unterschiedlichen Korpora neu trainiert und verglichen.

5.1 Evaluation

Für die Evaluation teilten wir das Lindenberg-Korpus in Trainingsdaten, Entwicklungsdaten und Testdaten auf. Wir wählten die ersten 75.189 Tokens als Trainingsdaten, die nächsten 7.269 Tokens als Entwicklungsdaten und die restlichen 7.333 Tokens als Testdaten.

Da bei der Erstellung des Goldstandard-Korpus einige Fehler in der Wortart-Annotation des Originalkorpus korrigiert wurden, haben wir diese Korrekturen auf das Originalkorpus übertragen, um faire Ergebnisse zu erhalten. In Fällen, wo ein apostrophiertes Token fälschlich nicht aufgespalten worden war, wurde das Token mit einem Doppeltag annotiert. In Fällen, in denen ein Token fälschlich in mehrere kleinere Tokens aufgespalten worden war, annotierten wir alle Tokens bis auf das erste mit dem neuen Tag *PART*.

Um den Einfluss der unterschiedlichen Tokenisierungen auf die Taggingergebnisse zu untersuchen, führten wir Experimente durch, bei denen wir den RNNTagger entweder nur auf dem Tigerkorpus (Tiger) oder zusätzlich auf dem Trainingsteil des Lindenberg-Korpus mit der originalen (Tiger+original) oder der verbesserten Tokenisierung des Goldstandards (Tiger+goldstandard) trainierten. Jede Tagger-Version wurde viermal mit den Standard-Hyperparametern und unterschiedlichen Startwerten des Zufallszahlengenerators trainiert. Nach jeder Trainingsepoche wurde das Taggermodell auf den Entwicklungsdaten evaluiert. Das Modell aus der besten Trainingsiteration wurde jeweils auf den Testdaten evaluiert. Für jede Taggervariante gab es somit 4 Evaluationsergebnisse. Die Tokenisierung der Trainings-, Entwicklungs- und Testdaten war in jedem Experiment einheitlich, d.h. entweder immer die Originaltokenisierung oder immer die neue Goldstandard-Tokenisierung.

Tabelle 4 zeigt die erzielten Genauigkeiten. Der Tagger, der nur auf dem Tiger-Korpus trainiert wurde, erzielte mit der Goldstandard-Tokenisierung im Mittel deutlich bessere Ergebnisse (+0.6%) als mit der Original-Tokenisierung. Zusätzliches Training auf dem Lindenberg-Korpus verbesserte die Ergebnisse sogar um +3,0% mit der Goldstandard-Tokenisierung (Tiger+goldst.) und um +3,7% mit der Original-Tokenisierung (Tiger+original).

Überraschenderweise erzielte der Tagger, der zusätzlich auf dem Korpus mit der originalen Tokenisierung trainiert wurde (Tiger+original), ebenso gute Ergebnisse wie der Tagger, der zusätzlich auf den Goldstandard-Daten trainiert wurde (Tiger+goldst.). Der

[9] `https://www.cis.lmu.de/~schmid/tools/RNNTagger`

Trainingsdaten	Entwicklungsd.	Testdaten	Genauigkeit in % (Mittel, Max., Min., Stdabw.)			
Tiger	original	original	92,58	92,80	92,44	0,18
Tiger	goldst.	goldst.	93,17	93,33	92,96	0,15
Tiger+original	original	original	96,24	96,33	96,15	0,09
Tiger+goldst.	goldst.	goldst.	96,15	96,37	96,03	0,15

Tabelle 4: Genauigkeit der verschiedenen Tagger-Varianten

Genauigkeitsunterschied von 0,09% war statistisch nicht signifikant.[10] Der RNNTagger ist also in der Lage, auch falsch tokenisierte Texte korrekt zu annotieren, wenn er die Problemfälle im Training kennenlernen konnte.

Eine weitere, allerdings weniger umfassende Evaluation haben wir mithilfe der Songtexte der Band *Fettes Brot* durchgeführt, um zu prüfen, inwieweit sich Verbesserungen durch die Verwendung der neu entwickelten Segmentierung – allerdings ohne manuelle Eingriffe für die Namenserkennung – ergeben. Im Original-Korpus ist diese Sammlung mit 59.269 Tokens verzeichnet, sie beinhaltet 1.309 Apostrophe. Das unveränderte Tokenisierungswerkzeug aus (Schmid, 2019) erzeugt 78.360 Tokens, das hier neu entwickelte 78.028 Tokens. Bereits mit der bisherigen Version des Segmentierers (Schmid, 2019) werden die folgenden Phänomene korrekt erkannt:

- Löschen des Schwa in oder am Ende eines Worts, wie in *hör', hör'n, Fernseh'n*
- Abtrennen der gekürzten unbestimmten Artikel *ein, einer, einem, einen* wie in *'n Knall, 'ner Weile, aus'm Häuschen, durch 'en Reifen*

Die neue Version des Segmentierers erkennt zusätzlich die folgenden Phänomene

- Abtrennen des Personalpronomens / des explikativen *es*, wie in *gibt's, geht's, habt's*: 222 Vorkommen
- Abtrennen des gekürzten unbestimmten Artikels *ein* wie in *und'n Kind* : 78
- Abtrennen des gekürzten bestimmen Dativartikels *dem* wie in *auf'm Dreier*

Satz (12) unten veranschaulicht, wie der Tokenisierer in manchen Fällen den Apostroph verdoppelt (12a). In Satz (13) ist uns die Verwendung des ersten *'n* zwar nicht ganz klar[11], das neue Werkzeug erkennt jedoch zumindest den Artikel korrekt als eigenständiges Token.

12 *Und würd's nochmal von vorne losgehen, wärst du wieder dabei.*
12a *Und würd' **'s** nochmal von vorne losgehen, wärst du wieder dabei.*
13 *..., was'n das für'n Name?*
13a *..., was **'n** das für **'n** Name?*

[10] Der t-Test lieferte einen p-Wert von 0.37.
[11] Bei dem Ausdruck *was'n* könnte es sich um eine Verkürzung des Ausdruckes *was ist denn* handeln.

6 Mögliche weitere Arbeiten

Wir zeigen in diesem Beitrag auf, dass die Texte Udo Lindenbergs eine besondere computerlinguistische Prozessierung benötigen, da insbesondere der Apostroph dort ambig auftritt. Segmentierungswerkzeuge, die von WebLicht derzeit zur Verfügung gestellt werden und die auf Standard-Texten trainiert wurden, können diese Texte nur eingeschränkt und z.T. überhaupt nicht handhaben.

Es ist also durchaus noch nötig, aber selbst bei diesem eher kleinen Korpus auch möglich, Segmentierungswerkzeuge spezifisch für diese Textart zu entwickeln unter Berücksichtigung des später anzuwendenden Tagsets für die Wortartenannotierung. Wir haben ein solches Werkzeug vorgestellt, welches mit dem EM-Algorithmus direkt auf den zu verarbeitenden Texten trainiert wird und Apostrophe recht zuverlässig desambiguieren kann. Mit unserem Segmentierungswerkzeug gehen wir weiter als in (Schiller et al., 1999) beschrieben und ermöglichen es, mehrteilige Namen bereits bei der Tokenisierung als eigenständige Tokens zu identifizieren und sie damit während der weiteren Prozessierung mit den richtigen Wortart-Tags zu annotieren. Die erstellten Werkzeuge sind für andere Anwendungen frei unter `https://github.com/helgihu/German-Song-Tagger` verfügbar.

Wir erwarten, dass die vorgestellte Verarbeitungspipeline auch auf das Songkorpus als Ganzes anwendbar ist, sofern nicht unvorhergesehene neue Problemstellungen in Erscheinung treten. Die Tokenisierung des Gesamtkorpus kann in derselben Weise erfolgen wie beim Lindenberg-Teilkorpus. Wenn auf die Identifizierung noch unbekannter mehrteiliger Namen verzichtet werden kann, sind hier keine manuellen Eingriffe erforderlich. Wegen der Ähnlichkeit der Textsorten sollte die Tagging-Genauigkeit auch bei den Songtexten anderer Interpreten von dem zusätzlichen Training auf den manuell annotierten Lindenbergtexten profitieren, auch wenn dort die Steigerung der Genauigkeit etwas kleiner ausfallen dürfte.

Unsere Evaluation auf den Songtexten der Band *Fettes Brot* zeigte, dass sich die Behandlung von Apostrophen mit dem neuen Werkzeug eindeutig verbessert hat. Weitere, detailliertere Tests sowie die Erstellung eines Goldstandards wären allerdings nötig, um eine vollständige Evaluation dieser und andere Songtexte durchzuführen.

Literatur

Bartz, T., Beißwenger, M. & Storrer, A. (2013). Optimierung des Stuttgart-Tübingen-Tagset für die linguistische Annotation von Korpora zur internetbasierten Kommunikation: Phänomene, Herausforderungen, Erweiterungsvorschläge. *Journal for Language Technology and Computational Linguistics (JLCL)*, *28* (1), 157–198.

Brants, S., Dipper, S., Eisenberg, P., Hansen, S., König, E., Lezius, W., ... Uszkoreit, H. (2004). Tiger: Linguistic interpretation of a german corpus. *Journal of Language and Computation*, *2*, 597-620.

Grefenstette, G. & Tapanainen, P. (1994). What is a word, what is a sentence? problems of tokenization. In *Proceedings of the 3rd International Conference on Computational Lexicography* (S. 79–87).

Hinrichs, M., Zastrow, T. & Hinrichs, E. (2010). WebLicht: Web-based LRT Services in a Distributional eScience Infrastructure. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)* (S. 489–493).

Kneser, R. & Ney, H. (1995). Improved backing-off for m-gram language modeling. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* (Bd. 1, S. 181–184).

Schiller, A., Teufel, S. & Stöckert, C. (1999). *Guidelines für das Tagging deutscher Textkorpora mit STTS.* Zugriff am 2022-08-29 auf `https://www.ims.uni-stuttgart.de/documents/ressourcen/lexika/tagsets/stts-1999.pdf`

Schmid, H. (2000). *Unsupervised learning of period disambiguation for tokenisation.* Zugriff am 2022-10-19 auf `https://www.cis.lmu.de/~schmid/papers/tokeniser.pdf`

Schmid, H. (2019, May). Deep Learning-Based Morphological Taggers and Lemmatizers for Annotating Historical Texts. In *Proceedings of the 3rd International Conference on Digital Access to Textual Cultural Heritage (DATeCH).*

Schneider, R. (2020). A Corpus Linguistic Perspective on Contemporary German Pop Lyrics with the Multi-Layer Annotated « Songkorpus ». In *Proceedings of the 12th International Conference on Language Resources and Evaluation (LREC 2020)* (S. 842–848).

Zinsmeister, H., Heid, U. & Beck, K. (2014). Adapting a part-of-speech tagset to non-standard text: The case of STTS. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2014)* (S. 4097–4104).

Korrespondenzanschrift

Gertrud Faaß
Universität Hildesheim
Institut für Informationswissenschaft und Sprachtechnologie
gertrud.faass@uni-hildesheim.de

Helmut Schmid
Ludwig-Maximilians Universität München,
Institut für Informations- und Sprachverarbeitung
schmid@cis.lmu.de

7 Anhang 1

Testsuite für die WebLicht-Tokenizer

Nr.	Kategorie	TSNr.	Testsatz (TS)
1	APPR'ART	1	Die kämpfen für's ewige Gestern.
		2	Ich war nie der Typ für'n Liebesfilm.
		3	Mit'm U-Boot fahren wollte ich schon immer mal.
		4	Ich sitze an der Bar mit'nem Drink.
		5	Und die tanzt auf'm Tisch wie'n Gogogo-Girl.
2	APPR_'ART	6	Für 'ne Woche nach Wien.
		7	Mit 'nem Star oder mit 'nem Statist.
		8	Und hab was mit 'ner anderen Frau.
3	^APPR'ART	9	Dass es viel mehr war als nur so'n kleiner Flirt am Rand.
		10	Die Frau schreibt'n läppischen Abschiedsbrief.
		11	Au ja, 'n Teenie mit Doktortitel lila Strapse und drüber'n weißer Kittel
4	^APPR_'ART	12	Dann reißt er noch 'nen blöden Witz.
		13	Es geht doch hier nicht um 'ne schnelle sexuelle Nacht.
		14	Ich kenn' 'ne Lady.
		15	. 'Ne dunkle Wand und ein Riss geht durch die Zeit
		16	Au ja, 'n Teenie mit Doktortitel lila Strapse und drüber'n weißer Kittel
		17	Er wär 'n Astronaut.
		18	Ich baute 'ne Mauer um mein Herz.
		19	In der der Hand 'ne Cognacflasche und 'n Autogramm von Klaus Kinski.
		20	Doch er ist so 'ne Art Traum-Dompteur.
		21	Aus irgend 'nem blöden Grund rasen die aneinander vorbei.
		22	Als nur 'ne schnelle Romanze am Strand.
		23	Ist alles erst 'n paar Stunden her.

		24	Ich fuhr mit dem Auto n' bißchen zu schnelle.
5	POS'PPER	25	Auf der Erde gibt's doch sowas noch.
		26	Wo's dauernd auf die Fresse gibt.
		27	Wie's lächerlicher nicht mehr geht.
		28	Alles im Lot und wenn's untergeht.
		29	Und da gibts's auch Wahnsinnsfrauen.
6	POS'	30	Ich denk' immer nur an dich.
		31	Hab' nicht gewusst, dass das alles so stark ist.
		32	Bis ich absolut keine Luft mehr hab'.
		33	Denn da wär' beinah ein Schiffsunglück passiert.
		34	Oder wenn ich durchdreh', die letzten Scheine auf den Tresen hau'.
		35	Das lässt mich einfach nicht mehr in Ruh'.
		36	Ich hätt' dich beinah' nicht geseh'n.
7	'POS	37	Haben wir uns ein Glas Nost 'reingeknallt.
		38	Was sonst so um mich 'rum passiert.
8	V'n	39	Einen Groupie hab'n die auch.
		40	Ich muss Dich wiederseh'n.
		41	Und da darf keiner zwischensteh'n.
		42	Kannst' auch mal abschmier'n.
9	^V'n	43	Ich hab was mit einer ander'n Frau.
		44	Da kommt man nicht so ohne weit'res rein.
		45	Die krabbeln durch die seid'nen Betten.
10	GEN's	46	Es sind des Haifisch's Flossen rot.
11	'YEAR	47	Im Sommer '84 kommen wir vorbei.

Tabelle 5: Die Testsuite: z.T. verkürzte Sätze aus dem Lindenberg-Korpus

8 Anhang 1

WebLicht: Tokenizer
Informationen aus `https://weblicht.sfs.uni-tuebingen.de`

Tokenizer	Attribute	Description
SFS Tübingen	Desc:	Tokenizer/sentences from the OpenNLP project. The 'newlineBounds'
	parameter	treats newlines as a hard break (a sentence boundary).
	Creator(s):	SfS: Uni-Tuebingen
	Contact:	wlsupport@sfs.uni-tuebingen.de
	PID:	`http://hdl.handle.net/11858/00-1778-0000-0004-BA7B-4`
SoJaMo	Desc:	SoMaJo is a state-of-the-art tokenizer and sentence splitter for German web and social media texts. You can find more information : `https://github.com/tsproisl/SoMaJo`
	Creator(s):	SfS: Uni-Tuebingen
	Contact:	wlsupport@sfs.uni-tuebingen.de
	PID:	`http://hdl.handle.net/11022/0000-0007-E7D0-9`
CLAR: ASV	Desc:	The sentence segmentizer used by the Wortschatz project for German texts (also containing a very simple tokenizer)
	Creator(s):	CLARIN-D center, Natural Language Processing Group, University of Leipzig
	Contact:	clarin@informatik.uni-leipzig.de
	PID:	`http://hdl.handle.net/11022/0000-0000-94F4-5`

Blingfire	Desc:	Tokenizer/Sentencer from Microsoft, called BlingFire. It is designed for fast-speed and quality tokenization of natural language. For more information about the quality and efficiency of that tokenizer, please check out the corresponding github page: `https://github.com/microsoft/BlingFire`.
	Creator(s):	SfS: Uni-Tuebingen
	Contact:	wlsupport@sfs.uni-tuebingen.de
	PID:	`http://hdl.handle.net/11022/0000-0007-DA1F-2`
SFS-OpenNLP	Desc:	Tokenizer from the OpenNLP Project
	Creator(s):	SfS: Uni-Tuebingen
	Contact:	wlsupport@sfs.uni-tuebingen.de
	PID:	`http://hdl.handle.net/11858/00-1778-0000-0004-BA63-7`
BBAW	Desc:	detects word- and sentence boundaries in raw text using WASTE (`http://www.dwds.de/waste/`)
	Creator(s):	BBAW: Berlin-Brandenburg Academy of Sciences and Humanities
	Contact:	jurish@bbaw.de
	PID:	`https://hdl.handle.net/21.11120/0000-0008-3183-C`
IMS	Desc:	Czech, Slovenian, Hungarian, Italian, French, German, English tokenizer and sentence boundary detector
	Creator(s):	IMS: University of Stuttgart
	Contact:	clarin@ims.uni-stuttgart.de
	PID:	`http://hdl.handle.net/11022/1007-0000-0000-8E1F-F`

Tabelle 6: WebLicht Informationen über die dort verfügbaren Tokenizer