INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Webtool zur Betrachtung und Annotation von Satzalignments aus Wikipedia in vereinfachtem Englisch

Alex Galkin

Bachelorarbeit

Beginn der Arbeit:	19. Mai 2019
Abgabe der Arbeit:	19. August 2019
Gutachter:	Prof. Dr. Stefan Conrad
	Prof. Dr. Stefan Conrad

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 19. August 2019

Alex Galkin

Zusammenfassung

Hier kommt eine ca. einseitige Zusammenfassung der Arbeit rein.

Inhaltsverzeichnis

1 Einleitung

Die Vereinfachung von Text ist ein Prozess bei welchem Sätze so verändert werden, dass dabei die Grammatik, Fachbegriffe und die Struktur vereinfacht, die Kernaussage und/oder wichtige Informationen erhalten bleiben.

Diese Arbeit befasst sich damit komplexe Sätze aus dem English Wikipedia ihrem vereinfachten Gegenstück aus dem Simple English Wikipedia zuzuordnen, dabei behandeln beide Artikel das gleiche Thema, sind jedoch völlig unabhängig von einander geschrieben worden. Dies führt zu mehreren Problemen:

- Manche Sätze verfügen über kein Gegenstück
- Da die Satzstruktur oft vereinfach wird, untersuchen wir eine 1:N Beziehung - ein komplexer Satz kann oft in zwei oder noch mehr Sätze aufgeteilt worden sein.
- Da Artikel verschieden strukturiert sind können wir nicht die Struktur von Artikeln zur Hilfe ziehen

2 Implementierung

Als Programmiersprache wurde *Python* benutzt da es viele gute NLP Libraries dafür gibt. Die zwei meistverwendeten Python Libraries für NLP sind *NLTK* und *spaCy*. Für die Problemstellung wurde aus zahlreichen Gründen *spaCy* verwendet. Zum Einen hat es eingebaute, vortrainierte Word embeddings - im folgenden "Wort-Vektoren"genannt, zum Anderen ist es schneller als *NLTK* bei der Word Tokenization. Da das Programm webbasiert sein sollte wurde *Flask* als das Web Application Framework verwendet.

3 NLP

NLP - Natural Language Processing ist ein Teilfeld der Informatik welches sich damit befasst wie natürliche Sprachen von Computern verarbeitet werden können. Für die vorhandene Problematik ist die Vorverarbeitung von Text interessant, da vom User ausgewählte Strings als Eingabe verwendet werden und diese mit Hilfe von Vorverarbeitung bereit für die Algorithmen gemacht werden. Bei der Vorverarbeitung von Text geht es darum den Text für die effektive Weiterverarbeitung in den Algorithmen vorzubereiten. Dabei wird durch Normalisierung (Lowercasing, Lemmatisierung), das Entfernen von unerwünschten Token (Entfernen von Interpunktion, Entfernen von Stoppwörtern) und der Zusammenfassung von Wörtern zu Token (Named-entity recognition), den Text so uniform wie möglich zu gestallten um mit den Algorithmen ein möglichst präzises Ergebnis erzielen zu können. Es wird versucht möglichst viel "Rauschenßu entfernen. Als "Rauschen"bezeichnen wir Token welche nicht dabei helfen oder es sogar schwerer machen Matches zu finden.

Als Ausgangslage liegen der Wikipedia Artikel in English und der Wikipedia Artikel in Simple English vor. Damit die Algorithmen präzisere Ergebnisse erzielen können müssen die beiden Texte vorbereitet und verarbeitet werden. Zuerst werden die Texte in einzelne Sätze zerlegt und danach in zwei Listen gespeichert. Da der Benutzer nur einen Satz aus dem English Artikel aussucht, wird um Rechenkraft zu sparen, nicht der gesamte Artikel weiterverarbeitet, sondern nur der vom Benutzer ausgewählte Satz. Der Simple English Artikel wird Satz für Satz analysiert werden und muss somit in seiner gesammten Länge vorverarbeitet werden. Die einzelnen Sätze werden in Wörter zerlegt und ebenfalls in einer Liste gespeichert.

Somit liegen uns als Input zwei verschiedene Listen vor. Die erste Liste besteht aus den Token des vom Benutzer ausgewählten Satzes. In der zweiten Liste sind alle Sätze des Simple English Wikipedia Artikels gespeichert welche ebenfalls in ihre Token zerlegt worden sind.

Ausgewählter Satz in Liste = ['This', 'is', 'the', 'selected', 'sentence', '.']
Simple English Artikel in Liste = [['This', 'is', 'a', 'Wikipedia', 'Article','.'], ['It', 'contains', 'multiple', 'sentences', '.'], ['It', 'is', 'about', 'the', 'Heinrich', 'Heine', 'University', '.']]

3.1 Named-entity recognition

Bei der named-entity recognition (NER) geht es darum Eigennamen zu finden und diese dann zu klassifizieren. Ein Eigenname wäre sowas wie „New York“ die Kategorie dazu wäre „Stadt“. Desweiteren kann auch der Kontext betrachtet werden, mit „Apple“ könnte die Frucht aber auch das Unternehmen gemeint sein.

Wenn also der Satz „David bought shares of Apple for $300 million.“betrachtet wird, dann würde dieser wie folgt annotiert werden: „$[\text{David}]_{\text{Person}}$ bought shares of $[\text{Apple}]_{\text{Corporation}}$ for $[\$300 \text{ million}]_{\text{Money}}$.“

Durch die Anwendung von NER ist es möglich den Input für die Algorithmen weiter zu verfeinern. Es liegen die Sätze „Burger King tastes good.“ und „This Burger tastes good.“vor. Wenn eine einfache Tokenization und eine Entfernung von Satzzeichen

durchgeführt wird, dann werden folgende Listen erstellt:
Satz 1 = ['Burger', 'King', 'tastes', 'good']
Satz 2 = ['This', 'Burger', 'tastes', 'good']

Hierbei würden die Wörter „Burger", „tastes" und „ good" zwischen den beiden Sätzen matchen. Wenn jedoch NER angewendet wird, dann werden folgende Listen erstellt:
Satz 1 = ['Burger King', 'tastes', 'good']
Satz 2 = ['This', 'Burger', 'tastes', 'good']

Somit würden nur noch die Wörter „tastes" und „good" jeweils ein Match erzeugen und damit den Algorithmus präziser machen, da jetzt auch der Kontext der Wörter in betracht gezogen wird und es somit zu weniger falschen Matches kommt.

Ausgewählter Satz in Liste = ['This', 'is', 'the', 'selected', 'sentence', '.']
Simple English Artikel in Liste = [['This', 'is', 'a Wikipedia Article','.'], ['It', 'contains', 'multiple', 'sentences', '.'], ['It', 'is', 'about', 'the Heinrich Heine University', '.']]

3.2 Entfernen von Stoppwörtern

Stoppwörter sind Wörter welche keine Relevanz für den Inhalt und Kontext des Satzes aufweisen. Überwiegend werden Synsemantika entfernt, dies sind im Englischen Wörter wie "the", "is", "at" etc. Dies sind Wörter die nur eine grammatische Funktion im Text haben. Somit bleiben in den Listen nur Token welche von inhaltlicher Bedeutung sind. Dies erleichtert die Suche nach passenden Alignments da nicht mehr so viel "Rauschen" in unseren Daten vorhanden ist.

Ausgewählter Satz in Liste = ['selected', 'sentence', '.']
Simple English Artikel in Liste = [['a Wikipedia Article', '.'], ['contains', 'multiple', 'sentences', '.'], ['the Heinrich Heine University', '.']]

3.3 Lowercasing und Entfernen von Interpunktion

Damit gleiche Wörter erkennt werden können ist Lowercasing der erste Schritt um eine Uniformität für den Vergleich von Strings gewährleisten zu können. Da in der englischen Sprache der erste Buchstabe in einem neuen Satz groß geschrieben wird, dient es vor allem dazu damit ein Wort am Anfang von einem Satz und das gleiche Wort in der Mitte eines Satzes auch als gleich erkannt werden. In dem Programm werden somit bei der weiteren Verarbeitung alle Wörter nur im Lowercase betrachtet. Desweiteren müssen alle Satzzeichen entfernt werden, da diese nicht von Hilfe für die Algorithmen sind.

Ausgewählter Satz in Liste = ['selected', 'sentence']
Simple English Artikel in Liste = [['a wikipedia article'], ['contains', 'multiple', 'sentences'], ['the heinrich heine university']]

3.4 Lemmatisierung

Um die Uniformität weiter zu verbessern wird die sogenannte Lemmatisierung angewandt. Da alle Algorithmen der Problemstellung darauf basieren, dass Token untereinander gematched werden ist es wichtig, dass Token mit dem gleichen Inhalt welche jedoch flektiert worden sind als identisch erkannt werden. Die Flexion ändert Wörter ab um diese an ihre grammatikalische Funktion im Satz anzupassen. Dabei können im Englischen der Tempus, Kasus, Aspekt, Person, Numerus, Genus und Modus angepasst werden. Ein Lemma ist die Grundform von einem Wort. Ein Beispiel für die Lemmatisierung sieht wie folgt aus: ”am”, ”are”, ”is” werden zu ”be”, ”helping”, ”helps”, ”helped” wird zu ”help”. Das Wort wird auf die Form zurückgeführt die man auch in einem Wörterbuch finden würde. Auch dieser Prozess hilft uns das Rauschen einzudämmen und Alignments zu finden welche sonst vielleicht nicht gefunden worden wären.

Ausgewählter Satz in Liste = [’select’, ’sentence’]
Simple English Artikel in Liste = [[’a wikipedia article’], [’contain’, ’multiple’, ’sentence’], [’the heinrich heine university’]]

4 Algorithmen

In diesem Kapitel werden Algorithmen betrachtet welche verwendet wurden um Stringketten untereinander zu matchen. Jeder von diesen Algorithmen basiert darauf, dass der vom Benutzer ausgewählte Satz mit einem String der aus einem oder mehreren Sätzen bestehen kann verglichen wird. Der Simple English Artikel wird von oben bis nach unten durchlaufen und der höchste Score zusammen mit der Satzposition gespeichert. Für jeden Algorithmus gibt es einen Threshold ab welchem wir sagen, dass der Satz ein Match ist.

4.1 Jaccard-Index

Der Jaccard-Index ist ein Maß um die Ähnlichkeit zweier Mengen, Vektoren oder Objekte zu bestimmen. Für die Problemstellung werden zwei Mengen betrachtet. Der Index ist eine Zahl zwischen 0 und 1, wobei 0 bedeutet, dass die Mengen komplett unterschiedlich, und eine 1, dass diese völlig identisch sind. Da dieser Index sich über die Zeit bei der Duplikaterkennung bewährt hat ist es sinnvoll diesen auch für unser Problem zu betrachten.

Die Menge A besteht aus Token von dem bereits vorverarbeiteten, vom Benutzer ausgewählten Satz. Die Menge B verändert sich und besteht dabei aus Token aus Strings von variabler Länge aus dem Simple English Text.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Im Zähler wird zunächst der Schnitt der zwei Mengen bestimmt. Der Schnitt besteht dabei aus den Token die zwischen den beiden Mengen gleich sind. Daraufhin wird ihre Anzahl bestimmt. Im Nenner wird die Vereinigung der beiden Mengen genommen, ihre Anzahl wird ebenfalls bestimmt. Es ist also die Anzahl aller Token aus den beiden Mengen, wobei Duplikate nicht mitgezählt werden da eine Menge das gleiche Element nicht zwei Mal enthalten kann.

Wenn also:

$$A = \{'Ich','spiele','oft','Basketball'\} und B = \{'Ich','liebe','Basketball'\}, dann$$
$$\frac{|A \cap B|}{|A \cup B|} = \frac{|\{'Ich','Basketball'\}|}{|\{'Ich','spiele','oft','Basketball','liebe'\}|} = \frac{2}{5} = 0{,}4$$

4.2 Kosinus-Ähnlichkeit von Wortvektoren

4.2.1 Wortvektoren

Wörter können als Vektoren repräsentiert werden. Ein Wort-Vektor ist ein Vektor welcher aus Gewichten besteht. Ähnliche Wörter haben ähnliche Vektoren, d.h. sie zeigen in

ungefähr die gleiche Richtung. Dies wird dadurch erreicht, dass Wörter die häufig im selben Kontext auftauchen ähnliche Vektoren zugewiesen bekommen. Mit diesen Vektoren kann dann auch gerechnet werden. Angenommen wir haben die Vektoren:

$$K\ddot{O}NIGIN, K\ddot{O}NIG, MANN, FRAU, BERLIN, DEUTSCHLAND, TOKYO, JAPAN$$

dann ergibt

$$K\ddot{O}NIGIN = K\ddot{O}NIG - MANN + FRAU$$

und

$$TOKYO = ROM - ITALIEN + JAPAN$$

Die Vektoren die für diesen Algorithmus verwendet worden sind bestehen aus jeweils 300 Dimensionen. Probleme können dann entstehen falls es in dem vortrainierten Modell keine Vektorrepräsentation von einem Wort gibt, dann wird dem Wort ein Nullvektor zugewiesen, darunter leidet die Präzision des Algorithmus. Wenn wir also Artikel haben in denen viele Fachwörter vorkommen, dann kann es sein, dass die Matchings schlechter sind als bei Artikeln mit mehr gewöhnlichen Wörtern. [Pennington et al., 2014]

4.2.2 Kosinus-Ähnlichkeit

Die Kosinus-Ähnlichkeit ist ein Maß um die Ähnlichkeit zweier Vektoren zu bestimmen. Dabei wird berechnet ob beide Vektoren in ungefähr die gleiche Richtung zeigen.

$$Kosinus - \ddot{A}hnlichkeit = \cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2} = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n} (a_i)^2} \cdot \sqrt{\sum_{i=1}^{n} (b_i)^2}}$$

Jedem Wort wird ein bestimmter Vektor zugewiesen. Je größer die Kosinus-Ähnlichkeit der zwei zu vergleichenden Vektoren ist, desto ähnlicher sind sich die dazugehörigen Wörter. Wenn also z.B. die Wörter *Hund, Banane, Katze* vorliegen, dann würden die Vektoren von *Hund* und *Katze* eine größere Kosinus-Ähnlichkeit aufweisen als *Hund* und *Banane*.

Da der Input aus ganzen Sätzen besteht und nicht nur zwei Wörter mit einander verglichen werden, muss der Algorithmus etwas angepasst werden. Der Vektor für den jeweiligen Satz wird dabei aus dem Durchschnitt von allen Wort-Vektoren die dieser Satz enthält gebildet.

4.3 TF-IDF

TF-IDF ist ein Maß welches dazu eingesetzt wird um die Wichtigkeit/Relevanz eines Terms in einem Dokument zu bestimmen. Dabei werden Termen größere Gewichte zugeordnet, je wichtiger diese für das Dokument sind. TF-IDF ist der am häufigsten eingesetzte Algorithmus für die Bestimmung von Gewichten für Terme.

IDF steht für Inverse Document Frequency Damit Token wie *und, oder, der*, also Token die häufig vorkommen aber weniger Relevanz für den Inhalt haben kein höheres Gewicht zugeteilt bekommen wird die Inverse Dokumenthäufigkeit mit in Betracht gezogen.

$$\mathrm{idf}(t) = \log \frac{N}{\sum_{D:t \in D} 1}$$

N ist die Anzahl der Dokumente und $\sum_{D:t \in D} 1$ die Anzahl der Dokumente wo der Term t vorkommt.

Das Gewicht $\mathrm{tfidf}(t, D)$ eines Termes t in Dokument D wird dann berechnet in dem man die Term Frequency mit der Inverse Document Frequency multipliziert.

$$\mathrm{tfidf}(t, D) = \mathrm{tf}(t, D) \cdot \mathrm{idf}(t)$$

Bei dem klassischen TF-IDF wird als Dokument meistens eine ganze Datei/Text angesehen. Für unsere Implementation definieren wir einen Satz als ein Dokument. Somit zählen wir bei der TF wie oft der Token/Term in dem Satz vorkommt. Bei der IDF zählen wir in wievielen Sätzen von dem Artikel der Token/Term vorkommt. Mit diesen Änderungen passen wir den Algorithmus an unsere Problemstellung an.

Wenn wir die Scores für den Satz erhalten haben können wir anfangen die Sätze zu matchen. Dafür wird ein einfacher Matching Algorithmus benutzt: Es wird Satz für Satz abgeglichen wieviele Token in dem vom Benutzer ausgewählten Satz und den Sätzen aus dem Artikel matchen. Jeder Satz aus dem Artikel bekommt einen Score der sich aus den aufsummierten Scores der matchenden Tokens zusammensetzt.

Eine weitere Besonderkeit für diesen Algorithmus ist, dass wir Zahlen einen höheren Score geben in dem wir den TF-IDF Score mit 1.8 multiplizieren. Zahlen sind in WIkipedia Artikeln oft ein Indikator für den selben Sachverhalt da sie oft entweder einer Jahreszahl entsprechen oder Teil einer Statistik sind.

5 Verbesserungen der Algorithmen

5.1 Variable Anzahl von Sätzen

Da 1:N Beziehungen von Sätzen untersucht werden, ist es eine gute Idee zu schauen wie groß „N“für jeden Satz überhaupt ist. Das Programm bestimmt zunächst wieviele Kommata „,“und „and“der Ausgangssatz besitzt. Kommata und „and“werden oft dazu benutzt um zwei Hauptsätze von einander zu trennen und damit einen einzelnen, langen Satz zu machen. Vereinfachte Sätze bestehen oftmals nur aus einem Hauptsatz da ihre grammatische Struktur weniger komplex ist. Da Kommata und „and“im Englischen aber auch für andere grammatikalische Funktionen wie z.B. Einschübe verwendet werden und im Simple English Text Informationen einfach ausgelassen werden können, betrachten wir eine „1 bis zu N Beziehung“. Dies bedeutet, dass ein langer Satz mit Kommata nicht unbedingt immer in mehrere Sätze aufgeteilt wird. Dadurch, dass alle, also die 1:1, 1:2 ... 1:N, Möglichkeiten durchgegangen werden, ist es möglich den besten Match zu finden.

Angenommen es liegt folgender Satz vor: „Soccer is played by 250 million players in over 200 countries, making it the world’s most popular sport.“, aus dem englischen Wikipedia Artikel. Um das Beispiel etwas deutlicher zu machen besteht der Simple English Wikipedia Artikel nur aus drei Sätzen: „Football is the world’s most popular sport. It is played in more than 200 countries. The length of a match is 90 minutes. “Da der Satz aus dem englischen Wikipedia Artikel ein Komma enthält gilt $N = 2$. Der Algorithmus würde nun folgende Vergleiche durchführen:

English	Simple Englisch
Soccer is played by 250 million players in over 200 countries, making it the world’s most popular sport.	Football is the world’s most popular sport.
”	Football is the world’s most popular sport. It is played in more than 200 countries.
”	It is played in more than 200 countries.
”	It is played in more than 200 countries. The length of a match is 90 minutes.
”	The length of a match is 90 minutes.

somit werden alle Möglichkeiten abgedeckt und die mit dem höchsten Score als Endergebnis genommen.

6 Evaluierung

In diesem Kapitel werden Evaluationsmaße betrachtet mit welchen die Ergebnisse der Algorithmen bewertet werden. Daraufhin werden die Ergebnisse der einzelnen Algorithmen verglichen.

Es liegt ein Klassifizierungsproblem vor da die Anwendung der Algorithmen zu vier ausgängen führen kann:

	Algorithmus Match	Algorithmus kein Match
Sätze matchen	True Positive	False Negative
Sätze matchen nicht	False Positive	True Negative

Wenn der Algorithmus tatsächlich matchende Sätze auch als solche erkennt, dann ist es ein True Positive = t_p. Wenn der Algorithmus diese als 'kein Match' klassifiziert, dann ist es ein False Negative = f_n. Umgekehrt gilt, dass wenn die Sätze nicht matchen, der Algorithmus diese aber als ein Match erkennt, es ein False Positive = = f_p ist. Wenn die Sätze nicht matchen und der Algorithmus diese als 'kein Match' klassifiziert dann liegt ein True Negative = t_n vor.

6.1 Precision

$$precision = p = \frac{|t_p|}{|t_p + f_p|}$$

$precision$ ist die Anzahl der True Positives geteilt durch die Anzahl aller vom Algorithmen gefundenen Positives. In unserem Fall ist es die vom Algorithmus gefundene Anzahl der tatsächlich matchenden Sätze geteilt durch die Anzahl von allen Sätzen die vom Algorithmus als matchend markiert worden sind. Je höher der precision Wert ist, desto höher ist die Wahrscheinlichkeit, dass ein als Match markierter Satz auch wirklich der positiven Klasse angehört.

6.2 Recall

$$recall = r = \frac{|t_p|}{|t_p + f_n|}$$

$recall$ ist die Anzahl der True Positives geteilt durch die Anzahl der True Positives summiert mit der Anzahl der False Negatives. In unserem Fall ist es die vom Algorithmus gefundene Anzahl der tatsächlich matchenden Sätze geteilt durch die Anzahl aller tatsächlich matchenden Sätze. Je höher der recall Wert ist, desto höher ist die Wahrscheinlichkeit, dass ein tatsächlich matchender Satz auch gefunden wird.

6.3 F1 Score

$$F_1 = \left(\frac{\text{recall}^{-1} + \text{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Der F_1 Score ist das harmonische Mittel aus $precision$ und $recall$. Dadurch können sowohl False Positives als auch False Negatives berücksichtigt werden. Somit ist es für unsere Ziele der aussagekräftigste Score.

6.4 Benutzer-Rating

Die Benutzer konnten die vom Algorithmus ausgewählten Satzpaare bewerten. Das Bewertungsverfahren wurde an Hwang et al., 2015 angelehnt.

Gut	Keine Information ist verloren gegangen
OK	Wenige Informationen sind verloren gegangen
Geringe Ähnlichkeit	Fast alle Informationen sind verloren gegangen
Keine Ähnlichkeit	Alle Informationen sind verloren gegangen

6.5 Bestimmung von Score-Thresholds

7 Fazit

Con97 hat ein Buch geschrieben. Es gibt auch andere Arbeiten (**PeHe97**) die referenziert sind. In Abbildung 1 ist ein Sachverhalt dargestellt.

1 Autor: **Con97** (**Con97**)
2 Autoren: **IWNLP** (**IWNLP**)
3 Autoren: **liebeck-esau-conrad:2016:ArgMining2016** (**liebeck-esau-conrad:2016:ArgMining2016**)

Online resource: **ILSVRC2016**

quotes:
„quote“.

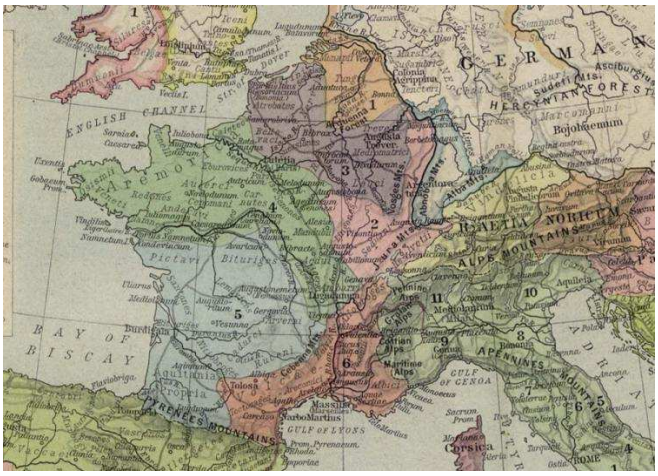

Abbildung 1: Gallien zur Zeit Caesars

Jahr	Erster Consul	Zweiter Consul
1	C. Caesar	L. Aemilius Paullus
2	P. Vinicius	P. Alfenus Varus
3	L. Aelius Lamia	M. Servilius
4	Sex. Aelius Catus	C. Sentius Saturninus
5	L. Valerius Messalla	Cn. Cornelius Cinna
suff.	C. Vibius Postumus	C. Ateius Capito
6	M. Aemilius Lepidus	L. Arruntius

Tabelle 1: Römische Konsulen

References

William Hwang, Hannaneh Hajishirzi, Mari Ostendorf und Wei Wu (Mai 2015). „Aligning Sentences from Standard Wikipedia to Simple Wikipedia“. In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, S. 211–217.

Jeffrey Pennington, Richard Socher und Christopher Manning (2014). „Glove: Global vectors for word representation“. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, S. 1532–1543.

Abbildungsverzeichnis

Tabellenverzeichnis