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 Methodik

lorem

3 Vorverarbeitung vom Text

Als Ausgangslage liegen uns 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 mit Hilfe von Spacy 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 haben wir den vom User ausgewählen Satz, aufgeteilt in einzelne Wörter in einer Liste und eine zweite Liste welche Listen von den einzelnen Sätzen welche einzelne Wörter enthält bei dem Simple English Artikel.

Ausgewählter Satz in Liste = ['Dies', 'ist', 'der', 'ausgewählte', 'Satz', '.']
Simple English Artikel in Liste = [['Dies', 'ist', 'ein', 'Wikipedia', 'Artikel','.'], ['Er', 'enthält', 'mehrere', 'Sätze', '.']]

Nun können wir mit der Vorverarbeitung von den Texten beginnen. Alle folgenden Prozesse wurden entweder mit der Python Standard Library oder mit spaCy durchgeführt.

3.1 Lowercasing und Entfernen von Interpunktion

Damit wir gleiche Wörter erkennen können ist Lowercasing der erste Schritt um eine Uniformität für den Vergleich von Strings gewährleisten zu können. Es dient 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

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 erhalten wir eine Liste an Wörtern die alle eine für den jeweiligen Satz inhaltliche Bedeutung aufweisen. Dies erleichtert uns die Suche nach passenden Alignments da nicht mehr so viel ”Rauschen” in unseren Daten vorhanden ist.

3.3 Lemmatisierung

Um die Uniformität weiter zu verbessern wenden wir die sogenannte Lemmatisierung an. 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.

4 Algorithmen

4.1 Jaccard-Index

Der Jaccard-Index ist eine Kennzahl um die Ähnlichkeit zweier Mengen zu messen. Dabei kommt man bei der Berechnung auf eine Zahl zwischen 0 und 1, wobei eine 0 bedeutet, dass die Mengen komplett unterschiedlich, und eine 1, dass diese völlig identisch sind.

In userem Fall besteht die Menge A aus Wörtern von dem bereits vorverarbeiteten, vom Benutzer ausgewählten Satz. Die Menge B verändert sich und besteht dabei aus Textabschnitten von variabler Länge aus dem Simple English Text.

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

4.2 Kosinus-Ähnlichkeit

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

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

Jedem Wort wird ein bestimmter Vektor zugewiesen, in unserem Fall benutzen wir ein vortrainiertes Model von spaCy. Je größer die Kosinus-Ähnlichkeit ist, desto ähnlicher sind sich die beiden Wörter. Wenn wir also z.B. die Wörter *Hund, Banane, Katze* haben, dann würden die Vektoren von *Hund* und *Katze* eine größere Kosinus-Ähnlichkeit aufweisen als *Hund* und *Banane*.

Da wir ganze Sätze als Input haben und nicht nur zwei Wörter mit einander vergleichen, muss der Algorithmus etwas angepasst werden. Der Vektor für den jeweiligen Satz(Satz, String, wie ab besten beschreiben???) wird dabei aus dem Durchschnitt von allen Vektoren die dieser Satz enthält gebildet.

4.3 Local 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 $\mathrm{tf.idf}(t, D)$ eines Termes t in Dokument D wird dann berechnet in dem man

die Term Frequency mit der Inverse Document Frequency multipliziert. $\mathrm{tf.idf}(t, D) = \mathrm{tf}(t, D) \cdot \mathrm{idf}(t)$ berechnet.

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 wir 1:N Beziehungen von Sätzen untersuchen ist es natürlich 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“.

Angenommen wir haben den Satz: „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 stellen wir uns vor, dass der Simple English Wikipedia Artikel nur aus diesen drei Sätzen besteht: „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.

5.2 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 wir also den Satz „David bought shares of Apple for $300 million.“haben, dann würde dieser wie folgt annotiert werden: „[David]$_{Person}$ bought shares of [Apple]$_{Corporation}$ for [$300 million]$_{Money}$.“

Durch die Anwendung von NER in unseren Algorithmen ist es möglich diese weiter zu

verfeinern. Angenommen wir hätten die Sätze „Burger King tastes good.“ und „This Burger tastes good.“. Wenn wir jetzt eine einfache Tokenization und eine Entfernung von Satzzeichen durchführen, dann erhalten wir für unseren Jaccard-Index Algorithmus folgende Listen:
Satz 1 = [’Burger’, ’King’, ’tastes’, ’good’]
Satz 2 = [’This’, ’Burger’, ’tastes’, ’good’]

Hierbei würden die Wörter „Burger“, „tastes“ und „ good“ matchen. Wenn wir jedoch NER anwenden, 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.

6 Evaluierung

6.1 User-Rating

6.2 Pearson correlation coefficient

6.3 Die Algorithmen im Vergleich

6.4 Besonderheiten von Wikipedia

7 Vergleich von word2vec und sense2vec

8 Aussicht

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“.

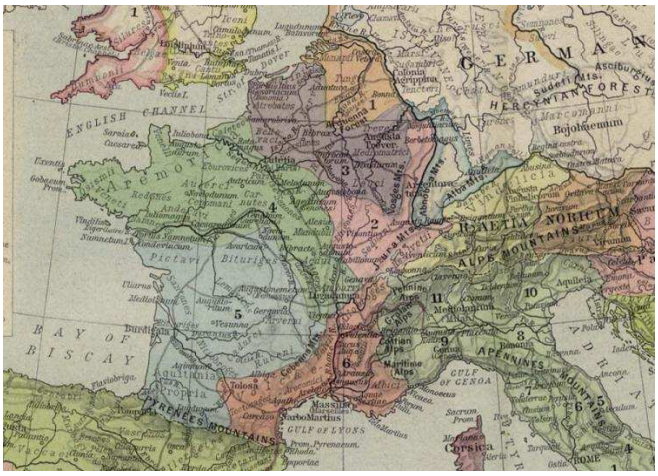

Abbildung 1: Gallien zur Zeit Caesars

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

Abbildungsverzeichnis

Tabellenverzeichnis