INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Webtool zur Betrachtung und Annotation von Satzmatchings 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. Michael Schöttner

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

Diese Bachelorarbeit befasst sich mit der Erstellung eines Webtools zur Betrachtung und Annotation von Satzmatchings aus Wikipedia in vereinfachtem Englisch.

Es wurden drei Algorithmen untersucht welche dabei helfen die Matchings von komplexen englischen Sätzen zu den einfachen englischen Sätzen zu finden. Die drei Algorithmen sind der Jaccard-Index Algorithmus, der Wortvektor Cosinus-Ähnlichkeit Algorithmus und der Term Frequency-Vektor Cosinus-Ähnlichekit Algorithmus.

Als erstes werden verschiedene Techniken beschrieben um den Input bereit für unsere Algorithmen zu machen damit diese präzisere Ergebnisse liefern können. Dabei wir anhand eines Beispiels die komplette NLP Pipeline durchlaufen, erklärt und anschaulich dargestellt. Daraufhin werden die verschiedenen Algorithmen vorgestellt und eine Verbesserung dieser durch eine variable Anzahl von Sätzen erläutert. Nach der Erklärung der Algorithmen und ihrer Verbesserungen wird auf die Implementierung des Annotationstools eingegangen. In diesem Kapitel wird ebenfalls die Bedienung des Tools erklärt. Im Evaluationskapitel werden die Evaluationsmaße und ihre Ergebnisse vorgestellt. Ebenfalls werden die Datensätze anhand denen die Evaluation stattgefunden hat vorgestellt. Desweiteren werden die Thresholds für jeden Algorithmus bestimmt. Dabei ist der Threshold der Wert ab welchem wir annehmen können, dass die zwei Sätze ein Match sind.

Um die Evaluationsmaße für jeden Algorithmus bestimmen zu können wurden 900 Satzpaare (300 pro Algorithmus) mit Hilfe des Tools annotiert und in einer Datenbank gespeichert. Es konnte gezeigt werden, dass der Jaccard-Index am besten für die Problemstellung geeignet ist.

Inhaltsverzeichnis

1 Einleitung

Simple English Wikipedia ist eine im Jahr 2001 gegründete Version von Wikipedia in welcher die Artikel in vereinfachtem Englisch geschrieben worden sind. Laut Wikipedia sollen ein vereinfachter Wortschatz und kürzere Sätze verwendet werden. Der Grundgedanke ist es eine Enzyklopädie für Kinder und Erwachsene welche Englisch lernen wollen zu haben. Im August 2019 sind auf der Simple English Wikipedia Webseite über 148000 Artikel zu finden (*Simple Wikipedia* 2019).

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 wichtige Informationen erhalten bleiben.

Diese Arbeit befasst sich damit komplexe Sätze aus dem English Wikipedia ihrem vereinfachten Gegenstück aus dem Simple English Wikipedia zuzuordnen.

Das Ziel ist es ein Annotationstool zu erstellen mit welchem ein Benutzer einen Satz in einem English Wikipedia Artikel markieren kann, einen oder mehrere ähnliche Sätze in dem dazugehörigen Simple English Wikipedia vorgeschlagen bekommt, den Vorschlag nach belieben modifizieren kann und anschließend bewerten kann wie ähnlich sich die beiden Sätze sind. So kann eine Datenbank mit komplexen und ihren dazugehörigen vereinfachten Sätzen aufgebaut werden, welche später z.B. für Machine Learning benutzt werden kann.

Dabei liegt der Fokus der Arbeit darauf, die Auswahl der Sätze, welche das Programm dem Benutzer vorschlägt, möglichst präzise zu machen. Für diesen Zweck werden mehrere Algorithmen implementiert und miteinander verglichen.

Es ist nicht immer leicht einen guten Vorschlag zu finden da die Problemstellung gleich mehrere Probleme mit sich bringt. Zum Ersten sind die Wikipedia Artikel unabhängig von einander, von verschiedenen Personen, geschrieben worden. Dies bedeutet, dass viele komplexe Sätze über kein vereinfachtes Gegenstück verfügen. Zum Zweiten sind die Artikel oft verschieden strukturiert. Da oft auch die Satzstruktur vereinfacht wird kann es passieren, dass ein komplexer Satz in mehrere, einfachere, kürzere Sätze aufgeteilt wird.

2 Natural Language Processing

Natural Language Processing (NLP) ist ein Teilfeld der Informatik welches sich damit befasst wie natürliche Sprachen von Computern verarbeitet werden können (Kannan und Gurusamy, 2014). 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 Rauschen zu entfernen. Als Rauschen bezeichnen wir Token welche nicht dabei helfen oder es sogar schwerer machen Matches zu finden.

Es wurde folgende Pipeline (Abfolge der Operationen) verwendet:

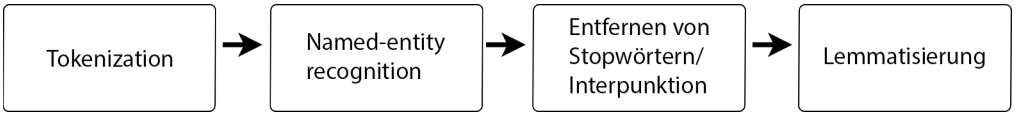

Abbildung 1: NLP Pipeline

Die Pipeline wird anhand dieses Beispiels durchlaufen:

Ausgewählter Satz in Liste =

$$['This' \quad 'is' \quad 'the' \quad 'selected' \quad 'sentence' \quad '.']$$

Simple English Artikel in Liste =

$$\begin{bmatrix} 'This' & 'is' & 'a' & 'Wikipedia' & 'Article' & '.' & & \\ 'It' & 'contains' & 'multiple' & 'sentences' & '.' & & & \\ 'It' & 'is' & 'about' & 'the' & 'Heinrich' & 'Heine' & 'University' & '.' \end{bmatrix}$$

Am Ende jedes Abschnittes werden nach der Erklärung der Technik diese auf die Sätze angewandt.

2.1 Named-entity Recognition

Bei der Named-entity Recognition (NER) geht es darum Eigennamen zu finden und diese dann zu klassifizieren (Jiang, 2012). 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 $300 million.“betrachtet wird, dann würde dieser wie folgt annotiert werden: „[David]$_{\text{Person}}$ bought shares of [Apple]$_{\text{Corporation}}$ for [$300 million]$_{\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:

$$Satz1 = \begin{bmatrix} 'Burger' & 'King' & 'tastes' & 'good' \end{bmatrix}$$

$$Satz2 = \begin{bmatrix} 'This' & 'Burger' & 'tastes' & 'good' \end{bmatrix}$$

Hierbei würden die Wörter „Burger“, „tastes“ und „ good“ zwischen den beiden Sätzen matchen. Wenn jedoch NER angewendet wird, dann werden folgende Listen erstellt:

$$Satz1 = \begin{bmatrix} 'Burger\ King' & 'tastes' & 'good' \end{bmatrix}$$

$$Satz2 = \begin{bmatrix} 'This' & 'Burger' & 'tastes' & 'good' \end{bmatrix}$$

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 =

$$\begin{bmatrix} 'This' & 'is' & 'the' & 'selected' & 'sentence' & '.' \end{bmatrix}$$

Simple English Artikel in Liste =

$$\begin{bmatrix} 'This' & 'is' & 'a\ Wikipedia\ Article' & '.' & \\ 'It' & 'contains' & 'multiple' & 'sentences' & '.' \\ 'It' & 'is' & 'about' & 'the\ Heinrich\ Heine\ University' & '.' \end{bmatrix}$$

2.2 Entfernen von Stoppwörtern

Stoppwörter sind Wörter welche keine Relevanz für den Inhalt und Kontext des Satzes aufweisen (Gaigole et al., 2013). Überwiegend werden Synsemantika entfernt, dies sind im Englischen Wörter wie ”the”, ”is”, ”at” etc. Synsemantika 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 =

$$\begin{bmatrix} 'selected' & 'sentence' & '.' \end{bmatrix}$$

Simple English Artikel in Liste =

$$\begin{bmatrix} 'a\ Wikipedia\ Article' & '.' & \\ 'contains' & 'multiple' & 'sentences' & '.' \\ 'the\ Heinrich\ Heine\ University' & '.' & \end{bmatrix}$$

2.3 Lowercasing und Entfernen von Interpunktion

Damit gleiche Wörter erkannt 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 =

$$\begin{bmatrix} 'selected' & 'sentence' \end{bmatrix}$$

Simple English Artikel in Liste =

$$\begin{bmatrix} 'a\ wikipedia\ article' & & \\ 'contains' & 'multiple' & 'sentences' \\ 'the\ heinrich\ heine\ university' & & \end{bmatrix}$$

2.4 Lemmatisierung

Um die Uniformität weiter zu verbessern wird die sogenannte Lemmatisierung angewandt. Da alle Algorithmen für diese 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 das Rauschen einzudämmen und Matchings zu finden welche sonst vielleicht nicht gefunden worden wären.

Das sogenannte Stemming hat eine ähnliche Funktion. Während bei der Lemmatisierung die Form von einem Wort aus einem Wörterbuch stammt, werden beim Stemming die Wörter mit Hilfe von Algorithmen welche Teile von dem Wort (Affixe) abschneiden auf ihre Stammform gebracht (Jivani et al., 2011). Dies passiert zum Beispiel in dem bestimmte Regeln vordefiniert werden, welche durchlaufen werden bis das Wort diesen Regeln entspricht. Ein weiterer Ansatz ist es, dass eine Liste mit Suffixen vorliegt. Mit Hilfe dieser Liste wird dann aus dem Wort der längste Suffix entfernt. Dabei können durch Stemming Grundformen von Wörtern entstehen welche keine Bedeutung haben und nicht im Wörterbuch vorkommen. Die meisten Fehler kann man dabei in zwei Kategorien unterscheiden: Over-Stemming und Under-Stemming. Beim Over-Stemming kommt es dazu, dass zwei verschiedene Wörter, welche eigentlich nicht den gleichen Stamm haben, auf die gleiche Stammform gebracht werden. Beim Under-Stemming werden zwei Wörter, welche den gleichen Stamm haben nicht auf die gleiche Stammform gebracht..

Im Gegensatz zum Stemming wird bei der Lemmatisierung auch die Funktion des Wortes im Satz analysiert, so wird auch festgestellt ob das Wort ein Nomen oder ein Verb ist, somit ist eine präzisere Zurückführung auf den Stamm möglich (Balakrishnan und Ethel, 2014). Desweiteren werden bei der Lemmatisierung auch Synonyme von Wörtern betrachtet. Dies ist für die Problemstellung besonders wichtig, da die Wikipedia Artikel unabhängig von einander und von verschiedenen Personen geschrieben worden sind. Somit ist die Wahrscheinlichkeit hoch, dass Synonyme benutzt worden sind um gleiche Sachverhalte zu beschreiben.

Ausgewählter Satz in Liste =

$$\begin{bmatrix} 'select' & 'sentence' \end{bmatrix}$$

Simple English Artikel in Liste =

$$\begin{bmatrix} 'a\ wikipedia\ article' & & \\ 'contain' & 'multiple' & 'sentence' \\ 'the\ heinrich\ heine\ university' & & \end{bmatrix}$$

3 Algorithmen zur Erkennung von Satzmatchings

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 der Satz als ein Match klassifiziert wird.

3.1 Jaccard-Index

Der Jaccard-Index ist ein Maß um die Ähnlichkeit zweier Mengen, Vektoren oder Objekte zu bestimmen (Eckey et al., 2002). 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. Somit ist die Kardinalität des Schnittes im Zähler und die Kardinalität der Vereinigung im Nenner.

Sei also:

$$A = \{'Ich', 'spiele', 'oft', 'Basketball'\}$$

und sei

$$B = \{'Ich', 'liebe', 'Basketball'\}$$

dann ist,

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

3.2 Kosinus-Ähnlichkeit von Wortvektoren

3.2.1 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:

$$KÖNIGIN, KÖNIG, MANN, FRAU, BERLIN, DEUTSCHLAND, TOKIO, JAPAN$$

dann ergibt

$$KÖNIGIN = KÖNIG - MANN + FRAU$$

und

$$TOKIO = ROM - ITALIEN + JAPAN$$

Die Vektoren die für diesen Algorithmus verwendet worden sind GloVe-Vektoren (Pennington et al., 2014) und bestehen aus jeweils 300 Dimensionen. Probleme können dann entstehen wenn 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.

3.2.2 Kosinus-Ähnlichkeit

Die Kosinus-Ähnlichkeit ist ein Maß um die Ähnlichkeit zweier Vektoren zu bestimmen (Haenelt, 2006). Dabei wird berechnet ob beide Vektoren in ungefähr die gleiche Richtung zeigen. Die Werte der Kosinus-Ähnlichkeit liegen dabei zwischen -1 und 1. Dabei bedeutet -1, dass die Vektoren in genau entgegengerichtete Richtungen zeigen, 1 bedeutet, dass die Vektoren genau gleichgerichtet sind und eine 0 bedeutet, dass sie orthogonal zu einander sind. Somit bedeuten die Werte zwischen -1 und 0 den Grad der Unähnlichkeit und Werte zwischen 0 und 1 den Grad der Ähnlichkeit.

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

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.

3.3 TF-IDF

TF-IDF ist ein Maß welches dazu eingesetzt wird um die Wichtigkeit/Relevanz eines Terms in einem Dokument zu bestimmen (Rajaraman und Ullman, 2011).

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 (Beel et al., 2016).

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.

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

Dabei ist N die Anzahl der Dokumente und $\sum_{D:t \in D} 1$ die Anzahl der Dokumente in denen 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.

Wie sich im Laufe der Arbeit herausgestellt hat ist dieses Verfahren für die Problemstellung nur bedingt geeignet. Auf Grund von dem Nichtvorhandensein von geeigneten Datensätzen um diesen Algorithmus zu testen (mehr dazu im Abschnitt [Bestimmung von Score-Thresholds > TF-IDF]) wurde ein vierter Algorithmus implementiert welcher TF und Kosinus-Ähnlichkeit vereint.

3.4 Kosinus-Ähnlichkeit von TF-Vektoren

3.4.1 Bag-of-words

Um die Implementierung von dem Algorithmus zu verstehen muss zuerst auf Bag-of-words Modell eingegangen werden. Bei dem Bag-of-words Modell befinden sich Wörter aus einem Textabschnitt in einem „Bag“ (dt. Beutel). Dabei ist der „Bag“ eine Multimenge. Die Besonderheit einer Multimenge gegenüber einer Menge ist die, dass Elemente

mehrmals darin vorkommen können. Somit ist es möglich alle Wörter in einem Satz und wie oft diese darin vorkommen abzuspeichern. Damit lässt sich die TF (auf welche bereits eingegangen worden ist) bestimmen. Dies ist auch der erste Schritt im Algorithmus, eine Multimenge könnte wie folgt aussehen:

$$Satz1 = \text{Der Ball ist rund, der Ball ist schön}$$

$$\mathrm{M}(Satz1) = \{„der“ : 2, „Ball“ : 2, „ist“ : 2, „rund“ : 1, „schön“ : 1\}$$

3.4.2 Umwandlung zu einem Vektor

Das Bag-of-words Modell kann in einen Vektor umgewandelt werden. Dies ist wichtig, da später die Vektoren von zwei Sätzen verglichen werden sollen.

Angenommen es liegen folgende Sätze vor:

$$Satz1 = \text{Der Ball ist rund, der Ball ist schön}$$

$$Satz2 = \text{Die Erde ist rund, das Fahrrad ist schön}$$

Dann würden diese Vektoren erstellt werden:

	der	ball	ist	rund	schön	die	erde	das	fahrrad
Satz1	2	2	2	1	1	0	0	0	0
Satz2	0	0	2	1	1	1	1	1	1

3.4.3 L2 Norm

Da die TF längere Sätze favorisiert, muss diesem entgegengesteuert werden. Ein möglicher Weg ist es die TF Werte zu normalisieren.

Es liegt der Vektor $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ vor, dann berechnet man die L2 Norm wie folgt:

$$\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}$$

Zunächst wird jeder Vektoreintrag quadriert, dann werden die Ergebnisse daraus aufsummiert und anschließend die quadratische Wurzel gezogen. Als Ergebniss erhält man eine Zahl welche die Länge des Vektors repräsentiert.

Damit die TF Werte jetzt normalisiert werden, muss jeder Wert durch die L2 Norm des Vektors dividiert werden.

$$\|\boldsymbol{Satz1}\|_2 := \sqrt{2^2 + 2^2 + 2^2 + 1^2 + 1^2 + 0^2 + 0^2 + 0^2 + 0^2} \approx 3,74$$

$$\|\boldsymbol{Satz2}\|_2 := \sqrt{0^2 + 0^2 + 2^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2} \approx 2,82$$

Wenn jetzt alle TF Werte durch die L2 Norm ihres Vektors dividiert werden, dann liegen die Vektoren wie folgt vor:

	der	ball	ist	rund	schön	die	erde	das	fahrrad
Satz1	0,53	0,53	0,53	0,27	0,27	0	0	0	0
Satz2	0	0	0,71	0,35	0,35	0,35	0,35	0,35	0,35

3.4.4 Kosinus-Ähnlichkeit durch das Skalarprodukt

Da die Vektoren normalisiert sind ist es möglich ihre Kosinus-Ähnlichkeit mit Hilfe des Skalarprodukts zu berechnen.

$$\vec{a} \cdot \vec{b} = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = a_1 b_1 + \cdots + a_n b_n.$$

Für das Beispiel würde es bedeuten:

$$\vec{Satz1} \cdot \vec{Satz2} = \begin{pmatrix} 0,53 \\ 0,53 \\ 0,53 \\ 0,27 \\ 0,27 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0,71 \\ 0,35 \\ 0,35 \\ 0,35 \\ 0,35 \\ 0,35 \\ 0,35 \end{pmatrix} =$$

$0,53 \cdot 0 + 0,53 \cdot 0 + 0,53 \cdot 0,71 + 0,27 \cdot 0,35 + 0,27 \cdot 0,35 + 0 \cdot 0,35 + 0 \cdot 0,35 + 0 \cdot 0,35 + 0 \cdot 0,35 = 0,5653$

4 Verbesserungen der Algorithmen

4.1 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: „Soccer is played by 250 million players in over 200 countries, making it the world’s most popular sport.“, aus dem English Wikipedia Artikel vor. 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 English Wikipedia Artikel ein Komma enthält gilt $N = 2$. Der Algorithmus würde nun folgende Vergleiche durchführen:

English	Simple English
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 Implementierung

Als Programmiersprache wurde *Python* benutzt da dies der heutige Standard ist und die meist benutzten NLP Libraries ebenfalls Python verwenden. 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 (*NLTK vs. spaCy: Natural Language Processing in Python* 2019). Da das Programm webbasiert sein sollte wurde *Flask* als das Web Application Framework verwendet. Die Datenbank in der die Annotationen gespeichert werden benutzt das Datenbankverwaltungssystem *MySQL*. Es wurde *Docker* benutzt damit es möglichst einfach ist die Anwendung zu starten und der Benutzer sich nicht um Pakete kümmern muss. Es wurde ein *Docker Container* für die *MySQL* Datenbank erstellt, einer für *Flask* und ein weiterer für einen *NGINX* Server.

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. Bei der Tokenization geht es darum Text in logische Elemente zu zerlegen. So werden die Sätze durch Tokenization in einzelne Wörter zerlegt und ebenfalls in einer Liste gespeichert. Somit liegen 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.

5.1 Der Annotationsprozess

In diesem Kapitel wird das Annotationstool vorgestellt und dessen Bedienung erklärt.

Abbildung 2: Suchleiste für Wikipedia Artikel

Beim Öffnen des Programms wird dem Benutzer eine Suchleiste angezeigt. Hier kann

man den Begriff eingeben nach dem man suchen möchte.

Abbildung 3: Auswahlliste

Nachdem der Benutzer entweder auf die Lupe geklickt oder mit der Eingabetaste seine Eingabe bestätigt hat, wird dem Benutzer eine Liste an Wikipedia Artikeln präsentiert. In der Liste befinden sich alle zum Suchbegriff relevante Artikel. Es werden nur Artikel angezeigt welche in Simple English und English vorliegen.

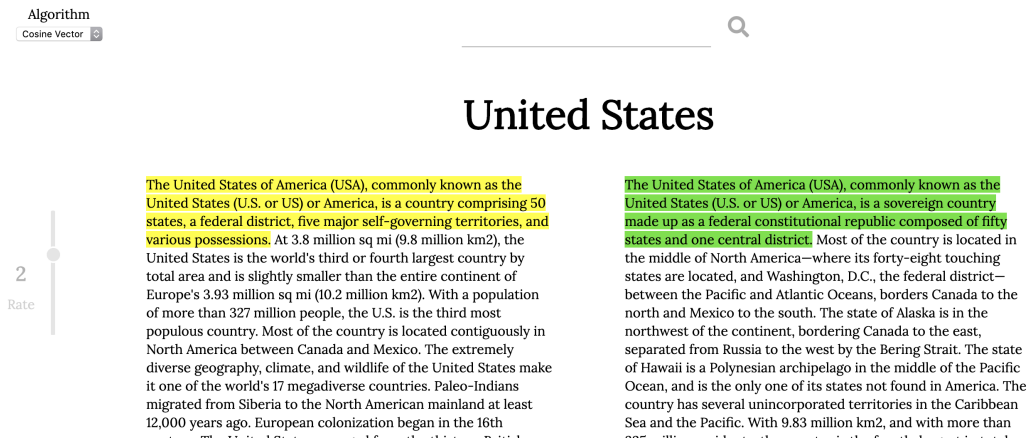

Abbildung 4: Annotationsinterface

Wenn der Benutzer einen Artikel aussucht, dann wird dieser einmal in English (linke Seite) und einmal in Simple English (rechte Seite) geladen. Jetzt kann der Benutzer sich links oben im Dropdown Menü einen Algorithmus aussuchen. Er hat die Auswahl zwischen 'Wordvector Cosine Similarity', 'TF-IDF' und 'Jaccard-Index'. Jeder Satz aus dem English Artikel kann angeklickt werden. Das Programm sucht dann mit Hilfe des ausgewählten Algorithmus nach den dazugehörigen Sätzen im Simple English Artikel. Der angeklickte Satz wird mit Gelb markiert. Das Programm gibt immer einen Match aus. Wenn der Match über dem Score-Threshold für den jeweiligen Algorithmus liegt dann wird der gematchte String grün hinterlegt, wenn der String unter dem Threshold liegt dann wird dieser orange markiert. Jetzt kann der Benutzer optional ein Rating vergeben wie gut der Text zu dem ausgewähltem Satz passt, es kann aber auch ein anderer Satz angeklickt werden ohne ein Rating zu vergeben. Wenn ein Rating vergeben wurde, dann springt die Markierung auf den nächsten Satz im Text und es wird automatisch der Match dazu gesucht. Wenn der Benutzer es wünscht, hat er auch die Möglichkeit Sätze im Simple English Artikel anzuklicken, diese werden daraufhin blau markiert und werden auch abgespeichert wenn ein Rating vergeben wird. Ein Klick auf einen bereits markierten Satz deselektiert diesen. Wenn die Auswahl die der Algorithmus getroffen hat vom Benutzer verändert worden ist, dann wird in der Datenbank für diese Kombination von Strings in dem Score für diesen Algorithmus eine -1 abgespeichert um darauf hinzuweisen, dass der Algorithmus diese auswahl nicht getroffen hat.

Association football

Association football, more commonly known as football or soccer, is a team sport played with a spherical ball between two teams of eleven players. It is played by 250 million players in over 200 countries and dependencies, making it the world's most popular sport. The game is played on a rectangular field called a pitch with a goal at each end. The object of the game is to score by moving the ball beyond the goal line into the opposing goal. Association football is one of a family of football codes, which emerged from various ball games played worldwide since antiquity. The modern game traces its origins to 1863 when the Laws of the Game were originally codified in England by The Football Association. Players are not allowed to touch the ball with hands or arms while it is in play, except for the goalkeepers

For the American sport, see American football. For other sports known as football, see Football Association football is a sport, played between two teams. There were various attempts to codify the rules of football in England in the mid-19th century. The present laws date back to 1863 where a ruleset was adopted in Rugby, Warwickshire by the newly formed Football Association. In its country of origin, United Kingdom, it is called football. In other countries, such as the United States and Canada, it is called soccer. In Australia, New Zealand, Ireland, South Africa, and Japan, both words are often used. Each team has 11 players on the field. One of these players is the goalkeeper, and the other ten are known as "outfield players". The game is played by kicking a ball into the opponent's goal. A match has 90 minutes of play, with a

Abbildung 5: Benutzerauswahl

Die Datenbank in der die Ratings von den Benutzern gespeichert werden sieht wie folgt aus:

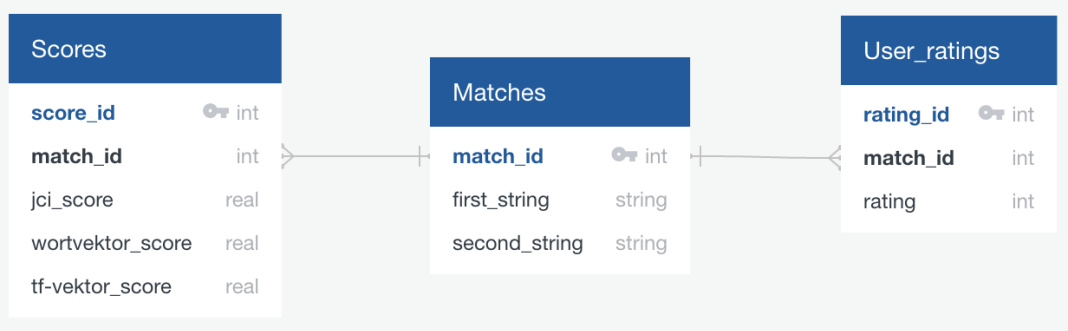

Abbildung 6: Datenbankschema

In der Tabelle „Matches“ werden der vom Benutzer ausgewählte Satz aus dem English Wikipedia Artikel und der dazugehörige String aus dem Simple English Wikipedia Artikel gespeichert. Diese Kombination von Strings bekommt eine ID. In der Tabelle „Scores“ werden die von den Algorithmen vergebene Scores von 0 bis 1 abgespeichert. In der Tabelle „User_ratings“ werden die Bewertungen der Benutzer gespeichert (mehr zum Bewertungssystem im Kapitel [Evaluation]). Es können mehrere Ratings pro Satzpaar gespeichert werden. Dies stellt sicher, dass mehrere Personen den gleichen Artikel annotieren können, da es oftmals subjektiv ist ob zwei Sätze matchen oder nicht. Für die Evaluation wurde der Durchschnitt aus den Bewertungen der Benutzer verwendet, falls ein Satzpaar verschiedene Bewertungen erhalten hat.

6 Evaluation

In diesem Kapitel werden Evaluationsmaße und ihre Ergebnisse im Bezug auf die benutzten Algorithmen betrachtet.

Die Anwendung der Algorithmen kann zu vier Ausgängen führen:

	Algorithmus - Match	*Algorithmus - kein Match*
Tatsächlich Match	True Positive	False Negative
Tatsächlich kein Match	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

$Precision$ ist die Anzahl der True Positives geteilt durch die Anzahl aller vom Algorithmen gefundenen Positives (Powers, 2011). 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.

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

6.2 Recall

$Recall$ ist die Anzahl der True Positives dividiert durch die Anzahl der True Positives summiert mit der Anzahl der False Negatives (Powers, 2011). 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.

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

6.3 Accuracy

$Accuracy$ ist die Anzahl der True Positives summiert mit der Anzahl der True Negatives geteilt durch die Anzahl aller Objekte (Powers, 2011). Für die Problemstellung bedeutet

es, dass die Anzahl der korrekt klassifizierten Sätze durch die Anzahl aller Sätze geteilt wird.

$$accuracy = \frac{|t_p + t_n|}{|t_p + t_n + f_p + f_n|}$$

Accuracy ist oftmals keine gute Metrik. Das sogenannte „accuracy paradox“ tritt dann auf wenn eine Klasse überrepräsentiert ist. Hier ein Beispiel:

	Algorithmus Positiv	*Algorithmus Negativ*
Tatsächlich Positiv	10 True Positives	20 False Negatives
Tatsächlich Negativ	30 False Positives	100 True Negatives

Die $accuracy$ wäre hier:

$$a = \frac{10 + 100}{10 + 100 + 30 + 20} = \frac{110}{160} = 0,6875 = 68,75\%$$

dies wäre ein guter Wert, da der Algorithmus ja anscheinend in 68,75% der Fälle die richtige Entscheidung trifft.

Wenn die gleichen Daten genommen werden, der Algorithmus aber jedes Objekt einfach als „Negativ“ klassifiziert, dann liegt folgendes vor:

	Algorithmus Positiv	*Algorithmus Negativ*
Tatsächlich Positiv	0 True Positives	30 False Negatives
Tatsächlich Negativ	0 False Positives	130 True Negatives

Die $accuracy$ wäre hier:

$$a = \frac{0 + 130}{0 + 130 + 0 + 30} = \frac{130}{160} = 0,8125 = 81,25\%$$

Obwohl der Algorithmus komplett nutzlos ist, da er alle Objekte als “Negativ“ klassifiziert, ist der $accuracy$ Wert um einiges gewachsen.

6.4 F_1 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 (Sasaki et al., 2007) ist das harmonische Mittel aus $precision$ und $recall$. Dadurch können sowohl False Positives als auch False Negatives berücksichtigt werden.

6.5 Datensätze

6.5.1 Sentence-aligned data

Einer der Datensätze welcher für die Auswertung verwendet worden ist, ist ein Datensatz welcher von Coster und Kauchak (2011) erstellt worden ist. Es wurden Sätze aus

Simple English Artikeln mit ihrem Gegenstück aus den English Artikeln aligned. Dieser Datensatz enthält 167k Satzpaare.

Der Datensatz ist wie folgt aufgebaut: Es liegen die Dateien „normal.aligned“ und „simple.aligned“ vor. Die beiden Dateien matchen Zeile für Zeile, d.h. die erste Zeile aus dem „simple.aligned“ matched mit der ersten Zeile aus „normal.aligned“ die zweite mit der zweiten usw. Jede Zeile ist gleich aufgebaut:

$$\text{Name des Artikels} < TAB > \text{Paragraph Nummer} < TAB > \text{Der Satz}$$

Mehr zum Prozess der Erstellung und Aufbau der Datenbank kann in Coster und Kauchak (2011) gefunden werden.

6.6 Bestimmung von Score-Thresholds

Ein Score-Threshold ist ein Grenzwert ab welchem angenommen wird, dass ein Satzpaar ein Match ist. Für jeden Algorithmus muss ein eigener Wert bestimmt werden, da verschiedene Verfahren und Formeln benutzt werden um diesen zu berechnen. Die Schwierigkeit bei der Bestimmung des Thresholds besteht dadrin, dass wenn dieser zu hoch gesetzt wird, viele Matches welche Benutzer auch als solche annotieren würden nicht zustande kommen. Das umgekehrte Problem tritt dann auf wenn dieser zu niedrig ansetzt wird. So würden Sätze als Match markiert werden obwohl Benutzer die Sätze nicht als Match empfinden würden.

Die Idee ist es, dass Diagramme gebildet werden und ihr Schnittpunkt als Threshold gesetzt wird. Der erste Graph bildet dabei ab welche Werte der Algorithmus für Satzpaare ausgibt welche bereits als ein Match vom Benutzer markiert worden sind (Grün in den Abbildungen). Der zweite Graph wird dann auf Satzpaare angewendet welche nicht mit einander matchen (Rot in den Abbildungen). Somit ist der Schnittpunkt der Punkt an dem die Wahrscheinlichkeit höher ist, dass ein Satzpaar ein *Match* ist, als *kein Match*, da es statistisch mehr *Matches* als *nicht Matches* mit diesem Score gibt. Um die Auswertung praxisnah und realistisch zu gestalten, wurden nur Satzpaare verwendet welche aus dem selben Wikipedia Artikel stammen. Das heißt, dass obwohl die Sätze nicht matchen, sie genau das gleiche Thema behandeln. Dies ist für die Problemstellung relevant, da das Tool immer nur in einem Artikel nach dem besten Satz sucht und somit alle Satzpaare das gleiche Thema behandeln. Die Algorithmen wurden auf den Datensatz 'Sentence-aligned data' von William Coster und David Kauchack angewendet. Es wurden jeweils 50000 Satzpaare betrachtet da pro Artikel oftmals nur ein oder wenige Satzpaare vorhanden waren und somit weniger *nicht Matches* gebildet werden konnten.

Auf der X-Achse sind dabei die Werte zwischen 0 und 1 welche vom Algorithmus ausgegeben werden können. Auf der Y-Achse ist die Anzahl der Satzpaare in Prozent. Somit liegt eine Kurve für *Matches* und eine Kurve für *nicht Matches* vor. Der Schnittpunkt dieser Kurven ist dann der Punkt an dem es dann mehr *Matches* als *nicht Matches* gibt.

Diese Methode den Threshold zu bestimmen ist eine Eigenentwicklung und nur bedingt brauchbar da *Matches* und *nicht Matches* als gleichwertig angesehen werden. Der Threshold sollte an den Zweck der Anwendung angepasst werden, je nachdem was wichtiger ist. Wenn es wichtiger ist, dass man überhaupt ein Ergebniss bekommt, dann kann man

den Threshold nach unten korrigieren, wird dadurch aber mehr False Positives erhalten. Wenn der Fokus jedoch auf der Genaugkeit liegt, dann kann man den Threshold nach oben korrigieren, nimmt dann aber in Kauf, dass Sätze welche vielleicht ein mögliches Match wären, nicht als solche erkannt werden.

6.6.1 Kosinus-Ähnlichkeit von Wortvektoren

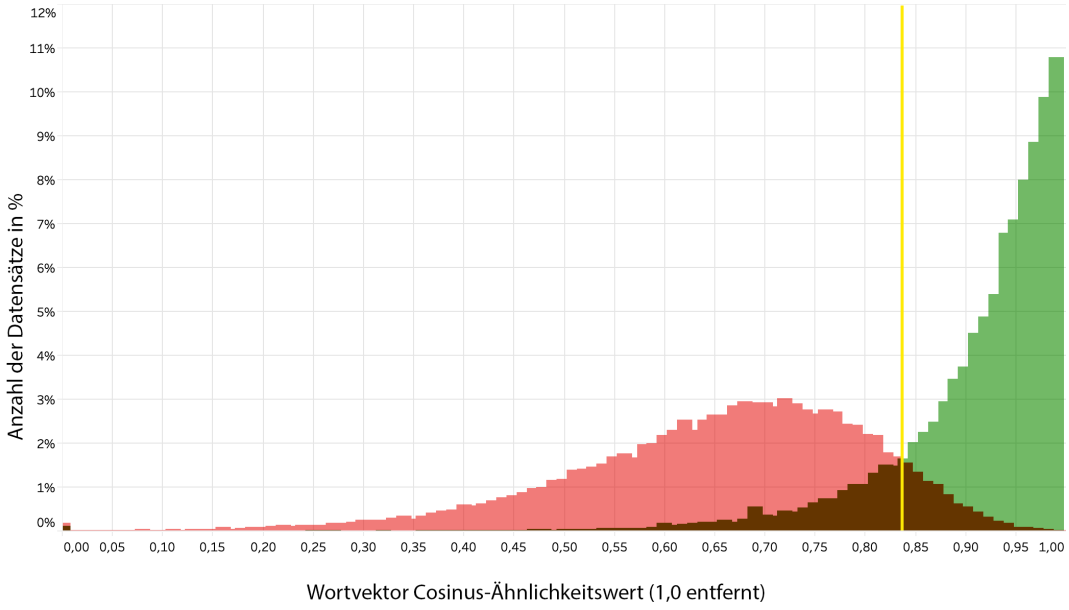

Abbildung 7: Diagramm Wortvektor Cosinus-Ähnlichkeit

In Abbildung 7 ist das Schnittdiagramm für die Wortvektor Cosinus-Ähnlichkeit zu sehen. Der Cosinus-Ähnlichkeitswert 1,0 wurde entfernt, da nur der Schnitt relevant ist und dieser Wert überproportional oft vertreten ist. Der Grund dafür ist, dass nach der Textvorverarbeitung das Satzpaar aus zwei identischen Sätzen besteht.

Wenn der Schnittpunkt der beiden Balkenandiagramme betrachtet wird, dann fällt auf, dass ab einem Cosinus-Ähnlichkeitswert von ca. 0,84 es Prozentmäßig anfängt mehr *Matches* zu geben als *nicht Matches*. Das bedeutet, dass die Wahrscheinlichkeit, dass ein Satzpaar ein *Match* ist, ab einem Cosinus-Ähnlichkeitswert von ca. 0,84, größer ist als, dass es *kein Match* ist. Somit ist es ein guter Wert der als Threshold für den Wortvektor Cosinus-Ähnlichkeits Algorithmus benutzt werden kann.

6.6.2 Jaccard-Index

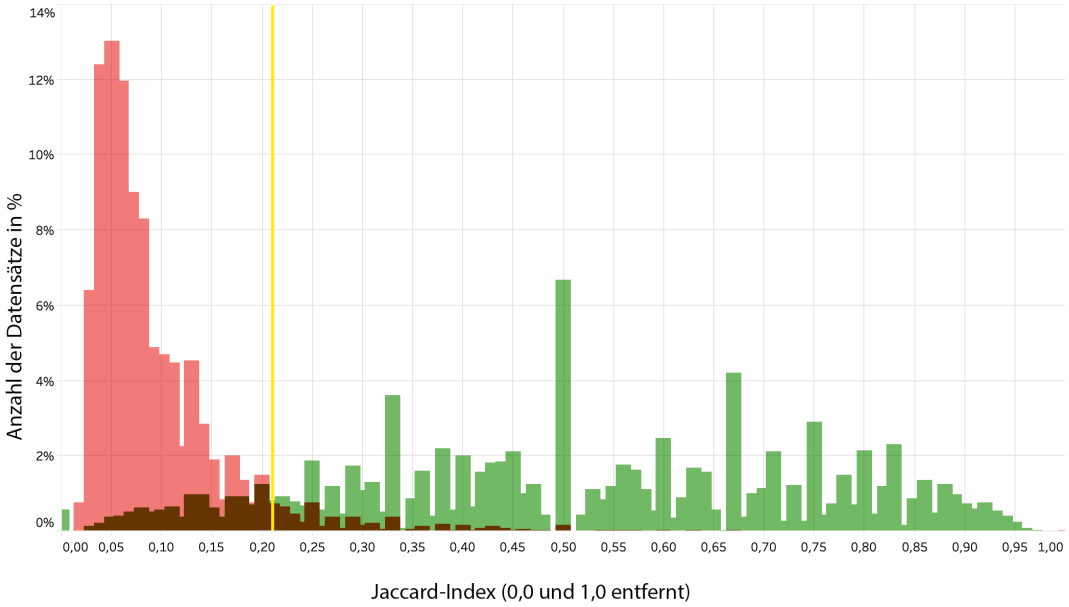

Abbildung 8: Diagramm Jaccard-Index

In Abbildung 8 ist das Schnittdiagramm für den Jaccard-Index zu sehen. Die Werte 0,0 und 1,0 wurden entfernt, da wir uns nur für den Schnitt interessieren und diese Werte überproportional oft vertreten sind.

Wenn der Schnittpunkt der beiden Balkendiagramme betrachtet wird, dann fällt auf, dass ab einem Jaccard-Index von ca. 0,21 es Prozentmäßig anfängt mehr *Matches* zu geben als *nicht Matches*. Das bedeutet, dass die Wahrscheinlichkeit, dass ein Satzpaar ein *Match* ist, ab einem Jaccard-Index von ca. 0,21, größer ist als, dass es *kein Match* ist. Somit ist es ein guter Wert der als Threshold für den Jaccard-Index Algorithmus benutzt werden kann.

6.6.3 TF-IDF

Mit dieser Methode ist es leider nicht möglich den Threshold für den TF-IDF Algorithmus zu bestimmen. Da der Algorithmus darauf basiert, dass ein ganzer Textkorpus vorliegt und die Gewichte der Wörter mit Hilfe von anderen Wörtern aus dem gleichen Artikel bestimmt werden, kann man den vorliegenden Datensatz 'Sentence-aligned data' von William Coster und David Kauchack nicht benutzen, da dieser nur aus wenigen Satzpaaren pro Artikel besteht. Wegen dieser Einschränkung ist der Algorithmus für unsere Ziele nicht gut geeignet.

6.6.4 TF-Vektor Cosinus-Ähnlichkeit

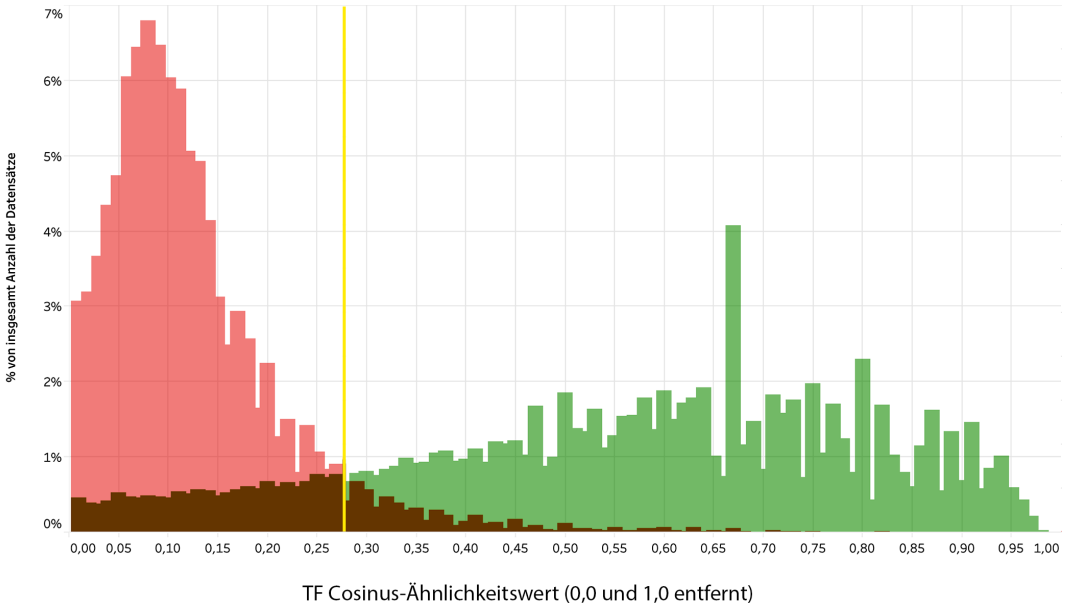

Abbildung 9: Diagramm TF-Vektor Kosinus-Ähnlichkeit

In Abbildung 8 ist das Schnittdiagramm für die Cosinus-Ähnlichkeit von TF-Vektoren zu sehen. Die Werte 0,0 und 1,0 wurden entfernt, da wir uns nur für den Schnitt interessieren und diese Werte überproportional oft vertreten sind.

Wenn der Schnittpunkt der beiden Balkendiagramme betrachtet wird, dann fällt auf, dass ab einem Kosinus Ähnlichkeitswert von TF-Vektoren von ca. 0,28 es Prozentmäßig anfängt mehr *Matches* zu geben als *nicht Matches*. Das bedeutet, dass die Wahrscheinlichkeit, dass ein Satzpaar ein *Match* ist, ab einer Kosinus-Ähnlichkeit von TF-Vektoren von ca. 0,28, größer ist als, dass es *kein Match* ist. Somit ist es ein guter Wert der als Threshold für den TF-Vektor Kosinus-Ähnlichkeit Algorithmus benutzt werden kann.

6.7 Benutzer-Bewertung

Um herauszufinden wie gut die Algorithmen Sätze matchen ist eine Annotation von den Ergebnissen nötig. Die Benutzer können die vom Algorithmus ausgewählten Satzpaare bewerten. Das Bewertungsverfahren wurde an Hwang et al. (2015) angelehnt. In manchen Fällen ist es subjektiv ob zwei Sätze matchen oder nicht, deshalb wurde folgendes System benutzt um die Satzpaare zu bewerten:

3	Informationsgehalt ist gleich geblieben
2	Geringe Unterschiede im Informationsgehalt
1	Große Unterschiede im Informationsgehalt
0	Alle Informationen sind verloren gegangen

Beispiele:

3	Goals are scored by moving the ball beyond the goal line into the opposing goal.	Goals are scored by getting the ball into the opponents goal.
2	It 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. It is played in more countries than any other game.
1	Providing continuous 24/7 service, the New York City Subway is the largest single-operator rapid transit system worldwide, with 472 rail stations.	Subway transportation is provided by the New York City Subway system, one of the biggest in the world. Pennsylvania Station, the busiest train station in the United States, is here.
0	The game is played on a rectangular field called a pitch with a goal at each end.	If a player kicks the ball out of play at the other end of the field, the other team kicks the ball back into play from directly in front of the goal (a goal kick).

6.7.1 Scores anhand der Benutzer-Bewertungen

Mit Hilfe der Benutzerbewertungen wurden pro Algorithmus 300 Satzpaare untersucht. Dazu wurden mehrere Wikipedia Artikel bewertet und in einer Datenbank gespeichert. Es wurden für alle Algorithmen genau die selben Sätze vom Benutzer ausgesucht (angeklickt), die daraus resultierenden Satzpaare sind jedoch oft unterschiedlich, da die Algorithmen oft unterschiedliche Sätze als *Match* aussuchen.

	t_p	t_n	f_p	f_n
Jaccard-Index	110	128	29	33
Kosinus-Ähnlichkeit von Wortvektoren	128	55	107	10
Kosinus-Ähnlichkeit von TF-Vektoren	94	108	66	32

Tabelle 1: Binäre Klassifikatoren

In der Tabelle 1 ist die Verteilung der True Positives, True Negatives, False Positives und False Negatives von dem annotierten Datensatz zu sehen.

	Precision	*Recall*	*Accuracy*	F_1
Jaccard-Index	**0,791**	0,769	**0,793**	**0,780**
Kosinus-Ähnlichkeit von Wortvektoren	0,545	**0,927**	0,610	0,686
Kosinus-Ähnlichkeit von TF-Vektoren	0,587	0,746	0,763	0,657

Tabelle 2: Precision, Recall, Accuracy und F_1

In der Tabelle 2 sind die Maße *Precision*, *Recall*, *Accuracy* und F_1 welche mit Hilfe der Tabelle 1 berechnet worden sind zu sehen.

Zunächst ist anzumerken, dass es für einen Algorithmus für die vorliegende Problemstellung wichtiger ist ähnliche Sätze bzw. gute Satzpaare zu finden, als diese korrekt zu klassifizieren. Da das entwickelte Programm ein Annotationstool ist, soll der Algorithmus dem User helfen die besten Sätze aus dem Artikel zu finden, ob der Algorithmus dabei ausgibt, dass diese matchen oder nicht ist dabei zweitrangig. Es ist besser wenn der Algorithmus den am besten matchenden Satz aus dem Artikel findet und diesen dabei nicht korrekt als nicht matchend klassifiziert als wenn ein nicht matchender Satz gefunden wird und korrekt klassifiziert wird.

Wenn wir also die $t_p + f_n$ betrachten, dann erhalten wir die Anzahl der matchenden Sätze:

$$|Matches(Jaccard - Index)| = 110 + 33 = 143$$
$$|Matches(Wortvektoren)| = 128 + 10 = 138$$
$$|Matches(TF)| = 94 + 32 = 126$$

Die meisten *Matches* wurden vom Jaccard-Index Algorithmus gefunden was etwa 47,66% vom gesammten Datensatz sind. Etwas schlechter hat der Kosinus-Ähnlichkeit von Wortvektoren Algorithmus abgeschnitten, welcher 46% der *Matches* fand. Der Kosinus-Ähnlichkeit von TF-Vektoren Algorithmus konnte in 42% einen *Match* finden. Hierbei ist es wichtig zu beachten, dass nicht in 100% der Fälle ein Match überhaupt möglich war. Die hängt damit zusammen, dass nicht jeder Satz aus dem English Wikipedia Artikel einen matchenden Satz im Simple English Wikipedia Artikel hatte.

Wenn wir die *kein Match / Match* Klassifizierung betrachten dann ist der hohe *Recall* Wert (0,927) des Kosinus-Ähnlichkeit von Wortvektoren Algorithmus auffällig. Da der *Pricision* Wert (0,545) im Vergleich dazu relativ niedrig ist können wir sehen, dass dieser Algorithmus dazu neigt die meisten Sätze als *Match* zu klassifizieren. Dies ist auch aus dem im Vergleich zu den anderen Algorithmen hohem Anteil an f_p zu sehen. Der Kosinus-Ähnlichkeit von Wortvektoren Algorithmus hat dabei 107 f_p, der Jaccard-Index Algorithmus hat 29 f_p und der Kosinus-Ähnlichkeit von TF-Vektoren Algorithmus 66 f_p.

Der Jaccard-Index Algorithmus ist in *Precision*, *Accuracy*, F_1 und der Anzahl der gefundenen Matches den anderen Algorithmen überlegen. Somit ist es der beste Algorithmus für die vorliegende Problemstellung. Der Kosinus-Ähnlichkeit von Wortvektoren Algorithmus ist ebenfalls sehr nützlich. Dieser findet fast genau so oft *Matches* wie der Jaccard-Index Algorithmus, klassifiziert jedoch die *nicht Matches* oftmals nicht korrekt. Der Kosinus-Ähnlichkeit von TF-Vektoren Algorithmus schneidet am schlechtesten ab.

7 Fazit

Nun soll auf die Ergebnisse, Verbesserungsvorschläge und mögliche Anwendungsmöglichkeiten eingegangen werden.

Es konnte gezeigt werden, dass der Jaccard-Index Algorithmus am besten von den untersuchten Algorithmen die ähnlichsten Sätze unter den English - Simple English Wikipedia Artikeln finden konnte. Der Wortvektor Cosinus-Ähnlichkeit Algorithmus kommt an zweiter Stelle, dieser klassifiziert die Sätze in *Match/nicht Match* zwar oft falsch, findet jedoch auch fast genau so viele Sätze wie der Jaccard-Index Algorithmus. Auffällig ist auch, dass der Jaccard-Index Algorithmus schneller ein Ergebnis findet, als der Wortvektor Cosinus-Ähnlichkeit Algorithmus, welcher bei langen Sätzen und Artikel mehrere Sekunden braucht, dies könnte eine große Rolle in der Auswahl des Algorithmus spielen falls Menschen mehrere tausende von Sätzen annotieren wollen.

In der Zukunft könnte man untersuchen ob sich bestimmte Algorithmen für bestimmte Arten von Wikipedia Artikeln besser eignen. Da der Jaccard-Index Algorithmus präzise Wortmatches sucht und der Wortvektor Cosinus-Ähnlichkeit Algorithmus die Bedeutung der Wörter betrachtet, könnte es sein, dass der Jaccard-Index Algorithmus sich eher für Artikel mit vielen harten Fakten eignet und der Wortvektor Cosinus-Ähnlichkeit Algorithmus besser für Artikel in denen eine gewisse Ambiguität herrscht, wie z.B. Artikel über Literatur, zu gebrauchen ist.

Desweiteren könnte die Bestimmung der variablen Anzahl der Sätze verbessert werden. Es sollte untersucht werden in welchen Fällen ein komplexer Satz in mehrere einfache Sätze aufgeteilt wird. In dieser Arbeit wurden nur die Anzahl der „and“ und Kommata betrachtet, es existieren mit hoher Wahrscheinlichkeit noch mehr Merkmale welche auf eine Aufteilung der Sätze hindeuten.

Auch können neue Algorithmen ausprobiert werden oder die untersuchten Algorithmen ausgebaut werden. So könnte z.B. mit „n-grams“ und „regular expressions“ gearbeitet werden um zu sehen ob mehr und/oder bessere *Matches* gefunden werden.

Die Datenbank welche mit Hilfe des Tools aufgebaut werden würde, könnte für Machine Learning verwendet werden. Die damit trainierte KI könnte dazu eingesetzt werden komplexe Sätze automatisch in einfache Sätze zu übersetzten. Davon könnten vor allem Fremdsprachler und allgemein Menschen mit einem niedrigen Sprachniveau profitieren.

References

Vimala Balakrishnan und Lloyd-Yemoh Ethel (Jan. 2014). „Stemming and Lemmatization: A Comparison of Retrieval Performances“. In: *Lecture Notes on Software Engineering* 2, S. 262–267.

Joeran Beel, Bela Gipp, Stefan Langer und Corinna Breitinger (2016). „paper recommender systems: a literature survey“. In: *International Journal on Digital Libraries* 17.4, S. 305–338.

William Coster und David Kauchak (Juni 2011). „Simple English Wikipedia: A New Text Simplification Task“. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, S. 665–669.

Hans-Friedrich Eckey, Reinhold Kosfeld und Martina Rengers (2002). *Multivariate Statistik: Grundlagen - Methoden - Beispiele*. Gabler, S. 219.

Pritam C Gaigole, LH Patil und PM Chaudhari (2013). „Preprocessing techniques in text categorization“. In: *National Conference on Innovative Paradigms in Engineering & Technology (NVIPET-2013), Proceedings published by International Journal of Computer Applications (IJCA)*.

Karin Haenelt (2006). „Ähnlichkeitsmaße für Vektoren“. In: *Kursfolien* 26, S. 1.

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.

Jing Jiang (2012). „Information extraction from text“. In: *Mining text data*. Springer, S. 11–41.

Anjali Ganesh Jivani et al. (2011). „A comparative study of stemming algorithms“. In: *Int. J. Comp. Tech. Appl* 2.6, S. 1930–1938.

Subbu Kannan und Vairaprakash Gurusamy (2014). „Preprocessing techniques for text mining“. In: *Conference Paper. India*.

NLTK vs. spaCy: Natural Language Processing in Python (2019). URL: https://blog.thedataincubator.com/2016/04/nltk-vs-spacy-natural-language-processing-in-python/ (besucht am 16. 08. 2019).

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.

David Martin Powers (2011). „Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation“. In:

Anand Rajaraman und Jeffrey David Ullman (2011). „Data Mining“. In: *Mining of Massive Datasets*. Cambridge University Press, S. 1–17.

Yutaka Sasaki et al. (2007). „The truth of the F-measure“. In: *Teach Tutor mater* 1.5, S. 1–5.

Simple Wikipedia (2019). URL: https://simple.wikipedia.org/wiki/Main_Page (besucht am 16. 08. 2019).

Abbildungsverzeichnis

Tabellenverzeichnis