Bauhaus-Universität Weimar
Fakultät Medien
Studiengang Medieninformatik

Eruierung von Methoden zur Exploration von Textwiederverwendung in großen Datenmengen am Beispiel der Wikipedia

Bachelorarbeit

Tristan Licht Matrikelnummer: 70688
geb. am. 31.05.1985 in Weimar

1. Gutachter: Junior-Prof. Dr. Matthias Hagen

Betreuer: Dr. Martin Potthast, Michael Völske

Datum der Abgabe: 6. Februar 2017

Eidesstattliche Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche gekennzeichnet.

Ort, Abgabedatum Unterschrift des Verfassers

Abstract

Das Ziel dieser Arbeit ist es, Methoden zur Textwiederverwendung zu begutachten, zu vergleichen und zu implementieren, um diese auf die explorative Untersuchung von ähnlichen Texten am Beispiel der Wikipedia anzuwenden.

Hierfür wurde der Datensatz aller Wikipedia Artikel mit dem Tf-Idf Model vektorisiert und auf Basis einer Spark-Applikation die Ähnlichkeiten aller Texte miteinander verglichen. In einem zweiten Schritt wurden Texte mit hoher Übereinstimmung genauer betrachtet und auf Wortebene gegenübergestellt.

Das Resultat der Arbeit ist eine Programmpipeline, mit deren Hilfe Textkorpora einer Größe von bis zu 10^8 Texten in wenigen Tagen exakt verglichen werden können. Des Weiteren wird ein Einblick über die gefundenen Plagiatsfälle gegeben.

Inhaltsverzeichnis

1 Einführung

In einem immer größer wachsenden Datennetz wie dem öffentlichen Internet, welches primär durch ökonomische Interessen vorangetrieben wird, werden Inhalte längst nicht mehr ausschließlich neu kreiert.

Der ursprünglich noble Gedanke, Wissen zusammen zu tragen, Gedanken zu veröffentlichen und dezentral weltweit zugänglich zu machen wurde durch die höhere Verbreitung und Kommerzialisierung zunehmend abgewandelt. Das öffentliche, mittlerweile zum Großteil durch Werbung finanzierte, Teil des World Wide Web (WWW) beherbergt nun unzählige Seiten mit kopierten und plagiierten Inhalten. Mit Hilfe von Algorithmen zur automatischen Generierung von Internetseiten entstand ein Markt, der mit wenig Aufwand, fremden Inhalten aus frei zugänglichen Quellen und finanziert durch Werbenetzwerke wachsen konnte. Die Frage, die sich hier stellt, ist wie hoch der Anteil der kopierten beziehungsweise plagiierten Seiten im WWW ist. Die Menge der öffentlich zugänglichen Websites lässt sich nicht adäquat abschätzen. Google, als größter Suchmaschinenbetreiber, hat 2008 [gb1] die letzten, offiziellen Zahlen veröffentlicht und erreichte damals die Marke von einer Billion indexierter Webseiten. Die öffentlichen Webseiten-Sammlungen (engl. Crawls) Common Crawl [cc1] und ClueWeb12 [cw1] beinhalten über zwei Milliarden beziehungsweise 733 Millionen Seiten. Als Quelle für etwaige Plagiate dienen aus empirischen Erkenntnissen in erster Linie freie Datensätze oder Archive wie die Wikipedia, eine öffentlich Zugängliche Enzyklopädie, welche durch ihre Nutzer geschrieben und stetig erweitert wird.

Abbildung 1.1 zeigt ein Beispiel eines Plagiats in Form eines Blogeintrags. Um den Beitrag attraktiver zu gestalten, wurden Teile des themenverwandten Wikipedia-Artikel übernommen und umformuliert.

Um einen Eindruck über die Anzahl und Arten von Plagiaten in solchen großen Datenmengen zu bekommen, müssen als erstes Methoden zur Repräsentation der Seiten evaluiert werden, um anschließend mit geeigneten Algorithmen diese Repräsentationen vergleichen zu können. Das Ziel ist eine "Pipeline", bestehend aus verschiedenen Funktionen und Prozessen, zu erstellen, um effizient die Wiederverwendung von Fließtext, als Replikat oder in abgewandelter Form, in großen Textmengen zu finden. Als Studienobjekt soll hier der Korpus aller aktuellen Wikipedia-Artikel dienen, welcher mit über fünf Millionen Dokumenten ausreichend groß ist, um die Effizienz der Algorithmen zu evaluieren und Textwiederverwendung in der Praxis zu analysieren.

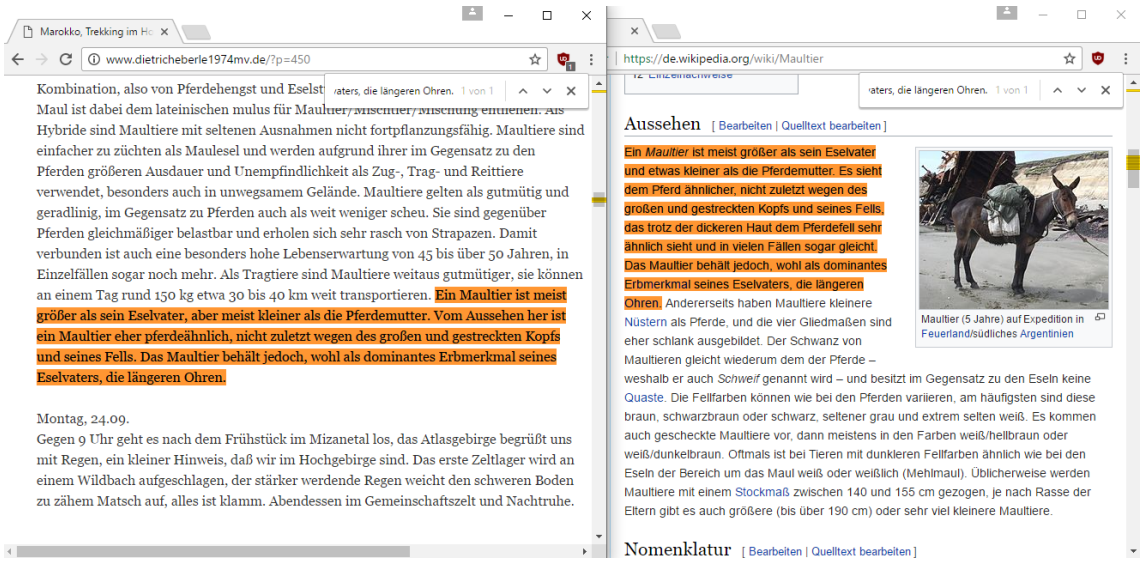


Abbildung 1.1: Textwiederverwendung [1] mit Paraphrasierung aus dem Web mit Wikipedia [2] als Quelle

Das grundlegende Problem der Suche nach Textwiederverwendung lässt sich in zwei Teile gliedern. Zum einen muss für ein potentiell plagiiertes Dokument ein großer Korpus aus Texten durchsucht werden, um etwaige Quelldokumente zu finden. Falls, wie im Fall von Wikipedia, jeder Artikel mit jedem verglichen werden soll, resultiert dies im schlechtesten Fall in einer quadratischen Rechenzeit im Hinblick auf die Eingabedaten, da jeder Text mit jedem anderen des Korpus verglichen werden muss. Hierzu muss ein Algorithmus gefunden werden, der auch bei Daten in Größenordnungen von 10^7 und mehr berechenbar bleibt. Das Ergebnis der Suche ist eine Liste von Dokumentenpaaren, bei denen potentiell gleiche Textpassagen verwendet werden. Als zweiten Schritt müssen die gefundenen, ähnlichen Textpaare näher mit einander verglichen werden (Text-Alignement), um Textwiederverwendung und Paraphrasierung auf Wortebene analysieren zu können. Das Resultat ist eine Reihe von Dokumentenpaaren und die Markierungen ihrer gleichen Textabschnitte. Anhand dieser kann ein Überblick über den Umfang von Textwiederverwendung und der verschiedenen Arten von Plagiaten geschaffen werden.

---

1 Eberle, Friedrich: Marokko, Trekking im Hohen Atlas (15.10.2012), URL: http://www.dietricheberle1974mv.de/?p=450 (Stand: 02.01.2017)

2 Wikipedia: Maultier (21.12.2016), URL: https://de.wikipedia.org/w/index.php?title=Maultier&oldid=160874566 (Stand: 02.01.2017)

Kapitel 3 der Arbeit behandelt die allgemeine Vorverarbeitung (Preprocessing) von Texten. Anschließend werden verschiedene Textrepräsentationen und Abbildungen, wie Hashverfahren und Vektormodelle, mit ihren zugehörigen Ähnlichkeitsoperationen verglichen. Hierzu werden diese an einem Beispielkorpus durchgeführt, um deren Eignung zur Erkennung von Textwiederverwendung zu evaluieren.

In Kapitel 4 wird die finale Implementierung mit Hilfe von Apache Spark erläutert, beginnend mit der Verarbeitung der Wikipedia-Daten, der Vektorisierung von Texten sowie der Berechnung aller Ähnlichkeiten.

Der 5. Abschnitt behandelt das nähere Abgleichen der ähnlichsten Artikel sowie der Umsetzung auf parallelen Rechensystemen.

Abschließend werden die Ergebnisse näher betrachtet sowie ein Ausblick auf potentielle Verbesserungen des Textabgleichs gegeben.

2 Verwandte Arbeiten

In Anbetracht der Aufgabe müssen für die Bearbeitung verschiedene Themen berücksichtigt werden.

„*Identifying Duplicate and Contradictory Information in Wikipedia*“ (Jimmy Lin et al.) [shl] beschäftigt sich auch mit Ähnlichkeiten in der Wikipedia. Hier wurden alle duplizierten und gegensätzlichen Sätze in der Wikipedia gesucht und kategorisiert. In dem Whitepaper werden alle Wikipedia-Artikel auf Satzebene mittels Local-Sensitive-Hashing (LSH) abgebildet. Jeder Satz wurde in n-Gramme auf Zeichenebene, also Zeichenketten mit Länge n, geteilt und mit Minhash [mh1] als Hashwerte gebildet. Die resultierenden Hashkollisionen wurden gruppiert, um ähnliche Sätze zu finden und zusammenzufassen. Im Folgenden wurden diese Gruppen explorativ analysiert und kategorisiert. Zu den größten, identifizierten Kategorien gehören identische Kopien von Sätzen, leichte Abänderungen von Sätzen, entweder durch inhaltliche oder förmliche Veränderung und Templates, bei denen Satzvorlagen mit gleicher Struktur nur durch themenbezogene Objekte ergänzt wurden.

Quoc V. Le et al. analysierten in dem Whitepaper „*Document Embedding with Paragraph Vectors*“ [pv2] die Mächtigkeit von Dokumenteinbettungen in Hinsicht auf die Abbildung der semantischen Bedeutung durch Vektoren. Hier wurden durch Methoden des maschinellen Lernens Vektorrepräsentationen von Wikipedia und arXiv Artikeln trainiert und mit verbreiteten Modellen wie BOW oder LDA verglichen. In den Experimenten konnte gezeigt werden, dass diese Paragraph Vectors in der Sentiment-Analyse die Genauigkeit der anderen Methoden erreichen und teilweise übertreffen konnten.

An der Bauhaus Universität Weimar wurde in „*Visualizing Article Similarities in Wikipedia*“ [bu1] eine Visualisierung der Similarität aller Wikipedia-Artikel entwickelt. Zu diesem Zweck wurden alle Texte mit dem Vector-Space-Model auf Basis von Tf-Idf Gewichten abgebildet und die Ähnlichkeiten mit der Kosinussimilarität berechnet.

Ein anderer Weg die Nähe von Dokumenten zu errechnen wird in „*From Word Embeddings To Document Distances*“ [wmd1] beschrieben. Diese dient eher der Klassifizierung von Texten, könnte aber auch dazu dienen, Verschleierung oder Paraphrasierung von Plagiatsfällen zu erkennen. Bei dem Modell werden die Word2Vec [w2v1] Worteinbettungen als Basis für den Textvergleich genutzt. Eine Eigenschaft von Word2Vec ist es, bedeutungsgleiche und bedeutungsähnliche Worte als Vektoren in enge Nachbarschaft zu modellieren. Dadurch kann der Abstand von einzelnen Worte in zwei Texten im Bedeutungsraum gemessen werden, was dabei hilft, eine Relation von syntaktisch verschiedenen Worten in den Textvergleich einzubeziehen.

3 Methoden zur Textabbildung

Die Untersuchung zweier Texte auf gleiche oder ähnliche Zeichensequenzen ist ein bekanntes und effizient gelöstes Problem der Informatik. Bei der Suche nach einer Textpassage in einem Text existiert eine Reihe von Methoden aus der Familie der String-Matching-Algorithmen. Bereits der naive Ansatz einen Suchstring Stück für Stück mit dem Text zu vergleichen, ist mit $O(N + M)$ berechenbar, wobei N die Textlänge und M die Länge des Suchstrings ist. Die Differenz zweier Texte zu erkennen, wurde bereits 1986 mit hinreichender Laufzeit in „*An O(ND) Difference Algorithm and its Variations*“ [sd1] und „*An O(NP) sequence comparison algorithm*“ [sd2] entwickelt.

Bei dem Vergleich eines Textes mit einer größeren Menge $M > 10^8$ von Texten stößt man mit solch genauen Volltextsuchalgorithmen an die Grenzen des Berechenbaren. Wenn nun jeder Text t_1 einer Menge M mit jedem anderen Text $t_2, t_3 \dots t_n$ verglichen werden soll, ist dies mit den erwähnten Algorithmen nicht mehr berechenbar. Wenn ein Textvergleich im praktischen Beispiel durchschnittlich 0.001s dauert, würde der Vergleich einer Menge von 10^8 Texten durch die quadratische Laufzeit 3.171 Jahre betragen.

Zur Beschleunigung der Suche nach Textwiederverwendung bedarf es einer effizienteren Repräsentation der Textdokumente sowie einer schnellen Vergleichsoperation, um die Größe der zu durchsuchenden Datenmenge für jeden Text zu reduzieren. Hypothetisch sei ein Beispielkorpus K mit n Dokumenten $\{d_1, d_2 \dots d_n\}$ gegeben. Es muss nun zu jedem Quelltext beziehungsweise Dokument d_s jeder ähnliche Text $\{d_{sp1}, d_{sp2} \dots d_{spn}\}$ aus dem Korpus K gefunden werden, der eine vorher definierte Ähnlichkeitsschwelle δ überschreitet. Das Ergebnis ist die Resultatmenge $R_s = \{d_{sp1}, d_{sp2} \dots d_{spn}\}$.

$$f(d_{s,}K, \delta) = \{d_{sp1}, d_{sp2} \dots d_{spn}\} = R_s \subseteq K \qquad mit\ R \ll K$$

Das Ziel muss es nun sein, einen Algorithmus f zu finden, der die Resultatmenge für jedes Dokument so exakt wie möglich errechnet.

Das beschriebene Problem kann man dem Information-Retrieval (IR) zuordnen. Dabei geht es um die Beschaffung von relevanten Information aus einer großen Informationsmenge. Im Grunde umfasst es Arbeitsweisen aller Suchmaschinen, bei denen anhand von Stichworten oder Textsequenzen alle relevanten Dokumente gefunden werden müssen. Zur Evaluierung von geeigneten Methoden werden nun folgende Verfahren näher betrachtet: Die Suche nach Duplicates und Near-Duplicates durch Hashing, Vektormodelle basierend auf dem Vector-Space-Model (Vektorraum Retrieval), Sentiment-Analysis-Verfahren durch Paragraph Vectors sowie Topic-Modelling durch Latent-Semantic-Indexing.

3.1 Text-Preprocessing

Für die maschinelle Analyse von Dokumenten muss der Fließtext vorverarbeitet werden. Dies ist einerseits sinnvoll, um die Geschwindigkeit der späteren Verarbeitung zu beschleunigen, als auch die Qualität des Ergebnisses zu verbessern.

Im ersten Schritt werden Textbausteine je nach Anforderung in Sätze, Wörter oder Buchstaben aufgeteilt. Diese aufgeteilten Elemente werden als Token bezeichnet, der Vorgang als Tokenization. Das geschieht vorwiegend durch die Analyse der Interpunktion und Leerzeichen. Abgesehen von Punkt und Komma, ist die korrekte Interpretation von Satzzeichen wie Apostroph oder Klammern durch Algorithmen aufwendig und benötigt mehr Aufwand. Wenn der Satz „*You can't do that.*“ in Token aufgeteilt werden soll, produziert ein naiver Algorithmus „*You*“, „*can't*“, „*do*“, „*that*“. Das „*can't*“ müsste in „*can*“ und „*not*“ oder „*cannot*“ substituiert werden, um eine adäquate Tokenisierung zu erzielen. An dieser Stelle ist es sinnvoll vorgefertigte Bibliotheken wie NLTK oder den Stanford-Tokenizer zu benutzen.

Je nach Aufgabenbereich, wie beispielsweise Plagiats- oder Bedeutungserkennung, sind linguistische Textbausteine und Wörter wie Artikel, Konjunktionen und Präpositionen für die Relevanz von Suchergebnissen ohne Mehrwert. Da diese sogenannten Stoppworte, unabhängig des Themas von Dokumenten, überproportional in der natürlichen Sprache auftreten, dominieren sie bei der Suche die relevanten, aber weniger auftretenden Begriffe. Im Folgenden werden drei Sätze als Beispiel anhand von gleichen Worten auf Textwiederverwendung mit und ohne Stoppworte geprüft.

1. Der Bundespräsident bereitet seine Rede im Garten von Schloss Bellevue vor.
2. Auch im Schloss Bellevue wird hart gearbeitet, Bundespräsident Gauck wirkt angespannt.
3. Der Fuchs lauert vor dem Hühnerstall, seine Beute sitzt im Gras.

Alle drei Sätze besitzen elf Worte, die Relevanz besteht nur zwischen Satz 1 und 2. Dennoch ist die prozentuale Ähnlichkeit der Worte zwischen Satz 1 und 2, sowie 1 und 3 mit 36% gleich. Das selbe Beispiel ohne Stoppworte führt zu einem besseren Ergebnis bei dem Vergleich in Hinblick auf die thematische Ähnlichkeit.

1. Bundespräsident bereitet Rede Garten Schloss Bellevue
2. Schloss Bellevue hart gearbeitet Bundespräsident Gauck wirkt angespannt
3. Fuchst lauert Hühnerstall Beute sitzt Gras

Wie zu sehen ist, besteht jetzt keinerlei Ähnlichkeit mehr zwischen Satz 1 und 3, wohingegen die Sätze 1 und 2 noch eine Übereinstimmung von 27,3% besitzen.

Ein weiterer Schritt ist das Stemming. Hierbei werden Wörter auf ihren Wortstamm reduziert beziehungsweise auf ihre Grundform zurückgeführt. Bei Verben wird die Konjugation aufgehoben, bei Nomen die Deklination. Dies wird mithilfe einfacher Substitutionsalgorithmen bewerkstelligt, die anhand von Listen die Endungen von Worten abändern und damit, besonders in der englischen Sprache, gute Ergebnisse erzielen [ku1], [po1].

Ein Beispiel für die Substitution:

Wörter	Substitution	Stamm
playing, played, plays	ing → ∅, ed → ∅, s→ ∅	play
houses, papers	s→ ∅	house, paper

Durch das Stemming werden Satz- und Textrelevanz bei der Suche beibehalten, obwohl die Syntax auf Zeichenebene unterschiedlich ist.

3.2 Precision und Recall

Um die Anwendbarkeit der Retrieval-Algorithmen im Hinblick auf Plagiats- und Textwiederverwendungserkennung zu prüfen, benötigt man ein Maß für deren Qualität. Im Information-Retrieval dienen hierzu Precision und Recall. Zuerst müssen allerdings andere Begrifflichkeiten definiert werden.

Gegeben sei eine Dokumentmenge D. In D sollen alle themenverwandten Texte eines Dokumentes d_1 gefunden werden. Alle relevanten Texte sind die Menge R, alle nicht relevanten Texte die Menge N. Somit ist die Vereinigung von R und N die Menge D aller Dokumente. Ein Algorithmus soll nun möglichst alle relevanten Texte aus D heraussuchen und zeitgleich alle nicht relevanten Daten ignorieren. Da diese Perfektion schlecht erreichbar ist, liefert ein Beispielalgorithmus nun eine Menge P an Dokumenten, die als positiv bewertet wurden. In dieser Menge befindet sich ein Teil P_T (True Positive), der richtig erkannt wurde. Der andere Teil wurde als richtig erkannt, gehört aber nicht zur gewünschten Abfragemenge P_F (False Positive). Die Dokumente, die der Algorithmus

als nicht relevant klassifiziert hat, sind die Menge N. Sie teilt sich in einen Teil, der zurecht als falsch eingestuft wurde N_T (True Negative) sowie in einen Teil N_P , der fälschlich als nicht relevant verworfen wurde (False Negative).

Precision definiert nun die Genauigkeit der Suchanfrage. Hier wird die Anzahl der als relevant klassifizierten Dokumente durch die Anzahl aller relevanten Elemente aus D geteilt.

$$Precision = \frac{|P_T|}{|P|}$$

Der Recall beschreibt das Verhältnis der True Positive Ergebnisse zu allen relevanten Dokumenten in D.

$$Recall = \frac{|P_T|}{|D|}$$

Das Ziel muss sein, durch eine geeignete Methode, den Recall zu maximieren ohne die Precision zu vernachlässigen. Man will alle relevanten Dokumente erhalten und die Anzahl der falsch positiven Resultate minimieren.

3.3 Retrieval-Methoden

3.3.1 Vector-Space-Model

Das Vector-Space-Model [vsm] ist eine indexbasierte Vektorisierungsmethode für Texte. Anstatt wie bei einfachen Vergleichsmethoden alle Texte als Zeichenketten im Speicher zu vergleichen, werden alle Dokumente in einem metrischen Vektorraum abgebildet. Um dies zu erreichen, wird zu Beginn ein fortlaufender Index aller vorkommenden Wörter einer Dokumentenmenge erstellt und abgespeichert. Das Resultat ist ein Wörterbuch, bei dem jedem verschiedenen Wort ein Indexwert von 1 bis n zugeordnet ist. Für jeden Artikel wird nun ein n-dimensionaler Vektor erstellt. Für jedes Wort in dem Artikel wird der Wörterbuchindexwert gesucht und die entsprechende Vektorindexposition inkrementiert.

Beispiel:

1. Der Mann steht auf der Leiter.
2. Die Frau sitzt vor der Leiter.

Indexierung:

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
	der	mann	steht	auf	leiter	die	frau	sitzt	vor

Abbildung:

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
1	2	1	1	1	1	0	0	0	0
2	1	0	0	0	1	1	1	1	1

Das Wörterbuch besitzt nun neun Einträge. Die entsprechenden Vektoren sind:

$$\overrightarrow{v_1} = (2,1,1{,}1,1,0,0,0,0)^T \qquad \overrightarrow{v_2} = (1,0,0,0,1{,}1,1,1,1)^T$$

Die Vektorrepräsentation hat den Vorteil, dass Methoden der linearen Algebra benutzt werden können, um die Similarität von Texten zu berechnen. Der Kosinus zwischen zwei Vektoren beschreibt nicht nur den Winkel der Vektoren im Raum, sondern auch die Ähnlichkeit der zugrundeliegenden Texte. Die Berechnung des Kosinus des eingeschlossenen Winkels ergibt sich wie folgt:

$$cossim(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Dieses Maß heißt Kosinus-Ähnlichkeit (1). Da alle Werte der Eingabevektoren ausschließlich positiv sind, ergibt die Berechnung der Kosinus-Ähnlichkeit immer Werte zwischen 0 und 1, wobei 1 bei gleichen Vektoren auftritt und 0 bei orthogonal orientierten Vektoren. Die Übereinstimmung der beiden Beispielsätze durch dieses Maß ist 0.43.

Ein weiteres Maß für die Ähnlichkeit ist der Jaccard-Koeffizient. Dieser berechnet sich aus dem Verhältnis der Schnittmenge von zwei Mengen mit der Vereinigung dieser.

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

Die beiden Beispielvektoren haben einen Jaccard-Koeffizienten von $\frac{2}{9} = 0.22$.

Je nach Anwendungsgebiet kann es sinnvoll sein das Maß zu ändern. Als Maß für Dokumentenähnlichkeit eignet sich die Kosinus-Ähnlichkeit am besten, wie „*Comparison of Jaccard, Dice, Cosine Similarity Coefficient To Find Best Fitness Value for Web Retrieved Documents Using Genetic Algorithm"* [cmp1] gezeigt werden konnte.

Eine Eigenschaft der Vektoren ist die zum Wörterbuch proportional wachsende Größe. Ein Textkorpus mit 10.000 verschiedenen Termen resultiert in 10.000-dimensionalen, dünnbesetzten Vektoren (sparse vectors) für jedes Dokument. Falls ein Korpus mit 1000 Texten vektorisiert wird, ergibt sich eine 10.000 x 1.000 große TD-Matrix (Term-Dokument-Matrix), in der jede Spalte für ein Dokument und jede Zeile für ein Wort steht. Die Anzahl der Stellen pro Spalte, die nicht 0 sind, entspricht der Anzahl der verschiedenen Worte des Ausgangstextes.

Umso größer die resultierende TD-Matrix eines vektorisierten Korpus, desto aufwendiger ist die Berechnung aller Similaritäten. Deerwester at al. beschreibt eine Methode [lsa] zur Dimensionsreduktion mittels Singulärwertzerlegung [svd] namens Latent-Semantic-Indexing. Dadurch lässt sich eine niedrig dimensionale Approximation der gegebenen Matrix finden. In der Praxis kann so eine Spaltenlänge von mehreren zehntausend auf wenige hundert verringert werden, ohne dass große Abweichungen bei dem Retrieval von Dokumenten im Vergleich zur Ausgangsmatrix entstehen.

3.3.2 Tf-Idf

Bei der VSM basierten Vektorisierung bekommen häufig auftretende Worte durch den höheren Wert an der entsprechenden Vektorindexposition eine stärkere Gewichtung als vereinzelt auftretende Elemente. Dies kann von Vorteil sein, wenn relevante Worte häufig vorkommen. Allerdings treten diese oftmals weniger auf als erwartet. Auch wenn Stoppworte vorher gefiltert wurden, ist es wahrscheinlich, dass bedeutungsschwache Elemente häufiger zu finden sind. Für ein einzelnes Dokument und ohne Vorwissen über einzelne Worte lässt sich so maschinell kein aussagekräftiges Relevanzurteil bilden. Wenn jedoch ein ganzer Dokumentenkorpus als Referenz für die Worthäufigkeit vorliegt, kann festgestellt werden, wie wichtig ein einzelnes Wort für ein Dokument ist. Falls ein Token in sehr vielen Artikeln häufig vorhanden ist, kann man davon ausgehen, dass dessen Relevanz nicht sehr groß ist. Wenn jedoch ein Element in wenigen Texten sehr häufig zu finden ist, scheint die Bedeutung des Wortes für diese speziellen Texte hoch zu sein.

Mit Tf-Idf, Term frequency - Inverse document frequency (3), existiert ein Maß, welches die Gewichtung aller Terme anhand der beschriebenen Vorgehensweise berechnet [tfi]. Der Aufbau der Vektoren ist vergleichbar mit denen des Vector-Space-Model, allerdings sind die Termgewichtungen hier angepasst.

$$term\ frequency: \quad tf(t,d) = f_{t,d} \quad t \in d \qquad (1)$$

$$inverse\ document\ frequency: \quad idf(t,D) = log\frac{N}{n_t} \quad N = |D| \qquad (2)$$

$$tfidf(t,d,D) = f_{t,d} \cdot log\frac{N}{n_t} \quad d \in D, n_t = |\{d \in D : t \in d\}| \qquad (3)$$

Die Termfrequenz (1) beschreibt die relative Häufigkeit eines Terms t in einem Dokument d. Die inverse Dokumentenfrequenz (2) ist das logarithmische Verhältnis von der Anzahl N aller Dokumente im Korpus zu der Anzahl der Dokumente n_t, in denen der Term t vorkommt. Dies hat den Vorteil, dass ein Term, der in vielen oder allen Dokumenten des Korpus vorkommt und dadurch wenig bis keinen Informationsgehalt trägt, durch $log\frac{N}{n_t}$ eine Gewichtung 0 oder fast 0 bekommt.

3.3.3 Local-Sensitive-Hashing

Eine weitere Alternative, um effektiv große Datenmengen auf Wiederverwendung zu prüfen, stellen Local-Sensitive-Hashing Verfahren dar. Auch hier wird die Dimension der Eingabedaten reduziert, um deren Vergleiche mit weniger Aufwand zu berechnen. Auf Basis eines Hash-Algorithmus wird die Eingabe, in dem Fall ein tokenisiertes Dokument, auf einen kleineren Hashwert (Fingerprint) abgebildet. Hashingmechanismen im Information-Retrieval gehören zur Problemklasse der Near-Duplicate-Detection. Hierbei sollen in einer Datenmenge gleiche und fast gleiche Elemente gefunden und zusammengefasst werden. Besonders bei der Web-Suche werden von Suchmaschinenbetreibern Hashingverfahren eingesetzt, um Duplikate bei den Suchergebnissen zu filtern. In „*Detecting near-duplicates for web crawling*“ [shg] wird diskutiert, welche Einsatzgebiete die Duplicate-Dection hat und welche Methoden ausreichend schnell sind, um bei der Echtzeitsuche Duplikate zu finden. Ihr Ergebnis ist, das Simhash [ch1], ein LSH-Verfahren mit 64Bit Hashwerten, ausreichend schnell ist, um in kurzer Zeit acht Milliarden Webseiten zu durchsuchen. Aufgrund dessen soll Simhash für die Erkennung von Textwiederverwendung näher betrachtet werden.

Bei der Arbeitsweise von Simhash wird ein Hashgenerator basierend auf einer Hashfunktion gewählt, um Worte der Dokumente auf Bitstrings abzubilden. Für diesen Schritt sollten kryptografische Hashfunktionen gewählt werden und eine genügende Länge von $n > 32$, um Kollisionen zu vermeiden. Es folgt ein Beispiel mit Adler-8 als Hashfunktion:

1. Der Mann steht aufrecht.
2. Die Frau steht aufrecht.

Word	*Hashvalue*
der	1000 0100
mann	1000 1011
steht	0111 0111
aufrecht	0001 0110
die	0011 1000
frau	1100 0010

Als nächstes werden Zwischenwerte entsprechend des Vorkommens der Worte in den Sätzen anhand ihrer Hashwerte berechnet. Für jede Indexposition wird ein Wert x mit 0 definiert. Dieser wird nun für jede 1 an der entsprechenden Position der Hashwerte aller Worte inkrementiert und für jede 0 dekrementiert. Das Ergebnis wird bei $x > 0$ auf 1 gesetzt, bei $x \leq 0$ auf 0.

der	1000 0100
mann	1000 1011
steht	0111 0111
aufrecht	0001 0110
	0000 0110

die	0011 1000
frau	1100 0010
steht	0111 0111
aufrecht	0001 0110
	0001 0110

Durch die Hamming-Distanz der Bitstrings kann nun die Ähnlichkeit gemessen werden.

Diese ergibt sich aus der Anzahl der benötigten Substitutionen, um einen String in einen anderen zu wandeln. In dem Beispiel beträgt die Hamming-Distanz 1. Die beiden Sätze besitzen bei dieser Repräsentation somit eine Ähnlichkeit von 0.875.

3.3.4 Paragraph Vectors

Vielversprechend scheint auch die Methode der Dokumenteneinbettung zu sein. In „*Document Embedding with Paragraph Vectors*“ [pv2] wird die neuartige Methode der Worteinbettung benutzt, um die Bedeutung von Wikipedia-Artikeln in Vektoren zu kodieren. Das Verfahren basiert auf einer in „*Efficient estimation of word representations in vector space.*“ [w2v1] beschriebenen Technik, die Worte, mit Betrachtung des Kontextes, in dem sie vorkommen, vektorisiert. Im Gegensatz zu den erläuterten Verfahren, werden hier die Vektoren mithilfe eines flachen, neuronalen Netzes trainiert. Von Mikolov et al. werden zwei Modelle vorgestellt. Mit Continuous Bag of Words"(CBOW) wird ein neuronales Netz darauf trainiert ein gesuchtes Wort anhand seiner im Text umgebenen Worte vorherzusagen. Im Gegensatz dazu soll das Skip-gram Model mit einem Wort als Eingabe den wahrscheinlichen Kontext des Wortes ausgeben. Umgebung und Kontext beschreibt hier eine fest definierte Anzahl an vorhergehenden und folgenden Termen, auch als Window bezeichnet. Die Vektoren von bedeutungsähnlichen Termen werden durch diese Methode nah beieinander platziert.

Die darauf aufbauenden Paragraph Vectors [pv1] erweitern das Prinzip der Worteinbettung auf Sätze, Paragraphen und ganze Texte. Die Erweiterung der CBOW-Methode besteht darin, neben dem Kontext der Worte nun auch die passende Paragraph-ID als Eingabe des Vektortrainings bei jedem Term eines Textabschnittes zu nutzen. Diese Paragraph-ID dient als zusätzlicher Kontext für jeden Abschnitt und verbindet so Worte eines Abschnitts, die mit CBOW noch außerhalb des Window waren. Dieses Verfahren heißt Paragraph Vectors - Distributed Memory (PV-DM). Das Skip-gram Model wurde ebenfalls in diese Richtung erweitert, in dem das Netz auch darauf trainiert wird, aus einer Paragraph-ID alle Worte vorherzusagen (PV-DBOW - Paragraph Vectors - distributed bag of words). Die Gewichtung des Hidden-Layer der trainierten neuronalen Netze stellt bei allen Modellen den Vektor dar.

3.4 PAN Experiment

Es stellt sich nun die Frage, welche der vorgestellten Methoden sich am besten eignet, um in einem gegebenen Korpus die Textwiederverwendungen mit höchster Präzision zu finden. Für diesen Zweck bot sich der PAN14 Plagiatskorpus an. In dem Korpus befinden sich 3.385 unterschiedliche Quelldokumente mit 103 bis 11.898 Worten. Zusätzlich beinhaltet der Korpus 498 verdächtige Dokumente, bei denen teilweise aus den Quelldokumenten plagiiert wurde. Die Anzahl der Plagiatsdokumente beträgt 179. Des Weiteren sind die wiederverwendeten Textstellen zum Teil verschleiert und ein verdächtiges Dokument kann aus mehreren Quellen kopiert worden sein. Der Datensatz wurde zum einen maschinell erstellt, zum anderen per Hand mittels Amazon-Mechanical-Turk. Für die Berechnung von Precision und Recall liegen alle Plagiatspaare vor.

Für die Implementation wurde Python verwendet. Als Framework diente Gensim [gen1], das die hier diskutierten Methoden des Vector-Space-Model, Tf-Idf, LSI sowie Paragraph Vectors als API bereitstellt. Das Simhashing musste selbst implementiert werden.

3.4.1 Programmablauf

Im ersten Experiment wurden alle Quell- und Plagiatsdokumente eingelesen, tokenisiert und die Stoppworte entfernt. Zur Bildung der Simhash-Werte wurden 3-gram Worttoken gebildet, welche drei aufeinanderfolgenden Worten in einem Satz entsprechen. Diese wurden anschließend mit MD5 gehasht. Die Hashwerte wurden dann nach dem Simhash-Prinzip auf Bit-Ebene an den entsprechenden Positionen addiert, um einen 64 Bit Fingerprint zu erstellen. In einem früheren Experiment wurden 3-Gramme auf Zeichenebene gebildet, angelehnt an das von Jimmy Lin et al. durchgeführte Hashing der Wikipedia auf Satzebene, was auf Dokumentebene allerdings zu unbrauchbaren Ergebnissen führte.

Zur Analyse der Vektormodelle wurde eine TD-Matrix aus allen Dokumenten erstellt. Diese hatte eine Spaltenlänge von 140.000, entsprechend dem generieten Wörterbuch und wurde nun mittels Tf-Idf neu gewichtet. Um den Qualitätsunterschied von VSM beziehungsweise Tf-Idf-Matritzen und ihren LSI Pendants festzustellen, wurden diese mittels SVD auf 200 Dimensionen approximiert. Als letztes wurden die auf Worteinbettung basierten Paragraphvektoren mit 100 Dimensionen aus den Quelldokumenten trainiert. Auch wenn Quoc V. Le et al. mit der Distrubuted Memory Methode bessere Ergebnisse erzielten, zeigten Paragraph Vectors mit DBOW auf dem PAN14 Korpus bessere Resultate und sind hier zum Vergleich genutzt wurden.

Im zweiten Schritt erfolgte die Abbildung der Plagiatsdokumente auf dieselbe Weise. Für jedes dieser Query-Dokumente konnten nun die Ähnlichsten ermittelt werden, indem für die Vektormodelle die Kosinus-Ähnlichkeit und für die Simhash-Werte der Hamming-Abstand ermittelt wurde. Die Similaritäten für alle Plagiatsdokumente wurden absteigend, geordnet gespeichert. Für die Berechnung des Recalls konnte nun die berechnete

Liste der ähnlichen Dokumente mit der bekannten Liste aller Textwiederverwendungen verglichen werden.

Anhand dessen kann jetzt der Recall von Plagiatsfällen in Bezug auf die Anzahl der ermittelten Quelldokumente gezeigt werden.

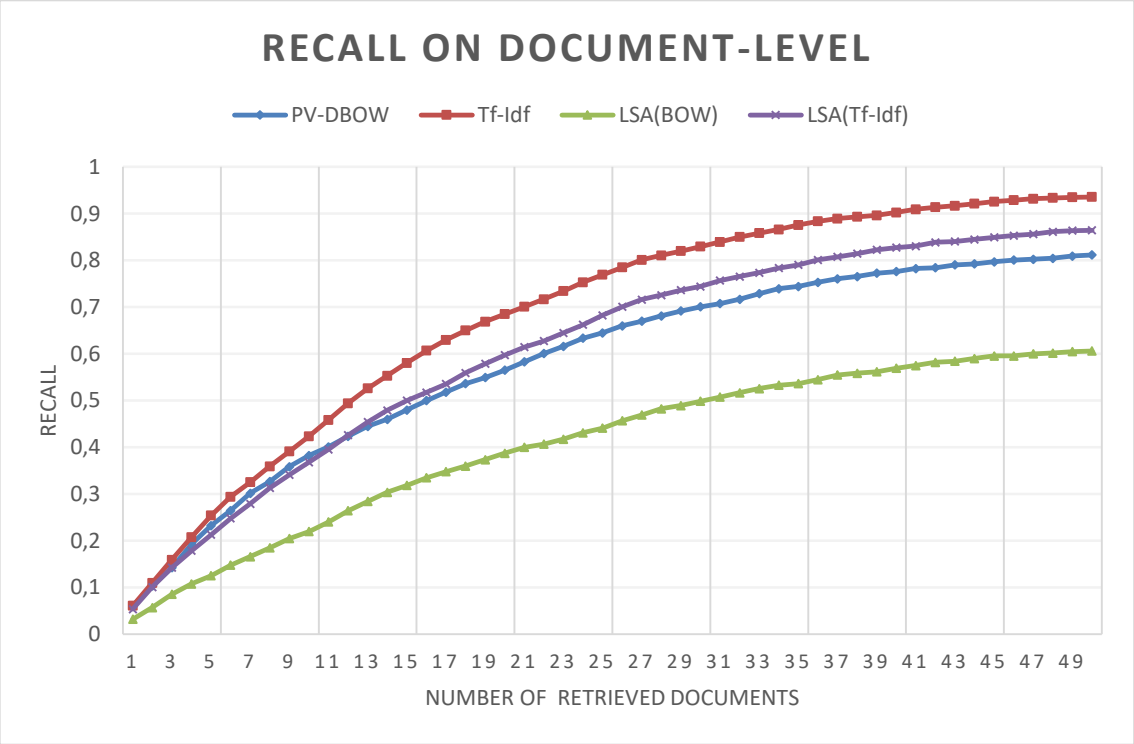


Abbildung 3.4.1: Recall von Plagiaten des PAN14-Korpus auf Dokumentenebene

Das Diagramm zeigt den Recall der Retrieval-Methoden im Verhältnis zur Anzahl der ermittelten Dokumente. Wie zu sehen ist, wird mit Tf-Idf Vektoren das beste Ergebnis erzielt. Es ist anzumerken, dass je nach Query-Dokument bis zu 16 Quelldokumente verwendet wurden. Daher können frühestens ab 16 ermittelten Dokumenten alle Quellen gefunden werden. Bei dieser Marke erreicht Tf-Idf bereits 60,69% und steigert sich bei 50 Dokumenten auf 93,6%. Der Qualitätsverlust durch Dimensionsreduktion mittels LSA und Singulärwertzerlegung zeigt sich deutlich. Bei 16 Dokumenten ist der Recall hier 16,6% niedriger als mit Tf-Idf und 9% nach 50 Dokumenten. Paragraph Vectors auf Dokumentenlevel besitzen ein niedrigeres Recall Niveau als LSA mit Gewichtungen. Etwas abgeschlagen ist LSA, basierend auf dem einfachen Bag-of-Words Modell. Die Simhash Ergebnisse waren so niedrig, dass sie aus der Übersicht entfernt wurden.

3.4.2 Dokumententeilung

Anhand von empirischen Plagiatsbeispielen lässt sich zeigen, dass in seltenen Fällen ganze Artikel als Quelle in einem Plagiatsdokument auftauchen. Einzelne Paragraphen oder nur Abschnitte von Dokumenten werden entweder als Duplikat übernommen oder in abgewandelter Form plagiiert. Durch Vergleiche auf Dokumentenebene, besonders bei langen Dokumenten, gibt die gewonnene Kosinus-Ähnlichkeit zwischen den Dokumenten einen fehlerbehafteten Hinweis auf Plagiatsfälle. Zwischen zwei Artikeln mit jeweils zehn Paragraphen, bei denen lediglich ein Paragraph ein Duplikat darstellt, beträgt die Cosine-Similarity, je nach Ähnlichkeit der anderen Paragraphen, mindestens 0.32. Wenn nun jedoch jeder einzelne Paragraph der beiden Dokumente direkt verglichen wird und lediglich die höchste gefundene Similarität als Hinweis auf Wiederverwendung zwischen den Dokumenten gespeichert wird, lässt sich der Recall mit den Kosten des höheren Aufwands vergrößern.

Im Folgenden wurden die Retrieval-Methoden auf Paragraphenebene untersucht. Jedes Quelldokument und verdächtige Dokument wurde in Paragraphen unterteilt. In praktischen Anwendungsgebieten lässt sich die Textunterteilung anhand von Absätzen oder durch Überschriften realisieren. In diesem Experiment wurden die Texte in Passagen von maximal 400 Termen unterteilt. Wie beschrieben, wurden die Ähnlichkeiten zwischen Dokumenten auf Basis ihrer ähnlichsten Paragraphen gespeichert.

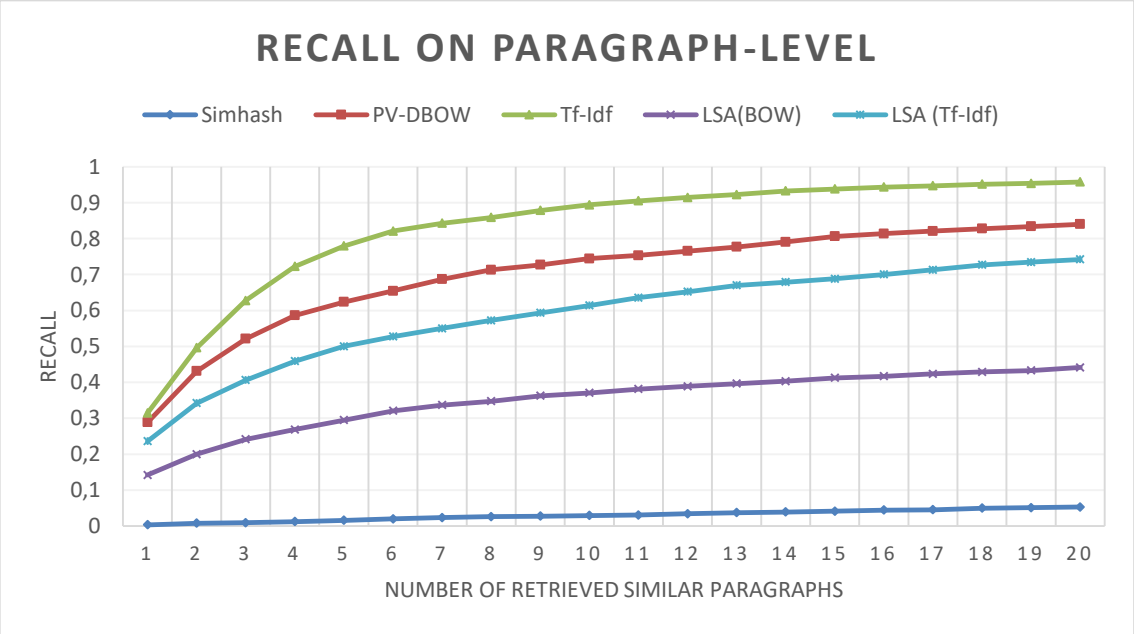


Abbildung 3.4.1: Recall von Plagiaten des PAN14-Korpus auf Paragrapheneben

In Abbildung 3.4.1 ist zu sehen, wie sich die Retrieval-Methoden bei der Suche nach Textwiederverwendung auf Paragraphenebene eignen. Die 3385 Quelldokumente wurden in 7080 Paragraphen geteilt. Alle 179 verdächtigen Texte besitzen 1080 Paragraphen. Das bedeutet, dass jedes Query-Dokument im Durchschnitt in sechs Paragraphen geteilt wurde und daher durchschnittlich sechs Quellkandidaten ermittelt wurden.

Wie schon auf Dokumentenebene werden durch das Tf-Idf Vektormodell die richtigen Quelldokumente am schnellsten erkannt. Allerdings liefern die Wordembeddings auf Paragraphenebene hier bessere Ergebnisse als die beiden LSA Varianten. Durch Simhash konnten wieder keine sinnvollen Kandidaten gefunden werden.

Leider lassen sich die beiden Experimente schlecht zusammenführen, da die Anzahl der Plagiate und die Dokumentlängen sehr stark variieren.

3.4.3 Diskussion

Wie zu sehen ist, liefern Tf-Idf Vektoren bei der Suche nach Textwiederverwendung die besten Ergebnisse. Dazu muss man anmerken, dass die Größe dieser Vektoren nicht reduziert wurde. Die Vektoren der Paragraphen auf Basis von Worteinbettungen zeigten auf Paragraphebene bessere Ergebnisse, was daran liegen kann, dass der Vektorraum von 100 Dimensionen zu niedrig gewählt war um große Texte adäquat abzubilden. In früheren Experimenten zeigten sich allerdings mit dieser Vektorart an dem PAN14 Korpus mit 200 Dimensionen schlechtere Ergebnisse als mit 100 Dimensionen. Aufgrund der geringen Größe des Korpus kann auch die Qualität des Vektortrainings leiden, weswegen mit einer größeren Eingabemenge und höher gewähltem Vektorraum bessere Resultate zu erwarten sind. Auch wenn Simhash prädestiniert für die schnelle Duplikatssuche in großen Datenmengen ist, eignet es sich offensichtlich nicht für die Suche nach Textwiederverwendung, wenn diese nur einen kleinen Teil eines Dokumentes ausmachen. Anhand des Kurvenverlaufs ist zu sehen, dass der Recall mit Anzahl der ermittelten Dokumente nur noch marginal steigt. Aufgrund dessen kann man davon ausgehen, dass durch den Kosinus der LSA-Vektoren ein Teil der Plagiate nicht ermittelt werden kann.

4 Implementierung der Similaritätssuche

Aufgrund der Resultate aus Kapitel 2 sollen nun Tf-Idf gewichtete Vektoren genutzt werden, um Plagiatskandidaten innerhalb der Wikipedia zu finden. Zu diesem Zweck, stand der BetaWeb-Cluster der Bauhaus Universität Weimar zur Verfügung.

In "*Visualizing Article Similarities in Wikipedia*" wurde von J.Kiesel bereits die Wikipedia nach dem Tf-Idf gewichteten Vektormodell berechnet. In dem Experiment wurden die Dokumente durch Stemming und Stoppwort-Entfernung vorbereitet und anschließend mit der Kosinus-Ähnlichkeit verglichen. Um den Rechenaufwand und damit einhergehend den Datensatz zu reduzieren (pruning), wurden alle Artikel mit weniger als 50 Worten ignoriert und alle Seiten ohne Informationsgehalt wie User-Pages, Talk-Pages und Ähnliche entfernt. Des Weiteren wurden die Vergleiche anhand der Artikellänge reduziert. Lange Artikel wurden mit kurzen verglichen, aber kürzere Artikel, wegen ihrer geringeren Relevanz, nicht untereinander. Die Berechnung der 65 Milliarden paarweisen Vergleiche erfolgte auf 4 NVidia GTX 480 Grafikkarten mittels CUDA und dauerte 22 Tage.

Im Gegensatz zu der in dem Paper beschriebenen Vorgehensweise, sollen hier die Vergleiche auf Paragraphenebene durchgeführt werden. Da in dieser Arbeit das Hauptaugenmerk auf Textwiederverwendung und nicht auf der Klassifizierung oder Modellierung von Bedeutungsräumen liegt, soll im Preprocessing auf das Stemming verzichtet werden. Die Paragraphenlänge ist außerdem unabhängig von der Relevanz für Textwiederverwendung. Passagen kurzer Artikel sowie langer Texte können als Quelle für Plagiate untereinander dienen. Dadurch lässt sich die Vergleichsmenge schlecht reduzieren, ohne etwaige Textwiederverwendung zu übersehen.

4.1 Wikipedia Datensatz

Im Folgenden wurde der englische Wikipedia Artikelsatz vom 1.5.2016 benutzt. Dieser Datensatz liegt als XML-Datei vor und beinhaltet alle Seiten aller Namensräume [wns], die zu diesem Zeitpunkt vorhanden waren. Um den Fließtext aller Artikel aus dem Dump zu bekommen, wurde ein Wikipedia-Text-Extractor [wte] von Guiseppe Attardi et al. verwendet. Dieses Python-Skript besitzt die Möglichkeit der Wikipedia-Template Erweiterung. Als Templates werden in der Wikipedia Satz- und Textpassagenvorlagen bezeichnet, die allen Artikeln ähnlicher Themen bereitstehen. Diese können themenspezifisch mit Schlüsselworten oder Datenwerten ergänzt und in Artikeln verwendet werden. Es ist diskutabel, Templates bei der Suche nach Textwiederverwendung einzubeziehen, da diese ohnehin schon zu den Near-Duplicates gehören. Allerdings ist es meiner Meinung nach unablässig diese einzubeziehen, da so die gleichen Texte verarbeitet werden, die ein

Nutzer im Browser auf Wikipedia zu sehen bekommt. Falls der Datensatz später noch mit anderen Textquellen verglichen werden sollte, liefert er mit Template-Erweiterung ein vollständiges Bild der Wikipedia.

Das Skript wurde dann erweitert, um einzelne Artikel anhand der vorhandenen Überschriften in Paragraphen zu teilen. Zusätzlich wurden alle Namensräume bis auf den Main-Namespace gefiltert, um ausschließlich die Artikel zu erhalten. Alle Artikel wurden mit Hilfe des Natural Language ToolKit [ntl] tokenisiert und die Stoppworte entfernt. Des Weiteren wurde der Datensatz auf Basis der Artikellänge extrahiert. Hierzu wurden alle Paragraphen mit einer Länge von weniger als 50 Worten mit dem vorherigen zusammengeführt und alle Artikel mit weniger als 100 Worten gänzlich ausgelassen. Die Filterung anhand der Textlängen geschah vor dem Preprocessing. Die Seiten mit Disambiguierungen, also Seiten zur Aufschlüsselung von mehrdeutigen Begriffen, wurden ebenfalls entfernt. Die Ausgabe wurde nun in ein neues XML-Format überführt, in dem alle Artikel mit Titel, Wikipedia-ID und einer Liste aller Paragraphen gespeichert wurden.

Tabelle 4.1.1: Übersicht der gefilterten Wikipedia Artikel

Artikel in Namespace 0	5.139.351
Artikel gefiltert durch Länge	2.400.125
Artikel mit Disambiguierung	118.468
Verbleibende Artikel	2.620.758
Anzahl der Paragraphen	8.507.799

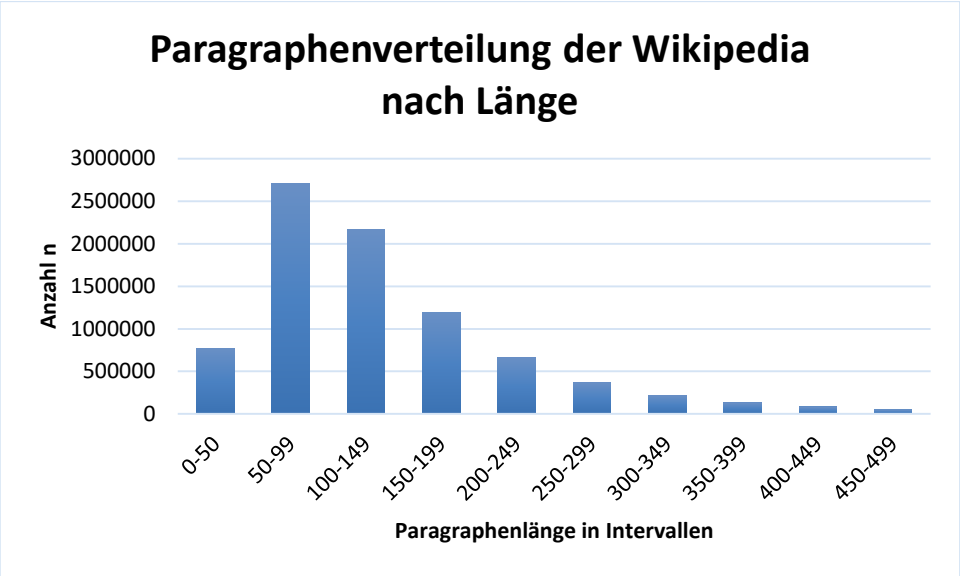


Abbildung 4.1.1: Verteilung der verarbeiteten Wikipedia-Paragraphen anhand ihrer Länge

Wie in dem Diagramm 4.1.1 zu sehen ist, besitzt der überwiegende Teil der Paragraphen eine Länge von bis zu 250 Worten. Diese 7.500.937 Paragraphen entsprechen 88,17% des Datensatzes. Es ist anzumerken, dass 128.252 Paragraphen mit 500 - 999 Token sowie 15.139 Paragraphen mit über 1000 Worten in dem Diagramm nicht bedacht wurden.

Ein Kreuzvergleich dieser 8.507.799 Paragraphen resultiert in 36,19 Billionen Vergleichsoperationen.

4.2 Cluster Computing

Für die weitere Arbeit wurde der BetaWeb-Cluster der Bauhaus Universität Weimar genutzt. Dieser Rechnerverbund besteht aus 130 Nodes mit jeweils zwei Intel Xeon E5-2620v2, also 12 CPU-Kernen respektive 24 Threads durch Hyperthreading und 64GB RAM. Als Framework diente Apache Spark, welches APIs in Python, Java und Scala bereitstellt. Dieses beinhaltet die Machine Learning Library, MLlib, eine Bibliothek mit verschiedenen Algorithmen des maschinellen Lernens sowie Feature-Extraktionsmethoden, wie der benötigten Vektorisierung auf Basis von Tf-Idf Gewichten. Im Gegensatz zu der beschriebenen Generierung der Vektoren durch ein Wörterbuch, werden jene durch Feature-Hashing erstellt. Die Vektorisierung resultierte in den 8.507.799 Sparse-Vektoren, welche den beschriebenen Paragraphen der Wikipedia entsprechen.

4.2.1 Apache Spark

Apache Spark ist ein Framework für Cluster-Computing. Im Gegensatz zu herkömmlichen MapReduce-Applikationen wie Apache Hadoop wurde Spark entwickelt, um unnötige Geschwindigkeitsengpässe zu vermeiden [sp1]. Eingabedaten müssen mit Spark nur einmal gelesen werden, da sie danach verteilt im Speicher residieren können, wohingegen Hadoop MapReduce nach Map- oder Reduce-Aufgaben immer auf persistente Speicher schreibt. Des Weiteren stellt Spark, besonders mit den Python und Scala APIs, Schnittstellen bereit, die den Programmieraufwand aufgrund von hoher Abstraktion der zugrundeliegenden Architektur sehr verkürzen und vereinfachen.

Die Architektur des Spark Clusters besteht aus einem Driver Node und mehreren Worker Nodes. Auf dem Driver Node läuft die eigentliche Spark-Applikation. Das dort kreierte SparkContext Objekt stellt die Verbindung zum Spark Cluster bereit. Die durch den Quellcode beschriebene Verteilung der Daten und Ausführung der Prozesse wird durch die API in Tasks gekapselt und auf die Worker verteilt. Die Ausgabe der Applikation wird entweder verteilt gespeichert oder auf den Driver zurückgegeben. Die Ausführung der Tasks auf den Worker geschieht in definierten Executorprozessen. Die maximale Anzahl an Executorprozessen ist abhängig von der Anzahl der Kerne der Worker Nodes. Der maximale Grad von Parallelität ergibt sich so durch die Anzahl der logischen Prozessorkerne aller Worker Nodes und beschreibt die Anzahl der gleichzeitig ausgeführten Tasks.

Da die API auf Java basiert, wird für jeden Executor eine eigene Java Virtual Machine gestartet. Die Konfiguration der JVM, wie die maximale Heapgröße oder der zu verwendende Garbage Collector, kann durch die Spark Konfiguration festgelegt werden und sollte je nach Anwendung angepasst werden. Die Geschwindigkeit einer Applikation kann je nach Architektur stark von der gewählten Größe des Speichers und der Anzahl der Kerne pro Executor abhängen. Abbildung 4.3.1 zeigt eine Skizze der Spark Architektur basierend auf YARN, wie sie auf dem Betaweb-Cluster installiert ist.

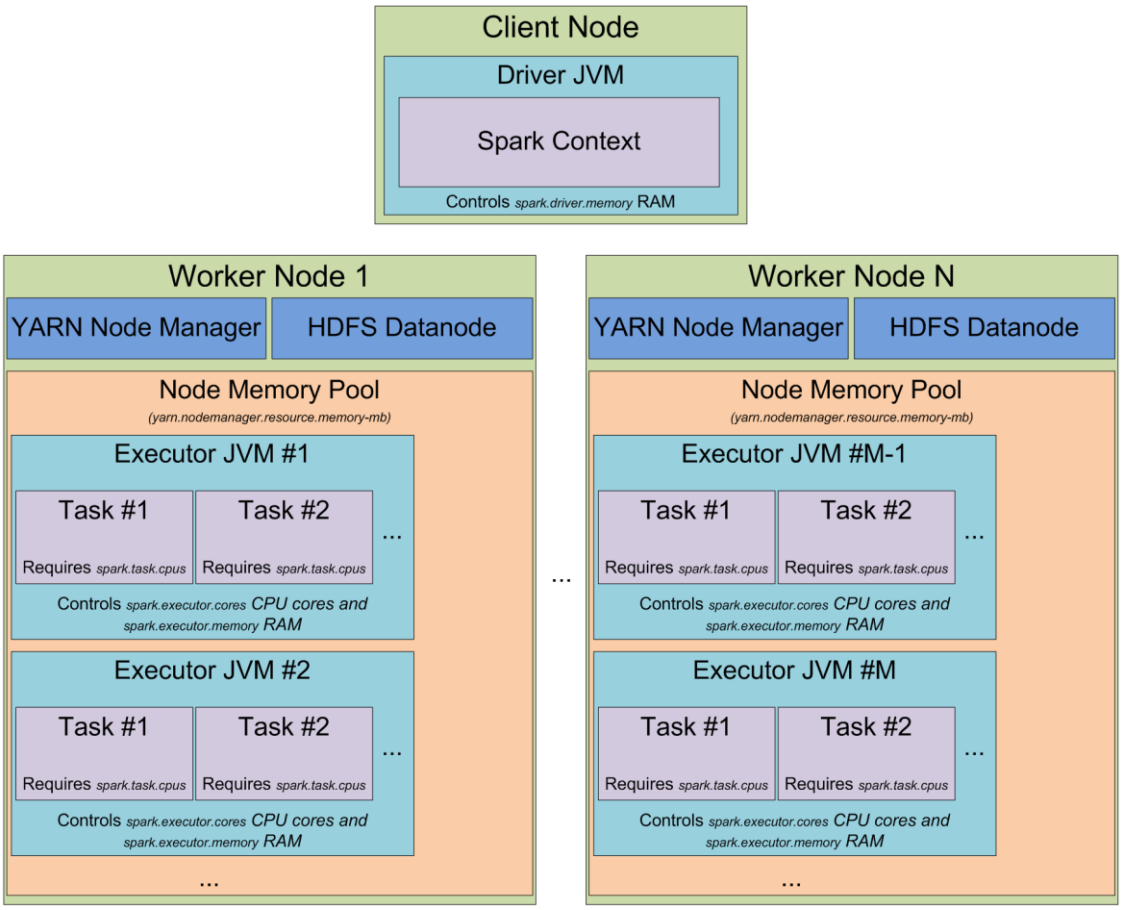


Abbildung 4.2.1: Spark Architektur basierend auf YARN, Quelle: Alexey Grishchenko[1]

Das Datenmodell basiert auf Resilient Distributed Datasets (RDD), die als verteilte Liste von primitiven Datentypen oder Objekten verstanden werden kann. Diese stellen ein Interface für Transformationen und Aktionen auf den zugrundeliegenden Daten bereit. Ein Datensatz wird bei der Erstellung des RDD in Partitionen geteilt, die jeweils einen Teil der Daten umfassen und auf den einzelnen Executor im Heap gespeichert werden. RDD werden „lazy-evaluated“, das bedeutet, dass Transformationen auf den Daten wie Filter- oder Map-Operationen nicht direkt durchgeführt werden. Erst wenn eine Aktion, die die Verarbeitung des gesamten RDD benötigt, durchgeführt wird, werden alle Transformationen der Reihe nach ausgeführt. Beispiele für solche Aktionen sind das Zählen der Elemente oder das Speichern des RDD. So beschreibt ein RDD bis zu einer Aktion lediglich die Verarbeitungsschritte der Daten, wodurch diese dann von Spark bei der Durchführung optimiert werden können.

[1] Alexey Grishchenko, Spark Architecture (14.3.2015),
URL: https://0x0fff.com/spark-architecture/ (Stand: 5.1.2017)

Im Grunde können alle Transformationen als Map-, Filter- oder Reduceprozess beschrieben werden oder als Kombination aus diesen. Durch Map wird jedes Element durch eine Funktion auf ein anderes abgebildet, Filter reduziert die Elementmenge anhand eine booleschen Funktion und Reduce fasst Elemente anhand von Schlüsselwerten zusammen. Im Vergleich zu iterativen oder rekursiven Prozessen muss der Programmablauf daher parallel geplant werden. Auch wenn die API Schleifen wie foreach bereitstellt, wird jeder Schritt intern als Map-, Filter- oder Reduceprozess durchgeführt.

4.2.2 Methoden zur Berechnung der Plagiatskandidaten

Die Suche nach ähnlichen Elementen und damit einhergehend ähnlichen Vektoren ist ein weitverbreitetes Problem in der Informatik, sei es Bilderkennung, Klassifizierung, Genomanalyse in der Bioinformatik oder Analyse von Nutzerverhalten in Empfehlungssystemen. Die naive Herangehensweise besteht in der linearen Suche, bei der ein Element mit jedem anderen verglichen wird und die nächsten Nachbarn gespeichert werden. Die Suche resultiert aber in einer Komplexität von $O(n)$. Im eindimensionalen Raum kann das Problem mithilfe von Binärbäumen und Reichweitensuche auf $O(logn)$ reduziert werden. Auch in zwei und mehr Dimensionen existieren mit k-d-Bäumen ($O(\sqrt{n})$) und Rangetrees durch *fractional cascading* ($O(logn + k)$) effektive Suchmethoden. Sobald die Anzahl der Dimensionen auf über acht steigt, verlieren diese k-Nearest-Neighbour Algorithmen (k-NN) ihren Geschwindigkeitsvorteil gegenüber linearer Suche [knnc]. In einem Datensatz wie dem der vektorisierten Wikipedia mit über 200.000 Dimensionen und selbst mit einem durch SVD reduzierten Raum von 300 Dimensionen bringt diese Art von Algorithmen keinen Vorteil. Eine andere Methode ist das Hashing der Vektoren mit LSH-Verfahren, das aufgrund der ungenügenden Ergebnisse in Kapitel 2 nicht weiter betrachtet wurde.

Approximate Nearest Neighbor Search ist eine weitere Problemklasse mit dem Unterschied, die Suchergebnisse zu approximieren und nicht genau zu berechnen. Zwangsläufig tausch man dadurch die Präzision der Suche gegen Geschwindigkeit. Radim Rehurek, Entwickler von Gensim, hat verschiedene ANN-Algorithmen wie ANNOY [annoy] von Spotify und Flann [flann] im Hinblick auf Genauigkeit und Anfragezeit getestet (Tabelle 4.2.2) [annb].

Tabelle 4.2.2: Vergleich von ANN-Algorithmen, Quelle: Eigene Darstellung in Anlehnung an: Performance Shootout of Nearest Neighbours[1]

Library	Parameter	N	Precision	Query time in ms	Index size in GB
openblas		N	1	678.9	6.6
annoy	100	1000	0.85	64.8	11
annoy	500	1000	0.99	287.7	29
flann	0.9	1000	0.33	1.6	10
flann	0.99	1000	0.37	1.6	10

In dem Experiment wurde die Wikipedia mit LSI und 3.5 Millionen Dokumenten auf 500 Dimensionen reduziert. Die Tabelle beschreibt die Anfragezeit für ein Anfragedokument und die Präzision der erhaltenden Ergebnisse in Hinblick auf die Methodenbedingten Parameter und den geforderten Speicherbedarf. Relevant ist die Präzision der Top 1000 Query-Ergebnisse, weswegen nur diese aus der Quelle übernommen wurden. Wie zu sehen ist, liefert ANNOY bei den ähnlichsten 1000 Query-Ergebnissen und einem Indexparameter von 500 eine Präzision von 0.99. Dies stellt ein hinreichend gutes Ergebnis dar und verbessert die Suchanfragezeit um das 2.4-fache im Vergleich zu Exhaustive-Search. FLANN bietet zwar sehr gute Anfragezeiten, allerdings ist die Präzision ungenügend.

4.2.3 Kosinussimilarität mit Spark

Die folgenden Experimente wurden mit Pyspark auf dem Betaweb-Cluster mit 100 Worker, 400 Executor mit jeweils 7 GB RAM und 5 Executor-Cores durchgeführt.

Gegeben waren 8.507.799 spärlich besetzte Vektoren mit 262.144 Dimensionen, deren "*all-pair similarity*" zu berechnen war. DIMSUM, Dimension Independent Matrix Square using MapReduce, ein von Twitter entwickelter Algorithmus, implementiert in der Machine Learning Library von Spark, soll den Rechenaufwand dessen um bis zu 40% reduzieren. Allerdings tritt dieser Fall nur bei Matrizen mit $m \gg n$ ein und wurde mit dichten Vektoren als Eingabe entwickelt. Ein Test der Bibliothek schlug allerdings fehl, da keinerlei Fortschritt des Programmablaufs nach 48 Stunden zu erkennen war.

[1] Performance Shootout of Nearest Neighbours: Querying, Radim Rehurek (12.01.2014), URL:https://rare-technologies.com/performance-shootout-of-nearest-neighbours-querying/ (Stand:10.01.2017)

In dem Whitepaper [dimsum] wird von $m = 2^{13}$ und $n = 2^4$ als Matrixbeispiel gesprochen und setzt voraus, dass die SVD von $n \times n$ Matrizen auf einen einzelnen Node durchgeführt werden kann. Dies ist mit den gegebenen $n = 2^{18}$ des Wikipediadatensatzes auf dem Betaweb-Cluster nicht möglich, da der Speicheraufwand pro Node allein für die $n \times n$ Matrix 68.7 GB beträgt und erklärt den Abbruch des Experiments.

Als nächstes wurde der Brute-Force Ansatz gewählt. Die Sparse-Vektoren wurden neu indiziert und mit ihrer Norm, Wikipedia-ID und Artikeltitel als RDD von Tupeln gespeichert. Auch wenn in der Literatur für Kreuzvergleiche die Bildung des kartesischen Produktes der Objektmenge empfohlen wird, ist dies bei der Datengröße unmöglich. Als Lösung wurde der Datensatz als RDD R_p anhand der Modulo-Restklassen des fortlaufenden Index auf Basis einer Primzahl gleichmäßig auf allen Executor verteilt $R_p = \{R_{p1}, R_{p2} \dots R_{p2}\}$. Jeder Executor besitzt nun eine RDD-Partition R_{px} der Daten. Die gleichen Daten wurden durch eine zweite Primzahl erneut in einzelne Mengen $R_q = \{R_{q1}, R_{q2} \dots R_{q3}\}$ geteilt. Nun ist es möglich iterativ alle Teilmengen von R_q durch Spark Broadcastvariablen jeden Worker zugänglich zu machen. Dies ermöglicht die sequentielle Berechnung aller Mengenpaare $\{(R_{p1}R_{q1}), (R_{p2}R_{q1}) \dots (R_{pn}R_{q1}) \dots (R_{pn}R_{qn})\}$ gleichmäßig verteilt auf allen Nodes. Die eigentliche Errechnung der Kosinussimilarität von zwei Vektormengen konnte nun mit der mapPartitons Funktion, welche auf jedem Executor mit einem anderen Teil der RDD-Partition durchgeführt wurde, realisiert werden. Hierzu wurden die MLLib-Funktionen für lineare Algebra genutzt.

In einem Testlauf mit 1% der Daten wurden alle Similaritäten in 7,7 Stunden errechnet, was circa 32 Tage für alle Vergleiche ergeben würde.

4.2.4 Optimierung der Implementierung

Um sicher zu stellen, dass die Ausführung der mathematischen Operationen von Pyspark, besonders die des Kreuzproduktes der Sparse-Vektoren, nicht mit langsamen Python-Funktionen geschieht, wurde die MLLib näher betrachtet. Es stellt sich heraus, dass bei Verwendung der MLLib-Typen für Vektoren im Hintergrund keine Scala oder Java Objekte verwendet werden, sondern Numpy Objekte. Diese sind aufgrund ihrer Implementierung als C-Array sehr effizient. Die Methoden sind allerdings alle in Python geschrieben, was die Ausführung ungemein verlangsamt. Im Hinblick auf die Erweiterbarkeit von Python-Skripten durch C-Methoden wurde überprüft, wie sich diese in Pyspark umsetzen lassen. Es zeigt sich, dass Cython-Code dynamisch im IPython-Notebook eingebettet werden kann [ccl], sowie als kompilierte C-Extensions, ähnlich wie in Numpy und Scipy, in Pyspark genutzt werden können [cyk]. Die erste Idee war die Nutzung einer effizienten

BLAS Library (Basic Linear Algebra Subprograms), wie die Open Source Variante ATLAS oder die INTEL Math Kernel Library. Diese sind hochoptimierte, teils in Assembler geschriebene Bibliotheken für Vektor- und Matrixoperationen. Laut Spezifikation [blas] unterstützen diese leider keine DOT-Produkte zwischen zwei Sparse-Vektoren. Mit einem Datensatz von Dense-Vektoren durch SVD oder Word-embeddings sollten diese mit hoher Wahrscheinlichkeit die schnellste Möglichkeit für „all-pair similarity“ Aufgaben darstellen.

Nach einem kurzen Test mit einer selbst geschriebenen Similaritätsfunktion in Cython zeigte sich, dass die Geschwindigkeit der Berechnung um Faktor 10 gesteigert werden konnte. Für die Umsetzung auf dem Cluster wurde zum einen die mapPartitions Funktion, welche auf jedem Executor ausgeführt wird und zum anderen die Kosinusfunktion mit Cython kompiliert. Dieses Modul wurde bei Programmstart allen Nodes zugänglich gemacht. Da die Sparse-Vektoren der MLlib intern als zwei dichte Numpy-Arrays auf C-Array Basis realisiert wurden, konnten diese einfach an das Cython-Modul übergeben werden. Dies hat zur Folge, dass alle abhängigen Datentypen der Ähnlichkeitsfunktion in C deklariert sind und alle Funktionen als reiner C-Code kompiliert werden konnten, ohne auf Python Objekte oder Funktionen angewiesen zu sein. Aufgrund dieser Erweiterung konnten alle Wikipedia-Vektoren innerhalb von vier Tagen und 13 Stunden verglichen werden, was ausreichend schnell war, um von weiteren Optimierungen abzusehen.

Die Ausgabe wurde in das CSV-Dateiformat überführt. Jede Zeile umfasst die Ähnlichkeiten für ein Dokument. Der erste Wert ist die Wikipedia ID des Dokumentes, gefolgt von Paaren aus ID und Kosinus aller gefundenen ähnlichen Dokumente. Es ist zu beachten, dass bei Artikelpaaren (a, b) die Similarität lediglich einmal gespeichert wurde und in dem Fall in den CSV-Dateien in der Zeile für Dokument a eine Relation zu b festgehalten ist, jedoch nicht in Dokumentzeile b die Ähnlichkeit zu a.

4.2.5 Ergebnisse

Kosinus Ähnlichkeit	**Anzahl der Dokumentenpaare**
[0.9,1.0]	1.440.388
[0.8,0.9)	9.983.895
[0.7,0.8)	160.716.484
[0.6,0.7)	273.926.278
[0.5,0.6)	287.084.295
[0.4,0.5)	307.513.389
[0.3,0.4)	627.619.558
[0.2,0.3)	3.060.431.052
[0.1,0.2)	31.624.808.958

Tabelle 4.2.5: Ähnlichkeiten aller Wikipedia Artikelpaare anhand ihrer ähnlichsten Paragraphen

Die oben dargestellte Tabelle zeigt die Verteilung aller Similaritäten der 2.620.758 Artikel anhand ihrer ähnlichsten Paragraphen untereinander. Die Anzahl aller Dokumentenpaare beträgt $3{,}4 * 10^{12}$. Durch die Filterung aller nicht relevanten Paaren mit einem Kosinus von unter 0.1 reduzieren sich diese auf circa $36{,}3 * 10^{9}$. Um diese Zahlen zu errechnen, wurden $36{,}2 * 10^{12}$ Vergleichsoperationen durchgeführt. Die Ausgabedaten haben eine Größe von 750 GB.

5 Textalignment

Das Textalignment beschreibt den Vergleich von zwei Texten auf Wiederverwendung. Hierbei werden Teile der Texte, die einen Schwellwert von Gleichheit überschreiten, markiert und gespeichert. Für den Textabgleich anhand der Kandidatenpaare wurde das Textalignment von PicaPica [pica] verwendet. Dieses Projekt der Bauhaus Universität Weimar hat das Ziel, Texte untereinander oder einzelne Texte mit größeren Textkorpora zu vergleichen. In PicaPica wurde schon viel Forschungsarbeit investiert, weswegen eine Evaluation der verwendeten Methoden hier nicht in Frage kommt. Im Folgenden wird die Arbeitsweise der Parallelisierung des Alignments durch die Spark Java-API beschrieben.

5.1 Picapica Textalignment

Die Arbeitsweise des Algorithmus wurde hier nicht im Detail betrachtet. Es werden zwei Eingabetexte in Token unterteilt und Wort n-Gramme gebildet. Durch DBScan werden Häufungen gleicher Textbausteine der Texte als Punkte potentieller Wiederverwendung gespeichert. Diese Kandidaten werden in einem Nachbearbeitungsschritt durch Parameter für die übereinstimmende Zeichenlänge und die prozentuale Übereinstimmung gefiltert. Im weiteren Verlauf wurde die Java-Klasse des Textalignment als Blackbox genutzt, um durch die Eingabe der Texte und der Parameter die ähnlichen Textpassagen zu finden.

5.2 Parallelisierung des Alignments

Durch die Parallelisierung des Textalignment sollen alle Plagiate anhand der Kandidatenpaare in einem Zeitraum von wenigen Tagen gefunden werden. Der Aufwand des Alignments hat sich durch die erhaltenen Plagiatskandidaten mit Ähnlichkeiten von < 0.5 um eine Größenordnung von 10^4 Operationen reduziert. Dennoch lässt sich das PicaPica Textalignment bei den verbleibenden $7{,}33 * 10^8$ Vergleichen nicht ohne Parallelisierung in der geforderten Zeit berechnen. Im Zuge dessen wurde die Java-Klasse des Textalignmnet durch die Spark Java-API verteilt ausgeführt.

Für jeden Wikipedia-Artikel lag eine Liste aller anderen Artikel vor, bei denen mindestens ein Paragraph beider Dokumente eine Kosinus-Ähnlichkeit von mehr als 0,1 besitzt. Die Größe der Eingabedaten beträgt 750 GB. Der zweite benötigte Datensatz ist der Klartext aller Wikipedia-Artikel. Als Vorbereitung wurde der Wikipedia-Artikelkorpus erneut vorverarbeitet. Da für die eigentlichen Textvergleiche die ganzen Artikel vonnöten sind, wurde kein Stemming durchgeführt und die Stoppworte blieben erhalten. Allerdings wurden Sonderzeichen durch reguläre Ausdrücke entfernt.

Die Paragraphen wurden zwar getrennt als XML gespeichert, allerdings innerhalb der Spark Applikation wieder zusammengeführt. Die Idee des Programmablaufs war es, auf jedem Executor die gesamte Wikipedia als Hashmap für schnelleren Zugriff im Speicher zu halten. Die Similaritäten sollten als RDD verteilt und dadurch partiell auf jedem Executor zur Verfügung stehen. Jeder Executor hatte nun die Aufgabe einen Teil der Ähnlichkeiten anhand der lokal vorhanden Wikipedia Kopie mit dem PicaPica Textalignment zu verarbeiten. Das Ergebnis dieser Operation ist eine Liste für jedes Dokument, in dem die gefundenen Textwiederverwendungen im Hinblick auf alle anderen Dokumente gespeichert wurden.

In der Praxis führte die maximal verfügbare Speichergröße von ~29GB pro Executor zu Problemen. Um die Wikipedia als Hashmap platzsparender zu speichern, wurden die Texte nicht als String sondern als komprimierter Bytecode gespeichert. Die Dekompressionszeit der Texte spielte im Hinblick auf die Ausführungszeit des Textalignments keine signifikante Rolle. Es zeigte sich, dass der Speicher unter diesen Voraussetzungen ausreichte.

Um den Ausgabedatensatz für dieses erste Experiment überschaubar zu halten, wurde der Postprocessor des Textalignments so konfiguriert, dass lediglich Wiederverwendungsfälle mit einer Ähnlichkeit von 0.7 und mehr zu berücksichtigen sind. Dadurch sollten neben Duplikaten auch im gewissen Maße Paraphrasen erfasst werden. Die Mindestlänge der ähnlichen Textpassagen wurde auf 200 Zeichen festgelegt. Die Daten wurden in der JavaScript Objekt Notation gespeichert.

Die Applikation wurde auf dem gesamten 130 Node Cluster ausgeführt und berechnete alle $7{,}33 * 10^8$ Textvergleiche innerhalb von 11 Stunden und 22 Minuten.

5.3 Ergebnisse

Der nähere Abgleich zeigte, dass in 210.479 Artikeln Textwiederverwendung gefunden wurde, was 8,03% der verarbeiteten Seiten entspricht. Insgesamt erfasste der Algorithmus 4.246.181 Dokumentenpaare mit Textabschnitten von 200 Zeichen oder mehr, die eine Gleichheit von 70% besitzen. Bei Stichproben dieser Plagiatsfälle fiel auf, dass Listenartikel im Preprocessing nicht gefiltert wurden und daher die Auflistung der ähnlichsten Artikel dominierten. Deswegen wurden alle Artikel anhand des Stichwortes „*List*“ im Titel gefiltert, was die Anzahl der Dokumentenpaare auf 3.807.793 reduzierte.

5.3.1 Stichprobenbetrachtung von Textwiederverwendung

Als erstes wurden alle Artikel anhand der Anzahl ihrer gefunden Alignments geordnet. Dabei ist anzumerken, dass es zwischen zwei längeren Artikeln schon mehr als 20 gemeinsame Textpassagen geben kann.

Tabelle 5.3.1: Liste der Dokumente mit den häufigsten Alignments

Nr.	Alignments	Titel
1	13810	Portland, Connecticut
2	12657	West Shore School District
3	11670	Dover Area School District
4	11653	York City School District
5	11557	Chester Upland School District
6	11425	Pleasant Valley School District (Pennsylvania)
7	11282	Brownsville Area School District
8	11279	Penn Manor School District
9	11064	Big Spring School District
10	11058	West Perry School District

Wie zu sehen ist, dominieren hier Artikel mit dem Bezug auf Schuldistrikte in den USA. Im Detail zeigte sich durch die Visualisierung der Alignments auf www.picapica.org mit den zwei Beispielartikeln „West Shore School District“ und „Dover Area School District“ die Arten der Wiederverwendung. Die gesamte Seite scheint auf einem Template für alle Schuldistrikte zu basieren, weswegen kleinere Auflistungen wie Jahrgangsstärken und allgemeine Statistiken sich lediglich in den Werten unterscheiden. Die gefundenen Alignments verweisen zum Teil auf Satztemplates, beinhalten allerdings auch gleiche Zitate in Bezug auf Quellen, die für beide Artikel relevant sind. Das erklärt die hohe Anzahl von Alignments. Wenn hypothetisch 100 Seiten an 20 Stellen ein Template nutzen, dass dem geforderten 200 Zeichen und 70% Übereinstimmung genügt, resultiert dies bei den 100 Dokumenten in jeweils 1980 gefundenen Alignments für jedes dieser Dokumente. Man kann diese Fälle von Wiederverwendung als Template-Cluster bezeichnen.

Aufgrund dessen wurde die Liste der meisten Alignments von Titeln mit dem Schlüsselwort „school“ oder „district“ gefiltert. Das Ergebnis ähnelt dem bisherigen, nur jetzt zeigten sich lediglich Artikel von Städten der USA, welche auf dem gleichen Template basieren. Da alle dieser Städte ihren Staat im Titel tragen, konnten die Vergleiche durch eine Liste der Staaten der USA erneut gefiltert werden. Dies reduzierte die Anzahl der

gefundenen Plagiatspaare von den ursprünglichen 4.246.181 auf 576.835. Mit jedem Filterschritt zeigen sich neue Template-Cluster, denn jetzt werden die obersten Positionen von Schweizer Städten bestimmt. An dieser Stelle führt diese Herangehensweise zu keinen neuen Erkenntnissen.

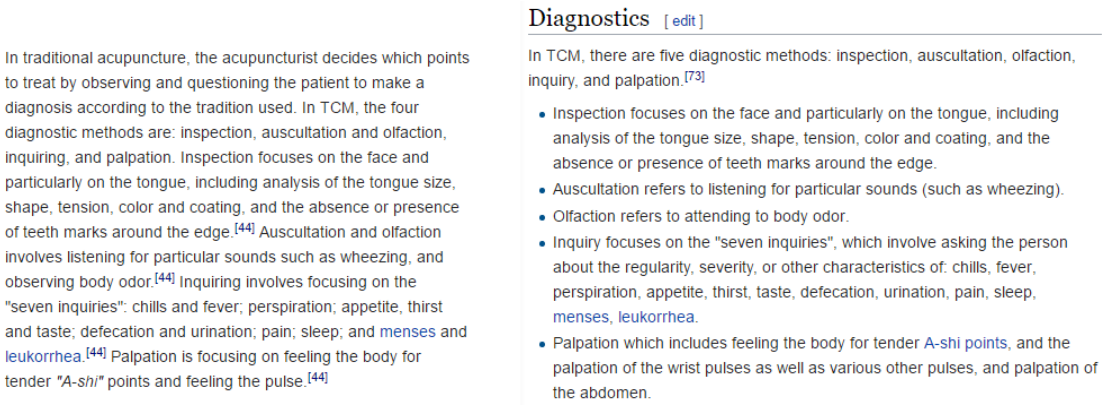
Danach wurden die Dokumentpaare im Hinblick auf die Anzahl ihrer gemeinsamen Textpassagen untereinander betrachtet.

Tabelle 5.3.2: Liste von Dokumentpaaren mit den häufigsten Wiederverwendungen

Nr.	**Seeds**	**Titel A**	**Titel B**
1	56	Operation Indian Ocean	2015 timeline of the War in Somalia
2	54	Operation Indian Ocean	2014 timeline of the War in Somalia
3	54	Timeline of Al-Shabaab ...	2014 timeline of the War in Somalia
4	51	Operation Indian Ocean	Timeline of Al-Shabaab…
5	50	Timeline of Al-Shabaab ...	2015 timeline of the War in Somalia

Alle in Tabelle 5.3.2 aufgeführten Artikel beschreiben chronologisch den Ablauf von Militäraktionen. Dabei werden die Geschehnisse eines jeden Tages in einzelnen Paragraphen geschildert. Bei näherer Betrachtung der Wiederverwendungen zeigte sich, dass die Texte einzelner Tage exakt übernommen wurden. Aufgrund der Bearbeitungshistorie kann man feststellen, dass beide Artikel aus Artikelpaar 1 fast ausschließlich von dem Autor „Middayexpress“ verfasst wurden. Auch die anderen aufgelisteten Artikel wurden entweder gänzlich, wie „*2014 timeline of the War in Somalia*“ oder partiell von demselben Autor geschrieben. Im Gegensatz zu der Häufung von Alignments aufgrund von Templates, kann man hier eine Ansammlung von gleichen Texten unter ähnlichem Kontext erkennen.

Ein anderes zufällig ausgewähltes Dokumentenpaar ist „*Traditional Chinese medicine*“ und „*Acupuncture*“ mit 33 Wiederverwendungen. Die hier gefundenen Passagen sind teilweise Zitate gleicher Quellen oder Verweise auf Quellen. Man findet auch unveränderte Kopien von ganzen Paragraphen, welche allerdings als solche gekennzeichnet sind und von einem gemeinsamen Dokument stammen. Nicht digitale Ursprünge führen dazu, dass ähnliche oder umformulierte Textabschnitte mit gleichem Inhalt zu finden sind (Abbildung 5.3.1).



In traditional acupuncture, the acupuncturist decides which points to treat by observing and questioning the patient to make a diagnosis according to the tradition used. In TCM, the four diagnostic methods are: inspection, auscultation and olfaction, inquiring, and palpation. Inspection focuses on the face and particularly on the tongue, including analysis of the tongue size, shape, tension, color and coating, and the absence or presence of teeth marks around the edge.[44] Auscultation and olfaction involves listening for particular sounds such as wheezing, and observing body odor.[44] Inquiring involves focusing on the "seven inquiries": chills and fever; perspiration; appetite, thirst and taste; defecation and urination; pain; sleep; and menses and leukorrhea.[44] Palpation is focusing on feeling the body for tender *"A-shi"* points and feeling the pulse.[44]

Diagnostics [edit]

In TCM, there are five diagnostic methods: inspection, auscultation, olfaction, inquiry, and palpation.[73]

- Inspection focuses on the face and particularly on the tongue, including analysis of the tongue size, shape, tension, color and coating, and the absence or presence of teeth marks around the edge.
- Auscultation refers to listening for particular sounds (such as wheezing).
- Olfaction refers to attending to body odor.
- Inquiry focuses on the "seven inquiries", which involve asking the person about the regularity, severity, or other characteristics of: chills, fever, perspiration, appetite, thirst, taste, defecation, urination, pain, sleep, menses, leukorrhea.
- Palpation which includes feeling the body for tender A-shi points, and the palpation of the wrist pulses as well as various other pulses, and palpation of the abdomen.

Abbildung 5.3.1: Textwiederverwendung aus nicht digitaler Quelle[1] der Artikel „Acupuncutre“[2] und „Traditional Chinese medicine“[3]

Als letztes wurden die Textpaare mit den größten, addierten, übereinstimmenden Passagenlängen betrachtet. Exemplarisch ist hier „Islamic philosophy“ und „Early Islamic philosophy“. Da der erste Artikel eine thematische Teilmenge des zweiten darstellt, ist zu erwarten, dass sich beide Artikel sehr ähneln. Es zeigt sich, dass auch auf syntaktischer Ebene der eine Artikel eine Teilmenge des anderen ist. Die Struktur wurde übernommen und ganze Paragraphen kopiert oder nur leicht abgewandelt. Aufgrund der Menge der Autoren ließ sich auf dem ersten Blick nicht erkennen, ob derselbe Autor für die kopierten Passagen verantwortlich ist. Auch bei dem Artikelpaar „Rock Music“ und „American Rock“, in denen einzelne Rockgenre beschrieben werden, wurden genau diese Beschreibungen präzise übernommen.

5.3.2 Diskussion

Es zeigten sich bei der ersten Betrachtung verschiedene Fälle von Textwiederverwendung in der Wikipedia. Bereits bei der Auswertung der Kosinus-Ähnlichkeiten viel auf, dass Paragraphenpaare mit einer Similarität von 1.0 gefunden wurden. Was anfangs wie ein Fehler in der Implementierung wirkte, stellte sich bei näherer Prüfung als korrekt heraus. Themenverwandte Artikel nutzen teilweise exakt die gleichen Textpassagen und Paragraphen. Leider lässt es sich statistisch nicht einfach berechnen, wie hoch der Anteil der kopierten Paragraphen oder Textteile tatsächlich ist.

---

1 Chinese Acupuncture and Moxibustion, Cheng, 1987, chapter 12.

2 Wikipedia: Acupuncture, https://en.wikipedia.org/wiki/Acupuncture (Stand: 20.01.2017)

3 Wikipedia: Traditional Chinese medicine, https://en.wikipedia.org/wiki/Traditional_Chinese_medicine (Stand: 20.01.2017)

Wie in der Abbildung der Paare mit den meisten Wiederverwendungen zu sehen ist, kann ein Paragraph in vielen anderen Artikeln verwendet worden sein. Wenn eine Passage t eines Artikels a in n anderen Artikeln $\{t_1, t_2 \dots t_n\} = T$ verwendet wurde, ergibt sich für die Anzahl der gefundenen Dokumentenpaare C die kombinatorische Gleichung $C = \binom{n}{2} = \frac{n!}{2 \cdot (n-2)!}$, wobei nur $|T| = n$ Fälle von Wiederverwendung vorliegen. Ein Paragraph, der neben dem Original noch zehn weitere Male im Korpus vorkommt, resultiert in 45 gefundenen Dokumentenpaaren.

Es wurden 120.043 Dokumentenpaare gefunden, bei denen mindestens ein Paragraph identisch ist. Daraus lässt sich ableiten, dass maximal 120.043 Plagiatsfälle vorliegen, was 4,6% aller verarbeiteten Artikel entspricht, wenn die vielfache Verwendung eines einzelnen Paragraphen ignoriert wird. Anhand die gezeigten Fälle von Mehrfachverwendung kann angenommen werden, dass die reale Zahl geringer ist.

Es kann bei diesen Fällen unterschieden werden, ob ein Wiederverwendung ein Plagiat im herkömmlichen Sinne darstellt oder aber ein Textabschnitt von einem Autor bewusst mehrmals verwendet wurde.

Es wurde gezeigt, dass ein nicht unerheblicher Teil der Artikelpaare aufgrund von Templates entstanden sind, was die Auswertung von Plagiaten erschwert. Aufgrund der gewählten Parameter für das Textalignment wurden einige kurze und irrelevante Fälle gefunden, anhand derer viele Templates als Plagiate eingestuft wurden. Beispielhaft hierfür sind Listen mit statistischen Angaben, bei denen die Beschreibung der Felder gleich ist, aber deren Werte differieren. Eine Vergrößerung der geforderten Mindestlänge von mehr als 200 Zeichen oder eine höhere prozentuale Übereinstimmung als 70% auf Zeichenebene sollte das Problem lösen.

Es zeigte sich außerdem, dass bei gefundene Wiederverwendungsfällen das Quelldokument nicht aus der Wikipedia stammen muss. Wie in Abbildung 5.3.1. zu sehen ist, kann auch eine externe Quelle, die von mehreren Artikeln genutzt wird, zu Wiederverwendung innerhalb des Korpus führen.

6 Zusammenfassung

In der Arbeit wurde gezeigt, wie sich aktuelle Methoden des Information-Retrieval auf das Problem der Erkennung von Textwiederverwendung in großen Datenmengen anwenden lassen. Die Berechnungen der Ähnlichkeiten von Texten anhand ihrer Vektorrepräsentationen lässt sich mit genügend Rechenleistung auch auf Korpussen wie dem der Wikipedia anwenden, ohne ungenaue Approximationen oder Hashfunktionen zu nutzen. Durch die Nutzung bestehender Frameworks wie Spark lassen sich die benötigten Kalkulationen effektiv verteilen und beschleunigen. Mit der Anpassung der zugrundeliegenden Bibliotheken konnte die Rechenzeit erheblich reduziert werden. Die genutzten Methoden lassen sich daher auch auf andere Textsammlungen anwenden oder in Echtzeitanwendungen nutzen, um Plagiate effektiv in großen Datenmengen zu finden. Mithilfe des verteilten Textalignments wurden die tatsächlichen Fälle von Textwiederverwendung in kurzer Zeit ermittelt. Hier zeigte sich, dass ein nicht unerheblicher Teil der Wikipedia-Artikel von anderen Wikipedia-Texten stammt. Dabei zeigten sich in einer ersten Betrachtung verschiedene Fälle von Wiederverwendung, sowie eine gewisse Clusterbildung von thematisch verwandten, plagiierten Artikeln.

7 Zukünftige Arbeit

Ein naheliegender Ansatz um die Arbeit fortzuführen, ist ein erneutes Preprocessing ohne Template-Erweiterung. Leider waren unter den Ergebnissen zu viele Dokumentenpaare, die nur aufgrund eines gemeinsamen Templates als solche erfasst wurden. Der errechnete Datensatz der Similaritäten eignet sich zwar gut für einen Abgleich mit anderen Textsammlungen, dennoch müssten die Alignment-Parameter angepasst werden, um dort nicht wieder alle Templates der Wikipedia zu erfassen.

Ein weiterer Punkt sind die Alignment-Parameter an sich. Mit den hier verwendeten Parametern von 70% Übereinstimmung und 200 Zeichen Länge werden einige irrelevante Wiederverwendungen gefunden. Durch Erhöhung der Parameter sollte sich ein besseres Bild über den Umfang von Wiederverwendung in der Wikipedia zeichnen lassen.

Des Weiteren könnte man versuchen Paragraphen, die in mehreren Artikeln verwendet werden, zusammenzufassen. Durch einen Abgleich dieser Paragraphen untereinander mit Simhash oder dem Kosinus kann man feststellen wie viele Verwendungen ein einziger Text in der Wikipedia hat.

Es könnte außerdem geprüft werden, ob die Berechnung der Similaritäten durch optimierten Code oder die Verwendung von Bibliotheken weiter beschleunigt werden kann. Alternativ kann die Nutzung aktueller Grafikkarten für die Vektoroperationen von Vorteil sein. Inwiefern sich die Approximation von Kandidatenpaaren durch ANNOY parallelisieren lässt, muss noch untersucht werden.

Eine vielversprechende Erweiterung für das Textalignment scheint „*From Word Embeddings To Document Distances*“ [wmd1] zu sein. Die Möglichkeit Paraphrasen auf Basis von Wortvektoren zu finden, wäre ein sinnvoller Zusatz bei der Plagiatssuche.

Literaturverzeichnis

[gb1] Alpert, J. und Hajaj, N. : "We knew the web was big..." (25.07.2008), URL: https://googleblog.blogspot.de/2008/07/we-knew-web-was-big.html (Stand: 10.01.2017)

[cc1] Nagel, s: "December 2016 Crawl Archive Now Available" (16.12.2016), URL: http://commoncrawl.org/2016/12/december-2016-crawl-archive-now-available/ (Stand: 10.01.2017)

[cw1] Lemur Project: "The ClueWeb12 Dataset" (05.2012), URL: http://www.lemurproject.org/clueweb12.php/ (Stand: 10.01.2017)

[shl] Weissman, S., Ayhan, S., Bradley, J., & Lin, J. (2015, June). Identifying duplicate and contradictory information in wikipedia. In Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries (pp. 57-60). ACM.

[mh1] Broder, A. Z. (1997, June). On the resemblance and containment of documents. In Compression and Complexity of Sequences 1997. Proceedings (pp. 21-29). IEEE.

[pv1] Le, Q. V., & Mikolov, T. (2014, June). Distributed Representations of Sentences and Documents. In ICML (Vol. 14, pp. 1188-1196).

[pv2] Dai, A. M., Olah, C., & Le, Q. V. (2015). Document embedding with paragraph vectors. arXiv preprint arXiv:1507.07998.

[bu1] Riehmann, P., Potthast, M., Gruendl, H., Kiesel, J., Jürges, D., Castiglia, G., ... & Froehlich, B. Visualizing Article Similarities in Wikipedia.

[wmd1] Kusner, M. J., Sun, Y., Kolkin, N. I., & Weinberger, K. Q. (2015, July). From Word Embeddings To Document Distances. In ICML (Vol. 15, pp. 957-966).

[w2v1] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[sd1] Myers, E. W. (1986). AnO (ND) difference algorithm and its variations. Algorithmica, 1(1-4), 251-266.

[sd2] Wu, S., Manber, U., Myers, G., & Miller, W. (1990). An O (NP) sequence comparison algorithm. Information Processing Letters, 35(6), 317-323.

[ku1] Kuhlen, W.: "Informationslinguistische Verfahren". in "Grundlagen der praktischen Information und Dokumentation" (2013), S. 275-277

[po1] Porter, M. F. (1980). An algorithm for suffix stripping. Program, 14(3), 130-137. ISO 690

[vsm] Salton, Gerard, Anita Wong, and Chung-Shu Yang. "A vector space model for automatic indexing." Communications of the ACM 18.11 (1975): 613-620.

[cmp1] Thada, V., & Jaglan, V. (2013). Comparison of jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm. International Journal of Innovations in Engineering and Technology, 2(4), 202-205.

[lsa] Deerwester, Scott, et al. "Indexing by latent semantic analysis." Journal of the American society for information science 41.6 (1990): 391.

[svd] Golub, Gene, and William Kahan. "Calculating the singular values and pseudo-inverse of a matrix." Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis 2.2 (1965): 205-224.

[tfi] Salton, Gerard, and Christopher Buckley. "Term-weighting approaches in automatic text retrieval." Information processing & management 24.5 (1988): 513-523.

[ch1] M. Charikar. Similarity estimation techniques from rounding algorithms. In Proc. 34th Annual Symposium on Theory of Computing (STOC 2002), pages 380–388, 2002.

[shg] Manku, Gurmeet Singh, Arvind Jain, and Anish Das Sarma. "Detecting near-duplicates for web crawling." Proceedings of the 16th international conference on World Wide Web. ACM, 2007.

[gen1] Řehůřek, R., and P. Sojka. "Gensim–Python Framework for Vector Space Modelling." NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic (2011).

[wns] Wikipedia: "Wikipedia:Namespace",
URL: https://en.wikipedia.org/wiki/Wikipedia:Namespace (Stand: 10.01.2017)

[wte] Attardi, G: "Wikiextractor", URL: https://github.com/attardi/wikiextractor (Stand: 10.09.2017)

[ntl] NLTK Project: "Natural Language Toolkit", URL: http://www.nltk.org/ (Stand: 10.10.2017)

[sp1] Laskowski, J: "Mastering Apache Spark 2.0", URL: "https://www.gitbook.com/book/jaceklaskowski/mastering-apache-spark/details" (Stand: 10.01.2017)

[knnc] Kouiroukidis, Nikolaos, and Georgios Evangelidis. "The effects of dimensionality curse in high dimensional knn search." Informatics (PCI), 2011 15th Panhellenic Conference on. IEEE, 2011.

[annoy] Bernhardsson, E. at al. : "Approximate Nearest Neighbors Oh Yeah", URL: "https://github.com/spotify/annoy" (Stand: 10.01.2017)

[flann] Marius Muja and David G. Lowe: "Fast Library for Approximate Nearest Neighbors" (14.12.2014). URL: "http://www.cs.ubc.ca/research/flann/" (Stand: 10.01.2017)

[annb] Rehurek, R.: Performance Shootout of Nearest Neighbours:Querying (1.12.2014), URL: "https://rare-technologies.com/performance-shootout-of-nearest-neighbours-querying/" (Stand: 10.01.2017)

[dimsum] Zadeh, Reza Bosagh, and Gunnar Carlsson. "Dimension independent matrix square using mapreduce." arXiv preprint arXiv:1304.1467 (2013).

[ccl] Amirbekian, N.: "Enhancing Spark with IPython Notebook and Cython" (30.10.2014), URL: "https://www.4info.com/Blog/October-2014/Enhancing-Spark-with-IPython-Notebook-and-Cython" (Stand: 10.01.2017)

[cyk] Smith, Kurt: "Cython: Blend the Best of Python and C++" (2015), URL: "http://public.enthought.com/~ksmith/scipy2013_cython/"

[blas] Basic Linear Algebra Subprograms (BLAS) Technical Forum Standard, URL: "http://www.netlib.org/blas/blast-forum/chapter3.pdf" (Stand: 10.01.2017)

[pica] Potthast, M et al.: PicaPica, URL: "www.picapica.org" (Stand: 10.01.2017)