UNIVERSITÄT ROSTOCK

FAKULTÄT FÜR INFORMATIK UND ELEKTROTECHNIK

INSTITUT FÜR INFORMATIK

Universität Rostock

Traditio et Innovatio

BACHELORARBEIT

AUTOMATISIERTE PRÜFUNG VON QUELLENANGABEN IN STUDENTISCHEN ARBEITEN

VORGELEGT VON: ADAM ABRAHAM

MATRIKELNUMMER: 219203217

ERSTGUTACHTER: DR.-ING. HANNES GRUNERT

ZWEITGUTACHTER: DR.-ING. HOLGER MEYER

ABGABEDATUM: 03.04.2024

Abstrakt

Diese Bachelorarbeit beschäftigt sich mit der Entwicklung eines automatisierten Systems zur Überprüfung von Quellenangaben in studentischen Arbeiten. Ziel ist es, die Integrität wissenschaftlicher Arbeiten zu gewährleisten und einen Beitrag zur Plagiatserkennung zu leisten. Zur Realisierung dieses Vorhabens wurden verschiedene Technologien und Methoden eingesetzt. Für die Extraktion von Texten aus PDF-Dateien kam die Bibliothek PyPDF2 zum Einsatz, während die Vorverarbeitung der natürlichen Sprache mithilfe der NLTK-Bibliothek durchgeführt wurde. Reguläre Ausdrücke (regex) dienten dazu, Zitate innerhalb der studentischen Arbeiten zu identifizieren sowie relevante Abschnitte aus den referenzierten wissenschaftlichen Publikationen zu extrahieren. Zur automatisierten Suche nach Quellen wurde die Datenbank DBLP[1] herangezogen. Zur Messung der Ähnlichkeit zwischen den extrahierten Zitaten aus studentischen Arbeiten und den Textpassagen aus den wissenschaftlichen Referenzen wurden zwei Methoden verglichen: die TF-IDF-Methode, implementiert mit scikit-learn, sowie die SIF-Methode unter Verwendung eines vortrainierten Word2Vec-Modells. Als Maßstab für die Ähnlichkeit diente die Kosinus-Ähnlichkeit. Die wichtigsten Ergebnisse der Arbeit zeigen, dass die automatisierte Messung der Ähnlichkeit zwischen Textpassagen effektiv durchgeführt werden kann, was bedeutende Implikationen für die Plagiatserkennung in akademischen Kontexten hat. Diese Forschung demonstriert die Machbarkeit der automatisierten Überprüfung von Quellenangaben und unterstreicht das Potential, die akademische Integrität durch den Einsatz fortschrittlicher Textanalysemethoden zu fördern.

[1] dblp.org

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listing-Verzeichnis

1 Einleitung

Dieses Kapitel bietet einen kurzen Überblick über die Arbeit. Als erstens werden die Gründe für diese Untersuchung und ihre Bedeutung im akademischen Kontext erläutert. Dann wird das Hauptziel dieser Arbeit klar definiert, was die Richtung der Forschung bestimmt und als Grundlage für den methodischen Ansatz sowie die Bewertung der Ergebnisse dient. Abschließend wird ein Überblick über die Struktur dieser Arbeit gegeben, um einen Einblick in die verschiedenen Phasen der Untersuchung zu geben. Dieses Kapitel bildet eine Grundlage für das Verständnis der gesamten Arbeit und leitet zur weiteren Analyse über.

1.1 Überblick

Das Verfassen studentischer Arbeiten erfordert beträchtliche Anstrengungen seitens der Studierenden und beansprucht einen erheblichen Teil der Zeit, die eigentlich für das Studium, die Recherche und innovative Entwicklungen vorgesehen wäre. Die Rolle des Prüfers ist jedoch keineswegs weniger anspruchsvoll. Die Studenten haben möglicherweise Wochen oder sogar Monate, um die erforderliche Arbeit abzuschließen, steht dem Prüfer diese umfangreiche Zeit in der Regel nicht zur Verfügung. Oftmals ist der Prüfer gezwungen, mehrere unterschiedliche studentische Arbeiten innerhalb einer kurzen Zeitspanne zu bewerten. Dies stellt eine beträchtliche Herausforderung dar.

Obwohl sich der Kern einer Forschungsarbeit auf ein spezifisches Thema konzentriert, ermöglicht die digitale Transformation und die Verfügbarkeit zahlreicher wissenschaftlicher Suchmaschinen heutzutage, leicht hunderte von Quellen zu diesem Thema zu finden. Und um die Integrität der wissenschaftlichen Forschung zu gewährleisten und eine Verletzung geistigen Eigentums zu vermeiden, muss der Studierende alle Quellen angeben, die während seiner Forschung verwendet wurden. Deshalb eine der Herausforderungen, denen der Prüfer bei der Bewertung von Forschungsarbeiten gegenübersteht, besteht in der Vielzahl der Quellen, die in einer einzigen studentischen Forschung verwendet werden. Diese umfassende Bewertung erfordert nicht nur Zeit, sondern auch eine tiefe Einsicht in die Vielfalt der verwendeten Quellen. Um dieser Herausforderung zu begegnen, sind neuartige Ansätze erforderlich, um den Prozess der Quellenbewertung zu optimieren. Außerdem stellt die stetig wachsende Menge an verfügbaren Forschungsressourcen nicht nur eine Herausforderung für die Prüfer dar, sondern auch für die Studierenden selbst. In einer Welt, in der Informationen im Überfluss vorhanden sind, ist es für Studierende von entscheidender Bedeutung, nicht nur auf eine breite Palette von Quellen zuzugreifen, sondern auch ihre Relevanz und Qualität zu bewerten. Daher gewinnt die Förderung von Fähigkeiten zur Informationsbewertung und Informationsauswahl eine zunehmende Bedeutung in der akademischen Ausbildung. Daher ist es zwingend erforderlich, effektive Lösungen zu entwickeln, um den Prozess der Bewertung des Quellenaspekts zu erleichtern und zu beschleunigen.

In diesem Kontext wird die Bedeutung von Lösungen zur automatisierten Quellenbewertung immer deutlicher. Eine vielversprechende Methode besteht in der Nutzung moderner Technologien, einschließlich maschinellem Lernen und künstlicher Intelligenz, um den Prüfer bei der Identifizierung von Zitaten und der Überprüfung ihrer Korrektheit zu unterstützen. Diese automatisierten Ansätze bieten Effizienzsteigerungen und tragen auch zur Entlastung des Prüfers bei, der oft vor der Aufgabe steht, in kurzer Zeit eine Vielzahl von Arbeiten zu korrigieren. Ein weiterer wichtiger Aspekt ist die kontinuierliche Anpassung der Bewertungsmethoden an die sich entwickelnde technologische Landschaft. Die Integration fortschrittlicher Technologien in den Bewertungsprozess ermöglicht eine effizientere Handhabung des wachsenden Umfangs an studentischen Arbeiten und fördert auch eine dynamischere Herangehensweise an die Herausforderungen der Informationsbewertung und Quellenüberprüfung. Die Synergie zwischen traditionellen Bewertungsmethoden und modernen technologischen Ansätzen schafft somit eine optimale Umgebung für die Entwicklung und Bewertung studentischer Arbeiten in einer zunehmend digitalisierten Welt. Abschließend lässt sich festhalten, dass die kontinuierliche Weiterentwicklung von Lösungsansätzen und Technologien im Bereich der Quellenbewertung und Plagiatserkennung einen bedeutenden Beitrag zur Förderung von Integrität und Qualität in der akademischen Welt leistet. Die Verbindung von traditionellen Werten wie akademischer Redlichkeit und dem neuesten Stand der Technologie schafft eine Grundlage für eine effiziente und anspruchsvolle Bewertung von studentischen Arbeiten.

1.2 Ziele

Die vorliegende Arbeit behandelt eine der möglichen Methoden zur automatischen Durchführung des Bewertungsprozesses von Quellenangeben. Der grundlegende Ansatz dieser Methode besteht im Wesentlichen aus:

1. **Extrahieren des Textes aus einer PDF-Datei:** Durch die Extraktion des Textes aus der PDF-Datei wird eine digitale Version des Inhalts erstellt.
2. **Lesen der vom Studenten verfassten Textquelle:** Die verfasste Textquelle des Studierenden wird sorgfältig analysiert, um den Inhalt und die Verwendung von Zitaten zu verstehen.
3. **Inhalt des Textes identifizieren**: Es erfolgt eine präzise Identifizierung des im Text genannten Inhalts, um mögliche Quellen zu erkennen.
4. **Vergleichen des genannten Textes mit der zugeordneten Quelle**: Ein detaillierter Vergleich zwischen dem im Text genannten Inhalt und der zugeordneten Quelle wird durchgeführt.
5. **Bestimmen des Ähnlichkeitsgrades zwischen den beiden Texten**: Anhand von Algorithmen wird der Ähnlichkeitsgrad zwischen dem studentischen Text und der Quelle ermittelt.

Darauf basierend wird überprüft, ob das Zitieren korrekt erfolgt ist oder nicht. Der oben beschriebene Ansatz stellt eine effektive und umfassende Lösung dar, um die Bewertung von studentischen Arbeiten zu beschleunigen. Dieser Prozess wird gemäß

vorab festgelegter und vereinbarter Standards durchgeführt, was die Effizienz und Genauigkeit der Bewertungsprozesse verbessert. Parallel dazu spielt dieser Prozess eine entscheidende Rolle bei der Bekämpfung von Plagiaten, indem er mithilfe einer automatisierten Herangehensweise Ähnlichkeitsraten zwischen Texten identifiziert und unrechtmäßige Überschreitungen oder unberechtigte Zitate aufdeckt.

Diese Methode zur automatischen Auswertung von Quellenangaben revolutioniert die Bewertung von studentischen Arbeiten durch ihre fortschrittlichen Analysewerkzeuge und digitalen Algorithmen. Die Extraktion von Text aus PDF-Dateien ermöglicht eine digitale Repräsentation des Inhalts. Die Identifizierung des im Text genannten Inhalts ermöglicht es, potenzielle Quellen zu erkennen und eine Grundlage für den anschließenden Vergleich mit der zugeordneten Quelle zu schaffen. Die Bestimmung des Ähnlichkeitsgrades zwischen den beiden Texten erfolgt auf intelligente Weise durch den Einsatz fortschrittlicher Algorithmen, die sowohl wörtliche Übereinstimmungen als auch inhaltliche Verknüpfungen berücksichtigen.

Diese Methode soll sicherstellen, dass die höchsten Standards bei der Bewertung von Arbeiten eingehalten werden. Der Fortschritt in der digitalen Bewertungstechnologie trägt dazu bei, die Herausforderungen im Zusammenhang mit dem wachsenden Umfang von studentischen Arbeiten zu bewältigen, indem er eine effiziente und zuverlässige Beurteilung ermöglicht.

1.3 Aufbau der Arbeit

Diese Arbeit beschäftigt sich mit der Untersuchung und Darstellung der effektiven Erkennung von Plagiaten durch automatisierte Überprüfung von Quellenangaben in studentischen Arbeiten. Nachdem in Kapitel 1 (Einleitung) die Problemstellung und die Ziele der Arbeit eingeführt wurden, wird in Kapitel 2 (Grundlagen) eine ausführliche Erläuterung der theoretischen Grundlagen gegeben. In Kapitel 3 (Stand der Technik) wird eine detaillierte Übersicht über die durchgeführte Recherche und die gegenwärtig verfügbaren Werkzeuge geboten, welche zur Zielerreichung dieser Arbeit beitragen. Kapitel 4 (Konzept) beschreibt das Konzept und den methodischen Ansatz dieser Arbeit im Detail. Hier wird Schritt für Schritt erklärt, wie das Vorgehen zur Erkennung von Plagiaten in studentischen Arbeiten konzipiert und umgesetzt wird. In Kapitel 5 (Implementierung) wird ein entwickelter Prototyp präsentiert, der die Anwendung der zuvor diskutierten Theorien und Tools demonstriert. Abschließend werden in Kapitel 6 (Zusammenfassung und Ausblick) die Ergebnisse zusammengefasst und die Effektivität des vorgestellten Ansatzes bewertet und mögliche Verbesserungen aufgezeigt.

2 Grundlagen

In diesem Kapitel erfolgt eine gründliche Erläuterung der theoretischen Grundlagen, mit dem Ziel, ein allgemein besseres Verständnis der in dieser Arbeit behandelten Themen zu fördern. Es wird angestrebt, durch eine tiefgehende und umfassende Darstellung der zentralen Konzepte und Prinzipien, ein fundiertes und ganzheitliches Verständnis der Thematik zu ermöglichen, was für die Einschätzung und Bewertung der weiterführenden Inhalte und Diskussionen der Arbeit von entscheidender Bedeutung ist.

2.1 Zitierstil

Die Zitierweise bezieht sich auf die Methode, wie Quellen oder Referenzen in einem Text angegeben werden. Es gibt verschiedene Zitierstile, darunter APA, MLA, Harvard, und viele andere, die in verschiedenen akademischen und professionellen Bereichen verwendet werden. Die Wahl des Zitierstils hängt oft vom Fachgebiet, der Institution oder den Anforderungen einer Publikation ab. Wichtig bei jeder Zitierweise ist die korrekte und vollständige Angabe der Quellen, um Plagiate zu vermeiden und die Nachvollziehbarkeit von Informationen zu gewährleisten.

Die Bedeutung der Zitierweise im akademischen Schreiben ist vielschichtig. Zum einen dient sie der Anerkennung und dem Respekt gegenüber der Arbeit anderer Wissenschaftler und Autoren. Durch korrektes Zitieren wird geistiges Eigentum anerkannt und Plagiat vermieden. Zum anderen verbessert die Zitierweise die Nachvollziehbarkeit und Überprüfbarkeit von Forschungsergebnissen. Sie ermöglicht Lesern, die Ursprünge von Ideen, Daten und Behauptungen zu verfolgen und stellt somit eine wesentliche Säule wissenschaftlicher Integrität dar [1].

2.1.1 APA-Stil

Der APA-Stil, der von der American Psychological Association entwickelt wurde, hat sich weit über die Grenzen der Psychologie und Sozialwissenschaften hinaus verbreitet und wird nun in vielen akademischen Disziplinen verwendet. Seine Beliebtheit und Verbreitung beruhen teilweise auf seiner klaren und konsistenten Struktur, die sowohl die Lesbarkeit als auch die Präzision in wissenschaftlichen Arbeiten fördert.

Eine ausführlichere Erklärung des APA-Stils [2][2] lautet wie folgt:

Im-Text-Zitierung: Im APA-Stil wird das Autor-Datum-System für Zitate innerhalb des Textes verwendet. Dies bedeutet, dass der Nachname des Autors und das Jahr der Veröffentlichung in Klammern direkt nach dem zitierten oder paraphrasierten Text angegeben werden. Wenn der Autor bereits im Satz erwähnt wird, steht das Jahr in Klammern direkt nach dem Namen des Autors. Zum Beispiel „stellt Klaus (2017) fest, dass...“ oder „(Klaus, 2017)“.

Literaturverzeichnis: Am Ende des Dokuments befindet sich ein umfangreiches Literaturverzeichnis mit allen verwendeten Quellen. Die Einträge im Literaturverzeichnis sind alphabetisch nach den Nachnamen der Autoren sortiert. Jeder Eintrag enthält wichtige Informationen wie den Namen des Autors, das Jahr der Veröffentlichung, den Titel des Werks und gegebenenfalls den Herausgeber sowie den Ort der Veröffentlichung. Bei Artikeln aus Fachzeitschriften werden auch Titel der Zeitschrift, Bandnummer und Seitenzahlen angegeben.

Formatierung: Der APA-Stil legt großen Wert auf eine einheitliche Gestaltung, einschließlich der Wahl einer bestimmten Schriftart, Schriftgröße und Randgestaltung. Dies gewährleistet eine kohärente Präsentation wissenschaftlicher Arbeiten.

Digitalquellen: Bei der Zitierung von Online-Quellen fordert der APA-Stil zusätzliche Informationen wie das Datum des letzten Zugriffs und, falls vorhanden, die DOI (Digital Object Identifier) des Dokuments.

Die Verwendung des APA-Stils in verschiedenen Fachbereichen zeigt seine Anpassungsfähigkeit an unterschiedliche wissenschaftliche Anforderungen. Um den APA-Stil korrekt anzuwenden, empfiehlt es sich, das aktuelle APA-Handbuch zu konsultieren, da sich die Richtlinien und Anforderungen im Laufe der Zeit ändern können.

Aufbau für Zitieren aus einem Buch[3]:

- **Verweis im Text**

  In Klammern: (Nachname der/des Verfassenden, Jahr)

  Im Fließtext: Nachname der/des Verfassenden (Jahr)

- **Literaturverzeichnis**

  Nachname der/des Verfassenden, Initial(en). (Jahr). Buchtitel (Auflagennummer Aufl.). Verlag.

[2] scribbr apa-standard, (abgerufen am 08.01.2024)
[3] scribbr richtig-zitieren, (abgerufen am 08.01.2024)

2.1.2 Harvard-Stil

Der Harvard-Stil wird häufig in den Natur- und Sozialwissenschaften verwendet und zeichnet sich durch seine Flexibilität und die verschiedenen Varianten aus, die von verschiedenen Universitäten und Institutionen verwendet werden.

Eine ausführlichere Erklärung des Harvard-Zitierstils [3][4] lautet wie folgt:

Im-Text-Zitierung: Im Harvard-Stil wird ein Autor-Datum-System für Zitate im Text verwendet. Der Nachname des Autors und das Erscheinungsjahr werden in Klammern angegeben. Wenn eine bestimmte Seite oder ein Seitenbereich zitiert wird, folgt nach einem Doppelpunkt die Seitenzahl. Zum Beispiel: (Wolf, 2012: 45). Wenn der Autor bereits im Text erwähnt wurde, wird nur das Jahr in Klammern angegeben, wie zum Beispiel „Wolf (2012) argumentiert...“.

Literaturverzeichnis: Ähnlich wie beim APA-Stil enthält der Harvard-Stil am Ende der Arbeit ein ausführliches Literaturverzeichnis. Dieses umfasst alle im Text zitierten Quellen und ist nach dem Nachnamen des Autors geordnet. Die Einträge enthalten normalerweise den Namen des Autors, das Erscheinungsjahr, den Titel des Werks sowie den Verlag und den Ort der Veröffentlichung. Bei Artikeln werden zusätzlich der Titel der Zeitschrift, Band- und Heftnummer sowie die Seitenzahlen angegeben.

Formatierung und Varianten: Da es keine einheitliche Version des Harvard-Stils gibt, können sich die genauen Formatierungsregeln je nach Universität oder Verlag unterscheiden. Es ist daher von großer Bedeutung, dass Autoren die spezifischen Richtlinien ihrer jeweiligen Institution beachten und diese konsequent anwenden.

Digitalquellen: Der Harvard-Stil erfordert auch präzise Angaben für Online-Quellen, einschließlich des Zugriffsdatums und gegebenenfalls der URL oder DOI.

Aufbau für Zitieren aus einem Buch[5]:

- **Verweis im Text**

  Paraphrase: (vgl. Nachname Jahr: Seite)

  Direktes Zitat: (Nachname Jahr: Seite)

  Im Fließtext: Nachname (Jahr: Seite)

- **Literaturverzeichnis**

  Nachname, Vorname (Jahr): Buchtitel, Auflage, Stadt, Land, Verlag.

[4] scribbr harvard-zitierweise, (abgerufen am 08.01.2024)
[5] scribbr richtig-zitieren, (abgerufen am 08.01.2024)

2.1.3 IEEE-Stil

Der Zitierstil, der vom Institute of Electrical and Electronics Engineers (IEEE) entwickelt wurde, ist besonders in technischen Studiengängen wie Informatik und Elektrotechnik weit verbreitet. Dieser Stil hat tatsächlich Einflüsse von der Chicago-Zitierweise, insbesondere in Bezug auf die Verwendung eines numerischen Systems.

Eine ausführlichere Erklärung des IEEE-Zitierstils [4] lautet wie folgt:

Numerische Zitate im Text: Im IEEE-Stil werden Quellenverweise im Text durch Zahlen in eckigen Klammern gekennzeichnet. Diese fortlaufenden Nummern verweisen auf entsprechende Einträge im Literaturverzeichnis. Nach einem Zitat oder einer Paraphrase würde also beispielsweise eine Zahl in eckigen Klammern wie „[1]“ stehen.

Literaturverzeichnis: Am Ende der Arbeit wird ein ausführliches Literaturverzeichnis angeführt, in dem die Quellen in der Reihenfolge ihres Vorkommens im Text aufgelistet sind. Jeder Eintrag im Literaturverzeichnis beginnt mit der entsprechenden Nummer und enthält alle Details zur Quelle einschließlich des Autorennamens, des Titels des Werks, der Publikationsdaten und weiterer relevanter Informationen.

Formatierung von Quellenangaben: Im IEEE-Stil werden die Quellenangaben je nach Art der Quelle (z.B. Buch, Zeitschriftenartikel, Konferenzbeitrag) unterschiedlich formatiert. Dies beinhaltet Angaben wie Autorennamen, Titel des Werks, Titel der Publikation (bei Artikeln), Bandnummer, Seitenzahlen und Erscheinungsjahr.

Digitalquellen: Die Verwendung von Online-Quellen in wissenschaftlichen Arbeiten erfordert gemäß dem IEEE-Stil bestimmte Richtlinien. Dies betrifft elektronische Zeitschriften, Datenbanken und andere digitale Ressourcen, bei denen oft die URL oder DOI der Quelle angegeben wird.

Aufbau für das Zitieren aus einem Buch [4]:

- **Mit fortlaufenden Nummern zitieren im Text**

  erste Quelle erhält die [1] und die zweite Quelle entsprechend die [2]

- **Literaturverzeichnis**

  Vornamens. Nachname, Titel des Buches. Stadt (ggf. Bundesstaat), Land, Verlag, Jahr.

2.1.4 Deutscher Zitierstil

Die Zitierweise (auch als Fußnoten-Stil bekannt), die in vielen geistes- und sozialwissenschaftlichen Disziplinen an deutschen Hochschulen üblich ist, besteht darin, im Haupttext lediglich auf Fußnoten zu verweisen.

Eine ausführlichere Erklärung des Deutschen-Zitierstils [5][6] lautet wie folgt:

Fußnoten im Text: es werden hochgestellte Zahlen verwendet, um auf die entsprechenden Quellenangaben am Ende der Seite im Fußnotenbereich zu verweisen. Dadurch wird der Lesefluss nicht gestört und es erscheinen keine ausführlichen Quellenangaben im Text selbst.

Quellenangaben in Fußnoten: In den Fußnoten finden sich normalerweise vollständige Angaben zu den Quellen, einschließlich Autorennamen, Titel des Werks, Publikationsort und -jahr. Bei direkten Zitaten wird auch die genaue Seitenzahl angegeben.

Literaturverzeichnis: Obwohl die Fußnoten alle erforderlichen Informationen enthalten, ist es trotzdem notwendig, ein alphabetisch sortiertes Literaturverzeichnis am Ende der Arbeit anzufertigen. Dieses Verzeichnis enthält alle im Text verwendeten Quellen in standardisierter Form.

Angabe des Seitenbereichs: Im Literaturverzeichnis wird in der Regel der gesamte Seitenbereich von Buchkapiteln oder Fachartikeln angegeben. Bei der Zitierung eines gesamten Werkes wie einem Buch ist dies jedoch nicht erforderlich.

Aufbau für Zitieren aus einem Buch[7]:

- **Mit Fußnoten zitieren im Text**

  Beispiele für Vollbeleg[1] und Kurzbeleg[2] – „Das ist ein wörtliches Zitat."[3]

  ---

  [1] Vgl. Nachname, Vorname: Buchtitel, Auflage, Stadt, Land: Verlag, Jahr, S. Seite.

  [2] Vgl. Nachname, Jahr, S. Seite.

  [3] Nachname, Jahr, S. Seite

- **Literaturverzeichnis**

  Nachname, Vorname: Buchtitel, Auflage, Stadt, Land: Verlag, Jahr, S. Seiten.

[6] scribbr deutsche-zitierweise, (abgerufen am 10.01.2024)
[7] scribbr richtig-zitieren, (abgerufen am 10.01.2024)

2.2 BibTeX

BibTeX ist ein Tool und ein Dateiformat, das in Verbindung mit LaTeX verwendet wird, um Literaturverzeichnisse und Quellenangaben in wissenschaftlichen Dokumenten zu organisieren und zu formatieren. Es wurde 1985 von Oren Patashnik und Leslie Lamport entwickelt[8].

Hier sind einige wichtige Aspekte von BibTeX [6, 7]:

Automatisches Literaturverzeichnis: Mithilfe von BibTeX können Benutzer ein zentrales Literaturverzeichnis erstellen, das alle Referenzen enthält. Dieses Verzeichnis wird in einer separaten ***.bib*** Datei gespeichert. In LaTeX-Dokumenten können dann Verweise auf die Einträge aus diesem Verzeichnis eingefügt werden, und BibTeX generiert automatisch das entsprechende Literaturverzeichnis im gewünschten Format.

Flexible Zitierweise: BibTeX unterstützt verschiedene Zitierstile und kann problemlos an die spezifischen Anforderungen verschiedener Publikationen oder akademischer Institutionen angepasst werden.

Einheitliche Struktur: Jeder Eintrag in einer BibTeX-Datei folgt einer bestimmten Struktur, die den Typ der Quelle (z.B. Buch, Artikel, Konferenzbeitrag), den Autor, den Titel, das Veröffentlichungsjahr und weitere relevante Informationen enthält.

Integration mit LaTeX: BibTeX arbeitet nahtlos mit LaTeX zusammen, einem weit verbreiteten System zur Erstellung wissenschaftlicher Dokumente. Die Kombination von LaTeX und BibTeX ermöglicht es Ihnen effizient komplexe Dokumente mit professioneller Typografie und korrekten Zitaten zu erstellen.

Anwendung in der Wissenschaft: Aufgrund seiner Effektivität und Vielseitigkeit ist BibTeX vor allem in den Naturwissenschaften, Mathematik und Technik weit verbreitet, wird aber auch in anderen wissenschaftlichen Fachbereichen genutzt.

Tabelle 1 stellt einige verschiedene Arten von Literaturangaben dar, die in BibTeX genutzt werden, sowie die dazu passenden Felder. In dieser Tabelle wird zwischen erforderlichen und optionalen Feldern unterschieden. Alle Felder, die über diese erforderlichen und optionalen Angaben hinausgehen, werden von BibTeX normalerweise nicht berücksichtigt.

[8] BibTeX – Wikipedia, (abgerufen am 16.01.2024)

Referenzart	erforderliche Felder	optionale Felder
Buch	Autor oder Herausgeber, Titel, Verlag, Jahr	Band oder Nummer, Reihe, Ort, Auflage, Monat, Anmerkung, ISBN
Zeitungs- oder Zeitschriftenartikel	Autor, Titel, Zeitschrift, Jahr	Band, Nummer, Seiten, Monat, Anmerkung
Wissenschaftliche Konferenz	Autor, Titel, Buchtitel, Jahr	Herausgeber, Band oder Nummer, Reihe, Seiten, Ort, Monat, Organisation, Verlag, Anmerkung
Gebundenes Druckwerk	Titel	Autor, Veröffentlichungsart, Ort, Monat, Jahr, Anmerkung
Teil eines Buches	Autor oder Herausgeber, Titel, Kapitel und/oder Seiten, Verlag, Jahr	Band oder Nummer, Reihe, Typ, Ort, Auflage, Monat, Anmerkung
Konferenzbericht	Titel, Jahr	Herausgeber, Band oder Nummer, Reihe, Ort, Monat, Organisation, Verlag, Anmerkung
veröffentlichter Bericht einer Hochschule oder anderen Institution	Autor, Titel, Institution, Jahr	Typ, Anmerkung, Nummer, Ort, Monat

Tabelle 1: Literaturtypen: einige Arten von Literaturangaben[9]

2.3 Information Retrieval

Information Retrieval bezeichnet den Prozess des Auffindens und Abrufens von Informationen aus einem großen Datensatz oder einer Datenbank, basierend auf Benutzeranfragen. Im Kern geht es darum, relevante Informationen aus einer Sammlung von Dokumenten (wie Texten, Bildern oder Datenbanken) zu extrahieren. Dies geschieht typischerweise über Suchmaschinen, die auf Algorithmen basieren, um die Relevanz und Bedeutung der Informationen im Hinblick auf die Benutzeranfrage zu bewerten [8, 9]. In wissenschaftlichen Kontexten bezieht sich IR auf das Auffinden akademischer Papiere, Artikel, Konferenzbeiträge und anderer wissenschaftlicher Publikationen. Hier spielen Aspekte wie die Genauigkeit, Relevanz, Zitierhäufigkeit und Aktualität der Informationen eine wichtige Rolle.

[9] BibTeX – Wikipedia, (abgerufen am 16.01.2024)

Die Suche nach wissenschaftlicher Literatur über Suchmaschinen, die APIs oder REST-Schnittstellen bieten, ist ein wichtiger Aspekt im Bereich der Informationsbeschaffung. In *Tabelle 2* sind einige Suchmaschinen und Datenbanken für wissenschaftliche Literatur aufgelistet, die API- oder REST-Zugänge anbieten:

Such-maschine	Schnitt-stelle	Fokus	Kosten-frei	Anfrage
PubMed[10]	API	Hauptsächlich aus den Bereichen Biomedizin und Gesundheit	✓	Beim Einfügen eines API-Schlüssels kann auf eine Webseite bis zu 10 Anfragen pro Sekunde gestellt werden
Crossref[11]	REST-API	Diverse wissenschaftliche Disziplinen	✓	Ab und zu wird eine Ratenbegrenzungen festlegt, dass es 50 Anforderungen pro Sekunde ausgeführt werden kann
arXiv[12]	API	Physik, Mathematik, Informatik, Quantenbiologie, Statistik…	✓	Eine Anforderung alle drei Sekunden
Europe PMC[13]	RESTful API	Biowissenschaftlicher Literatur	✓	Die genaue Anfragerate ist nicht spezifiziert
Semantic Scholar[14]	API	Diverse wissenschaftliche Disziplinen	✓	Für alle nicht authentifizierten Benutzer gilt ein Grenzwert von ungefähr 16 Anforderungen pro Sekunde
CORE[15]	API	Diverse wissenschaftliche Disziplinen	✓	Eine Anforderung alle zwei Sekunden

Tabelle 2: Einige wissenschaftliche Suchmaschinen, die über kostenlose APIs verfügen

[10] PubMed, (abgerufen am 28.01.2024)
[11] Crossref, (abgerufen am 28.01.2024)
[12] arXiv, (abgerufen am 28.01.2024)
[13] Europe PMC, (abgerufen am 29.01.2024)
[14] Semantic Scholar, (abgerufen am 29.01.2024)
[15] CORE, (abgerufen am 29.01.2024)

2.4 Herausforderungen bei der Prüfung von Quellenangaben

Zum Abschluss dieses Kapitels nach einer Betrachtung verschiedener Zitierstile, sowie Betrachtung der Information Retrieval und einige wissenschaftliche Suchmaschinen, sollen nun einige der Herausforderungen, die bei der Überprüfung von Quellenangaben auftreten können, diskutiert werden:

1. **Vielfalt der Zitierstile:** Unterschiedliche akademische Disziplinen und Publikationen verlangen verschiedene Zitierstile. Ein Zitat im APA-Stil sieht anders aus als im Harvard- oder IEEE-Stil. Diese Stile können sich in der Anordnung, Formatierung oder den benötigten Informationen unterscheiden. Deshalb muss das automatisierte System in der Lage sein, diese Unterschiede zu erkennen und korrekt zu interpretieren, was eine komplexe Anpassungsfähigkeit erfordert.
2. **Inkorrekte oder unvollständige Zitate:** Fehler wie das Fehlen von Autorennamen, Veröffentlichungsjahren oder Titel können vorkommen. Ein Fehler kann aber auch ein Tippfehler enthalten. Deshalb muss das System in der Lage sein, fehlerhafte Einträge zu identifizieren und idealerweise zu korrigieren. Dies erfordert das Verständnis der Zitierregeln und die Fähigkeit, Informationen gemäß diesen Regeln zu überprüfen und anzupassen.
3. **Erkennung von Plagiaten:** Plagiate können subtil sein, wie das Umschreiben von Text mit ähnlicher Bedeutung ohne angemessene Zitation. Dies erfordert fortschrittliche Algorithmen, die in der Lage sind, Ähnlichkeiten zwischen Texten zu erkennen, auch wenn diese umformuliert oder leicht verändert wurden.
4. **Zugang zu Datenbanken und Quellen:** Viele akademische Quellen sind hinter Bezahlschranken versteckt oder nicht online verfügbar, was den Zugriff erschwert. Einschränkungen beim Zugang oder bei der Integration dieser Datenbanken können ein Hindernis darstellen.
5. **Qualitätsbewertung von Quellen:** Neben der Überprüfung der Korrektheit der Zitate ist es auch wichtig, die Qualität und Glaubwürdigkeit der zitierten Quellen zu bewerten, denn nicht jede Quelle ist gleich vertrauenswürdig oder relevant, einige Quellen könnten veraltet oder inakzeptabel für akademische Standards sein. Deshalb das System sollte in der Lage sein, die Glaubwürdigkeit und Relevanz einer Quelle zu bewerten, was komplexe analytische Fähigkeiten erfordert.
6. **Sprachliche Unterschiede:** Unterschiedliche Sprachen können die Interpretation von Zitaten beeinflussen, insbesondere wenn es um Übersetzungen geht. Deshalb muss das System mehrere Sprachen unterstützen können, um eine breite Palette von Quellen abdecken zu können.
7. **Datenschutz:** Es müssen Datenschutzbestimmungen und ethische Richtlinien strikt eingehalten werden, was die Gestaltung des Systems beeinflussen kann.

3 Stand der Technik

In diesem Kapitel erfolgt eine detaillierte Darstellung der durchgeführten Literaturrecherche, die als fundamentale Basis für die vorliegende Arbeit dient. Dabei wird speziell auf die angewandte Suchmethodik eingegangen. Zudem wird ein Vergleich einige dieser wesentlichen Ergebnisse untereinander vorgenommen, um Unterschiede, Gemeinsamkeiten und spezifische Erkenntnisse herauszuarbeiten. Die sorgfältige Auswahl und Analyse der Quellen sind entscheidend, um die vielschichtigen Aspekte der automatisierten Prüfung von Quellenangaben umfassend zu beleuchten. Durch diese systematische Untersuchung wird sowohl die Relevanz der einzelnen Beiträge für das Forschungsfeld bewertet als auch ein solides Fundament für die weitere Analyse und Diskussion innerhalb dieser Arbeit gelegt.

3.1 Literaturrecherche

In diesem Abschnitt wird die Methodik der Literatursuche schrittweise und systematisch dargestellt. Es wird erläutert, welche Suchbegriffe verwendet wurden und wie die Suche zu relevanten Arbeiten geführt hat. Anschließend erfolgen eine detaillierte Betrachtung und ein Vergleich der einigen wesentlichen Ergebnisse, die aus dieser Suche hervorgegangen sind.

3.1.1 Zweck der Literatursuche

Ein umfassendes Verständnis der geforderten Arbeit sowie eine detaillierte Auseinandersetzung mit den potenziellen Problematiken und Herausforderungen sind entscheidende Schritte, die eine methodische und wissenschaftlich fundierte Zielerreichung gewährleisten. Das grundlegende Prinzip dieser Arbeit erleichterte die Entwicklung eines systematischen Suchplans nach den erforderlichen Quellen. Zweifellos führten viele der identifizierten Quellen auf weitere, eröffneten neue Perspektiven und vertieften das Verständnis für die Realisierung des angestrebten Ziels dieser Arbeit.

3.1.2 Suchbegriffe

Nachdem ein fundiertes Verständnis des Problems erlangt und das Ziel der Arbeit klar definiert wurde, konnte eine gezielte und präzise Literaturrecherche durchgeführt werden. Die sorgfältige Auswahl und Formulierung der Suchbegriffe waren dabei entscheidend, um relevante und qualitativ hochwertige Quellen zu identifizieren. Zu den verwendeten Suchbegriffen zählten unter anderem:

- "Automatisierte Prüfung von Quellenangaben"
- "Plagiatserkennungstechnologien"
- "Information Retrieval in akademischen Arbeiten"
- "Zitierstile und Zitationsanalyse"
- "Techniken der Texterkennung und -verarbeitung"
- "Schwellenwert in der Ähnlichkeitsanalyse"
- "Messung der semantischen Ähnlichkeit zwischen zwei Texten"

Diese Suchbegriffe wurden sowohl einzeln als auch in Kombination verwendet, um die Breite und Tiefe der vorhandenen Literatur auszuschöpfen. Durch die Anwendung dieser spezifischen Ausdrücke in verschiedenen akademischen Datenbanken und Suchmaschinen, darunter Google Scholar, IEEE Xplore, Crossref und arXiv, konnte eine umfassende Sammlung relevanter Studien, Artikel und Berichte zusammengestellt werden. Dieser Ansatz ermöglichte es, ein breites Spektrum an Perspektiven und Methoden zu erfassen, die zur Lösung der in dieser Arbeit adressierten Problematik beitragen. Die daraus resultierende Literaturbasis dient nicht nur als Grundlage für die theoretische Fundierung der Arbeit, sondern auch als Anstoß für die Entwicklung fortschrittlicher Ansätze im Bereich der automatisierten Quellenprüfung.

3.1.3 Auswahl der Ergebnisse

Im Anschluss an den initialen Schritt der Literatursuche ergab sich eine umfangreiche Sammlung von Publikationen. Um aus dieser Vielzahl die tatsächlich relevanten Arbeiten präzise auszuwählen und die Anzahl der Ergebnisse effektiv zu reduzieren, wurde ein zweistufiger Bewertungsprozess angewendet. In der ersten Stufe erfolgte eine sorgfältige Durchsicht von Titel und Abstrakt jeder identifizierten Publikation. Dieser Schritt diente dazu, die Arbeiten zu identifizieren, die auf den ersten Blick relevant erschienen oder einen direkten Bezug zur Thematik der automatisierten Prüfung von Quellenangaben aufwiesen.

Die Publikationen, die aufgrund ihres Titels und Abstrakts als relevant eingestuft wurden, wurden daraufhin einer eingehenden Analyse unterzogen. Ziel dieser intensiven Durchsicht war es, ein tiefgreifendes Verständnis für die spezifischen Ergebnisse und den Beitrag jeder einzelnen Arbeit zum Forschungsfeld zu erlangen. Dieser Prozess ermöglichte es, die Literaturbasis auf die Studien zu beschränken, die relevante Einsichten oder methodische Ansätze zur Lösung der Problematik bieten, die in dieser Arbeit behandelt werden.

Durch diesen methodischen Ansatz war es möglich, die Auswahl auf jene Arbeiten zu beschränken, die wesentliche Einblicke und wertvolle Beiträge zur Thematik liefern. Dieser Prozess gewährleistete, dass die finale Zusammenstellung der Literatur die relevantesten und aussagekräftigsten Arbeiten umfasst, die als Grundlage für die weitere Analyse und Diskussion innerhalb dieser Arbeit dienen.

3.2 Ergebnisse der Literaturrecherche

Es werden einige wesentliche Ergebnisse, die aus der sorgfältig durchgeführten Literatursuche hervorgegangen sind, dargelegt und eingehend beschrieben. Das Hauptaugenmerk liegt dabei auf der Vorstellung und Diskussion der essenziellen Ergebnisse, die für das Forschungsthema von besonderer Bedeutung sind.

3.2.1 Texte aus PDF lesen und extrahieren

Der erste Schritt bei der Arbeit mit PDF-Dateien besteht darin, eine geeignete Bibliothek auszuwählen, die das Lesen von PDF-Dokumenten und die Extraktion von Texten daraus ermöglicht: (Die genaue Beschreibung, wie Texte aus PDFs extrahiert werden, erfolgt hier nicht, weil diese auf der Arbeit von Alkhamis basiert [10]. In dieser Arbeit wird PyPDF2 verwendet, da es eine einfachere und schnellere Durchführung ermöglicht.)

1. **PyPDF2** stellt eine frei verfügbare und im Open-Source-Verfahren entwickelte Python-Bibliothek dar, die speziell für die Bearbeitung von PDF-Dateien konzipiert wurde. Mit dieser leistungsfähigen Bibliothek ist es möglich, PDF-Dokumente zu teilen, zusammenzuführen, den Inhalt zu beschneiden oder die Seiten auf vielfältige Weise zu modifizieren. Darüber hinaus ermöglicht PyPDF2 das Hinzufügen von individuellen Daten, das Festlegen von spezifischen Ansichtsoptionen sowie das Implementieren von Passwortschutz für PDF-Dateien. Ein weiterer wesentlicher Vorteil dieser Bibliothek ist die Fähigkeit, Textinhalte sowie Metadaten effizient aus PDF-Dokumenten zu extrahieren [11, 10].
2. **PDFMiner** dient der Informationsgewinnung aus PDF-Dokumenten und hebt sich von anderen PDF-Werkzeugen dadurch ab, dass es sich ausschließlich auf die Erfassung und detaillierte Analyse von Textdaten spezialisiert hat. Es ermöglicht die präzise Bestimmung der Position des Textes auf einer Seite und bietet Zugang zu weiteren Daten wie Schriftarten und Linienführungen. Ein weiterer nützlicher Aspekt von PDFMiner ist sein integrierter PDF-Konverter, der es erlaubt, PDF-Dokumente in verschiedene andere Textformate, wie zum Beispiel HTML, zu überführen. Zusätzlich ist PDFMiner mit einem flexiblen PDF-Parser ausgestattet, der nicht nur für Textanalysen, sondern auch für eine Reihe anderer Aufgaben eingesetzt werden kann, was die Flexibilität und Einsatzmöglichkeiten des Werkzeugs erweitert [10, 12]
3. **PyMuPDF** stellt eine effiziente Python-Bibliothek dar, die auf die Extraktion von Daten, Analyse, Umwandlung und Bearbeitung von PDF-Dokumenten sowie weiteren Dateiformaten ausgerichtet ist. Es ist bekannt für seine Geschwindigkeit und Effizienz im Vergleich zu anderen Bibliotheken. Mit ihren vielfältigen Funktionen ermöglicht sie es Anwendern, eine breite Palette von Dokumentenverarbeitungsaufgaben durchzuführen. Dies reicht von der simplen Herausziehung von Text bis zu fortgeschrittenen Manipulationen verschiedenster

Dokumenttypen, was PyMuPDF zu einem vielseitigen Werkzeug in der digitalen Dokumentenverwaltung macht [10][16].

Die nachstehende *Tabelle 3* bietet einen Vergleich der Bibliotheken PyPDF2, PDFMiner und PyMuPDF, indem sie einige ihrer Hauptunterschiede hervorhebt[17]. Sie stellt die Schlüsseleigenschaften und Funktionalitäten dieser drei Bibliotheken gegenüber.

Feature	PyPDF2	PDFMiner	PyMuPDF
Dokumentunterstützung	PDF	PDF	PDF und viele weiter Formate
Text extrahieren	✓	✓	PDF und weitere Formate
Geschwindigkeit bei Textextraktion	Mittel	Langsam	Schnell
Extrahieren von Tabellen	✗	✓	✓
Extrahieren von Vektorgrafiken	✗	begrenzt	✓
Zusatzfunktionen	Einfache Bearbeitung von PDFs	Konvertierung in andere Formate (z.B. HTML)	Umfangreiche Bearbeitungs- und Konvertierungsoptionen

Tabelle 3: Vergleich der drei Bibliotheken

3.2.2 Textverarbeitung und Zitatextraktion

Um Textverarbeitung im Rahmen von Natural Language Processing (NLP) durchführen, Textanalysen ermöglichen und Zitate aus dem Text identifizieren und extrahieren zu können, war es erforderlich, die Möglichkeiten der Textverarbeitung genauer zu untersuchen. Diese Untersuchung hat zu den folgenden bekannten Bibliotheken geführt:

3.2.2.1 Textverarbeitung

1. **NLTK** steht als führende Python-Plattform für die Verarbeitung und Analyse von menschlichen Sprachdaten im Vordergrund. Mit benutzerfreundlichen Zugängen zu mehr als 50 Korpora und lexikalischen Ressourcen, darunter WordNet, bietet

[16] PyMuPDF, (abgerufen am 14.02.2024)
[17] Features Comparison, (abgerufen am 14.02.2024)

NLTK eine umfassende Palette an Textverarbeitungstools. Diese Tools decken Bereiche wie Klassifizierung, Tokenisierung, Stemming, Tagging, Parsing und semantische Analyse ab und werden ergänzt durch Anbindungen an leistungsstarke NLP-Bibliotheken sowie durch grafische Demonstrationen und Beispieldaten. Ziel von NLTK ist es, die Forschung und Lehre in den Bereichen der natürlichen Sprachverarbeitung (NLP), empirischen Linguistik, Kognitionswissenschaft, künstlichen Intelligenz sowie in der Informationsrückgewinnung und im maschinellen Lernen zu unterstützen [13][18].
Listing 1 ist ein einfaches Beispiel, wie NLTK verwendet werden kann, um einen Text in Sätze und Wörter zu tokenisieren, Stoppwörter zu entfernen:

```
1 | import nltk
2 | from nltk.corpus import stopwords
3 | from nltk.tokenize import sent_tokenize, word_tokenize
4 | # Downloading the required NLTK resources
5 | nltk.download('punkt') # For tokenizing sentences and words
6 | nltk.download('stopwords') # For removing stopwords
8 | text = "NLTK stands as a leading Python platform for processing and
analyzing human language data. With user-friendly access to more than 50
corpora and lexical resources, including WordNet, NLTK offers a
comprehensive suite of text processing tools. These tools cover areas such
as classification, tokenization, stemming, tagging, parsing, and semantic
analysis, and are complemented by connections to powerful NLP libraries,
as well as by graphical demonstrations and sample data. The goal of NLTK
is to support research and teaching in fields such as natural language
processing (NLP), empirical linguistics, cognitive science, artificial
intelligence, information retrieval, and machine learning."
9 | # Tokenization of the text into sentences and words, removal of
    stopwords
10| sentences = sent_tokenize(text)
11| words = [word_tokenize(sentence) for sentence in sentences]
12| stop_words = set(stopwords.words(english))
13| filtered_sentences = [[word for word in sentence if word.lower() not
                                        stop_words] for sentence in words]
14| print("Sätze:", sentences)
15| print("Wörter (Stoppwörter entfernt):", filtered_sentences)
```

Listing 1: Textverarbeitung mit NLTK

Der Code beginnt mit dem Import von notwendigen NLTK-Modulen, die für die Tokenisierung und Entfernung von Stoppwörtern verwendet werden. Zunächst werden die erforderlichen NLTK-Ressourcen heruntergeladen, darunter Tokenisierung-Werkzeuge und eine Liste von Stoppwörtern. Ein Beispieltext wird als Grundlage für die Analyse definiert. Dieser Text wird dann in Sätze und anschließend in Wörter unterteilt. Im nächsten Schritt werden Stoppwörter aus dem tokenisierten Text entfernt, um den Textinhalt für die Analyse aufzubereiten. Dieser Prozess demonstriert, wie NLTK für grundlegende NLP-Aufgaben eingesetzt werden kann. Der Code liefert folgende Ausgabe:

```
1 | Sätze: ['NLTK stands as a leading Python platform for processing and
analyzing human language data.', 'With user-friendly access to more than
50 corpora and lexical resources, including WordNet, NLTK offers a
comprehensive suite of text processing tools.', 'These tools cover areas
such as classification, tokenization, stemming, tagging, parsing, and
semantic analysis, and are complemented by connections to powerful NLP
libraries, as well as by graphical demonstrations and sample data.', 'The
```

[18] NLTK, (abgerufen am 14.02.2024)

```
goal of NLTK is to support research and teaching in fields such as natural
language processing (NLP), empirical linguistics, cognitive science,
artificial intelligence, information retrieval, and machine learning.']
2 | Wörter (Stoppwörter entfernt): [['NLTK', 'stands', 'leading',
'Python', 'platform', 'processing', 'analyzing', 'human', 'language',
'data', '.'], ['user-friendly', 'access', '50', 'corpora', 'lexical',
'resources', ',', 'including', 'WordNet', ',', 'NLTK', 'offers',
'comprehensive', 'suite', 'text', 'processing', 'tools', '.'], ['tools',
'cover', 'areas', 'classification', ',', 'tokenization', ',', 'stemming',
',', 'tagging', ',', 'parsing', ',', 'semantic', 'analysis', ',',
'complemented', 'connections', 'powerful', 'NLP', 'libraries', ',',
'well', 'graphical', 'demonstrations', 'sample', 'data', '.'], ['goal',
'NLTK', 'support', 'research', 'teaching', 'fields', 'natural',
'language', 'processing', '(', 'NLP', ')', ',', 'empirical',
'linguistics', ',', 'cognitive', 'science', ',', 'artificial',
'intelligence', ',', 'information', 'retrieval', ',', 'machine',
'learning', '.']]
```

Listing 2: Ausgabe des NLTK-Codes

2. **TextBlob** ist eine Python-Bibliothek für die Verarbeitung von Texten, die auf NLTK aufbaut. Sie bietet eine einfache API für gängige Aufgaben der natürlichen Sprachverarbeitung (NLP), darunter Part-of-Speech-Tagging, Extraktion von Nominalphrasen, Sentiment-Analyse, Klassifizierung und Übersetzung. TextBlob ignoriert unbekannte Begriffe und verwendet bekannte Wörter und Ausdrücke, um Endbewertungen zu berechnen. Diese Bibliothek erfordert kein vorheriges Training, da sie bereits vordefinierte Funktionen enthält, was sie zu einem benutzerfreundlichen Werkzeug für Operationen wie Tokenisierung und Sentiment-Analyse macht [14, 15].
   Listing 3 ist ein einfaches Beispiel, wie TextBlob verwendet werden kann, um einen Text in Sätze und Wörter zu tokenisieren:

```
1 | from textblob import TextBlob
2 | text = "NLTK is a leading Python platform for working with human
language data. With user-friendly access to over 50 corpora and lexical
resources such as WordNet, NLTK offers a comprehensive suite of text
processing tools. These tools cover areas such as classification,
tokenization, stemming, tagging, parsing, and semantic analysis, and are
complemented by connections to powerful NLP libraries as well as by
graphical demonstrations and sample data. The goal of NLTK is to support
research and teaching in fields such as natural language processing (NLP),
empirical linguistics, cognitive science, artificial intelligence,
information retrieval, and machine learning."
3 | # Tokenization of the text into sentences and words
4 | blob = TextBlob(text)
5 | sentences = blob.sentences
6 | words = [word for sentence in sentences for word in sentence.words]
7 | print("Sätze:", [str(sentence) for sentence in sentences])
8 | print("Wörter:",words)
```

Listing 3: Textverarbeitung mit TextBlob

In dieser Code wird zuerst der Text in Sätze und dann in Wörter unterteilt, um eine detaillierte Analyse zu ermöglichen. Es ist zu beachten, dass TextBlob nicht direkt für die Entfernung von Stoppwörtern verwendet wird, für diese Aufgabe muss auf andere Methoden zurückgegriffen werden. Es wird folgende Ausgabe geliefert:

```
1 | Sätze: ['NLTK is a leading Python platform for working with human language data.', 'With user-friendly access to over 50 corpora and lexical resources such as WordNet, NLTK offers a comprehensive suite of text processing tools.', 'These tools cover areas such as classification, tokenization, stemming, tagging, parsing, and semantic analysis, and are complemented by connections to powerful NLP libraries as well as by graphical demonstrations and sample data.', 'The goal of NLTK is to support research and teaching in fields such as natural language processing (NLP), empirical linguistics, cognitive science, artificial intelligence, information retrieval, and machine learning.']
2 | Wörter: ['NLTK', 'is', 'a', 'leading', 'Python', 'platform', 'for', 'working', 'with', 'human', 'language', 'data', 'With', 'user-friendly', 'access', 'to', 'over', '50', 'corpora', 'and', 'lexical', 'resources', 'such', 'as', 'WordNet', 'NLTK', 'offers', 'a', 'comprehensive', 'suite', 'of', 'text', 'processing', 'tools', 'These', 'tools', 'cover', 'areas', 'such', 'as', 'classification', 'tokenization', 'stemming', 'tagging', 'parsing', 'and', 'semantic', 'analysis', 'and', 'are', 'complemented', 'by', 'connections', 'to', 'powerful', 'NLP', 'libraries', 'as', 'well', 'as', 'by', 'graphical', 'demonstrations', 'and', 'sample', 'data', 'The', 'goal', 'of', 'NLTK', 'is', 'to', 'support', 'research', 'and', 'teaching', 'in', 'fields', 'such', 'as', 'natural', 'language', 'processing', 'NLP', 'empirical', 'linguistics', 'cognitive', 'science', 'artificial', 'intelligence', 'information', 'retrieval', 'and', 'machine', 'learning']
```

Listing 4: Ausgabe des TextBlob-Codes

Beim Vergleich der Wörter-Tokenisierung zwischen NLTK und TextBlob fällt wesentlicher Unterschied auf:

- Es wurden NLTK-Satzzeichen, einschließlich Kommas, als separate Tokens behandelt. Diese Methode, implementiert durch die **`word_tokenize`** Funktion von NLTK, führt dazu, dass Satzzeichen in der Liste der tokenisierten Wörter sichtbar bleiben. Im Gegensatz dazu entfernt TextBlob in seiner Standard Tokenisierung, die durch Aufrufen von **`.words`** auf einem TextBlob-Objekt erfolgt, Satzzeichen aus der Liste der Tokens.

Dieser Unterschied unterstreicht die Bedeutung der Werkzeugauswahl je nach Analysebedarf.

3. **Stanza** ist ein fortschrittliches Python-NLP-Toolkit, das die linguistische Analyse einer Vielzahl von menschlichen Sprachen unterstützt. Es bietet eine vollständig neurale Pipeline, die von rohem Text ausgeht und eine Reihe von Annotationen erzeugt, darunter Tokenisierung, Lemmatisierung, POS-Tagging, morphologische Merkmalszuweisung, Abhängigkeitsanalyse und Erkennung benannter Entitäten. Stanza zeichnet sich durch seine Multilingualität aus und unterstützt Designs Modelle für viele Sprachen. Zusätzlich bietet Stanza eine Python-Schnittstelle zum CoreNLP (Java-Paket von Stanford), wodurch Benutzer Zugriff auf weitere Funktionen wie „Relationsextraktion erhalten“. Das Toolkit ist vollständig Open-Source, und für alle unterstützten Sprachen sind vortrainierte Modelle verfügbar. Stanza integriert sich nahtlos mit PyTorch[19] und profitiert von beschleunigter Leistung auf GPU-unterstützten Maschinen. Zusammengefasst kombiniert Stanza eine native Python-Implementierung für einfache Einrichtung mit einer umfassenden, neuronenbasierten Textanalysepipeline. Es unterstützt über 70

[19] PyTorch

Sprachen mit vortrainierten Modellen und bietet zusätzlich Zugang zu erweiterten CoreNLP-Funktionen, um die Forschung und Anwendung im Bereich der multilingualen NLP zu erleichtern und voranzutreiben [16, 17] [20].
Listing 5 ist ein einfaches Beispiel, wie Stanza verwendet werden kann, um einen Text in Sätze und Wörter zu tokenisieren.

```
1 | import stanza
2 | stanza.download('en')  # Downloads the English model
3 | nlp = stanza.Pipeline('en')  # Creates a pipeline for English
4 | text = "NLTK stands as a leading Python platform for processing and
analyzing human language data. With user-friendly access to more than 50
corpora and lexical resources, including WordNet, NLTK offers a
comprehensive suite of text processing tools. These tools cover areas such
as classification, tokenization, stemming, tagging, parsing, and semantic
analysis, and are complemented by connections to powerful NLP libraries,
as well as by graphical demonstrations and sample data. The goal of NLTK
is to support research and teaching in fields such as natural language
processing (NLP), empirical linguistics, cognitive science, artificial
intelligence, information retrieval, and machine learning."
5 | # Processes text with Stanza
6 | doc = nlp(text)
7 | # Tokenization of the text into sentences and words
8 | sentences = [sentence.text for sentence in doc.sentences]
9 | words = [[word.text for word in sentence.words] for sentence in
                                                        doc.sentences]
10| print("Sätze:", sentences)
11| print("Wörter:", words)
```

Listing 5: Textverarbeitung mit Stanza

Der Beispielcode stellt dar, wie mit der Stanza-Bibliothek Textverarbeitung durchgeführt werden kann. Zuerst wird das englische Modell heruntergeladen, um die Sprachdatenverarbeitung zu ermöglichen. Anschließend wird eine NLP-Pipeline für Englisch erstellt, die für verschiedene Textanalyseaufgaben wie Tokenisierung genutzt werden kann. Der Beispieltext wird durch die Stanza-Pipeline verarbeitet, wobei der Text zuerst in Sätze und dann in Wörter tokenisiert wird. Die Ergebnisse dieser Operationen werden ausgegeben, um die Struktur des Textes zu zeigen. Es ist wichtig zu erwähnen, dass obwohl Stanza leistungsstarke Funktionen für die Sprachverarbeitung bietet, hat aber keine integrierte Funktion zur Entfernung von Stoppwörtern. Für solche spezifische Aufgabe müssten zusätzliche Schritte oder andere Bibliotheken integriert werden. Ausgabe des Beispielcodes ist in *Listing 6* veranschaulicht:

```
1 | Sätze: ['NLTK stands as a leading Python platform for processing and
analyzing human language data.', 'With user-friendly access to more than
50 corpora and lexical resources, including WordNet, NLTK offers a
comprehensive suite of text processing tools.', 'These tools cover areas
such as classification, tokenization, stemming, tagging, parsing, and
semantic analysis, and are complemented by connections to powerful NLP
libraries, as well as by graphical demonstrations and sample data.', 'The
goal of NLTK is to support research and teaching in fields such as natural
language processing (NLP), empirical linguistics, cognitive science,
artificial intelligence, information retrieval, and machine learning.']
2 | Wörter: [['NLTK', 'stands', 'as', 'a', 'leading', 'Python',
'platform', 'for', 'processing', 'and', 'analyzing', 'human', 'language',
```

[20] stanza, (abgerufen am 06.03.2024)

```
'data', '.'], ['With', 'user', '-', 'friendly', 'access', 'to', 'more',
'than', '50', 'corpora', 'and', 'lexical', 'resources', ',', 'including',
'WordNet', ',', 'NLTK', 'offers', 'a', 'comprehensive', 'suite', 'of',
'text', 'processing', 'tools', '.'], ['These', 'tools', 'cover', 'areas',
'such', 'as', 'classification', ',', 'tokenization', ',', 'stemming', ',',
'tagging', ',', 'parsing', ',', 'and', 'semantic', 'analysis', ',', 'and',
'are', 'complemented', 'by', 'connections', 'to', 'powerful', 'NLP',
'libraries', ',', 'as', 'well', 'as', 'by', 'graphical', 'demonstrations',
'and', 'sample', 'data', '.'], ['The', 'goal', 'of', 'NLTK', 'is', 'to',
'support', 'research', 'and', 'teaching', 'in', 'fields', 'such', 'as',
'natural', 'language', 'processing', '(', 'NLP', ')', ',', 'empirical',
'linguistics', ',', 'cognitive', 'science', ',', 'artificial',
'intelligence', ',', 'information', 'retrieval', ',', 'and', 'machine',
'learning', '.']]
```

Listing 6: Ausgabe des Stanza-Codes

In der Ausgabe des Stanza-Codes, siehe *Listing 6*, wird folgendes festgestellt: Stanza behandelt zusammengesetzte Wörter oder Wörter, die durch Bindestriche verbunden sind, differenzierter, indem es sie in ihre Bestandteile zerlegt. Das bedeutet, dass '**user-friendly**' in drei separate Tokens aufgeteilt wird: '**user**', '**-**', '**friendly**'. Diese Art der detaillierten Tokenisierung kann in bestimmten analytischen Kontexten vorteilhaft sein. NLTK und TextBlob hingegen behandeln '**user-friendly**' als ein einziges Token, was in vielen Anwendungsfällen, insbesondere bei der oberflächlichen Textverarbeitung, praktisch sein kann, da es die Komplexität der Token-Liste reduziert.

4. **Flair** ist ein fortschrittliches NLP-Framework, das speziell für die einfache Anwendung und Weiterentwicklung von neuesten Sequenz-Labeling-, Textklassifikations- und Sprachmodellen entwickelt wurde. Es zeichnet sich durch eine einfache, einheitliche Schnittstelle aus, die es ermöglicht, unterschiedliche Arten von Wort- und Dokumenten-Embeddings mühelos zu kombinieren und zu verwenden. Zusätzlich bietet Flair standardisierte Routinen für Modelltraining und Hyperparameter-Optimierung sowie ein Modul zum Herunterladen und Konvertieren öffentlich verfügbarer NLP-Datensätze. Dies erleichtert das schnelle Einrichten von Experimenten. Da Flair auf PyTorch basiert, profitieren Anwender von einer nahtlosen Integration und der Möglichkeit, eigene Modelle zu trainieren und neue Ansätze mit Flair-Embeddings und -Klassen zu experimentieren [18] [21]. *Listing 7* ist ein einfaches Beispiel, wie Flair verwendet werden kann, um einen Text in Wörter zu tokenisieren:

```
1 | from flair.data import Sentence
2 | text = "NLTK stands as a leading Python platform for processing and
analyzing human language data. With user-friendly access to more than 50
corpora and lexical resources, including WordNet, NLTK offers a
comprehensive suite of text processing tools. These tools cover areas such
as classification, tokenization, stemming, tagging, parsing, and semantic
analysis, and are complemented by connections to powerful NLP libraries,
as well as by graphical demonstrations and sample data. The goal of NLTK
is to support research and teaching in fields such as natural language
processing (NLP), empirical linguistics, cognitive science, artificial
intelligence, information retrieval, and machine learning."
3 | # Creating a Flair Sentence object
```

[21] Flair, (abgerufen am 09.03.2024)

```
4 | sentence = Sentence(text)
5 | # Tokenization
6 | tokens = [token.text for token in sentence]
7 | print("Wörter:", tokens)
```

Listing 7: Textverarbeitung mit Flair

Dieser Beispielcode verwendet Flair, ein NLP-Framework, um die Wörter darin zu tokenisieren. Der Text wird in einzelne Wörter (Tokens) aufgeteilt und diese werden ausgegeben.

Die Ausgabe des Flair-Codes ist :

```
1 | Wörter: ['NLTK', 'stands', 'as', 'a', 'leading', 'Python',
'platform', 'for', 'processing', 'and', 'analyzing', 'human', 'language',
'data', '.', 'With', 'user-friendly', 'access', 'to', 'more', 'than',
'50', 'corpora', 'and', 'lexical', 'resources', ',', 'including',
'WordNet', ',', 'NLTK', 'offers', 'a', 'comprehensive', 'suite', 'of',
'text', 'processing', 'tools', '.', 'These', 'tools', 'cover', 'areas',
'such', 'as', 'classification', ',', 'tokenization', ',', 'stemming',
',', 'tagging', ',', 'parsing', ',', 'and', 'semantic', 'analysis', ',',
'and', 'are', 'complemented', 'by', 'connections', 'to', 'powerful',
'NLP', 'libraries', ',', 'as', 'well', 'as', 'by', 'graphical',
'demonstrations', 'and', 'sample', 'data', '.', 'The', 'goal', 'of',
'NLTK', 'is', 'to', 'support', 'research', 'and', 'teaching', 'in',
'fields', 'such', 'as', 'natural', 'language', 'processing', '(', 'NLP',
')', ',', 'empirical', 'linguistics', ',', 'cognitive', 'science', ',',
'artificial', 'intelligence', ',', 'information', 'retrieval', ',',
'and', 'machine', 'learning', '.']
```

Listing 8: Ausgabe des Flair-Codes

5. **spaCy** ist eine fortschrittliche Open-Source-Bibliothek für die natürliche Sprachverarbeitung (NLP) in Python, die auf neuesten Forschungsergebnissen basiert. Die Bibliothek ermöglicht die Durchführung grundlegender NLP-Aufgaben wie Tokenisierung, Part-of-Speech-Tagging, regelbasiertes Matching, Lemmatisierung, Dependency-Parsing, Erkennung von Satzgrenzen, Named Entity Recognition und Ähnlichkeitserkennung. Das zugrundeliegende Modell ist stochastisch und nutzt Gradienten und Verlustfunktionen für das Training, wobei das vortrainierte Modell auch für Transferlernen eingesetzt werden kann. spaCy bietet vortrainierte Pipelines, unterstützt die Tokenisierung und das Training für mehr als 70 Sprachen, nutzt neueste neuronale Netzwerkmodelle und ermöglicht Multi-Task-Learning mit vortrainierten Transformern [19] [22].

6. **Polyglot** ist ein vielseitiges NLP-Framework, das für umfangreiche mehrsprachige Anwendungen konzipiert wurde. Mit einer GPL v3 Lizenz ist es frei verfügbar. Polyglot bietet eine breite Palette von Funktionen, darunter Tokenisierung in 165 Sprachen, Spracherkennung in 196 Sprachen, Named Entity Recognition in 40 Sprachen, Part-of-Speech-Tagging in 16 Sprachen, Sentiment-Analyse in 136 Sprachen, Wort-Embeddings in 137 Sprachen, morphologische Analyse in 135 Sprachen und Transliteration in 69 Sprachen. Diese Fähigkeiten machen Polyglot zu einem mächtigen Werkzeug für die Verarbeitung und Analyse von Text in fast jeder Sprache [20].

[22] spaCy, (abgerufen am 09.03.2024)

7. **Pattern** ist ein vielseitiges Python-Modul, spezialisiert auf Web-Mining, und bietet eine umfassende Palette an Werkzeugen in verschiedenen Bereichen. Im Bereich der Verarbeitung natürlicher Sprache bietet Pattern Funktionen wie Part-of-Speech-Tagging, n-Gramm-Suche, Sentiment-Analyse und Zugriff auf WordNet. Für maschinelles Lernen stehen Methoden wie der Vektorraum, Clustering und Klassifizierungstechniken (KNN, SVM, Perzeptron) zur Verfügung. Zudem ermöglicht es die Netzwerkanalyse, einschließlich der Untersuchung von Graphenzentralität und Visualisierung von Netzwerken. Pattern ist umfangreich dokumentiert, durch mehr als 350 Unit-Tests abgesichert und mit über 50 Beispielen ausgestattet, wodurch es eine wertvolle Ressource für Entwickler und Forscher darstellt [21].

Die *Tabelle 4* bietet einen Überblick über die Fähigkeiten jeder Bibliothek in Bezug auf die NLP-Aufgaben:

	NLTK[23]	Text-Blob[24]	Stanza[25]	Flair[26]	spaCy[27]	Poly-glot[28]	Pattern[29]
Tokenisierung	✓	✓	✓	✓	✓	✓	✓
Part-of-Speech Tagging	✓	✓	✓	✓	✓	✓	✓
Lemmatisierung	✓	✓	✓	✓	✓	✗	✓
Named Entity Recognition (NER)	✓	✗	✓	✓	✓	✓	✓
Sentiment Analyse	✓	✓	✓	✓	✗	✓	✓
Stoppwörter Entfernung	✓	✗	✗	✗	✓	✗	✗
Parsing	✓	✓	✓	✓	✓	✗	✓
N-Gramme	✓	✓	✗	✗	✗	✗	✓

Tabelle 4: Vergleichstabelle zwischen den Bibliotheken mit Fokus auf NLP-Aufgaben

[23] NLTK guides library, (abgerufen am 17.03.2024)
[24] TextBlob Features, (abgerufen am 17.03.2024)
[25] Stanza, (abgerufen am 17.03.2024)
[26] Flair-python-library, (abgerufen am 17.03.2024)
[27] spaCy Features, (abgerufen am 17.03.2024)
[28] Polyglot Features, (abgerufen am 17.03.2024)
[29] Pattern wiki, (abgerufen am 17.03.2024)

3.2.2.2 Auswahl der NLP-Bibliothek

Für diese Arbeit wurde aufgrund verschiedener Aspekte die NLTK-Bibliothek ausgewählt:

- NLTK bietet umfangreiche Werkzeuge für zahlreiche NLP-Aufgaben, was es zu einem flexiblen und anpassbaren Toolkit macht.
- NLTK hat eine aktive Community und eine reich an Ressourcen, Tutorials und Beispielen. Dies erleichtert den Einstieg und die Weiterentwicklung Ihrer NLP-Fähigkeiten.
- NLTK kann gut mit anderen Bibliotheken wie TextBlob, re (Reguläre Ausdrücke), Pattern und Flair zusammenarbeiten.

3.2.2.3 Extraktion von Zitaten

Um explizite Zitate im zuvor extrahierten Text zu identifizieren, können verschiedene Methoden angewandt werden. In dieser Arbeit konzentrieren wir uns darauf, nach spezifischen Mustern im Text zu suchen. Als Beispiel wird der IEEE-Zitierstil herangezogen. Dieser Stil zeichnet sich durch eine einfache, strukturierte und explizite Art der Referenzierung im Fließtext aus. Zitate werden dabei durch Nummern in eckigen Klammern gekennzeichnet. Einige Möglichkeiten umfassen:

1. **Fuzzy-Wuzzy** ist eine Bibliothek für das Fuzzy-String-Matching, auch bekannt als approximatives String-Matching. Es handelt sich dabei um den Prozess, Zeichenketten zu identifizieren, die einem vorgegebenen Muster annähernd ähneln. Dies findet Anwendung in Bereichen wie der Rechtschreibprüfung, der Erkennung von Textduplikaten, Spamfilterung und dem Abgleichen von DNA-Sequenzen in der Bioinformatik. Die Ähnlichkeit zwischen zwei Ausdrücken wird mit Fuzzy-Wuzzy gemessen und durch einen Wert zwischen 0 und 1 dargestellt, wobei ein Wert nahe 1 eine starke Übereinstimmung und ein Wert nahe 0 eine geringe oder keine Ähnlichkeit anzeigt. Die Methode nutzt die Levenshtein-Distanz, um die Anzahl der benötigten Bearbeitungsschritte für eine vollständige Übereinstimmung zu bestimmen, und bietet Techniken für das direkte Matching (Ratio) sowie für das Matching basierend auf den besten Teilstrings (Partial Ratio) [22, 23]. Mit Hilfe von Fuzzy-Wuzzy ist es möglich, Zitate auf folgende Weise zu extrahieren.

```
1 | from fuzzywuzzy import fuzz
2 | Ein extrahierter Text
3 | text = "NLTK stands as a leading Python platform for processing and
analyzing human language data[1], With user-friendly access to more than
50 corpora and lexical resources[ 1], including WordNet, NLTK offers a
comprehensive suite of text processing tools(1]"
4 | # The desired pattern
5 | pattern = "[1]"
6 | # Split the text into segments (e.g., based on punctuation)
7 | segments = extracted_text.split(',')
8 | # Iterate through the segments and use fuzzy matching to search for
    the pattern
9 | for segment in segments:
```

```
10|      similarity = fuzz.partial_ratio(pattern, segment.strip())
11|      if similarity > 80:  # The threshold
12|          print(f"Gefunden: {segment}, \nÄhnlichkeit: {similarity}")
```

Listing 9: einfaches Beispiel: Zitate mithilfe von Fuzzy-Wuzzy extrahieren

In diesem Beispielcode wird die Bibliothek Fuzzy-Wuzzy genutzt, um Zitate in einem Text zu identifizieren, die dem Muster **[1]** ähneln. Zuerst wird der Text in einzelne Segmente aufgeteilt, wobei als Trennzeichen das Komma (**,**) dient. Für jedes Segment wird dann die Ähnlichkeit zum gesuchten Muster **[1]** mit der Methode **fuzz.partial_ratio** berechnet. Diese Funktion vergleicht das Muster mit einem Textsegment und gibt eine Ähnlichkeitspunktzahl zurück, die zwischen 0 und 100 liegt. Segmente, deren Ähnlichkeitspunktzahl über einem festgelegten Schwellenwert von 80 liegt, werden als mit dem Muster übereinstimmend betrachtet. Für diese Segmente wird eine Ausgabe erzeugt, die das gefundene Segment und die ermittelte Ähnlichkeitspunktzahl anzeigt.

```
1 | Gefunden: NLTK as a leading Python platform for processing and
              analyzing human language data[1],
2 | Ähnlichkeit: 100
```

Listing 10: Ausgabe des Fuzzy-Wuzzy-Codes mit Schwellenwert von 80

In diesem spezifischen Fall hat der Code erfolgreich das Segment identifiziert, das genau das Muster **[1]** enthält, und dieses mit einer Ähnlichkeit von 100% als Übereinstimmung ausgegeben. Die Variationen des Musters, wie **[1]** (mit einem zusätzlichen Leerzeichen) und **(1]** (mit einer runden Klammer statt einer eckigen), wurden jedoch nicht als Übereinstimmungen erkannt, Solche Abweichungen könnten auf Tippfehler oder andere Flüchtigkeitsfehler zurückzuführen sein. Dies liegt daran, dass die partielle Übereinstimmungsrate (partial_ratio) des Fuzzy-Wuzzy-Algorithmus die genaue Position und Art der Zeichen berücksichtigt. Ohne Anpassung des Musters oder der Anwendung einer umfassenderen Matching-Strategie können solche Variationen unentdeckt bleiben.
Die Anpassung des Schwellenwerts im beschriebenen Beispiel *Listing 9* mit der Fuzzy-Wuzzy-Bibliothek, ermöglicht eine flexiblere Identifikation von Mustern, die einem vorgegebenen Kriterium annähernd entsprechen. Durch die Senkung des Schwellenwerts von 80 auf 50 konnten neben der exakten Übereinstimmung "[1]" auch leicht abweichende, in *Listing 11* veranschaulicht, Muster wie "[1]" und "(1]" erkannt werden, die vorher unentdeckt blieben.

```
1 | Gefunden: NLTK stands as a leading Python platform for processing and
              analyzing human language data[1],
2 | Ähnlichkeit: 100
3 | Gefunden: With user-friendly access to more than 50 corpora and lexical
              resources[ 1],
4 | Ähnlichkeit: 67
5 | Gefunden: NLTK offers a comprehensive suite of text processing
              tools(1],
6 | Ähnlichkeit: 67
```

Listing 11: Ausgabe des Fuzzy-Wuzzy-Codes mit Schwellenwert von 50

Es existieren zwei Ansätze zur Verbesserung der Mustererkennung haben jedoch ihre eigenen Herausforderungen und mögliche Nachteile:

- Schwellenwert senken: Indem der Schwellenwert für die Ähnlichkeitsbewertung reduziert wird, erhöht sich die Wahrscheinlichkeit, dass auch Variationen eines Musters als Treffer erkannt werden. Dies kann besonders nützlich sein, wenn Variationen des Musters aufgrund von Tippfehlern oder Formatierungsunterschieden zu erwarten sind. Allerdings kann eine zu starke Senkung des Schwellenwerts auch zu falsch positiven Ergebnissen führen, bei denen irrelevante Segmente fälschlicherweise als Übereinstimmungen klassifiziert werden.
- Musteranpassungen und -erweiterungen: Eine präzisere Lösung kann darin bestehen, das Muster so anzupassen, dass es alle erwarteten Variationen der Zitate abdeckt. Obwohl dies zu einer höheren Genauigkeit bei der Identifizierung von Zitaten führt, kann es auch sehr zeitaufwendig sein, insbesondere wenn die Variationsbreite groß ist (wie bei einigen anderen Zitierstilen). Zudem besteht das Risiko, dass unvorhergesehene Musterformen, die in der Analysephase nicht erkannt wurden, weiterhin unentdeckt bleiben.

Beide Ansätze haben ihre Berechtigung, abhängig von den spezifischen Anforderungen und dem Kontext der Analyse. Die Wahl zwischen ihnen erfordert eine sorgfältige Abwägung zwischen der gewünschten Genauigkeit und dem Aufwand für die Implementierung.

2. **re (regular expression):** Reguläre Ausdrücke, auch bekannt als Regex oder Regex-Muster, sind eine spezialisierte Mini-Sprache, die innerhalb von Python durch das **`re`** Modul zur Verfügung gestellt wird. Durch diese Sprache werden Regeln für eine Menge möglicher Zeichenfolgen definiert, die abgeglichen werden sollen – seien es englische Sätze, E-Mail-Adressen, TeX-Befehle oder beliebige andere Daten. Es kann dann überprüft werden, ob eine Zeichenfolge einem Muster entspricht oder ob ein Muster irgendwo in der Zeichenfolge gefunden werden kann. Reguläre Ausdrücke können auch dazu genutzt werden, eine Zeichenfolge zu modifizieren oder auf verschiedene Arten aufzuteilen [24] [30].
   Die Verwendung der Bibliothek **`re`** ermöglicht eine effiziente Identifikation von Zitaten im extrahierten Text.

```
1 | import re
2 | # An extracted text
3 | text = "NLTK stands as a leading Python platform for processing and
analyzing human language data[1], With user-friendly access to more than
50 corpora and lexical resources[ 2], including WordNet[3   ], NLTK offers
a comprehensive suite of text processing tools(4]"
4 | # The desired pattern
5 | pattern = r'\[\s*\d+\s*\]|\(\s*\d+\s*\]'
6 | # Split the text into sentences
7 | sentences = re.split(r'\,\s*', text)
8 | # Iterate through the sentences and use regular expressions to search
    for the pattern
9 | for sentence in sentences:
10|     if re.search(pattern, sentence):
11|         print(f"Gefunden: {sentence}")
```

Listing 12: einfaches Beispiel: Zitate mithilfe von re extrahieren

[30] howto regex, (abgerufen am 10.03.2024)

Dieser Beispielcode nutzt die **re** Bibliothek in Python, um spezifische Muster in einem Text zu identifizieren, die Zitierungen im IEEE-Stil repräsentieren. Das Muster **pattern** sucht nach Zahlen innerhalb von eckigen Klammern oder Klammern mit möglichen Leerzeichen davor oder danach. Der Text wird anhand von Kommas in einzelne Sätze aufgeteilt. Für jeden Satz überprüft der Code dann, ob das definierte Muster vorhanden ist. Wird eine Übereinstimmung gefunden, zeigt der Code den entsprechenden Satz an, in dem das Zitat vorkommt. Dieser Ansatz ist effektiv, um explizite Zitate, die nach einem bestimmten Schema formatiert sind, aus einem umfangreicheren Text zu extrahieren. Die Ausgabe dieses Codes ist in *Listing 13* veranschaulicht.

```
1 | Gefunden: NLTK stands as a leading Python platform for processing and
              analyzing human language data[1]
2 | Gefunden: With user-friendly access to more than 50 corpora and lexical
              resources[ 2]
3 | Gefunden: including WordNet[3 ]
4 | Gefunden: NLTK offers a comprehensive suite of text processing
              tools(4]
```

Listing 13: Ausgabe des re-Codes

3.2.2.4 Auswahl der Methode zur Zitatextraktion

Für die Extraktion von expliziten Zitaten aus Texten wird die Verwendung von **re** gewählt, da **re** eine präzise und leistungsfähige Methode bietet, spezifische, streng formatierte Muster zu erkennen und zu extrahieren. Im Gegensatz zu **fuzzywuzzy**, das auf Ähnlichkeitsprüfung basiert und weniger präzise sein kann, ermöglichen reguläre Ausdrücke eine exakte Übereinstimmungssuche, was für die zuverlässige Identifizierung von Zitaten notwendig ist.

3.2.3 Text-Vektorisierung und Ähnlichkeitsmessung

Um die semantische Ähnlichkeit zwischen zwei Sätzen zu messen, ist die Vektorisierung des Textes ein entscheidender Schritt. Im Folgenden sind einige Ergebnisse der Recherche dargestellt:

1. **Gensim** ist eine populäre, open-source Python-Bibliothek, die für das eigenständige Lernen und die Verarbeitung großer Textdatensätze entwickelt wurde. Sie zeichnet sich durch die effiziente Implementierung von Algorithmen wie Word2Vec, FastText und Latent Dirichlet Allocation (LDA) aus, die es ermöglichen, Dokumente als semantische Vektoren darzustellen. Diese Algorithmen erkunden automatisch die semantische Struktur von Dokumenten, indem sie statistische Muster im gemeinsamen Auftreten von Wörtern innerhalb eines Trainingskorpus analysieren. Gensim benötigt dafür keine menschliche Eingabe und kann rein auf Basis von unstrukturierten Textdaten arbeiten. Ursprünglich für die Schätzung der Dokumentenähnlichkeit entwickelt, unterstützt Gensim effektiv Aufgaben wie die semantische Analyse von Texten und das Auffinden thematisch ähnlicher

Dokumente. Trotz seiner Fähigkeit, mit großen Datensätzen umzugehen, stößt Gensim bei extrem umfangreichen Korpora aufgrund seiner Abhängigkeit von Python-Wörterbüchern an Grenzen. Im Vergleich zu anderen NLP-Bibliotheken wie NLTK zeichnet sich Gensim besonders durch seine Skalierbarkeit und die Fokussierung auf semantische Vektormodelle aus [25, 26][31].

2. **scikit-learn:** eine umfassende Python-Bibliothek für maschinelles Lernen, erleichtert die Integration von Lernmethoden in Python-Anwendungen. Diese Bibliothek bietet ein breites Spektrum an Werkzeugen für Klassifizierung, Regression, Dimensionsreduktion und mehr, und ist sowohl für Forschungsprojekte als auch kommerzielle Anwendungen beliebt. Besonders hervorzuheben ist, dass Scikit-learn Funktionen wie CountVectorizer und TfidfVectorizer zur Textanalyse bereitstellt. Diese Werkzeuge sind in der Lage, automatisch Wortgrenzen zu erkennen, Satzzeichen zu entfernen und können für fortgeschrittene Textverarbeitungsprozesse wie Stemming oder Lemmatisierung angepasst werden. Obwohl Scikit-learn primär für maschinelles Lernen konzipiert ist, wird es in dieser Arbeit speziell für seine Fähigkeiten zur Text-Vektorisierung und Ermittlung von Textähnlichkeiten herangezogen, um die semantische Nähe zwischen Texten zu messen [27, 28].
   Darüber hinaus können mithilfe von ***sklearn.metrics.jaccard_score*** und ***sklearn.metrics.pairwise.cosine_similarity*** aus der Bibliothek Scikit-learn der Jaccard-Index sowie die Kosinus-Ähnlichkeit berechnet werden.

3. **Die Kosinus-Ähnlichkeit** ist eine weit verbreitete Metrik in der Informationswiedergewinnung und verwandten Forschungsbereichen, die zwei Textdokumente durch Berechnung des Kosinus des Winkels zwischen ihren Vektoren vergleicht. Diese Vektoren repräsentieren die Dokumente als Sammlung von Begriffen. Die Ähnlichkeit zwischen zwei Dokumenten wird durch den Kosinuswert zwischen ihren Begriffsvektoren ermittelt. Ein Kosinuswert von eins deutet darauf hin, dass zwei Vektoren genau in die gleiche Richtung zeigen, was maximale Ähnlichkeit bedeutet. Bei jedem anderen Winkel ist der Kosinuswert kleiner als eins, was auf eine geringere Ähnlichkeit hinweist. Diese Methode kann auf jeglichen Text angewendet werden, egal ob es sich um Sätze, Absätze oder ganze Dokumente handelt, und dient als Maß dafür, wie ähnlich sich zwei Texte inhaltlich sind [29, 30][32].

4. **Jaccard-Ähnlichkeit** stellt eine einfache und benutzerfreundliche Methode dar, die in zahlreichen Anwendungsbereichen zuverlässige Ergebnisse erzielt. Sie basiert auf dem Jaccard-Koeffizienten, der die Übereinstimmung zwischen zwei endlichen Mengen durch das Verhältnis der Größe ihres Schnittbereichs zur Größe ihrer Vereinigung quantifiziert. Als Maßstab zur Bewertung der Ähnlichkeit in Anfragen und Dokumenten variiert der Jaccard-Index zwischen 0 für vollständige Unähnlichkeit und 1 für vollständige Ähnlichkeit, wodurch ein intuitiver Rahmen für die Beurteilung von Übereinstimmungen geschaffen wird [31, 32].

[31] Gensim intro, (abgerufen am 10.03.2024)
[32] Kosinus-Ähnlichkeit, (abgerufen am 22.03.2024)

5. **Dice-Koeffizient** ist eine Methode zum Vergleich der Ähnlichkeiten zwischen zwei verschiedenen Textproben. Er stellt eine semimetrische Variante des Jaccard-Koeffizienten dar. Diese Methode gewährleistet Genauigkeit über verschiedene Datensätze hinweg und weist Datensätzen, die unzusammenhängende Merkmale enthalten, ein geringeres Gewicht zu [33].

6. **TF-IDF:** Die Umwandlung von Text in Vektoren kann auf verschiedene Arten erfolgen, wobei TF-IDF (Term Frequency-Inverse Document Frequency) eine beliebte Methode darstellt. TF-IDF gewichtet Wörter basierend auf ihrer Häufigkeit und Wichtigkeit über Dokumente hinweg. Es hilft, die Bedeutung von häufig auftretenden, aber weniger informativen Wörtern zu reduzieren, um die semantische Relevanz in der Textanalyse zu verbessern. Die **`TfidfVectorizer`**-Klasse gehört zur Scikit-learn-Bibliothek und bietet eine Methode zur Umwandlung von Textdaten in eine Matrix von TF-IDF-Features. Es kombiniert die Funktionalitäten von **`CountVectorizer`** für die Umwandlung von Text in eine Token-Zählmatrix und **`TfidfTransformer`** für die Umrechnung der Zählmatrix in eine normierte TF oder TF-IDF-Darstellung. Die Verwendung von TF-IDF ermöglicht es, die Bedeutung eines Wortes im Kontext des gesamten Datensatzes zu bewerten, indem häufig vorkommende, aber weniger informative Wörter heruntergestuft werden [34, 35][33].
   Das folgende Codebeispiel zeigt, wie die Bibliothek Scikit-learn und die TF-IDF-Methode zur Text-Vektorisierung und Ähnlichkeitsmessung eingesetzt werden können:

```
1 | from sklearn.feature_extraction.text import TfidfVectorizer
2 | # Two example texts
3 | text1 = "NLTK stands as a leading Python platform for processing and
analyzing human language data. With user-friendly access to more than 50
corpora and lexical resources, including WordNet, NLTK offers a
comprehensive suite of text processing tools. These tools cover areas such
as classification, tokenization, stemming, tagging, parsing, and semantic
analysis, and are complemented by connections to powerful NLP libraries,
as well as by graphical demonstrations and sample data."
4 | text2 = "NLTK is recognized as a premier Python framework for the
processing and analysis of human language information. Providing easy
access to over 50 corpora and lexical assets, such as WordNet, NLTK
presents an extensive array of text processing functionalities. These
functionalities span classification, tokenization, stemming, tagging,
parsing, and semantic analysis, enhanced by integration with potent NLP
libraries and enriched by visual demos and sample datasets."
5 | # Initialization of the TfidfVectorizer
6 | tfidf_vectorizer = TfidfVectorizer()
7 | # Transformation of texts into TF-IDF vectors
8 | tfidf_vectors = tfidf_vectorizer.fit_transform([text1, text2])
9 | # Output of the TF-IDF matrix
10| print(tfidf_vectors.toarray())
11| # Output of the shape of the TF-IDF matrix
12| print("Shape of TF-IDF Vectors:", tfidf_vectors.shape)
```

Listing 14: TfidfVectorizer Beispiel

Dieser Code nutzt den ***`TfidfVectorizer`*** aus ***`scikit-learn`***, um zwei Texte ***`text1`*** und ***`text2`*** in TF-IDF-Vektoren umzuwandeln. Dabei werden die Texte in eine Matrix

[33] using-countvectorizer & scikit-learn text-feature-extraction & tfidftransformer-tfidfvectorizer-usage-differences, (abgerufen am 10.03.2024)

von TF-IDF-Features transformiert, die die Wichtigkeit der Wörter in den Dokumenten und im Gesamtkorpus widerspiegelt. Die Ausgabe, in *Listing 15* veranschaulicht, umfasst die TF-IDF-Werte in Form eines Arrays und die Dimension der Matrix, die die Anzahl der Texte und die Größe des Wortschatzes zeigt. Diese Transformation ermöglicht es, die Texte anhand ihrer Wortbedeutung quantitativ zu vergleichen.

```
1 | [[0.08443707 "……" 0.08443707]
2 | [0.0900358 "……" 0.0900358 ]]
2 | Shape of TF-IDF Vectors: (2, 77)
```

Listing 15: Ausgabe des TfidfVectorizer-Codes

Es ist möglich, die Ähnlichkeit zwischen den beiden Texten durch die Berechnung der Kosinus-Ähnlichkeit der erzeugten TF-IDF-Vektoren zu ermitteln.
Der Code in *Listing 16* demonstriert, wie dies umgesetzt werden kann:

```
1 | from sklearn.feature_extraction.text import TfidfVectorizer
2 | from sklearn.metrics.pairwise import cosine_similarity
3 | # Two example texts
4 | text1 = "NLTK stands as a leading Python platform for processing and
analyzing human language data. With user-friendly access to more than 50
corpora and lexical resources, including WordNet, NLTK offers a
comprehensive suite of text processing tools. These tools cover areas such
as classification, tokenization, stemming, tagging, parsing, and semantic
analysis, and are complemented by connections to powerful NLP libraries,
as well as by graphical demonstrations and sample data."
5 | text2 = "NLTK is recognized as a premier Python framework for the
processing and analysis of human language information. Providing easy
access to over 50 corpora and lexical assets, such as WordNet, NLTK
presents an extensive array of text processing functionalities. These
functionalities span classification, tokenization, stemming, tagging,
parsing, and semantic analysis, enhanced by integration with potent NLP
libraries and enriched by visual demos and sample datasets."
6 | # Initialization of the TfidfVectorizer
7 | tfidf_vectorizer = TfidfVectorizer()
8 | # Transformation of texts into TF-IDF vectors
9 | tfidf_vectors = tfidf_vectorizer.fit_transform([text1, text2])
10| # Calculation of the cosine similarity between the two texts
11| cosine_sim = cosine_similarity(tfidf_vectors[0],tfidf_vectors[1])
12| # Output of the cosine similarity
13| print("Kosinusähnlichkeit zwischen den beiden Texten:",
                                                    cosine_sim[0][0])
```

Listing 16: Kosinus-Ähnlichkeit ermitteln

Der Code demonstriert, wie die Kosinus-Ähnlichkeit zwischen zwei Texten unter Verwendung der TF-IDF-Vektoren berechnet wird. Nachdem die Texte in TF-IDF-Vektoren umgewandelt wurden, wird die ***cosine_similarity*** Funktion aus dem ***sklearn.metrics.pairwise*** Modul angewendet, um die Ähnlichkeit zwischen den Vektoren der beiden Texte zu messen. Die ***cosine_similarity*** Funktion berechnet den Kosinus des Winkels zwischen den beiden Vektoren, was ein Maß für ihre Ähnlichkeit ist. Ein Wert nahe 1 bedeutet eine hohe Ähnlichkeit, während ein Wert nahe 0 eine geringe Ähnlichkeit anzeigt. Die berechnete Kosinus-Ähnlichkeit wird schließlich in *Listing 17* ausgegeben, um zu zeigen, wie ähnlich die beiden Texte in Bezug auf ihren Inhalt und ihre Semantik sind.

```
1 | Kosinusähnlichkeit zwischen den beiden Texten: 0.5549721847482533
```

Listing 17: Ausgabe des Kosinus-Ähnlichkeit Codes

7. **GloVe:** steht für Global Vectors for Word Representation und ist eine Methode zur Darstellung von Wortvektoren. Im Gegensatz zu dem weit verbreiteten word2vec, das hauptsächlich auf lokalen Kontextinformationen beruht, kombiniert GloVe lokale und globale statistische Informationen durch Analyse der word-to-word co-occurrences in einem Textkorpus. Diese Methode des Lernens ohne direkte Überwachung hebt bedeutende lineare Unterstrukturen im Wortvektorraum hervor, was tiefe Einblicke in die semantische Struktur der Sprache ermöglicht. Die von Stanford vorab trainierten GloVe-Embeddings nutzen globale Aggregatstatistiken, um leistungsfähige und aussagekräftige Wortvektoren für Anwendungen wie Ähnlichkeitsbestimmung und Entitäten Erkennung zu erstellen [35, 36, 37].
   Die Vektorisierung eines Textes mit GloVe erfordert den Zugriff auf vorab trainierte GloVe-Modelle, welche Wörter in Embeddings umwandeln. Diese Embeddings repräsentieren die semantische Bedeutung der Wörter basierend auf deren Kontext innerhalb großer Textkorpora[34].

8. **Word2Vec** ist ein einfaches, aber einflussreiches Modell im Bereich der Textverarbeitung, das Wörter in Vektorräume kodiert, um die semantische Information jedes Begriffs zu erfassen. Es basiert auf neuronalen Netzwerken und nutzt unbeschriftete Trainingsdaten, um Wörter basierend auf ihrer kontextuellen Nähe in ähnliche Vektorrepräsentationen zu überführen. Dies ermöglicht es, die semantische Ähnlichkeit zwischen Wörtern durch Berechnung der Kosinus-Ähnlichkeit ihrer Vektoren zu bestimmen, wobei ähnliche Wörter nahe beieinander und unähnliche weit voneinander entfernt im Vektorraum liegen. Trotz seiner Einfachheit wird Word2Vec häufig in verschiedensten Sprachverarbeitungsaufgaben wie Sentiment-Analyse, Named Entity Recognition (NER), Part-of-Speech-Tagging und Dokumentenanalyse eingesetzt. Es dient oft als Brücke zwischen rohen Textdaten und einem für neuronale Netze geeigneteren Eingabeformat, obwohl es nicht immer im Mittelpunkt der Forschungsarbeiten steht. Dennoch hat Word2Vec insgesamt einen gewaltigen Einfluss auf das Feld [35, 38, 39].

9. **SIF:** Smooth Inverse Frequency, die Methode zur Erstellung von Texteinbettungen, weist Wörtern in einem Satz Gewichte zu und ermittelt daraus einen Durchschnittsvektor. Die Gewichtung basiert auf der geschätzten Frequenz der Wörter im Korpus, wobei häufig vorkommende Wörter weniger gewichtet werden. Nach der Berechnung des Durchschnitts wird die Variation, die durch Frequenz und Syntax bedingt und semantisch weniger relevant ist, durch Entfernen der Projektion auf den ersten Hauptkomponenten reduziert. Diese Methode verbessert die Repräsentation von Sätzen, indem sie unwichtige Informationen reduziert und ist insbesondere robust gegenüber verschiedenen Gewichtungsschemata [40, 41].

[34] GloVe, (abgerufen am 10.03.2024)

3.2.4 Auswahl des Ansatzes in Text-Vektorisierung und Ähnlichkeitsmessung

Die Entscheidung für eine geeignete Methode zur Vektorisierung von Texten ist ein wesentlicher Schritt im Prozess der Implementierung, vor allem wenn es um die Messung der Ähnlichkeit zwischen zwei Texten geht. Um die am besten geeignete Methode auszuwählen, werden verschiedene Ansätze praktisch erprobt. Diese Vorgehensweise erlaubt es, die Stärken und Schwächen jeder Methode im Kontext des spezifischen Ziels – der präzisen Messung von Textähnlichkeiten – zu evaluieren. Durch das Testen unterschiedlicher Vektorisierung-Techniken können wertvolle Einblicke in die Struktur und Eigenheiten der Textdaten gewonnen werden, was zur Optimierung der Ähnlichkeitsmessung beiträgt. Die endgültige Auswahl einer Methode zur Text-Vektorisierung erfolgt daher erst nach einer umfassenden Bewertung und dem Vergleich der Ergebnisse im Laufe der Implementierungsphase.

3.2.5 Web-Datenextraktion

In Anbetracht der Notwendigkeit, zitierte Quellen mithilfe einer Suchmaschine zu identifizieren, wurde im Rahmen der Literaturrecherche nach Tools und Bibliotheken gesucht, die eine effiziente Handhabung von HTML und Web-Scraping ermöglichen. Im Folgenden sind die wesentlichen Ergebnisse dargestellt:

1. **BeautifulSoup** ist eine leistungsfähige Python-Bibliothek, die für die Verarbeitung von HTML- und XML-Daten konzipiert ist. Sie ermöglicht es Entwicklern, durch intuitive Methoden durch den Dokumentenobjektmodell (DOM) Baum zu navigieren, Suchvorgänge durchzuführen und Daten zu modifizieren, was oft eine erhebliche Zeitersparnis bedeutet. Mit der Unterstützung verschiedener Parser, wie lxml, html5lib und anderen, bietet Beautiful Soup Flexibilität beim Parsen von Dokumenten. Entwickler können den gewünschten Parser bei der Initialisierung von Beautiful Soup einfach angeben, um spezifische Anforderungen an die Datenextraktion und -analyse effizient zu erfüllen. Dadurch eignet sich Beautiful Soup besonders gut für das Prototyping und die schnelle Datengewinnung über verschiedene Plattformen hinweg [42, 43].

2. **Scrapy**[35] ist ein in Python geschriebenes Open-Source-Framework zum Crawlen und Scrapen von Webseiten. Seit seiner ersten Veröffentlichung im Juni 2008 hat es sich kontinuierlich weiterentwickelt und wird von vielen Unternehmen für das Web-Scraping eingesetzt. Das Framework vereint verschiedene Komponenten wie einen Datenfluss-Controller, einen Scheduler für Anfragen, einen Downloader sowie benutzerdefinierte Spider-Klassen zur Analyse und Extraktion von Daten. Besonders geschätzt wird Scrapy für seine Benutzerfreundlichkeit und die aktive Open-Source-Gemeinschaft, die Unterstützung für Neueinsteiger bietet. Mit seiner Hilfe können Nutzer effizient Daten erfassen und strukturieren, wobei der Prozess

[35] docs scrapy, (abgerufen am 30.03.2024)

typischerweise in zwei Phasen unterteilt wird: Crawling und Scraping. Scrapy zeichnet sich durch seine Geschwindigkeit und Vielseitigkeit in Anwendungsbereichen wie Datengewinnung, Monitoring und automatisiertes Testen aus [44].

3. **Requests[36]:** Die Bibliothek ***`requests`*** in Python ist eine der meistgenutzten Möglichkeiten, HTTP/1.1-Anfragen zu senden. Sie erleichtert den Umgang mit Anfragen erheblich, da beispielsweise das manuelle Hinzufügen von Query-Strings zu URLs oder das Kodieren von Daten für Formulare entfällt. Einfache Methoden wie PUT oder POST und das Senden von JSON-Daten gestalten sich unkompliziert. Mit wöchentlich rund 30 Millionen Downloads und der Abhängigkeit von über einer Million Repositories auf GitHub genießt ***`requests`*** großes Vertrauen in der Entwickler-Community. Diese unter der Apache2-Lizenz stehende Bibliothek ist benutzerfreundlich und erhöht die Produktivität bei der Arbeit mit HTTP-Anfragen, indem sie eine Vielzahl von Funktionen bietet und dabei eine intuitive Handhabung ermöglicht, die der normalen Navigation im Web sehr nahe kommt [45, 46, 47].

Die Auswahl der passenden Werkzeuge wird im Implementierungskapitel anhand konkreter Anwendungsszenarien vorgenommen. Dies geschieht mit dem Ziel, ihre Angemessenheit und Effektivität im Kontext der spezifischen Projektanforderungen zu demonstrieren. Zudem ermöglicht diese Herangehensweise eine flexible Anpassung an sich verändernde Bedürfnisse und Erkenntnisse im Laufe des Projekts, um die Relevanz und Anwendbarkeit der Tools zu jeder Zeit zu sichern.

Die *Tabelle 5* liefert einen detaillierten Vergleich[37] der Stärken und Schwächen von Beautiful Soup, Scrapy und Requests.

	BeautifulSoup	**Scrapy**	**Requests**
Vorteile	Ist benutzerfreundlich, besonders geeignet für Einsteiger	Ist ein umfassendes Web-Scraping-Framework, das Werkzeuge für effizientes Datenextrahieren, Verarbeiten und Speichern bereitstellt.	Ist einfach in der Handhabung für HTTP-Anfragen
	Unterstützt diverse Parser wie lxml und bietet Flexibilität im Umgang mit HTML	Durch das asynchrone Netzwerk-Framework Twisted kann Scrapy viele Anfragen gleichzeitig verarbeiten, ideal für umfangreiche Scraping-Projekte.	Bietet umfangreiche Funktionen für Sitzungen, Cookies sowie SSL- und Proxy-Management.
	Besitzt eine leicht verständliche, umfangreiche Dokumentation	Bietet Unterstützung für diverse Middlewares und Erweiterungen für erweiterte Funktionen wie Cookie-Management und User-Agent-Fälschung.	Profitiert von einer aktiven Community und detaillierter Dokumentation.

[36] pypi requests, (abgerufen am 30.03.2024)
[37] scrapy vs beautifulsoup vs requests, (abgerufen am 30.03.2024)

Nachteile	Dient lediglich dem Parsing, ohne HTTP-Anfragen oder weitere Web-Scraping-Funktionen zu unterstützen.	kann aufgrund seines Umfangs und der Lernkurve einschüchternd wirken.	bearbeitet nur HTTP-Anfragen, ohne HTML-Parser oder JavaScript-Handling
	Setzt auf externe Bibliotheken wie Requests für Anfragen.	Für einfache Scraping-Projekte kann Scrapy mehr bieten als nötig.	Für aufwendige Scraping-Vorhaben mit JavaScript oder asynchronen Anfragen ist es weniger geeignet.

Tabelle 5: Vergleich der Web-Datenextraktion-Werkzeuge

4 Konzept

In diesem Kapitel wird das vorgeschlagene Konzept dargestellt, was in späteren Phasen implementiert und umgesetzt wird. Mit dem Ziel, ein fundiertes Verständnis der Absichten und Erwartungen in Bezug auf die Ergebnisse zu erhalten. Jede Phase wird beschrieben und ihre Bedeutung im Gesamtkontext erläutert. Durch diese Herangehensweise wird angestrebt, die strategische Planung und die damit verbundenen Ziele transparent zu machen, um die Wichtigkeit jedes Schrittes im Entwicklungsprozess zu unterstreichen.

Die untenstehende *Abbildung 1* stellt den gesamten Ablauf der Konzeption dar, beginnend bei den Datenquellen bis hin zur Messung der Ähnlichkeit. Zunächst wird der Text aus der studentischen Arbeit (später auch als Hauptarbeit bezeichnet) extrahiert, woraufhin Zitate erkannt und herausgefiltert werden. Das Literaturverzeichnis wird daraufhin identifiziert und gesichert. Das erkannte Zitat wird im Literaturverzeichnis gesucht, um die dazugehörige Quellenangabe zu finden. Diese identifizierte Quelle wird anschließend mittels einer Suchmaschine recherchiert und im besten Fall gefunden. Mithilfe von Information Retrieval-Techniken werden der zitierte Text und der gefundene Text miteinander verglichen. Zum Schluss wird der Grad der Übereinstimmung beider Texte ermittelt.

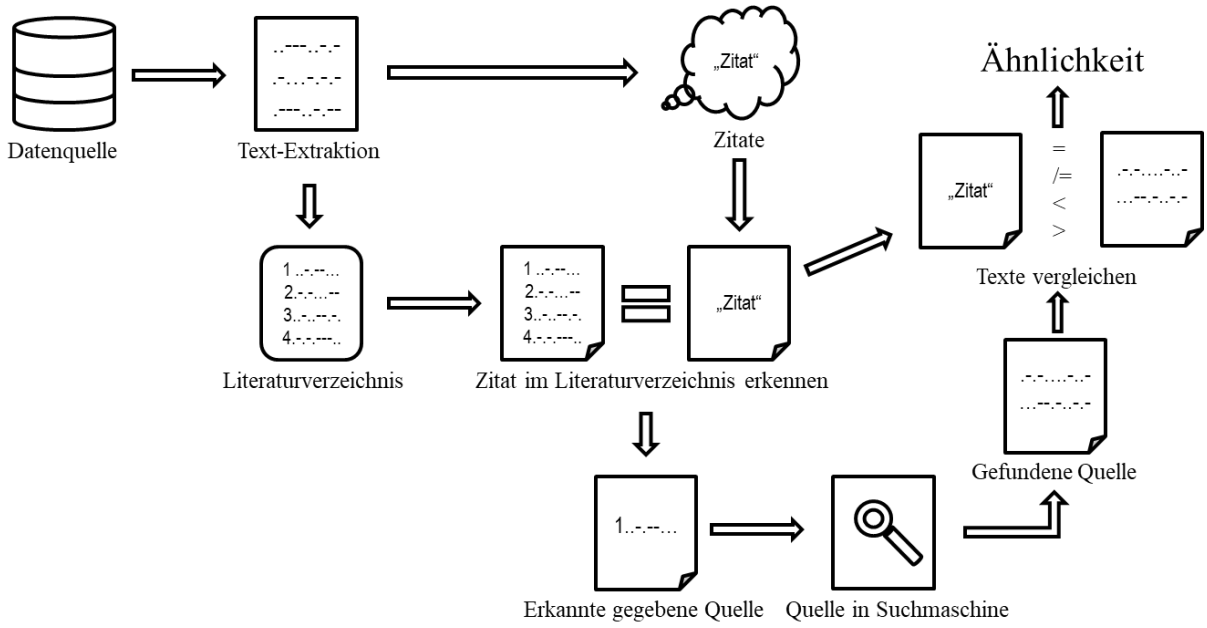

Abbildung 1: Prozessablauf der automatisierten Prüfung von Quellenangaben

4.1 Datenvorverarbeitung

In der Vorbereitungsphase geht es darum, Textmaterial aus der Hauptarbeit und den zugehörigen Referenzarbeiten für analytische Verfahren aufzubereiten. Durch eine umfassende Reinigung werden störende Bestandteile entfernt und bereitet den Text in einer detailliert strukturierten Weise für die Bearbeitung mit Natural Language Processing (NLP) vor.

4.1.1 Ziel

Das Ziel dieses Schrittes, was in *Abbildung 2* dargestellt ist, besteht darin, die Textdaten sowohl aus der Hauptarbeit als auch aus den Referenzarbeiten für die weitere Analyse vorzubereiten. Dies beinhaltet das gründliche Bereinigen des Textes, um irrelevante Elemente zu entfernen, die die Analyse stören könnten. Anschließend erfolgt die Tokenisierung des Textes, bei dem der Text in kleinere Einheiten zerlegt wird und das Umwandeln in ein für NLP-Aufgaben geeignetes Format. Schließlich wird der vorbereitete Text in ein Format konvertiert, das für die Verarbeitung durch NLP-Tools (Natural Language Processing) geeignet ist.

Abbildung 2: Schritte der Datenvorverarbeitung

4.1.2 Methodik

Im Rahmen der methodischen Herangehensweise wird der Fokus auf der systematischen Aufbereitung und Analyse von Textdaten aus Haupt- und Referenzarbeiten. Durch den Einsatz spezialisierter Techniken für Textextraktion, -bereinigung, Tokenisierung und Vektorisierung sollen die Textdaten optimal für die Anwendung von Technologien des Natural Language Processing (NLP) aufbereitet werden. Diese vorbereitenden Schritte sind entscheidend, um die Effizienz der Datenanalyse zu steigern und tiefgehende Einblicke in das analysierte Textmaterial zu ermöglichen:

Textextraktion:

- Der Text wird aus der Hauptarbeit und den Referenzarbeiten mittels ***PyPDF2*** gelesen und extrahiert

Textbereinigung:

- Um die Qualität der Datenanalyse zu erhöhen und mittels ***NLTK***-Bibliothek, werden neben Sonderzeichen und Satzzeichen auch nicht-alphanumerische Symbole und Formatierungen aus dem Text entfernt. Diese Bereinigung dient dazu, den Text von unnötigem Rauschen zu befreien und seine Verarbeitbarkeit zu verbessern.
- Durch das Entfernen von Stoppwörtern (wie "the", "and", "is" usw.) konnte der Fokus gezielt auf die für die Analyse relevanten Inhalte gerichtet werden, was eine effizientere Verarbeitung und tiefere inhaltliche Einsichten ermöglichen.

Tokenisierung:

- Die Tokenisierung des Textes – mittels ***NLTK***-Bibliothek – in einzelne Wörter oder Phrasen ist ein kritischer Schritt, um die Textdaten für komplexere NLP-Operationen vorzubereiten.

Textvektorisierung:

- Durch die Anwendung von Textvektorisierungsmethoden wie TF-IDF (Term Frequency-Inverse Document Frequency) oder Wort-Einbettungen (wie Word2Vec, GloVe) wird der tokenisierte Text in eine numerische Form überführt, die maschinelles Lernen und tiefergehende Textanalysen unterstützt.
- Diese Vektoren repräsentieren die Bedeutung der Wörter oder Phrasen im Kontext des gesamten Textkorpus und ermöglichen es, Muster und Beziehungen im Text auf eine Weise zu erkennen, die für Computer verständlich ist.

4.1.3 Ergebnisse

Die Textdaten sowohl aus der Hauptarbeit als auch aus den Referenzarbeiten wurden erfolgreich vorverarbeitet und in ein Format überführt, das für die Analyse mittels Natural Language Processing (NLP) geeignet ist. Diese Vorverarbeitung umfasste das Bereinigen von unnötigem Rauschen im Text, wie zum Beispiel das Entfernen von Sonderzeichen und Stoppwörtern, sowie die Tokenisierung und Vektorisierung der Textinhalte. Durch diese Aufbereitung soll eine solide Grundlage für die Anwendung fortschrittlicher NLP-Techniken geschafft werden, die darauf abzielen, tiefergehende Einblicke in die Inhalte und thematischen Strukturen der analysierten Texte zu gewinnen.

4.2 Extraktion von Zitaten

In diesem Abschnitt wird der Prozess beschrieben, bei dem Zitate aus der Hauptarbeit identifiziert und extrahiert werden. Die herausgefilterten Zitate bilden die Basis, um die zugehörigen Referenztexte für spätere Vergleiche aufzuspüren. Diese methodische

Herangehensweise trägt zur möglichen automatisierten Überprüfung der Quellenverweise in studentischen Arbeiten. Sie ermöglicht ebenfalls eine Beurteilung der Relevanz der zitierten Quellen.

4.2.1 Ziel

Das primäre Ziel dieses Schrittes, was in *Abbildung 3* dargestellt ist, besteht darin, Zitate innerhalb der Hauptarbeit zu erkennen und herauszufiltern. Diese identifizierten Zitate dienen als Schlüssel, um die dazugehörigen Referenzarbeiten aufzufinden, welche in nachfolgenden Phasen des Prozesses für detaillierte Vergleiche herangezogen werden. Diese Extraktion ermöglicht eine mögliche automatisierte Prüfung der Quellenangeben in den studentischen Arbeiten, die in der Hauptarbeit genutzt wurden. Durch die sorgfältige Analyse und den Abgleich mit den Referenzarbeiten kann auch die Relevanz der zitierten Literatur bewertet werden.

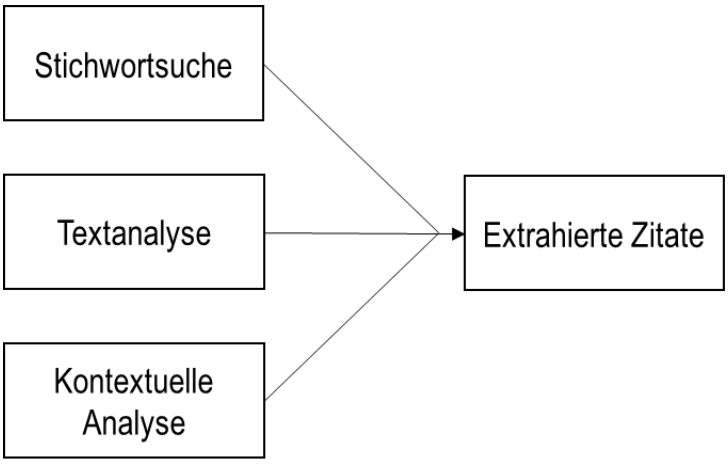

Abbildung 3: Extraktion der Zitate

4.2.2 Methodik

Bei der Untersuchung studentischer Arbeiten ist die genaue Identifikation von Zitaten essenziell. Durch die Anwendung von den unten gegliederten Methoden wie der Stichwortsuche, dem Text-Parsing und einer kontextbezogenen Analyse gelingt es, Zitate gezielt aus dem Haupttext zu isolieren und für nachfolgende Analysen bereitzustellen. Diese Techniken erlauben eine genaue Lokalisierung von Zitaten und ermöglichen es, deren Kontext zu berücksichtigen.

Stichwortsuche:

- Für die Suche nach Zitaten in der Hauptarbeit werden gängige Zitationsmuster, wie zum Beispiel numerische Kennzeichnungen, die in eckigen Klammern stehen (z.B. IEEE-Zitierstil: "[1]", "[2]"), als Suchbegriffe verwendet. Diese Methode zielt darauf ab, eine effiziente und zielgerichtete Identifizierung von Verweisen innerhalb der Arbeit zu ermöglichen.

- Um diese spezifischen Zitationsmuster präzise aus dem Text herauszufiltern, kommen reguläre Ausdrücke ***re*** zum Einsatz. Diese Werkzeuge der Textverarbeitung erlauben es, komplexe Suchanfragen zu formulieren, die genau auf die charakteristischen Merkmale der Zitierweise abgestimmt sind. Die Verwendung regulärer Ausdrücke unterstützt somit nicht nur die Identifikation von Zitaten, sondern auch deren Extraktion aus dem Fließtext, was eine saubere und strukturierte Erfassung der referenzierten Quellen gewährleistet.

Text-Parsing:

- Für die Extraktion von Zitationsinformationen aus der Hauptarbeit kommen Text-Parsing-Techniken zum Einsatz. Dies umfasste die Nutzung regulärer Ausdrücke und spezifischer Parsing-Regeln, die auf die verschiedenen Zitierstile abgestimmt sind. Diese Methoden ermöglichen eine präzise Identifizierung und Extraktion von Zitaten direkt aus dem Fließtext, indem sie spezifische Muster und Strukturen, die für Zitate charakteristisch sind, zielgerichtet suchen und verarbeiten.
- Um eine umfassende Analyse und einen Vergleich der Zitate zu ermöglichen, wurden diese nicht isoliert extrahiert, sondern zusammen mit dem umgebenden Kontext. Dieser Schritt ist entscheidend, um zusätzliche Einblicke in die Verwendung und Bedeutung der Zitate innerhalb der Hauptarbeit zu gewinnen. Der erfasste Kontext umfasst typischerweise den Satz oder Absatz, in dem das Zitat platziert ist, und bietet zusätzliche Informationen über den Zusammenhang, in dem die referenzierte Quelle diskutiert oder erwähnt wird.

Kontextuelle Analyse:

- Um die Zuverlässigkeit der Zitatidentifizierung zu erhöhen, wird der Kontext potenzieller Zitate analysiert. Diese Vorgehensweise dient dazu, die Wahrscheinlichkeit von falsch positiven Ergebnissen zu minimieren und die Genauigkeit der erfassten Zitate zu steigern. Durch die Betrachtung des textuellen Umfelds konnte besser unterschieden werden, ob es sich bei einem Element tatsächlich um ein Zitat handelt oder nicht.
- Phrasen, die auf die Einführung einer neuen Quelle oder die Diskussion verwandter Arbeiten hinweisen, wurden als potenzielle Zitationspunkte betrachtet.
- Phrasen werden betrachtet, die auf die Einführung einer neuen Quelle hinweisen oder die eine Diskussion über verwandte wissenschaftliche Arbeiten einleiten. Solche Ausdrücke werden als Indikatoren für potenzielle Zitationspunkte gewertet. Die Identifizierung dieser spezifischen Phrasen unterstützt die präzise Extraktion von Zitaten, indem sie sicherstellt, dass nur relevante und tatsächlich auf Quellen verweisende Textstellen berücksichtigt werden.

4.2.3 Ergebnisse

Diese Phase soll folgende Ergebnisse liefern:

- Durch die Anwendung einer kombinierten Methodik, die sowohl Schlüsselwortsuche als auch Text-Parsing-Techniken umfasst, konnten Zitate erfolgreich aus der Hauptarbeit extrahiert werden. Diese Herangehensweise ermöglichte eine effiziente und präzise Identifizierung der Referenzen innerhalb des Dokuments.
- Die extrahierten Zitate beinhalten sowohl direkte Zitate, die typischerweise durch numerische Kennzeichnungen in eckigen Klammern (z.B. "[1]", "[2]") markiert sind, als auch kontextuelle Zitate, die zusammen mit dem umgebenden Text erfasst wurden, was zusätzliche Informationen für die Analyse und den Vergleich bereitstellt.

4.3 Zitierte Quelle suchen

In diesem Abschnitt wird der Prozess erläutert, zitierte Quellen aus der Hauptarbeit identifiziert und anschließend durch den Einsatz einer Suchmaschine recherchiert werden. Diese identifizierten Quellen werden daraufhin als fundamentale Basis genutzt, um sowohl die Quellen selbst als auch die entsprechenden Zitate aus der Hauptarbeit umfassend für die anschließende Ähnlichkeitsanalyse vorzubereiten. Dieser Schritt ist entscheidend, um die inhaltliche Zusammenhang zwischen den Zitaten und ihren Quellen präzise zu bewerten.

4.3.1 Ziel

Das Hauptziel dieses Schrittes, wie in *Abbildung 4* dargestellt, ist es, nach den zuvor identifizierten zitierten Quellen mittels einer Suchmaschine zu suchen, um relevante Daten für die Ähnlichkeitsanalyse zu extrahieren. Diese Daten können von PDF-Dateien der zitierten Publikationen bis hin zu Abstrakt, Autorennamen oder Titeln der Arbeiten reichen. Die gezielte Suche und Extraktion dieser Informationen ermöglicht eine fokussierte und effiziente Vorbereitung für die anschließende Analyse der Zitate im Kontext der Hauptarbeit.

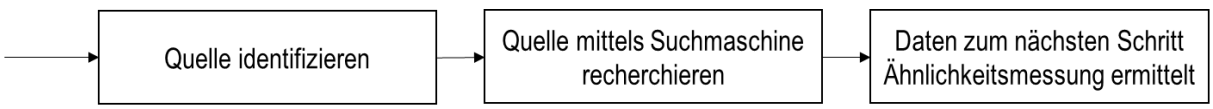

Abbildung 4: Schritte zur Suche nach zitierten Quellen

4.3.2 Methodik

Im vorherigen Schritt wurden bereits Zitate identifiziert, die auf Einträge im Literaturverzeichnis verweisen. Die nachstehenden Methoden ermöglichen die Erkennung der entsprechenden Quellen sowie die gezielte Suche nach diesen identifizierten Quellen:

- Die im vorigen Schritt identifizierten Zitate geben spezifische Hinweise auf ihre Quellen im Literaturverzeichnis, die eine detaillierte Analyse ermöglichen. Diese Phase umfasst das gründliche Untersuchen der Quellenangaben, um wesentliche Informationen wie den Namen des Autors, den Titel der Arbeit und gegebenenfalls das Erscheinungsjahr herauszuarbeiten.
- Durch die Zuordnung der Zitate zu ihren entsprechenden Quellen kann eine strukturierte Datenbasis erstellt werden. Diese ermöglicht eine gezielte Überprüfung der Genauigkeit und Vollständigkeit der Zitierungen und eröffnet die Möglichkeit, eine effiziente Identifizierung und Korrektur potenzieller Fehler oder Unstimmigkeiten in den Zitaten vorzunehmen, die durch Tippfehler oder versehentliche Ungenauigkeiten entstanden sein könnten, aufgedeckt und behoben werden.
- Es wird dann basierend auf der Zuordnung der Zitate zu ihren jeweiligen Quellen, die gezielte Suche nach diesen Quellen über Suchmaschinen eingeleitet. Ziel ist es, Zugang zu den vollständigen Dokumenten oder zu spezifischen Daten wie Abstrakt, Volltexten oder bibliographischen Informationen zu erhalten. Die Daten werden mittels spezialisierter Werkzeuge und Methoden für die weitere Analyse extrahierten. Dieser Schritt ist entscheidend für die Ermittlung der inhaltlichen Übereinstimmung zwischen den Zitaten und den Originalquellen.
- Die aus den Quellen extrahierten Informationen, darunter Abstrakt, Autorennamen oder Schlüsselbegriffe, dienen als robuste Basis für die nachfolgende Überprüfung von Ähnlichkeiten. Sie ermöglichen einen umfassenden und detaillierten Vergleich zwischen der studentischen Arbeit und den zitierten Referenzen, indem sie essenzielle Datenpunkte für die Analyse bereitstellen.

4.3.3 Ergebnisse

Diese Phase soll folgende Ergebnisse liefern:

- Jedes Zitat wurde erfolgreich seiner referenzierten Quelle zugeordnet. Diese Zuordnung ermöglicht die Analyse und Bewertung der Zitiergenauigkeit.
- Die zugehörigen Quellen wurden mittels einer Suchmaschine lokalisiert und identifiziert, was den Prozess der Datenbeschaffung automatisiert und beschleunigt.
- Von den gefundenen Quellen wurden relevante Informationen wie Abstrakt, Autorennamen und weitere essenzielle Details extrahiert. Diese Informationen

dienen als Basis für die anschließende Ähnlichkeitsprüfung, um die Übereinstimmung zwischen den Zitaten in der studentischen Arbeit und den Angaben in den referenzierten Quellen zu bewerten.

4.4 Ähnlichkeitsmessung

In dieser Untersuchungsphase liegt die sorgfältige Bewertung der Übereinstimmung zwischen den in der Hauptarbeit zitierten Textstellen und ihren jeweiligen Quellen. Durch die initiale Aufbereitung der Texte und die folgende Analyse der Ähnlichkeitswerten wird das Fundament für eine genaue Überprüfung der Zitierqualität gelegt.

4.4.1 Ziel

Das primäre Ziel dieses Schrittes, was in *Abbildung 5* dargestellt ist, besteht darin, die Ähnlichkeit zwischen den aus der Hauptarbeit extrahierten Zitaten und den zugehörigen Referenzarbeiten zu bewerten. Hierfür werden die Texte zunächst vorverarbeitet, um eine einheitliche Basis für die Analyse zu schaffen. Anschließend erfolgt die Berechnung von Ähnlichkeitswerten mittels spezifischer Methoden, die sowohl direkte als auch inhaltliche Übereinstimmungen berücksichtigen. Abschließend wird beurteilt, ob die Zitate ihre Quellen angemessen repräsentieren, indem die Ähnlichkeitswerte analysiert werden, um die Genauigkeit und Angemessenheit der Verweise in der Hauptarbeit zu bestimmen.

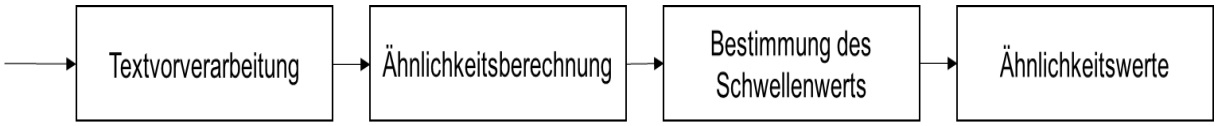

Abbildung 5: Schritte der Ähnlichkeitsmessung

4.4.2 Methodik

Durch die Entfernung von irrelevanten Elementen und die Anwendung von Tokenisierung wird das Fundament für eine differenzierte Analyse der Textähnlichkeit gelegt. Ein wesentliches Merkmal dieser Phase ist auch die Ermittlung eines Schwellenwerts für die Ähnlichkeitsbewertung, der basierend auf empirischen Erkenntnissen oder durch spezifische Validierungstechniken festgelegt wird, und somit eine fundierte Einschätzung der Präzision in der Zitierung ermöglicht. Die vorgesehenen Methoden sind wie folgt definiert:

Textvorverarbeitung:

- Im ersten Schritt der Textvorverarbeitung werden der Zitattext und der Referenztext bereinigt, um jegliche Form von Störgeräuschen zu eliminieren. Dies umfasst das Entfernen von Sonderzeichen, Satzzeichen und Stoppwörtern, die für die Analyse der inhaltlichen Ähnlichkeit irrelevant sind.
- Anschließend kommen Tokenisierung-Techniken zum Einsatz, um die vorverarbeiteten Texte in einzelne Einheiten, sogenannte Token, zu zerlegen. Diese Token repräsentieren in der Regel Wörter oder Phrasen und sind essenziell für die Durchführung detaillierter textbasierter Analysen. Dadurch wird auch eine strukturierte Form erreicht, die sowohl die Verarbeitung durch Algorithmen der Ähnlichkeitsmessung erleichtert als auch eine präzise Untersuchung der textuellen Übereinstimmungen zwischen Zitaten und Referenztexten ermöglicht.

Ähnlichkeitsberechnung:

- Nach der Vorverarbeitung der Texte wurden Ähnlichkeitswerte zwischen dem Zitattext und dem Referenztext ermittelt, indem passende Ähnlichkeitsmetriken angewendet werden. Diese Berechnung ist entscheidend, um zu bestimmen, wie ähnlich die Inhalte der Zitate den Originaltexten der Referenzen sind.
- Für die Ermittlung der Ähnlichkeit kommen verschiedene anerkannte Maße zum Einsatz. Die Kosinus-Ähnlichkeit und die Jaccard-Ähnlichkeit werden als mögliche Methoden herangezogen. Zusätzlich werden einbettungsbasierte Methoden wie Word2Vec berücksichtigt. Diese Methoden ermöglichen eine tiefgreifende Analyse der Textähnlichkeit, indem sie nicht nur die Präsenz gleicher Wörter, sondern auch die Bedeutungskontexte und die Beziehungen zwischen den Wörtern in Betracht ziehen.

Bestimmung des Schwellenwerts:

- Um zu entscheiden, ob ein Zitat seine Quelle in einer angemessenen Weise referenziert, wird ein spezifischer Schwellenwert für die Ähnlichkeitswerte definiert. Diese Vorgehensweise ermöglicht eine objektive Beurteilung, indem sie klare Kriterien dafür vorgibt, ab welchem Punkt die Übereinstimmung zwischen Zitat und Referenztext als ausreichend betrachtet wird.
- Die Festlegung dieses Schwellenwertes erfolgt auf empirischem Wege oder durch die Anwendung von Validierungsverfahren auf einen Datensatz, dessen tatsächliche Übereinstimmungen bekannt sind. Dieser Prozess gewährleistet, dass der Schwellenwert nicht nur theoretisch fundiert, sondern auch praktisch erprobt ist und somit eine zuverlässige Grundlage für die Bewertung der Zitatgenauigkeit bietet.

4.4.3 Ergebnisse

Diese Phase soll folgende Ergebnisse liefern:

- Die Berechnung der Ähnlichkeitswerte zwischen den aus der Hauptarbeit extrahierten Zitaten und den dazugehörigen Referenztexten wurde erfolgreich durchgeführt. Diese quantitative Analyse ermöglichte eine objektive Bewertung der Übereinstimmung zwischen den Zitaten und ihren Quellen, basierend auf den zuvor festgelegten Ähnlichkeitsmetriken.
- Jedes Zitat wurde anhand des vordefinierten Schwellenwerts bewertet, um festzustellen, ob es angemessen auf seine Quelle verweist.
- Anschließend wurde jedes einzelne Zitat mit dem vorab definierten Schwellenwert verglichen, um festzustellen, ob es die Quelle in einer wissenschaftlich angemessenen Weise referenziert. Dieser Schritt ist entscheidend, um die Qualität der Zitatverweise zu überprüfen und sicherzustellen, dass die Zitate tatsächlich relevante und korrekte Bezüge zu den angegebenen Quellen herstellen.

4.5 Bestimmung des Schwellenwerts

Die Definition eines Schwellenwertes für die Bewertung von Ähnlichkeiten ist ein kritischer Schritt, der darauf ausgerichtet ist, die Genauigkeit der in studentischen Arbeiten verwendeten Zitate zu bewerten. Diese Maßnahme schafft ein klares Kriterium zur Einschätzung, ob Zitate ihre Referenztexte richtig spiegeln.

4.5.1 Ziel

Das Hauptziel dieses Schrittes, was in *Abbildung 6* dargestellt ist, besteht darin, einen klar definierten Schwellenwert für die Ähnlichkeitswerte zu etablieren. Dieser Schwellenwert dient als maßgebliche Richtlinie, um zu bewerten, ob ein Zitat in der Hauptarbeit seine Quellen in einer wissenschaftlich geeigneten Weise referenziert. Durch die Festlegung dieses Schwellenwertes wird ein objektives Kriterium geschaffen, anhand dessen entschieden werden kann, ob die Übereinstimmung zwischen einem Zitat und dem dazugehörigen Referenztext ausreichend ist, um als angemessene Quellenangabe zu gelten.

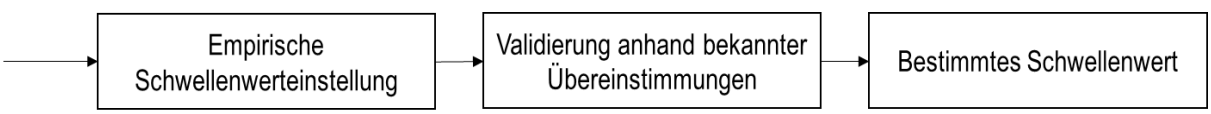

Abbildung 6: Schritte der Bestimmung des Schwellenwertes

4.5.2 Methodik

Dieser Abschnitt fokussiert auf der sorgfältigen Bestimmung und Prüfung von Schwellenwerten, die essenziell für die Evaluierung der Zitiergenauigkeit sind. Basierend auf empirischen Daten und fachspezifischem Wissen werden potenzielle Schwellenwerte ermittelt und anhand eines speziell zusammengestellten Datensatzes getestet. Dieses Vorgehen gewährleistet, dass die endgültigen Schwellenwerte eine zuverlässige Unterscheidung zwischen angemessen und unangemessen zitierten Quellen bieten. Die vorgesehenen Methoden sind wie folgt definiert:

Empirische Schwellenwertfestlegung:

- Bei der Festlegung des Schwellenwertes werden verschiedene Werte in Betracht gezogen, die auf empirischen Beobachtungen und dem spezifischen Fachwissen des jeweiligen Forschungsbereichs basieren. Diese Herangehensweise ermöglicht eine erste Einschätzung, welche Schwellenwerte realistisch sind und potenziell eine präzise Unterscheidung zwischen angemessen und unangemessen referenzierten Zitaten erlauben.
- Um die optimale Leistungsfähigkeit des Bewertungssystems zu gewährleisten, werden die ausgewählten Schwellenwerte in einem systematischen Prozess angepasst und anhand eines Datensatzes mit bekannten Übereinstimmungen überprüft. Diese methodische Vorgehensweise gewährleistete, dass der endgültig festgelegte Schwellenwert nicht nur theoretisch fundiert, sondern auch praktisch erprobt ist, um eine effektive und genaue Differenzierung bei der Bewertung der Zitate zu ermöglichen.

Validierung anhand bekannter Übereinstimmungen:

- Für die Validierung der festgelegten Schwellenwerte kann ein zusammengestellter Datensatz zum Einsatz kommen, der eine bekannte Übereinstimmungen aufweist. Dieser Datensatz umfasst eine Reihe von manuell annotierten Zitaten sowie deren zugehörige Referenztexte, was eine präzise und fundierte Grundlage für die Überprüfung bietet.
- Die Bewertung der Schwellenwerte erfolgte durch die Analyse ihrer Effektivität bei der korrekten Einordnung der Zitate. Konkret wird untersucht, inwieweit die Schwellenwerte in der Lage sind, zwischen Zitaten, die ihre Quellen angemessen referenzieren, und solchen, die dies nicht tun, zu differenzieren. Diese Vorgehensweise ermöglichte es, die Zuverlässigkeit jedes Schwellenwertes zu beurteilen und sicherzustellen, dass die gewählten Werte eine hohe Genauigkeit bei der Identifizierung korrekt referenzierter Zitate gewährleisten.

4.5.3 Ergebnisse

Diese Phase soll folgende Ergebnisse liefern:

- Basierend auf empirischen Beobachtungen und einer Validierung an einem Datensatz mit bekannten Übereinstimmungen wurde ein spezifischer Schwellenwert für die Ähnlichkeitswerte erfolgreich etabliert. Diese methodische Herangehensweise gewährleistete, dass der festgelegte Schwellenwert sowohl wissenschaftlich fundiert als auch praktisch erprobt ist.
- Der ausgewählte Schwellenwert zeigte eine hohe Effektivität bei der Differenzierung zwischen Zitaten, die ihre Quellen angemessen referenzieren, und solchen, bei denen dies nicht der Fall ist. Die Validierung dieses Schwellenwerts an einem annotierten Validierungsdatensatz bestätigte seine Zuverlässigkeit und Genauigkeit.

4.6 Evaluierung und Berichterstattung

In diesem Teil der Untersuchung liegt der Fokus auf der Überprüfung und Beurteilung der Ähnlichkeitswerte, die durch die Analyse des Textes erzielt wurden. Durch diese gründliche Analyse entsteht ein detaillierter Bericht, der die allgemeinen Ergebnisse der Bewertung zusammenfasst.

4.6.1 Ziel

Das Hauptziel dieses Schrittes, was in *Abbildung 7* dargestellt ist, besteht darin, die im Rahmen der Analyse ermittelten Ähnlichkeitswerte sorgfältig zu prüfen und sie mit dem zuvor festgelegten Schwellenwert zu vergleichen. Diese Bewertung dient dazu, festzustellen, inwieweit die Zitate aus der Hauptarbeit ihre jeweiligen Quellen korrekt und angemessen referenzieren. Basierend auf dieser Evaluation wird ein detaillierter Bericht erstellt, der nicht nur die Gesamtqualität der Zitierweise innerhalb der Arbeit aufzeigt, sondern auch speziell jene Zitate markiert, bei denen die Möglichkeit besteht, dass sie nicht den wissenschaftlichen Standards der Quellenangabe entsprechen.

Abbildung 7: Schritte der Evaluierung und Berichterstattung

4.6.2 Methodik

Der Prozess teilt sich in zwei wesentliche Hauptteile: Erstens die Einschätzung der Ähnlichkeitswerte im Verhältnis zu vorab definierten Schwellenwerten und zweitens die Erstellung eines detaillierten Berichts, der die Befunde dieser Überprüfung umfassend dokumentiert.

Bewertung des Schwellenwerts:

- In diesem Schritt der Analyse wird jeder Ähnlichkeitswert, der für die aus der Hauptarbeit extrahierten Zitate und ihre jeweiligen Referenztexte ermittelt wird, sorgfältig mit dem im Vorfeld festgelegten Schwellenwert verglichen. Diese systematische Überprüfung diente dazu, eine objektive Einschätzung darüber zu treffen, inwiefern die Zitate in ihrer inhaltlichen Nähe zu den Referenzen den definierten Anforderungen entsprechen.
- Zitate, bei denen die ermittelten Ähnlichkeitswerte unterhalb des festgelegten Schwellenwertes liegen, werden als potenziell unzureichend referenziert gekennzeichnet. Diese Kennzeichnung signalisiert, dass diese spezifischen Zitate möglicherweise nicht die notwendige inhaltliche Übereinstimmung oder Angemessenheit in der Wiedergabe der Quellen aufweisen.

Berichterstellung:

- Zur Zusammenfassung und Darstellung der Ergebnisse aus der Schwellenwertbewertung wird ein detaillierter Bericht generiert. Dieser Bericht hebt insbesondere jene Zitate hervor, die den festgelegten Ähnlichkeitsschwellenwert nicht erreichen konnten, und bietet somit einen klaren Überblick über potenzielle Schwachstellen in der Quellenreferenzierung innerhalb der Hauptarbeit.
- Um eine umfassende Analyse und Nachvollziehbarkeit zu gewährleisten, beinhaltet der Bericht spezifische Informationen zu jedem einzelnen Zitat, das die Kriterien nicht erfüllt. Dazu gehören die Identifikatoren der Zitate, die erreichten Ähnlichkeitswerte sowie die Bezeichnungen der jeweils zugeordneten Referenzarbeiten.

4.6.3 Ergebnisse

Diese Phase soll folgende Ergebnisse liefern:

- Im Rahmen der Bewertung wurden alle Zitate gegen den zuvor festgelegten Ähnlichkeitsschwellenwert geprüft. Dabei konnten spezifische Zitate identifiziert werden, deren Ähnlichkeitswerte unter diesem kritischen Wert lagen, was sie als potenziell unzureichend in Bezug auf das Referenzieren ihrer Quellen kennzeichnet. Diese Erkenntnis ist entscheidend, da sie direkt auf Bereiche hinweist, in denen die Genauigkeit und Vollständigkeit der Zitierpraxis möglicherweise überprüft werden muss.

- Basierend auf dieser detaillierten Analyse wurde ein umfassender Bericht generiert, der nicht nur die allgemeinen Ergebnisse der Bewertung zusammenfasst, sondern auch gezielt jene Zitate beleuchtet, die unter den festgelegten Ähnlichkeitswert gefallen sind. Der Bericht dient somit als wertvolles Werkzeug, um auf spezifische Fälle hinzuweisen, die einer genaueren Betrachtung oder Überprüfung bedürfen.

4.7 Gesamtübersicht

Die folgende *Abbildung 8* bietet einen Überblick und veranschaulicht die Verarbeitung der Zitate gemäß dem vorgeschlagenen Konzept.

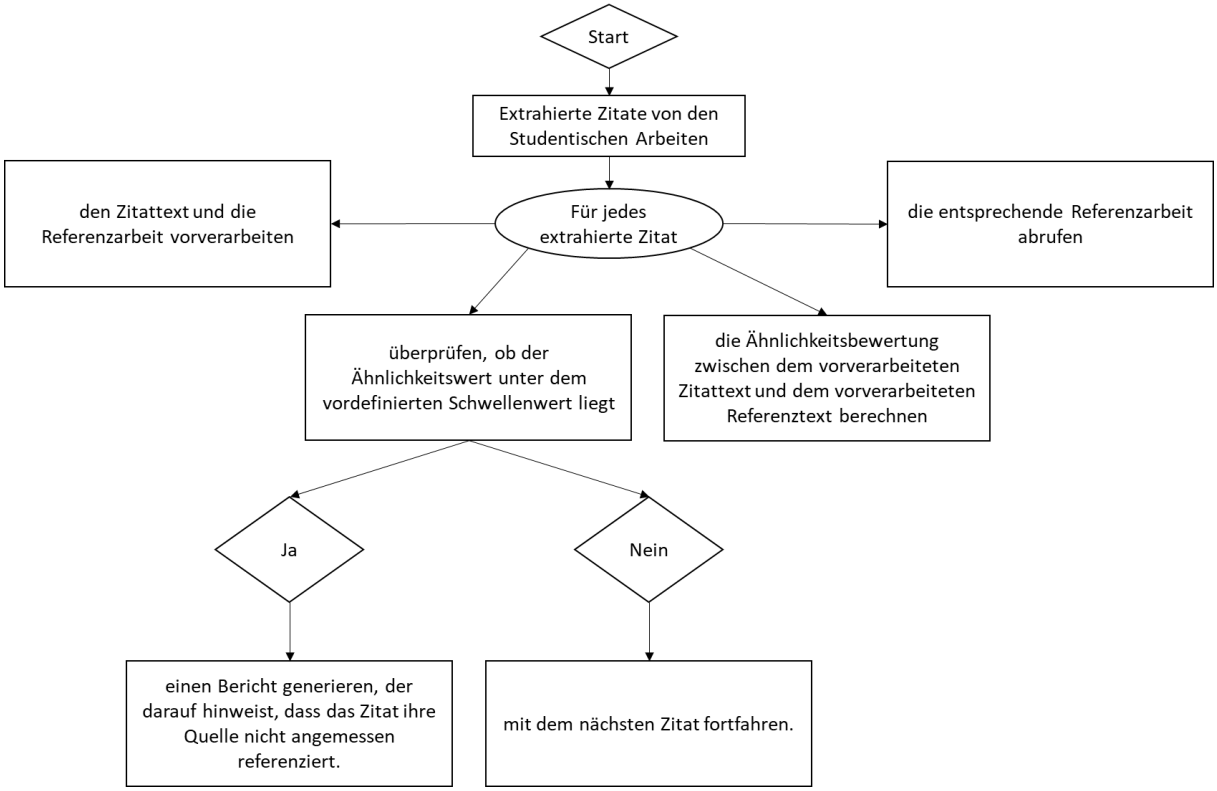

Abbildung 8: Gesamtübersicht, wie die Zitate verarbeitet werden

Die *Abbildung 8* präsentiert den Prozess der Überprüfung von Zitaten in studentischen Arbeiten. Es beginnt mit dem Startpunkt und folgt dann einem wiederholenden Zyklus, der für jedes extrahierte Zitat aus den Arbeiten durchlaufen wird. Zunächst wird die entsprechende Referenzarbeit abgerufen, danach erfolgt die Vorverarbeitung sowohl des Zitattexts als auch der Referenzarbeit. Anschließend wird eine Ähnlichkeitsbewertung zwischen dem vorverarbeiteten Zitattext und dem vorverarbeiteten Referenztext durchgeführt. Wenn der Ähnlichkeitswert unter einem

vordefinierten Schwellenwert liegt, wird ein Bericht generiert, der darauf hinweist, dass das Zitat seine Quelle nicht angemessen referenziert. Andernfalls wird mit dem nächsten Zitat fortgefahren. Dieser Prozess wiederholt sich, bis alle extrahierten Zitate geprüft werden.

Nachdem das vorgesehene Konzept zur Überprüfung von Zitaten in studentischen Arbeiten detailliert vorgestellt wurde, schließt sich nun das Kapitel der Implementierung an. wo das zuvor diskutierte Konzept praktisch umgesetzt wird. Ziel ist es, die theoretischen Grundlagen und geplanten Abläufe in einem demonstrativen Prototyp zu verwirklichen, um so die Funktionsfähigkeit und Effizienz des Ansatzes in der Praxis zu überprüfen und zu veranschaulichen.

5 Implementierung

In diesem Kapitel wird die Implementierungsphase eines Prototyps umfassend dargestellt. Hierzu gehören die detaillierte Erörterung und Analyse wichtiger Codeabschnitte sowie die Begründung der dabei getroffenen Entscheidungen. Darüber hinaus wurde der entwickelte Prototyp auf verschiedene Szenarien angewandt, um seine Anwendbarkeit und Leistungsfähigkeit in der Praxis zu überprüfen. Die daraus resultierenden Ergebnisse werden anschließend präsentiert und erläutert. Das Hauptziel dieses Kapitels ist es, die technische Umsetzung transparent zu machen und die praktische Bedeutung sowie Einsatzmöglichkeiten des Prototyps herauszustellen, was für die zukünftige Entwicklung und Nutzung in ähnlichen Kontexten wertvolle Einsichten liefert.

5.1 Anforderungen

Der entwickelte Code stützt sich auf die leistungsfähige und vielseitige Programmiersprache Python, konkret auf die Version 3.12.2. Für die Extraktion und das Lesen von Texten aus PDF-Dokumenten wird PyPDF2 in der Version 3.0.1 eingesetzt, ein Werkzeug, das den Umgang von PDF-Dateien vereinfacht. Die Verarbeitung natürlicher Sprache (NLP), ein wichtiger Aspekt dieses Projekts, erfolgt mithilfe von NLTK 3.8.1, einer umfassenden Bibliothek, die eine Vielzahl von Werkzeugen für die Analyse menschlicher Sprache bietet. Für die spezifische Aufgabe Erkennung von Zitaten in studentischen Arbeiten und die Erkennung bestimmter Abschnitte aus der referenzierten Arbeiten kommen reguläre Ausdrücke zum Einsatz, realisiert durch die Bibliothek regex in der Version 2023.12.25, die eine präzise und flexible Methode zur Textmustererkennung ermöglicht. Um die semantische Ähnlichkeit zwischen Texten zu bewerten, werden die fortschrittlichen Fähigkeiten von scikit-learn in der Version 1.4.1.post1 und gensim in der Version 4.3.2 genutzt. Diese Bibliotheken stellen Methoden zur Text-Vektorisierung und -Analyse bereit, die es ermöglichen, tiefergehende Einblicke in die Bedeutung der untersuchten Texte zu gewinnen. Darüber hinaus wird Kaggle in der Version 1.6.8 als Ressource für das Herunterladen vortrainiertes Modell verwendet, der mithilfe von gensim weiterverarbeitet werden kann. Zusätzlich wird beautifulsoup4 in der Version 4.12.3 und requests in der Version 2.31.0 für das Web Scraping und das Abfragen von Webseiten verwendet. Die Installation dieser Bibliotheken kann einfach über das Paketverwaltungstool ***pip*** erfolgen.

Das erforderliche vortrainierte Word2Vec-Modell kann über einen spezifischen Link[38] heruntergeladen werden. Für das Herunterladen ist es erforderlich, zuerst ein Konto auf der entsprechenden Plattform zu erstellen und einen persönlichen API-Schlüssel zu generieren. Das vortrainierte Word2Vec-Modell kann aber auch mit dem Code, wie in *Listing 18* veranschaulicht, heruntergeladen werden.

[38] kaggle word2vec-pretrained, (abgerufen am 15.03.2024)

```
# Importing necessary Python libraries
import os
import json
# Creating the Kaggle directory
kaggle_path = os.path.join(os.path.expanduser('~'), '.kaggle')
os.makedirs(kaggle_path, exist_ok=True)
# Copying the content of kaggle.json into the .kaggle directory
# Replace YOUR_KAGGLE_KEY and YOUR_KAGGLE_USERNAME with your Kaggle API keys
kaggle_key_content = {
    "username": "YOUR_KAGGLE_USERNAME ",
    "key": "YOUR_KAGGLE_KEY "
}
with open(os.path.join(kaggle_path, 'kaggle.json'), 'w') as file:
    json.dump(kaggle_key_content, file)
kaggle datasets download -d pkugoodspeed/nlpword2vecembeddingspretrained
```

Listing 18: Word2Vec-Modell herunterladen

Das Modell wird als ZIP-Datei heruntergeladen und kann mit dem folgenden Code entpackt werden:

```
from zipfile import ZipFile
zip_file_path = 'nlpword2vecembeddingspretrained.zip'
extract_to_directory = '.'
with ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_to_directory)
print("ZIP-Datei erfolgreich entpackt.")
```

Listing 19: ZIP-Datei entpacken

5.2 Lesen der Daten

Im folgenden Code dient Die Funktion ***read_pdf_paper(file_path)*** dem Einlesen und Extrahieren von Textdaten aus PDF-Dokumenten. Hierfür wird die Bibliothek PyPDF2 genutzt, um den Inhalt jeder Seite der PDF-Datei zu erfassen. Die Funktion durchläuft alle Seiten, extrahiert den Text und fügt ihn zu einem durchgehenden Text zusammen, der den gesamten Inhalt des Dokuments repräsentiert. Abschließend gibt sie den vollständigen, extrahierten Text zurück.

```
import PyPDF2
def read_pdf_paper(file_path):
    with open(file_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        paper_text = ""
        for page_number in range(len(reader.pages)):
            page = reader.pages[page_number]
            paper_text += page.extract_text()
    return paper_text
paper_file_path = 'desired file.pdf'
paper_text = read_pdf_paper(paper_file_path)
```

Listing 20: Text aus PDF-Datei lesen und extrahieren

5.3 Extraktion von Zitaten und Abschnitte

In diesem Prozess wird zwischen zwei Arten der Extraktion unterschieden: Zum einen die Extraktion von Zitaten aus der Hauptarbeit (studentischen Arbeiten) und zum anderen die Extraktion von Textabschnitten aus den referenzierten Arbeiten.

5.3.1 Erkennen und Extrahieren der Zitate

Die Funktion ***extract_referenced_text*** dient dazu, Textstellen aus dem Haupttext einer studentischen Arbeit zu extrahieren, die durch bestimmte Zitatnummern referenziert werden.

```
1 | def extract_referenced_text(main_paper_text):
2 |     referenced_texts = {}
3 |     # Regular expression pattern to match reference numbers like [1],[2],
      etc.
4 |     ref_pattern = r'\[\d+\]\. (.+?)(?=\[\d+\]\.|$)'
5 |     # Tokenize text into sentences using NLTK
6 |     sentences = nltk.sent_tokenize(main_paper_text)
7 |     # Flag to indicate whether we're within the REFERENCES section
8 |     in_references_section = False
9 |     # Iterate over the sentences and extract referenced text
10|     for i in range(len(sentences)):
11|         sentence = sentences[i]
12|         # Check if the sentence indicates the start of the REFERENCES
          section
13|         if "REFERENCES" in sentence or "References" in sentence:
14|             in_references_section = True
15|             break
16|         # If not within the REFERENCES section, extract referenced text
17|         if not in_references_section:
18|             # Find the reference number in the sentence
19|             reference_match = re.search(r'\[(\d+)\]\.', sentence)
20|             if reference_match:
21|                 reference_number = reference_match.group(1)
22|                 # Find the last non-empty sentence before the reference
                  number
23|                 for j in range(i-1, -1, -1):
24|                     if sentences[j].strip():
25|                         last_sentence = sentences[j+1].strip()
26|                         break
27|                 # Store referenced text in the dictionary
28|                 if reference_number in referenced_texts:
29|                 referenced_texts[reference_number] += ' ' + last_sentence
30|                 else:
31|                     referenced_texts[reference_number] = last_sentence
32|     return referenced_texts
```

Listing 21: Zitate, die nach IEEE-Stil zitiert sind, extrahieren

Der Code definiert eine Funktion namens ***extract_referenced_text***, die dazu dient, Textstellen aus der Hauptarbeit zu extrahieren, die über Zitatnummern referenziert werden. Die Funktion nimmt den Haupttext der Hauptarbeit entgegen und initialisiert ein leeres Dictionary ***referenced_texts***, um die referenzierten Texte und deren

Zitatnummern zu speichern. Mit einem regulären Ausdruck ***ref_pattern*** sucht sie nach Zitatnummern und dem dazugehörigen Text. Der Haupttext wird durch ***nltk.sent_tokenize*** in Sätze aufgeteilt, und es wird durch ***in_references_section*** überprüft, ob der Satz den Beginn des Referenzabschnitts markiert. Ist dies der Fall, wird die Suche beendet. Andernfalls extrahiert die Funktion den Text, der direkt vor jeder gefundenen Zitatnummer steht, und speichert ihn im Dictionary. Die Funktion endet mit der Rückgabe dieses Dictionary, das als einfache Möglichkeit dient, auf referenzierte Textstellen basierend auf ihren Zitatnummern zuzugreifen.

Um die Funktionsweise des Codes aus *Listing 21* zu veranschaulichen, bietet das folgende Anwendungsbeispiel einen Einblick in die erwarteten Ergebnisse.

```
# Example usage:
referenced_texts = extract_referenced_text(paper_text)
print("Referenced Texts:")
for ref_number, text in referenced_texts.items():
    print(f"Reference {ref_number}: {text}")
```

Listing 22: Anwendungsbeipiel: Extraktion von Zitaten

Der Code wurde auf die studentische Arbeit von Mohamad Algoabra [48] angewendet und erzeugte eine umfangreiche Liste von Ergebnissen. Aus Platzgründen kann jedoch nur ein Ausschnitt dieser Ergebnisse in dieser Arbeit präsentiert werden.

```
Referenced Texts:
Reference 3: Artificial Neural Network Artificial neural networks are one of the most important concepts in deep learning today, because they are so versatile and have impressive results, which makes them ideal for handling large and complex machine learning tasks such as classifying lots of images, running speech recognition services, and recommending the best videos to watch to hundreds of millions of users or even to produce new data that is very similar to real data [3].
Reference 7: The purpose of this algorithm is to minimize the loss function by iteratively updating the weights in the direction of the negative gradient [7]. So, in a single pass through a dataset of 1000 rows, SGD will update the model parameters 1000 times, compared to just once with Gradient Descent [7].
Reference 9: •AdaGrad (Adaptive Gradient): is an adaptive learning optimization algorithm that can automatically adjust the learning rate for each parameter based on its occurrence frequency, making it suitable for sparse data since it can adaptively adjust the learning rate for each parameter based on the available information [9].
Reference 12: Normalization aids in stabilizing gradient descent, facilitating the use of larger learning rates, and faster convergence of the models [12].
```

Listing 23: Ausgabe des Codes aus Listing 21

Es ist anzumerken, dass nicht alle Zitate mit dem angewendeten Verfahren extrahiert werden konnten, was auf verschiedene Faktoren zurückzuführen ist. Zum einen lieferten die Textextraktionstools PDFMiner und PyMuPDF deutlich bessere Ergebnisse, was die Bedeutung der Auswahl des richtigen Werkzeugs für die Textextraktion unterstreicht. Zum anderen kann die Formatierung der Texte während der Extraktion Veränderungen im Zitierstil bewirken, was eine Anpassung und Optimierung des Codes notwendig macht, um die Genauigkeit der Zitatextraktion zu verbessern.

5.3.2 Erkennen und Extrahieren der Textabschnitte

Die Funktion ***extract_text_from_sections*** dient dazu, Text aus bestimmten Abschnitten (z. B. ***abstract*** und ***conclusion***) der referenzierten Arbeiten.

```
1 | def extract_text_from_sections(reference_paper_text):
2 |     # Regular expression patterns to match abstract and conclusion
      sections
3 |     abstract_pattern = r'(?:Abstract\.|ABSTRACT\.|ABSTRACT|Abstract)
                          (.*?)(?=INTRODUCTION|KEYWORDS|\n\n)'
4 |     conclusion_pattern = r'CONCLUSION\s*(.*?)
                            (?=REFERENCES|ACKNOWLEDGMENT|\Z|Future work)'
5 |     # Extract abstract and conclusion sections from the reference paper
      text
6 |     abstract_matches = re.findall(abstract_pattern,
                          reference_paper_text, re.IGNORECASE | re.DOTALL)
7 |     conclusion_matches = re.findall(conclusion_pattern,
                          reference_paper_text, re.IGNORECASE | re.DOTALL)
8 |     # Initialize variable to store extracted text
9 |     extracted_text = ""
10|     # Extract text from abstract section
11|     for abstract_match in abstract_matches:
12|         extracted_text += abstract_match.strip() + "\n"
13|     # Extract text from conclusion section, stop if 'Future work' is
      encountered
14|     for conclusion_match in conclusion_matches:
15|         if 'Future work' in conclusion_match:
16|             break
17|         extracted_text += conclusion_match.strip()
18|     return extracted_text
```

Listing 24: Abschnitte aus einem Text extrahieren

Der Prozess beginnt mit der Definition von regulären Ausdrücken, die die Start- und Endpunkte dieser Abschnitte im Text identifizieren sollen. Nachdem die Muster festgelegt wurden, verwendet die Funktion die ***re.findall*** Methode, um den gesamten Referenztext nach Übereinstimmungen mit diesen Mustern zu durchsuchen. Diese Methode gibt eine Liste von Textstücken zurück, die den gefundenen Abschnitten entsprechen. Für den Abstrakt wird nach einer Zeichenkette gesucht, die mit ***Abstract*** oder ***ABSTRACT*** beginnt und bis zum nächsten Abschnitt, typischerweise ***INTRODUCTION***, oder ***KEYWORDS*** reicht. Ähnlich wird für die Zusammenfassung eine Zeichenkette extrahiert, die mit ***CONCLUSION*** beginnt und bis zu den ***REFERENCES***, ***ACKNOWLEDGMENT***, ***Future work*** oder dem Dokumentende verläuft. Eine Besonderheit hierbei ist die Berücksichtigung von ***Future work*** als Abbruchkriterium, um sicherzustellen, dass nur der tatsächliche Schlussteil und keine zukünftigen Forschungsaussichten miterfasst werden. Abschließend gibt die Funktion diesen gesammelten Text zurück, der eine zusammengeführte Darstellung der extrahierten Abstrakts und Schlussfolgerungen bietet, wobei ein Zeilenumbruch die Trennung zwischen Abstrakt und Zusammenfassung markiert.

Um die Funktionsweise des in *Listing 24* dargestellten Codes zu demonstrieren, liefert das nachstehende Anwendungsbeispiel, *Listing 25*, einen Einblick in die zu erwartenden Ergebnisse.

```
# Example usage:
text_extracted = extract_text_from_sections(paper_text)
print("Text Extracted:")
print(text_extracted)
```

Listing 25: Anwendungsbeispiel: Exrtaktion von Textabschnitte

Der Code wurde auf den Artikel [49] angewendet und extrahiert umfangreiche Textabschnitte. Aufgrund von Platzbeschränkungen wird in dieser Arbeit lediglich ein Auszug – speziell der Abschnitt des Abstrakt – dargestellt.

```
Text Extracted:
Machine learning for text classification is the cornerstone of document categorization, news filtering, document routing, and personalization. In text domains, effective feature selection is essential to make the learning task efficient and more accurate. This paper presents an empirical comparison of twelve feature selection methods (e.g. Information Gain) evaluated on a benchmark of 229 text classification problem instances that were gathered from Reuters, TREC, OHSUMED, etc. The results are analyzed from multiple goal perspectives—accuracy, F-measure, precision, and recall—since each is appropriate in different situations. The results reveal that a new feature selection metric we call 'Bi-Normal Separation' (BNS), outperformed the others by a substantial margin in most situations. This margin widened in tasks with high class skew, which is rampant in text classification problems and is particularly challenging for induction algorithms. A new evaluation methodology is offered that focuses on the needs of the data mining practitioner faced with a single dataset who seeks to choose one (or a pair of) metrics that are most likely to yield the best performance. From this perspective, BNS was the top single choice for all goals except precision, for which Information Gain yielded the best result most often. This analysis also revealed, for example, that Information Gain and Chi-Squared have correlated failures, and so they work poorly together. When choosing optimal pairs of metrics for each of the four performance goals, BNS is consistently a member of the pair—e.g., for greatest recall, the pair BNS + F1-measure yielded the best performance on the greatest number of tasks by a considerable margin.
```

Listing 26: Ausgabe des Codes aus Listing 24

5.4 Datenvorverarbeitung

Im Rahmen der Datenvorverarbeitung werden drei wesentliche Phasen durchlaufen. Diese Schritte dienen als Vorbereitung für die anschließende Messung der Ähnlichkeit zwischen der Hauptarbeit und den referenzierten Arbeiten.

5.4.1 Bereinigung eines Textes

In dem folgenden Code wird der Prozess der Datenvorverarbeitung realisiert, bei dem Sonderzeichen, Satzzeichen, nicht-alphanumerische Zeichen sowie Stoppwörter aus dem Text entfernt werden. Dieser Vorgang dient dazu, die Textdaten zu säubern und sie für die weiterführende Analyse vorzubereiten.

```
1 | def clean_text(text):
2 |     # Remove special characters, punctuation, and non-alphanumeric
      symbols
3 |     cleaned_text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
4 |     # Tokenize the text
5 |     words = nltk.word_tokenize(cleaned_text)
6 |     # Remove stopwords
7 |     stop_words = set(stopwords.words('english'))
8 |     filtered_words = [word for word in words if word.lower() not in
                                                          stop_words]
9 |     # Join the words back into a string
10|     cleaned_text = ' '.join(filtered_words)
11|     return cleaned_text
```

Listing 27: Textbereinigung

5.4.2 Tokenisierung

Der nachstehende Codeabschnitt führt eine Tokenisierung des bereinigten Textes durch. Dabei werden die Textdaten in einzelne Worte (Tokens) zerlegt, die als Basis für die weiterführende Verarbeitung im Bereich des Natural Language Processing (NLP) dienen. Dieser Schritt wandelt den Text in eine für NLP-Anwendungen zugängliche Struktur um.

```
1 | # Tokenize the cleaned text
2 | tokenized_text_extracted = nltk.word_tokenize(cleaned_text_extracted)
```

Listing 28: Tokenisierung des bereinigten Textes

5.4.3 Text-Vektorisierung

Im nächsten Codeabschnitt erfolgt die Umwandlung des tokenisierten Textes in TF-IDF-Vektoren. Jeder dieser Vektoren repräsentiert ein Dokument innerhalb des Textkorpus. Durch die Anwendung des TF-IDF-Verfahrens wird die Wichtigkeit jedes Wortes innerhalb eines Dokuments in Bezug auf den gesamten Textkorpus ermittelt, wodurch die semantische Bedeutung des Textes effektiv erfasst wird.

```
1 | from sklearn.feature_extraction.text import TfidfVectorizer
2 | # Initialize TF-IDF vectorizer
3 | tfidf_vectorizer = TfidfVectorizer()
4 | # Fit and transform the tokenized text
5 | tfidf_vectors = tfidf_vectorizer.fit_transform(tokenized_text_extracted)
```

Listing 29: Vektorisierung des tokenisierten Textes

5.5 API-Suche und Abstrakt-Retrieval

In diesem Abschnitt wurde der Fokus darauf gelegt, nach wissenschaftlichen Arbeiten zu suchen, die in Zitaten studentischer Arbeiten erwähnten Titeln entsprechen, und zwar innerhalb der DBLP-Datenbank. Zudem zielte der Prozess darauf ab, die Abstrakt dieser Arbeiten aus ihren jeweiligen URLs zu extrahieren. Hierfür wurden zwei zentrale Funktionen entwickelt: ***search_dblp***, um die Suche nach den entsprechenden Arbeiten in der DBLP-Datenbank durchzuführen, und ***get_abstract_from_url***, um die Abstrakt direkt aus den URLs der gefundenen Arbeiten zu beziehen. Dieser Code, *Listing 30* demonstriert die Implementierung der Suche in der DBLP-Datenbank und das Extrahieren von Abstrakt.

```
1 | def search_dblp(title):
2 |     base_url = "https://dblp.org/search/publ/api"
3 |     params = {
4 |         "q": title,
5 |         "format": "json"
6 |     }
7 |     response = requests.get(base_url, params=params)
8 |     data = response.json()
9 |     return data
10| def get_abstract_from_url(url):
11|     # Send a GET request to the URL
12|     response = requests.get(url)
13|     if response.status_code == 200:
14|         # Parse the HTML content of the webpage
15|         soup = BeautifulSoup(response.content, 'html.parser')
16|         # Find the element containing the abstract
17|         abstract_tag = soup.find('div', class_='abstractSection
                                                    abstractInFull')
18|         # Extract the text of the abstract if found
19|         if abstract_tag:
20|             abstract = abstract_tag.get_text(strip=True)
21|             return abstract
22|     # Return None if abstract is not found or if there's an error
23|     return None
```

Listing 30: DBLP-Datenbank suchen und Abstrakt extrahieren

Das Code, was in *Listing 30* veranschaulicht, besteht aus zwei Funktionen:

- ***search_dblp(title)***:
  - Diese Funktion nimmt einen Titel als Argument und führt eine Suche in der DBLP-Datenbank durch.
  - Sie baut eine URL für die DBLP-Such-API auf, indem sie den Titel als Suchparameter ***"q": title*** und das Format des Rückgabewerts als JSON ***"format": "json"*** angibt.
  - Ein GET-Request wird an die DBLP-API gesendet, und die Antwort wird als JSON zurückgegeben.
  - Die Antwort enthält Informationen über die Suchergebnisse, die dann von der Funktion zurückgegeben werden.
- ***get_abstract_from_url(url)***:
  - Diese Funktion erhält die URL einer spezifischen akademischen Arbeit als Argument.

- Sie sendet einen GET-Request an die übergebene URL, um die Webseite abzurufen.
- Wenn der Request erfolgreich war (HTTP-Statuscode 200), wird der HTML-Inhalt der Seite mit BeautifulSoup geparst.
- Die Funktion sucht dann nach einem ***div*** Element mit der Klasse ***'abstractSection abstractInFull'***, welches den Abstrakt der Arbeit enthält.
- Wenn ein solches Element gefunden wird, wird der Text des Abstrakt extrahiert und zurückgegeben.
- Falls kein Abstrakt gefunden wird oder ein Fehler auftritt, gibt die Funktion ***None*** zurück.

Zusammengefasst erlauben diese Funktionen, durch Angabe eines Titels nach akademischen Arbeiten in der DBLP-Datenbank zu suchen und deren Abstrakt zu extrahieren, sofern verfügbar.

Dieses Anwendungsbeispiel, *Listing 31*, demonstriert, wie man mit der zuvor definierten ***search_dblp*** Funktion nach wissenschaftlichen Arbeiten in der DBLP-Datenbank sucht und wie man mithilfe der ***get_abstract_from_url*** Funktion der Abstrakt von den URLs dieser Arbeiten extrahiert.

```
1 | # Example usage:
2 | title = "Integrating Feature and Instance Selection for Text
                                                        Classification"
3 | result = search_dblp(title)
4 | # Check if the paper is found and print its full name and abstract
5 | if 'hit' in result['result']['hits']:
6 |     first_hit = result['result']['hits']['hit'][0]
7 |     paper_info = first_hit['info']
8 |     paper_title = paper_info['title']
9 |     # paper_abstract = paper_info.get('abstract', 'N/A')
10|     paper_ee = paper_info.get('ee')
11|     print("Full Name of the Paper:", paper_title)
12|     # print("Abstract:", paper_abstract)
13|     if paper_ee:
14|         abstract_from_url = get_abstract_from_url(paper_ee)
15|         print("Abstract from URL:", abstract_from_url)
16|     else:
17|         print("No URL found to retrieve abstract.")
18| else:
19|     print("Paper not found.")
```

Listing 31: Anwendungsbeispiel: DBLP-Datenbank suchen und Abstrakt extrahieren

In *Listing 31* dient der Titel ***"Integrating Feature and Instance Selection for Text Classification"*** als Beispiel für die Suche. Zuerst wird die Funktion ***search_dblp*** mit dem Titel als Argument aufgerufen. Diese Funktion sendet eine Anfrage an die DBLP-API und erhält die Suchergebnisse zurück. Anschließend wird überprüft, ob die Suche erfolgreich war und Treffer erzielt wurden. Bei einem positiven Ergebnis werden aus dem ersten Treffer weitere Informationen wie der vollständige Titel der Arbeit extrahiert. Um das Abstrakt der gefundenen Arbeit zu erhalten, wird – sofern vorhanden – der Link zur elektronischen Ausgabe (EE) verwendet. Die Funktion ***get_abstract_from_url*** wird dann genutzt, um das Abstrakt von dieser URL abzurufen, indem der HTML-Inhalt der Webseite geparst und der Text des Abstrakts extrahiert wird. Die Ergebnisse dieses Prozesses, einschließlich des vollständigen Namens des Papiers und des Abstrakts,

werden ausgegeben. Falls keine URL vorhanden ist, um das Abstrakt abzurufen, oder kein Papier gefunden wurde, wird eine entsprechende Meldung angezeigt. In *Listing 32* ist die Ausgabe dieses Codes veranschaulicht.

```
1 | Full Name of the Paper: Integrating feature and instance selection for
text classification.
2 | Abstract from URL: Instance selection and feature selection are two
orthogonal methods for reducing the amount and complexity of data. Feature
selection aims at the reduction of redundant features in a dataset whereas
instance selection aims at the reduction of the number of instances. So far,
these two methods have mostly been considered in isolation. In this paper,
we present a new algorithm, which we callFIS(Feature and Instance Selection)
that targets both problems simultaneously in the context of text
classificationOur experiments on the Reuters and 20-Newsgroups datasets show
that FIS considerably reduces both the number of features and the number of
instances. The accuracy of a range of classifiers including Naïve Bayes, TAN
and LB considerably improves when using the FIS preprocessed datasets,
matching and exceeding that of Support Vector Machines, which is currently
considered to be one of the best text classification methods. In all cases
the results are much better compared to Mutual Information based feature
selection. The training and classification speed of all classifiers is also
greatly improved.
```

Listing 32: Ausgabe des Codes aus Listing 31

5.6 Ähnlichkeitsmessung durch Kosinus-Ähnlichkeit

In dieser Arbeit wird die Methode der Kosinus-Ähnlichkeit verwendet, um die Ähnlichkeiten zwischen zwei Textpassagen zu evaluieren: dem aus der Hauptarbeit extrahierten Text und dem aus anderen wissenschaftlichen Arbeiten referenzierten Text. Dabei wird jede Textpassage als Vektor repräsentiert, wobei jede Dimension des Vektors einem einzigartigen Begriff im gesamten Vokabular entspricht und der Wert in jeder Dimension die Häufigkeit oder die Gewichtung des entsprechenden Begriffs im Text angibt. Durch die Berechnung der Kosinus-Ähnlichkeit zwischen diesen Vektoren kann quantitativ erfasst werden, wie ähnlich sich die beiden Textsegmente inhaltlich sind. Um die optimale Strategie für die Analyse zu identifizieren, werden zwei verschiedene Ansätze untersucht. Durch den Vergleich dieser Methoden soll eine effektive und präzise Technik zur Bestimmung der Ähnlichkeit zwischen der Hauptarbeit und den referenzierten Arbeiten ausgewählt werden. Diese systematische Herangehensweise gewährleistet, dass die gewählte Methode die bestmögliche Übereinstimmung und damit die höchste Zuverlässigkeit bei der Ermittlung von Ähnlichkeiten bietet.

5.6.1 TF-IDF-Methode

Als erster Ansatz ist die Methode TF-IDF (Term Frequency-Inverse Document Frequency) wie folgt angewendet worden:

- Die Vorverarbeitung der Textdaten umfasste eine Bereinigung, Tokenisierung sowie die Entfernung von Stoppwörtern und irrelevanten Zeichen. Diese Maßnahmen gewährleisten, dass die Textdaten in einem geeigneten Format für die nachfolgende Ähnlichkeitsberechnung vorbereitet sind.
- Die tokenisierten Textdaten wurden mittels der TF-IDF-Methode (Term Frequency-Inverse Document Frequency) in numerische Vektoren konvertiert. Dabei bewertet TF-IDF die Relevanz jedes Terms sowohl aufgrund seiner Häufigkeit in einem spezifischen Dokument als auch seiner Bedeutung im gesamten Dokumentenkorpus. Dieser Schritt transformiert die Textdaten in ein für die Ermittlung der Kosinus-Ähnlichkeit passendes Format.
- Die Berechnung der Kosinus-Ähnlichkeit zwischen den TF-IDF-Vektoren der beiden Textabschnitte erfolgte mit der Funktion ***cosine_similarity*** aus dem Modul ***sklearn.metrics.pairwise***. Diese Funktion ermöglicht eine effiziente Ermittlung der Kosinus-Ähnlichkeit für Paare von Vektoren.

Dieser Code veranschaulicht die Umsetzung der TF-IDF-Methode:

```
1 | def compute_cosine_similarity(text1, text2):
2 |     # Convert tokenized text lists back to strings
3 |     text1_str = ' '.join(text1)
4 |     text2_str = ' '.join(text2)
5 |     # Store text in a list
6 |     list_text = [text1_str, text2_str]
7 |     # Initialize the TF-IDF vectorizer
8 |     vectorizer = TfidfVectorizer(stop_words='english')
9 |     # Fit and transform the text
10|     tfidf_vectors = vectorizer.fit_transform(list_text)
11|     # Extract the TF-IDF vectors for the two texts
12|     tfidf_text1, tfidf_text2 = tfidf_vectors[0], tfidf_vectors[1]
13|     # Compute the cosine similarity
14|     cosine_sim = cosine_similarity(tfidf_text1, tfidf_text2)
15|     return cosine_sim[0][0]
16| def process_text_similarity(text1, text2):
17|     # Clean the extracted text
18|     cleaned_text1 = clean_text(text1)
19|     cleaned_text2 = clean_text(text2)
20|     # Tokenize the cleaned text
21|     tokenized_text1 = nltk.word_tokenize(cleaned_text1)
22|     tokenized_text2 = nltk.word_tokenize(cleaned_text2)
23|     # Compute cosine similarity
24|     similarity_score = compute_cosine_similarity(tokenized_text1,
                                                     tokenized_text2)
25|     return similarity_score
26| similarity_score = process_text_similarity(text_extracted,
                                    referenced_texts['reference number '])
27| print("Cosine Similarity Score:", similarity_score)
```

Listing 33: TF-IDF-Methode

Der vorliegende Code besteht aus zwei Funktionen: ***compute_cosine_similarity*** und ***process_text_similarity***.

Die Funktion ***process_text_similarity(text1, text2)*** dient dazu, den Ähnlichkeitswert zwischen zwei Roh-Texten zu verarbeiten. Zuerst werden die Texte bereinigt, durch die Funktion in *Listing 27*. Anschließend werden die bereinigten Texte tokenisiert. Die tokenisierten Texte werden dann an die ***compute_cosine_similarity*** Funktion übergeben, die den Kosinus-Ähnlichkeitswert zwischen ihnen berechnet und

zurückgibt. Die Funktion ***compute_cosine_similarity(text1, text2)*** nimmt zwei Listen von tokenisierten Texten als Argumente. Sie fügt die Token jedes Textes zu einer Zeichenkette zusammen, sodass aus den Listen einfache Strings werden. Diese Strings werden in einer Liste gespeichert, auf die dann ein ***TfidfVectorizer*** angewendet wird. Der ***vectorizer*** wird mit einer Liste von Stoppwörtern initialisiert, die ignoriert werden sollen (hier: englische Stoppwörter). Der ***vectorizer*** transformiert die Liste der Textstrings in eine Matrix von TF-IDF-Features. Mit ***cosine_similarity*** wird dann die Kosinus-Ähnlichkeit berechnet. Die Funktion gibt den Ähnlichkeitswert für die beiden Texte zurück. ***process_text_similarity*** wird mit einem ***text_extracted*** (die Abschnitte aus der referenzierten Arbeiten) und einem ***referenced_texts*** (Zitat aus den studentischen Arbeit) aufgerufen, und das Ergebnis, der Ähnlichkeitswert, wird ausgegeben.

Der Code in *Listing 33* wurde auf die studentische Arbeit von Algoabra [48] angewandt, um zwei zitierte Textpassagen mit den zugehörigen Quellenangaben zu vergleichen.

Der erste zitierte Text aus Algoabras Arbeit:

> *•Least Squares GAN Loss: Least Squares GAN or LSGAN loss is a type of loss function used in generative adversarial networks, which was introduced in [29].*

zitiert Quelle in seinem Literaturverzeichnis:

> *[29] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least Squares Generative Adversarial Networks. 2017 IEEE International Conference on Computer Vision (ICCV), pages 2813–2821, 2016.*

Es wurden das folgende Ergebnis erzielt:

```
1 | Cosine Similarity Score: 0.17782646931726606
```

Listing 34: Anwendungsbeispiel des TF-IDF-Methode (erste zitierte Quelle)

Der erste zitierte Text aus Algoabras Arbeit:

> *The results we obtained are as follows: •Pix2Pix: Pix2Pix is a GAN-based model for supervised image-to-image translation introduced in [34].*
> *Moreover, dropout layers were added to the first three blocks of the decoder [34].*

zitiert Quelle in seinem Literaturverzeichnis:

> *[34] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 5967–5976, 2017.*

Es wurden das folgende Ergebnis erzielt:

```
1 | Cosine Similarity Score: 0.06310378557563243
```

Listing 35: Anwendungsbeispiel des TF-IDF-Methode (zweite zitierte Quelle)

Die erzielten Ergebnisse wiesen auf eine sehr geringe Ähnlichkeit zwischen den zitierten Texten und den entsprechenden Referenzquellen hin. Dies deutete zunächst

auf eine Störung in der Ähnlichkeitsmessung hin, da eine genauere Überprüfung zeigte, dass die Texte tatsächlich korrekt zitiert worden waren. Verschiedene Faktoren könnten zu diesem unerwartet niedrigen Ähnlichkeitswert beigetragen haben, darunter:

- Unterschiede in der Länge der beiden Textpassagen.
- Das TF-IDF-Verfahren gewichtet Wörter nach ihrer Häufigkeit und Relevanz, siehe Seite 31 Punkt 6 TF-IDF.
- Nicht ausreichende Berücksichtigung von Synonymen und semantischen Zusammenhängen, die die tatsächliche Übereinstimmung unterschätzen kann.
- Verwendung von Fachsprache in den Quellen, die in den Zitaten vielleicht einfacher oder anders formuliert wird.

Diese Ergebnisse führten zur Demonstration einer zweiten Methode, die als Nächstes betrachtet wird.

5.6.2 SIF-Methode

Die semantische Ähnlichkeit zwischen Textpassagen wurde mithilfe des Word2Vec-Modells und der Smooth Inverse Frequency (SIF),siehe Seite 33 Punkt 9 SIF, Methode berechnet. Word2Vec, siehe Seite 33 Punkt 8 Word2Vec, wandelt Wörter in Vektoren um und SIF generiert daraus Texteinbettungen, wodurch die Bedeutung des Textes erfasst wird. die Methode ist wie folgt angewendet worden:

1. Das genutzte vortrainierte Word2Vec-Modell wurde über die Kaggle-Plattform heruntergeladen. Dieses Modell wurde auf einem umfangreichen Korpus von Google News-Artikeln trainiert und bietet Worteinbettungen mit einer Vektorgröße von 300 Dimensionen[39]. Die Gensim, siehe Seite 29 Punkt 1 Gensim, Bibliothek in Python ermöglicht die einfache Einbindung und Anwendung dieses Modells in Projekte zur Verarbeitung natürlicher Sprache[40].
2. Im Prozess der Datenvorbereitung wurden die Texte zunächst in einzelne Wörter zerlegt und von gängigen Stoppwörtern sowie Satzzeichen bereinigt. Dies gewährleistet eine aufbereitete Datenstruktur, die für die Erstellung von Word2Vec-Einbettungen erforderlich ist, indem sie irrelevante und häufige Elemente eliminiert, die keinen Mehrwert für die semantische Analyse bieten.
3. Dokumenteinbettungen wurden mittels der SIF-Methode generiert, indem zunächst für jedes Wort im Text entsprechende Word2Vec-Einbettungen ermittelt wurden. Anschließend wurde der gewichtete Durchschnitt dieser Worteinbettungen bestimmt, wobei die Gewichtung sich invers zur Häufigkeit des jeweiligen Wortes im Text verhält (SIF-Gewichtung). Dieser Prozess resultierte in der Erstellung einer einzelnen Vektordarstellung für jede Textpassage, die als Einbettung dient.
4. Abschließend wurde die Kosinus-Ähnlichkeit zwischen den Dokumenteinbettungen der Textpassagen berechnet. Diese Methode bewertet die Ähnlichkeit der Richtung zweier Vektoren im Einbettungsraum, um ein Maß für die semantische Nähe

[39] Word2Vec-GoogleNews-Vectors, (abgerufen am 16.03.2024)
[40] Pretrained word2vec Implementation, (abgerufen am 15.13.2024)

zwischen den Texten zu bieten. Die Anwendung der SIF-Methode ermöglicht es, über die Grenzen herkömmlicher Kosinus-Ähnlichkeitsberechnungen hinauszugehen, indem sie semantische Informationen aus den Word2Vec-Einbettungen nutzt und gleichzeitig die Wortfrequenz im Text berücksichtigt. Durch den Einsatz von Word2Vec-Einbettungen in Kombination mit der SIF-Methode konnten subtilere semantische Verbindungen zwischen den Textpassagen erfasst und präzisere Ähnlichkeitsbewertungen erzielt werden.

Der Code in *Listing 36* veranschaulicht die Umsetzung der SIF-Methode:

```
# using pretrained model
model = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
def compute_sif_similarity(text1, text2, word_embeddings_model):
    # Tokenize the texts
    tokens_text1 = text1.split()
    tokens_text2 = text2.split()
    # Create TF-IDF vectors
    vectorizer = TfidfVectorizer(stop_words='english')
    tfidf_vectors = vectorizer.fit_transform([text1, text2])
    # Get word embeddings for all words in the vocabulary
    word_embeddings = {}
    for word in vectorizer.vocabulary_.keys():
        word_embeddings[word] = word_embeddings_model[word]
    # Compute the weighted average of word embeddings for each text
    embedding_dim = len(word_embeddings_model.vectors[0])
    embeddings_text1 = np.zeros(embedding_dim)
    embeddings_text2 = np.zeros(embedding_dim)
    total_weight_text1 = 0
    total_weight_text2 = 0
    for word, index in vectorizer.vocabulary_.items():
        tfidf_weight_text1 = tfidf_vectors[0, index]
        tfidf_weight_text2 = tfidf_vectors[1, index]
        if word in word_embeddings:
           embeddings_text1 += tfidf_weight_text1 * word_embeddings[word]
           embeddings_text2 += tfidf_weight_text2 * word_embeddings[word]
            total_weight_text1 += tfidf_weight_text1
            total_weight_text2 += tfidf_weight_text2
    # Normalize the embeddings
    embeddings_text1 /= max(total_weight_text1, 1)
    embeddings_text2 /= max(total_weight_text2, 1)
    # Compute cosine similarity between the normalized embeddings
    similarity_score = cosine_similarity([embeddings_text1], [embeddings_text2])[0][0]
    return similarity_score
def preprocess_text(text, model_vocab):
    # Remove special characters, punctuation, and non-alphanumeric symbols
    cleaned_text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    # Convert to lowercase
    cleaned_text = cleaned_text.lower()
    # Tokenize the text
    tokens = cleaned_text.split()
    # Filter out words not in the model's vocabulary
    tokens = [token for token in tokens if token in model_vocab]
    return tokens
# Get the vocabulary of the word embeddings model
model_vocab = model.key_to_index.keys()
cleaned_tokens_text1 = preprocess_text( text_extracted,model_vocab)
cleaned_tokens_text2 = preprocess_text(referenced_texts['6'],
```

```
                                                                model_vocab)
48| # Compute similarity using preprocessed tokens
49| similarity_score = compute_sif_similarity(' '.join(cleaned_tokens_text1)
                                    , ' '.join(cleaned_tokens_text2), model)
```

Listing 36: SIF-Methode

Das Word2Vec-Modell ***GoogleNews-vectors-negative300*** wird geladen, das auf einem Datensatz von Google News-Artikeln trainierte. Dann wird die Variable ***model_vocab*** als eine Liste der Schlüssel im Vokabular des Word2Vec-Modells definiert. Dies ist notwendig, weil im nächsten Schritt die Tokens gefiltert werden, um sicherzustellen, dass nur solche verwendet werden, die auch im Modell vorhanden sind. ***cleaned_tokens_text1*** wird durch Anwendung der ***preprocess_text*** Funktion auf ***text_extracted*** und das Modellvokabular erzeugt. Dieser Schritt beinhaltet das Entfernen von Sonderzeichen, das Umwandeln in Kleinbuchstaben und das Aufteilen in Tokens, die dann gegen das Vokabular des Modells gefiltert werden. Ähnlich wird ***cleaned_tokens_text2*** erzeugt. ***similarity_score*** wird berechnet, indem die ***compute_sif_similarity*** Funktion auf die bereinigten und zu einem String verbundenen Tokens von ***text_extracted*** und ***referenced_texts*** angewendet wird. Hierbei wird das vortrainierte Word2Vec-Modell als dritter Parameter übergeben, um die notwendigen Wortvektoren für die Berechnung zu erhalten.

Die Funktion ***compute_sif_similarity*** nimmt zwei Texte sowie das geladene Word2Vec-Modell entgegen und berechnet deren Ähnlichkeit basierend auf der Smooth Inverse Frequency (SIF)-Methode:

- Die Texte werden tokenisiert (in einzelne Wörter aufgespalten).
- Ein ***TF-IDF-Vectorizer*** wird initialisiert und auf die Token angewendet, um ihre Gewichtung zu erhalten.
- Für jedes Token werden die entsprechenden Word2Vec-Worteinbettungen aus dem Modell geholt.
- Es wird der gewichtete Durchschnitt dieser Einbettungen für jeden Text berechnet.
- Die Vektoren werden normalisiert.
- Die Kosinus-Ähnlichkeit zwischen den resultierenden normalisierten Vektoren wird als Maß für die Ähnlichkeit der beiden Texte verwendet.

Schließlich gibt diese Funktion die Ähnlichkeitswert ***similarity_score*** zurück.

Der in *Listing 36* präsentierte Code wurde auf die *oben* in TF-IDF-Methode verwendeten Beispieltexte angewendet, um deren Ähnlichkeit zu bestimmen. Dies ermöglicht außerdem einen direkten Vergleich zwischen den TF-IDF- und SIF-Methoden.

Ergebnis des ersten zitierten Textes:

```
1 | Smooth Inverse Frequency Similarity Score: 0.6897904820739631
```

Listing 37: Anwendungsbeispiel des SIF-Methode (erste zitierte Quelle)

Ergebnis des ersten zitierten Textes:

```
1 | Smooth Inverse Frequency Similarity Score: 0.5811373121476677
```

Listing 38: Anwendungsbeispiel des SIF-Methode (zweite zitierte Quelle)

Die Ergebnisse der TF-IDF- und SIF-Methoden haben deutlich unterschiedliche Werte geliefert. Die Effektivität und deutliche Überlegenheit der SIF-Methode werden im Abschnitt unten näher beleuchtet. Ein entscheidender Faktor ist das verwendete vortrainierte Word2Vec-Modell, siehe 1, das auf einem umfangreichen Korpus von Google News-Artikeln basiert und Worteinbettungen in 300 Dimensionen liefert. Dies ermöglicht eine tiefgreifende Erfassung semantischer Ähnlichkeiten und die Relevanz der Begriffe.

5.6.3 Auswahl der Methode

Die Auswahl der SIF-Methode (Smooth Inverse Frequency) für dieser Arbeit wurde getroffen, da sie effektiver mit den Herausforderungen umgeht, die durch Längenunterschiede zwischen zwei Textpassagen entstehen. Im Vergleich traditioneller Methoden, die alle Wörter gleich gewichten und somit die Vielfalt ihrer Bedeutungen im Text nicht ausreichend differenzieren, ermöglicht die SIF-Methode eine feinere Abstimmung der Bedeutungsgewichte. Sie nutzt Worteinbettungen zur Darstellung der semantischen Bedeutung einzelner Wörter, was die Vergleichbarkeit zwischen Texten unterschiedlicher Länge verbessert. Folglich erweist sich die SIF-Methode als präziseres Werkzeug für unser Vorhaben, die semantische Ähnlichkeit zwischen Textabschnitten zu bestimmen.

Hier sind die Gründe, warum die SIF-Methode für diese Arbeit genauer ist:

- Word2Vec-Modelle stellen Wörter als dichte Vektoren in einem hochdimensionalen Raum dar und erfassen semantische Bedeutungen basierend auf dem Kontext, in dem die Wörter im Korpus erscheinen. Diese Worteinbettungen reflektieren semantische Relationen und ermöglichen es der SIF-Methode den semantischen Inhalt von Textpassagen präziser zu ermitteln. Dadurch wird eine fundierte Analyse der semantischen Ähnlichkeit ermöglicht.
- Die SIF-Methode nutzt ein gewichtetes Schema für Worteinbettungen, das auf der Frequenz der Wörter im Text basiert, wodurch häufig vorkommende, aber inhaltlich weniger aussagekräftige Wörter eine geringere Gewichtung erhalten. Dies verringert den Einfluss von Stoppwörtern und allgemeinen Begriffen und führt zu semantisch reichhaltigeren Dokumenteinbettungen.
- Die SIF-Methode standardisiert Dokumenteneinbettungen, um eine einheitliche Repräsentation des semantischen Inhalts der Textabschnitte zu gewährleisten. Durch diese Standardisierung werden Unterschiede in der Textlänge ausgeglichen, wodurch ein objektiverer Vergleich von Textpassagen unterschiedlicher Längen ermöglicht wird.
- Die SIF-Methode verbessert die Genauigkeit der Ähnlichkeitsbewertungen, indem sie semantische Informationen aus Worteinbettungen einbezieht und gleichzeitig die Häufigkeit von Wörtern im Text berücksichtigt. Dies ist besonders vorteilhaft für diese Arbeit, da es ermöglicht, die semantische Ähnlichkeit von Textpassagen zu beurteilen, ohne dass deren unterschiedliche Längen das Ergebnis beeinflussen.

Insgesamt stellt die SIF-Methode einen robusteren und präziseren Ansatz dar, um die semantische Ähnlichkeit zwischen Textpassagen zu bewerten. Dies macht sie zu einer angemessenen Methode für die Zielsetzungen dieser Arbeit.

5.7 Bestimmung des Schwellenwerts

In der durchgeführten Implementierung wurde ein Ähnlichkeitsschwellenwert von 0,7 festgelegt. Dies bedeutet, dass Textpassagen als ähnlich betrachtet werden, sobald der Ähnlichkeitswert zwischen ihnen den Wert von 0,7 übersteigt. Die Entscheidung für diesen Schwellenwert basierte auf einer Reihe von Experimenten und Untersuchungen:

- **Empirische Evaluation:** in der Phase der empirischen Evaluation wurden verschiedene Ähnlichkeitsschwellenwerte experimentell getestet, um deren Effektivität auf den Datensatz zu bewerten. Es stellte sich heraus, dass ein Schwellenwert von 0,7 zufriedenstellende Ergebnisse erzielte, indem er eine präzise Unterscheidung zwischen ähnlichen und unähnlichen Textpassagen ermöglichte.
- **Optimierung von Präzision und Recall:** Die Wahl eines Ähnlichkeitsschwellenwerts von 0,7 zielte darauf ab, ein ausgewogenes Verhältnis zwischen Präzision und Recall bei der Erkennung ähnlicher Textpassagen zu erreichen. Ein Schwellenwert, der zu niedrig angesetzt ist, könnte zu einer erhöhten Anzahl falsch positiver Ergebnisse führen, bei denen unähnliche Passagen irrtümlich als ähnlich klassifiziert werden. Ein zu hoher Schwellenwert hingegen könnte dazu führen, dass tatsächlich ähnliche Passagen übersehen werden. Es wurde ermittelt, dass ein Schwellenwert von 0,7 für das Vorhaben dieser Arbeit eine effektive Balance zwischen Präzision und Recall darstellt.
- **Anpassung an die Projektanforderungen:** Die Festlegung des Ähnlichkeitsschwellenwerts wurde maßgeblich von den speziellen Anforderungen dieser Arbeit beeinflusst. In bestimmten Anwendungen, wie beispielsweise der Plagiatserkennung oder der Identifikation doppelter Inhalte, könnte ein strengerer Schwellenwert vorteilhaft sein, um ausschließlich Passagen mit hoher Übereinstimmung zu markieren. Für unser Vorhaben, das auf die Identifizierung relevanter Referenzen in akademischen Texten abzielt, erlaubt ein moderater Schwellenwert eine weitgefasste Auslegung von Ähnlichkeit und gewährleistet dabei ein akzeptables Maß an Präzision.
- **Domänenspezifische Betrachtungen:** Die Festlegung der Ähnlichkeitsschwelle wird ebenfalls durch domänenspezifische Gegebenheiten beeinflusst. Besonders in der wissenschaftlichen Forschung, wo es gängig ist, auf vorherigen Arbeiten aufzubauen und sich auf verwandte Konzepte zu beziehen, kann eine weniger strikte Ähnlichkeitsschwelle vorteilhaft sein, um solche Verknüpfungen zwischen Textpassagen aufzuzeigen.

Nach umfangreichen Experimenten und unter Berücksichtigung verschiedener Aspekte wie Genauigkeit, Recall, Arbeitsanforderungen und fachspezifischen Kriterien

wurde ermittelt, dass ein Ähnlichkeitsschwellenwert von 0,7 ein ausgewogenes Verhältnis für die Ziele dieser Arbeit bietet.

6 Zusammenfassung und Ausblick

Diese Arbeit hat sich der Herausforderung gewidmet, Quellenangaben in studentischen Arbeiten automatisiert zu überprüfen. Durch den Einsatz und Vergleich verschiedener Methoden zur Messung der Textähnlichkeit, insbesondere der SIF-Methode (Smooth Inverse Frequency) und der TF-IDF-Methode (Term Frequency-Inverse Document Frequency), konnte gezeigt werden, dass die SIF-Methode aufgrund ihrer höheren Präzision und Effektivität, besonders im Umgang mit Textlängenunterschieden, vorzuziehen ist. Der Beitrag dieser Arbeit liegt vor allem in der automatisierten Überprüfung von Quellen, was nicht nur zur Bekämpfung von Plagiaten beiträgt, sondern auch die Einhaltung der Zitationsprinzipien fördert und die Prüfer bei der Korrektur entlastet.

Allerdings stößt die Arbeit auch auf Limitationen. So können Quellen aus nicht-wissenschaftlichen Arbeiten und wissenschaftliche Arbeiten, die nicht in den Datenbanken DBLP gelistet sind, nicht geprüft werden. Zudem beschränkt sich die Erkennung von Zitaten auf den IEEE-Zitierstil. Für zukünftige Forschungen ergibt sich daher ein breites Spektrum an Möglichkeiten. Eine Erweiterung um weitere Zitierstile und die Anbindung an zusätzliche wissenschaftliche Suchmaschinen könnten die Abdeckung und Präzision der Quellenprüfung wesentlich verbessern. Des Weiteren wurden in dieser Arbeit nur studentische Arbeiten betrachtet und geprüft, die in Englisch verfasst wurden. Darüber hinaus könnte die Einbeziehung weiterer Methoden zur Ähnlichkeitsmessung die Ergebnisse der Prüfung optimieren. Die praktischen Anwendungen dieser Arbeit liegen insbesondere im Bereich der Plagiatserkennung, wo sie einen signifikanten Beitrag zur Sicherstellung der akademischen Integrität leisten kann.

Literaturverzeichnis

[1] T. Hug, K. Niedermair und A. Drexler, Wissenschaftliches Arbeiten eine Handreichung, Innsbruck: STUDIA Universitätsbuchhandlung und –verlag, 2019.

[2] C. Lipson, Cite right: a quick guide to citation styles--MLA, APA, Chicago, the sciences, professions, and more, Chicago: The University of Chicago Press, 2011.

[3] J. Bahr und M. Frackmann, Richtig zitieren nach der Harvard-Methode, Schweiz: Institut für Praxisforschung, 2011.

[4] IEEE, *Guide, IEEE Reference,* 445 Hoes Lane, Piscataway, NJ 08854 USA: IEEE Periodicals, 2018.

[5] M. Feidel, „mentorium," 15 03 2022. [Online]. Available: https://www.mentorium.de/deutsche-zitierweise/. [Zugriff am 10 01 2024].

[6] J. Fenn, „Managing Citations and Your Bibliography with BibTEX," *The PracTEX Journal,* Bd. 4, pp. 2006-4, 2006.

[7] H. Voß, Bibliografien mit LaTeX: 3, Berlin: Lehmanns Media, 2016.

[8] A. Goker und J. Davies, Information retrieval: Searching in the 21st century, John Wiley \& Sons, 2009.

[9] S. Ceri, A. Bozzon, M. Brambilla, E. D. Valle, P. Fraternali und S. Quarteroni, Web Information Retrieval, Berlin, Heidelberg: Springer Berlin, Heidelberg, 2013.

[10] M. A. Alkhamis, „Extraktion textueller Informationen aus heterogenen PDF-Dokumenten," Universität Rostock, Rostock, 2023.

[11] M. Fenniak, „PyPDF2," 2006 - 2008. [Online]. Available: https://pypdf2.readthedocs.io/en/3.0.0/index.html#. [Zugriff am 14 02 2024].

[12] Y. Shinyama, „PDFMiner," 2004 - 2013. [Online]. Available: https://pdfminer-docs.readthedocs.io/pdfminer_index.html. [Zugriff am 14 02 2024].

[13] A. Farkiya, P. Saini, S. Sinha und S. Desai, „Natural Language Processing using NLTK and WordNet," *International Journal of Computer Science and Information Technologies,* Bd. 6, pp. 5465-5469, 2015.

[14] O. Abiola, A. Abayomi-Alli, O. A. Tale, S. Misra und O. Abayomi-Alli, „Sentiment analysis of COVID-19 tweets from selected hashtags in Nigeria using VADER and Text Blob analyser," *Journal of Electrical Systems and Information Technology,* Bd. 10, Nr. 1, p. 5, 2023.

[15] C. P. W. Roma und P. L. E. Bladimir, „Comparative Analysis of Libraries for the Sentimental Analysis," 2023.

[16] P. Qi, Y. Zhang, Y. Zhang, J. Bolton und C. D. Manning, *Stanza: A Python Natural Language Processing Toolkit for Many Human Languages,* Stanford: Stanford University, 2020.

[17] A. Kumar, V. Katiyar und P. Kumar, „A Comparative Analysis of Pre-Processing Time in Summary of Hindi Language using Stanza and Spacy," *IOP Conference Series.Materials Science and Engineering,* Bd. 1110, Nr. 1, 03 2021.

[18] A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter und R. Vollgraf, „FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP," in *Proceedings of the 2019 Conference of the North {A}merican Chapter of the Association for Computational Linguistics (Demonstrations)*, Minneapolis, Minnesota, 2019.

[19] C. Chantrapornchai und A. Tunsakul, „Information Extraction Tasks based on BERT and SpaCy on Tourism Domain," *ECTI Transactions on Computer and Information Technology,* Bd. 15, Nr. 1, pp. 108-122, 01 2021.

[20] R. Al-Rfou, „Polyglot," MIT License, 2014-2015. [Online]. Available: https://polyglot.readthedocs.io/en/latest/. [Zugriff am 14 03 2024].

[21] T. D. Smedt und W. Daelemans, „Github," BSD License (BSD), [Online]. Available: https://github.com/clips/pattern?tab=readme-ov-file. [Zugriff am 15 03 2024].

[22] P. J. Rao, D. K. N. Rao und D. S. Gokuruboyina, „An Experimental Study with Fuzzy-Wuzzy (Partial Ratio) for Identifying the Similarity between English and French Languages for Plagiarism Detection," *International Journal of Advanced Computer Science and Applications,* Bd. 13, Nr. 10, 2022.

[23] A. Dhakal, A. Poude, S. Pandey, S. Gaire und H. P. Baral, „Exploring Deep Learning in Semantic Question Matching," in *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, Kathmandu (Nepal), 2018.

[24] J. Hunt, „Regular Expressions in Python," in *Advanced Guide to Python 3 Programming*, Chippenham, Wiltshire, UK, Springer, Cham, 2019, pp. 257-271.

[25] Y. Zahidi, Y. E. Younoussi und C. Azroumahli, „Comparative Study of the Most Useful Arabic-supporting Natural Language Processing and Deep Learning Libraries," in *2019 5th International Conference on Optimization and Applications (ICOA)*, 2019.

[26] M. M. Haider, M. A. Hossin, H. R. Mahi und H. Arif, „Automatic Text Summarization Using Gensim Word2Vec and K-Means Clustering Algorithm," in *2020 IEEE Region 10 Symposium (TENSYMP)*, Dhaka, Bangladesh, 2020.

[27] O. Kramer, „ Scikit-Learn," in *Machine Learning for Evolution Strategies.*, Cham, Springer International Publishing, 2016, pp. 45-53.

[28] L. Huang, „temple university," 30 03 2017. [Online]. Available: https://sites.temple.edu/tudsc/2017/03/30/measuring-similarity-between-texts-in-python/. [Zugriff am 14 03 2024].

[29] F. Rahutomo, T. Kitasuka und M. Aritsugi, „Semantic Cosine Similarity," in *The 7th international student conference on advanced science and technology ICAST*, South Korea: University of Seoul, 2012.

[30] B. Li und L. Han, „Distance Weighted Cosine Similarity Measure for Text Classification," in *Intelligent Data Engineering and Automated Learning -- IDEAL 2013*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2013, pp. 611-618.

[31] S. Bag, S. K. Kumar und M. K. Tiwari, „An efficient recommendation generation using relevant Jaccard similarity," *Information Sciences,* Bd. 483, pp. 53-64, 2019.

[32] A. Jain, A. Jain, N. Chauhan, V. Singh und N. Thakur, „Information retrieval using cosine and jaccard similarity measures in vector space model," *Int. J. Comput. Appl,* Bd. 164, Nr. 6, pp. 28-30, 2017.

[33] T. Wahyuningsih, H. Henderi und W. Winarno, „Text Mining an Automatic Short Answer Grading (ASAG), Comparison of Three Methods of Cosine Similarity, Jaccard Similarity and Dice's Coefficient," *Journal of Applied Data Sciences,* Bd. 2, Nr. 2, pp. 45-54, 2021.

[34] S. Tata und J. M. Patel, „Estimating the selectivity of tf-idf based cosine similarity predicates," Bd. 36, Nr. 2, pp. 7-12, 06 2007.

[35] H. K. C und K. V. K, „Topic Recognition and Correlation Analysis of Articles in Computer Science," in *2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2021.

[36] E. Hindocha, V. Yazhiny, A. Arunkumar und P. Boobalan, „Short-text Semantic Similarity using GloVe word embedding," *International Research Journal of Engineering and Technology (IRJET),* Bd. 06, Nr. 04, pp. 553-558, 04 2019.

[37] S. M. Mohammed, K. Jacksi und S. R. M. Zeebaree, „Glove Word Embedding and DBSCAN algorithms for Semantic Document Clustering," in *2020 International Conference on Advanced Science and Engineering (ICOASE)*, 2020.

[38] K. Ward und Church, „Word2Vec," *Natural Language Engineering,* Bd. 23, Nr. 1, pp. 155-162, 2017.

[39] S. Sivakumar, L. S. Videla, R. K. T., N. J., S. Itnal und D. Haritha, „Review on Word2Vec Word Embedding Neural Net," in *2020 International Conference on Smart Electronics and Communication (ICOSEC)*, 2020.

[40] M. N. Moghadas und Y. Zhuang, „Sent2Vec: A New Sentence Embedding Representation With Sentimental Semantic," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020.

[41] S. Agarwalaa, A. Anagawadi und R. M. R. Guddeti, „Detecting Semantic Similarity Of Documents Using Natural Language Processing," *Procedia Computer Science,* Bd. 189, pp. 128-135, 2021.

[42] L. Richardson, *Beautiful Soup Documentation,* 2019.

[43] C. Zheng, G. He und Z. Peng, „A Study of Web Information Extraction Technology Based on Beautiful Soup," *Journal of Computers,* Bd. 10, Nr. 6, pp. 381-387, 2015.

[44] D. Myers und J. W. McGuffee, „Choosing Scrapy," *Journal of Computing Sciences in Colleges,* Bd. 31, Nr. 1, pp. 83-89, 10 2015.

[45] A. Pan und H. Pan, „Design and Implementation of Web Crawler System Based on Python," in *2023 8th International Conference on Information Systems Engineering (ICISE)*, Dalian, China, 2023.

[46] S. Kumar, J. Thakur, D. Ekka und I. Sahu, „Web scraping using Python," *International Journal of Advances in Engineering and Management,* Bd. 4, Nr. 9, pp. 235-237, 09 2022.

[47] R. V. Chandra und B. S. Varanasi, Python requests essentials, Packt Publishing Birmingham, UK, 2015.

[48] M. Algoabra, „Removal of Imperfections in Digital Scans using Generative Adversarial Networks," Universität Rostock., Rostock, 2023.

[49] G. Forman, „An Extensive Empirical Study of Feature Selection Metrics for Text Classification," *Journal of Machine Learning Research,* Bd. 3, pp. 1289-1305, 03 2003.

[50] P. Willett, Readings in Information Retrieval, San Francisco: Morgan Kaufmann Publishers, 1997.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht. Zusätzlich habe ich zur ausschließlichen Verbesserung und Korrektur der Arbeit Unterstützung von ChatGPT erhalten, insbesondere hinsichtlich sprachlicher Formulierungen sowie grammatikalischer und orthografischer Korrekturen. Es wurden keine Ideen oder Texte von ChatGPT als eigene wissenschaftliche Leistung übernommen.

X Adam Abraham

Adam Abraham
Rostock , 03 . 04 . 2024