KI MV ZENTRUM FÜR KÜNSTLICHE INTELLIGENZ IN MV

Konzeption eines ETL-Prozesses zur Ermittlung von Kennzahlen aus Texten als Vorbereitungsschritt einer Plagiatserkennung

Carla Kalaf
Matrikel-Nr.: 219202923

Abgabedatum: 03.04.2024

Erstgutachter:
Dr.-Ing. Hannes Grunert
Universität Rostock
Albert-Einstein-Str. 22
18059 Rostock

Zweitgutachter:
Dr.-Ing. Holger Meyer
Universität Rostock
Albert-Einstein-Str. 22
18059 Rostock

Kurzfassung

Diese Bachelorarbeit diskutiert den Entwicklungsprozess eines ETL-Prozesses, das darauf abzielt, spezifische Kennzahlen aus Texten zu extrahieren. Die Texte wurden aus PDF-Dateien unter Verwendung der PyPDF2-Bibliothek in der Programmiersprache Python extrahiert. Diese Arbeit bietet eine umfassende Analyse der Arten von Kennzahlen, die extrahiert werden können, mit einem besonderen Schwerpunkt auf der Art und Weise, wie Metadaten extrahiert und als effektive Indikatoren genutzt werden können, auf diese Kennzahlen kann dann später verlassen werden, um Plagiate in akademischen Arbeiten von Studenten zu identifizieren. Die Arbeit hebt auch eine Reihe von in Python verfügbaren Programmbibliotheken hervor, erläutert ihre Eigenschaften und die Vorteile, die sie bieten, und kommt zu dem Schluss, dass in dieser Arbeit die PyPDF2-Bibliothek die bessere Option für die Textextraktion darstellt. Darüber hinaus wird die Rolle der NLTK-Bibliothek bei der effizienten Extraktion von Kennzahlen und Textanalyse diskutiert. Zusätzlich wurden die Ergebnisse der Text- und Kennzahlenextraktion sowie der Metadaten sorgfältig bewertet, um ihre Gültigkeit und Zuverlässigkeit zu überprüfen. Die Arbeit schließt mit einer Reihe von Empfehlungen für zukünftige Forschungen in diesem vielversprechenden Feld.

Abstract

This bachelor thesis discusses the development process of an ETL (Extract, Transform, Load) process aimed at extracting specific features from texts. The texts were extracted from PDF files using the PyPDF2 library in the Python programming language. This work provides a comprehensive analysis of the types of features that can be extracted, with a particular focus on how metadata can be extracted and used as effective indicators. These features can then be relied upon later to identify plagiarism in students' academic works. The thesis also highlights a number of programming libraries available in Python, explains their features and the advantages they offer, and concludes that the PyPDF2 library is the better option for text extraction in this work. Furthermore, the role of the NLTK (Natural Language Toolkit) library in the efficient extraction of features and text analysis is discussed. In addition, the results of the text and features extraction, as well as the metadata, were carefully evaluated to verify their validity and reliability. The thesis concludes with a series of recommendations for future research in this promising field.

Inhaltsverzeichnis

1 Einleitung

Obwohl wissenschaftliche Studien und Forschungsarbeiten oftmals Monate kontinuierlicher Anstrengung sowie möglicherweise Jahre intensiver Forschung und Studium erfordern, ist der Zugang zu diesen Werken angesichts der gegenwärtigen Informations- und digitalen Explosion außerordentlich erleichtert worden. Durch die schlichte Anwendung der Funktion des Kopierens und Einfügens kann eine Idee mühelos einer anderen Person zugeordnet werden. Des Weiteren haben Künstliche-Intelligenz-Programme durch ihre bemerkenswerten Ergebnisse einen erheblichen Eindruck hinterlassen. Es ist nun möglich, einen umfassenden Text in nur wenigen Minuten oder sogar Sekunden zu generieren. In Anbetracht dieser Entwicklungen ist es von entscheidender Bedeutung, effektive Methoden zur Entlarvung von Plagiaten zu entwickeln. Der wissenschaftliche Diebstahl konstituiert eine erhebliche Gefahr für die wissenschaftliche Forschung, indem er sich nicht lediglich auf den Transfer von Ideen zwischen Individuen beschränkt, sondern vielmehr als eine Verletzung und Täuschung gegenüber Bildungseinrichtungen sowie Arbeitsunternehmen fungiert.

Diese Forschungsarbeit behandelt die Konzeption und Entwicklung eines ETL-Modells mit dem Ziel, spezifische Kennzahlen aus Texten zu extrahieren. Diese Kennzahlen spielen eine entscheidende Rolle bei der Gestaltung des initialen und grundlegenden Schritts für die spätere Erkennung von Plagiaten.

Der ETL-Prozess (Extraktion, Transformation und Laden) ist ein grundlegender Schritt in der Datenverarbeitung, der die Erfassung von Rohdaten aus verschiedenen Quellen, ihre strukturierte Organisation sowie Formatierung und letztendlich die Erstellung aussagekräftiger Auswertungen umfasst. Diese Prozessabläufe sind in drei Hauptphasen unterteilt [1]:

1. **Extraktion:** In dieser Phase werden die Daten aus den verschiedenen Quellen extrahiert, sei es aus Datenbanken, Dateien oder anderen Datenquellen. Dieser Schritt beinhaltet die Gewinnung von Rohdaten in ihrer ursprünglichen Form.
2. **Transformation:** Die extrahierten Daten werden in dieser Phase gereinigt, umgestaltet und standardisiert, um ihre Qualität zu verbessern und sie für die Verarbeitung und Analyse vorzubereiten. Dies kann die Bereinigung von fehlerhaften Einträgen, die Zusammenführung von Datensätzen oder die Umwandlung von Formaten umfassen.

3. **Laden:** Nach der Transformation werden die bearbeiteten Daten in das Zielsystem oder die Datenbank geladen. Sie werden in einer für die weitere Analyse oder Nutzung geeigneten Struktur gespeichert.

Der ETL-Prozess bildet somit einen kritischen Teil der Datenverarbeitung, der es ermöglicht, unstrukturierte oder ungeordnete Rohdaten in wertvolle und nutzbare Informationen umzuwandeln, die für Entscheidungsfindung, Analyse und Berichterstattung verwendet werden können.

1.1 Motivation

Im Zuge des deutlichen Anstiegs von Daten im Laufe der Zeit manifestierte sich die Notwendigkeit eines effizienten und verlässlichen Ansatzes, dessen Ziel darin besteht, Daten aus diversen Quellen zu extrahieren, zu transformieren und zu organisieren oder zu filtern, um sie den Anforderungen des Zielsystems anzupassen. Als einfache Erklärung des ETL-Prozesses, in *Abbildung 1* dargestellt, kann dieser als die Extraktion von Daten aus unterschiedlichen Quellen, ihre anschließende Verarbeitung und Integration sowie die Speicherung in einem definierten System beschrieben werden. Das übergeordnete Ziel dieses Prozesses liegt in der Schaffung einer vernetzten, einheitlichen und transparenten Datenbasis. Dies erleichtert die Analyse und den Zugriff auf die Daten und trägt zur Steigerung ihrer Qualität bei.

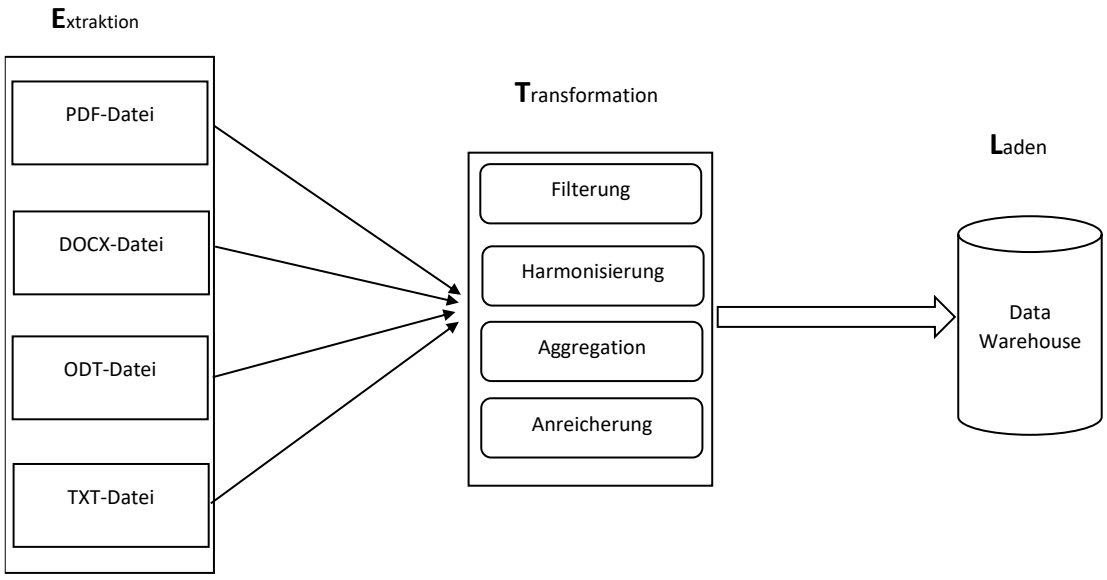

Abbildung 1: Die fundamentalen Schritte des ETL-Prozesses

1.2 Problemstellung und Zielsetzung

Selbstverständlich begegnen der Implementierung dieses Prozesses zahlreiche Herausforderungen und Schwierigkeiten. Hierzu zählen beispielsweise:

1. **Datenverlust:** Ein möglicher Verlust von Daten aufgrund der Vielfalt der Datenquellen erfordert ein effektives Management dieser Diversität, um sicherzustellen, dass sämtliche Informationen erfasst und genutzt werden.
2. **Aufrechterhaltung der Systemstabilität bei steigendem Datenvolumen:** Es gestaltet sich anspruchsvoll, die Kontinuität und Stabilität des Systems bei wachsender Datenmenge aufrechtzuerhalten.
3. **Gewährleistung der Informationsintegrität während der Übertragung:** Prozesse der Datenübertragung bergen sicherheitsrelevante Herausforderungen. Die Vertraulichkeit und Integrität der Informationen müssen während ihres Transfers zwischen unterschiedlichen Quellen gewährleistet sein.

Das primäre Ziel dieser These besteht darin, ein ETL-Prozess zu konzipieren, das spezifische Kennzahlen aus Texten extrahiert, beispielsweise: Metadaten, die den Ursprung und Kontext der Textsammlungen beleuchten, textuelle Merkmale, die die Struktur und den Inhalt der Texte darstellen, und numerische Merkmale, die die Daten der Texte in Zahlen fassen. Diese Extraktion dient als initialer Schritt zur nachfolgenden Erkennung von Plagiaten. Die Bachelorthesis befasst sich mit diesen Herausforderungen, um dieses Hauptziel zu erreichen.

1.3 Aufbau der Arbeit

Die vorliegende These gliedert sich in sechs Hauptkapitel, die eine detaillierte Untersuchung der thematischen Schwerpunkte umfassen. Im ersten Kapitel wurden die Problemstellung und die Ziele dieser Arbeit dargelegt. Das zweite Kapitel widmet sich einer umfassenden Erörterung der Metadaten und diversen Textdateien, ergänzt durch eine tiefgreifende Analyse des Information Retrieval und des ETL-Prozesses (Extraktion, Transformation, Laden). Im dritten Kapitel wird eine detaillierte Untersuchung der

aktuellen Technologien und Methoden im Bereich des ETL-Prozesses vorgenommen, mit einem besonderen Fokus auf die Extraktion von Metadaten und Texten aus PDF-Dokumenten sowie auf Techniken der Textverarbeitung. Das vierte Kapitel präsentiert das in dieser Arbeit verfolgte Konzept und erläutert detailliert die einzelnen zu implementierten Schritte. Im fünften Kapitel wird der entwickelte Prototyp vorgestellt, der die praktische Anwendung der theoretischen Konzepte demonstriert. Abschließend bietet das sechste Kapitel eine umfassende Zusammenfassung und Reflexion der wesentlichen Erkenntnisse der Arbeit und diskutiert, wie diese in Zukunft erweitert werden könnten.

2 Theoretische Grundlagen

In diesem Kapitel werden Grundlagen und Anwendungen erläutert, beginnend mit einem Überblick über Metadaten. Es folgt eine Vorstellung verschiedener Textdateiformate wie PDF, TXT usw.. Anschließend wird das Augenmerk auf den Prozess der Information Retrieval selbst gerichtet und eine umfassende Übersicht über dessen Prinzipien gegeben. Ein besonderer Fokus liegt auf dem Konzept von ETL (Extraktion, Transformation, Laden), dessen Hauptfunktion darin besteht, Kennzahlen aus Texten zu extrahieren.

2.1 Metadaten

Metadaten stellen eine essenzielle Kategorie von Daten dar, welche dazu dient, detaillierte und strukturierte Informationen über andere Datensätze zu liefern. Primär zielen sie darauf ab, die Administration, den Zugang, das Verständnis und die effektive Nutzung dieser Daten zu optimieren. Insbesondere deskriptive Metadaten spielen eine zentrale Rolle, indem sie eine Gruppe von Daten aggregieren und charakterisieren. Es ist jedoch von Bedeutung zu erkennen, dass solche deskriptiven Metadaten nicht die konkrete Bedeutung der betreffenden Daten widerspiegeln, sondern vielmehr als Schlüsselinformationen für deren Identifikation und Kontextualisierung dienen. Beispielsweise können deskriptive Metadaten Aufschluss über den Zeitpunkt des Versands einer Textnachricht geben, jedoch nicht über deren spezifischen Inhalt informieren. Diese Charakteristik unterstreicht die signifikante Rolle von Metadaten als Werkzeuge zur Datenkategorisierung und -analyse in verschiedenen Informationsumgebungen [2, 3].

2.1.1 Bedeutung

Die Bedeutung von Metadaten zeigt sich in den folgenden Aspekten [4]:

1. **Verbesserung der Entdeckung und des Zugangs zu Ressourcen:** Metadaten erleichtern die Entdeckung von Ressourcen und bieten zentrale Informationen über ihre Identität und Zugänglichkeit, was die Effizienz von Such- und Forschungsprozessen verbessert.
2. **Bereitstellung einzigartiger Kriterien zur Unterscheidung von Informationsentitäten:** Metadaten bieten eine Reihe von festen und einzigartigen digitalen Identifikatoren, die die Fähigkeit von Systemen zur Unterscheidung verschiedener Informationsentitäten erhöhen.
3. **Unterstützung der langfristigen Speicherung und Archivierung digitaler Ressourcen:** Metadaten spielen eine entscheidende Rolle bei der Gewährleistung der dauerhaften Speicherung und Archivierung digitaler Ressourcen, was ihre zukünftige Rückgewinnung und Nutzung sicherstellt.
4. **Dokumentation von Informationen zu Urheberrechten und Vervielfältigungen:** Metadaten tragen zur Nachverfolgung und Dokumentation von Informationen über Urheberrechte und Vertriebsrechte bei, was Transparenz und Compliance mit rechtlichen Systemen fördert.
5. **Ermöglichung der Interoperabilität zwischen verschiedenen Systemen:** Metadaten erleichtern die Kompatibilität und Interoperabilität zwischen verschiedenen Informationssystemen, indem sie die Harmonie zwischen Geräten und Software herstellen.

Auf diese Weise erlangen Metadaten eine bedeutende Rolle im Bereich des Informationsmanagements, da sie ein grundlegendes Werkzeug für die Organisation, Analyse und Sicherstellung der Nachhaltigkeit in der heutigen digitalen Umgebung darstellen.

2.1.2 Arten von Metadaten

Metadaten können in drei Hauptkategorien klassifiziert werden [2, 3, 4]:

1. **Deskriptive Metadaten:** Diese Kategorie von Metadaten umfasst identifizierende Informationen, die sich auf die Ressource beziehen; wie zum Beispiel den Titel des Dokuments, den Namen des Autors und die Schlüsselwörter. Diese Daten verbessern die Zugänglichkeit und Auffindbarkeit der betreffenden Ressourcen und erleichtern deren Klassifizierung und Organisation innerhalb von Informationssystemen.
2. **Strukturelle Metadaten:** Diese Art von Metadaten enthält Informationen darüber, wie die Elemente einer Ressource oder Daten organisiert und angeordnet sind. Zum Beispiel können diese Daten die Anordnung der Seiten innerhalb eines Dokuments zur Bildung von Kapiteln verdeutlichen, was das Verständnis der allgemeinen Struktur der Ressource und deren Organisation erleichtert.
3. **Administrative Metadaten:** Diese Kategorie bezieht sich auf Informationen, die den Ursprung der Ressource, den Besitz der Daten und die Zugriffsberechtigungen umfassen. Beispielsweise können Informationen über das Datum und die Art und Weise der Erstellung eines Bildes, den verwendeten Kameratyp und die Auflösung des Objektivs enthalten sein. Diese Daten sind entscheidend für das Management von Urheberrechten, die Konservierung und die digitale Sicherheit.

Jeder dieser Typen spielt eine zentrale Rolle bei der Steigerung der Effizienz und Effektivität im Informations- und Ressourcenmanagement im digitalen Bereich und gilt als wichtiger Bestandteil der Grundlage moderner Informationssysteme.

2.2 Textdateien

In diesem Abschnitt werden verschiedene Arten von Textdateien vorgestellt, die jeweils spezifische Eigenschaften besitzen und somit für unterschiedliche Einsatzgebiete geeignet sind. Beispielsweise werden im akademischen Bereich häufig PDF-Dateien verwendet. Es wird ein Überblick über vier unterschiedliche Textdateiformate gegeben, um zu beleuchten, wo ihre jeweiligen Stärken und Anwendungsbereiche liegen.

2.2.1 PDF

Die PDF-Datei, eine Abkürzung für „Portable Document Format“, repräsentiert ein digitales Dateiformat, das in den frühen 1990er Jahren entwickelt wurde. Ihr herausragendes Merkmal liegt in der Fähigkeit, den Nutzern den Austausch digitaler Dokumente in einem konsistenten Format zu ermöglichen, unabhängig von der spezifischen Software, die zur Erstellung dieser Dokumente verwendet wurde. Diese Eigenschaft stellt sicher, dass die Dokumente auf jeder Plattform identisch dargestellt werden. PDFs finden breite Anwendung in diversen Bereichen, einschließlich, aber nicht beschränkt auf elektronische Bücher, akademische Formulare und juristische Dokumente [5, 6].

Eines der signifikantesten Merkmale von PDF-Dokumenten ist ihre hohe Sicherheit und Vertrauenswürdigkeit. Dies wird durch fortschrittliche Sicherheitsfunktionen erreicht, die den Schutz von Inhalten durch Mechanismen wie digitale Signaturen und die Implementierung von Passwortschutz für Dokumente gewährleisten. Diese Sicherheitsaspekte sind besonders relevant in Kontexten, in denen die Integrität und Authentizität von Dokumenten von essenzieller Bedeutung sind [5, 6].

2.2.2 DOCX

Eine DOCX-Datei ist eine Erweiterung des Microsoft Word-Dokumentformats, das auf dem Open XML-Standard basiert. Dieses Format wurde 2007 als Nachfolger des älteren DOC-Formats eingeführt. Der Hauptunterschied zwischen DOC und DOCX liegt in der Verwendung von XML, einer Auszeichnungssprache, die eine effizientere Speicherung und Datenübertragung ermöglicht. DOCX-Dateien sind in der Regel kleiner und weniger anfällig für Beschädigungen als DOC-Dateien [7, 8].

Die Dateien im DOCX-Format zeichnen sich durch mehrere charakteristischen Merkmale aus, die wesentlich zu ihrer Effizienz und Funktionalität beitragen. Zu den Eigenschaften zählt die reduzierte Dateigröße im Vergleich zu den älteren DOC-Formaten, eine Entwicklung, die vornehmlich auf die erweiterten Komprimierungstechniken zurückzuführen ist. Darüber hinaus bietet das DOCX-Format eine umfangreiche Palette an Formatierungsoptionen, die eine Vielzahl von Elementen wie Bilder, Texte und Grafiken einschließen. Des Weiteren ist das Format für seine fortgeschrittenen Sicherheitsfunktionen bekannt, einschließlich des Schutzes durch Passwörter, was zu einer erhöhten Sicherheit und Integrität der in diesen Dateien gespeicherten Informationen beiträgt [7, 8].

2.2.3 TXT

TXT-Dateien, auch bekannt als Plain-Text-Dateien, sind einfache Textdokumente, die keine spezielle Formatierung enthalten. Sie bestehen ausschließlich aus Textzeichen und sind in einem menschenlesbaren Format, im Gegensatz zu Binärdateien. Das Besondere am TXT-Format ist seine Einfachheit und breite Kompatibilität. TXT-Dateien verwenden häufig den ASCII-Zeichensatz, können aber auch im Unicode-Format gespeichert werden, um eine breitere Palette von Zeichen international zu unterstützen [9].

Eine der Hauptstärken von TXT-Dateien ist ihre Einfachheit und Zugänglichkeit. Sie benötigen keine speziellen Programme zur Erstellung oder Anzeige, da sie in den meisten Betriebssystemen mit vorinstallierten Texteditoren geöffnet und bearbeitet

werden können. Diese Dateien sind oft in Programmier- und Entwicklungsumgebungen zu finden, wo sie zum Schreiben und Speichern von Code verwendet werden. Ein beliebtes Beispiel ist Notepad++, ein Texteditor, der von Webdesignern und Frontend-Entwicklern genutzt wird. Er bietet eine benutzerfreundliche Oberfläche, schnelle Verarbeitung und fortgeschrittene Funktionen für die Codebearbeitung, was ihn zu einem nützlichen Werkzeug in der Programmierung macht [9].

2.2.4 ODT

Das Open Document Format (ODF), offiziell als "OASIS Open Document Format for Office Applications" bezeichnet, repräsentiert einen international genormten Standard für Bürodokumente, der sich durch die Verwendung einer XML-basierten Auszeichnungssprache auszeichnet. Diese Technologie ermöglicht es, Textdokumente so zu strukturieren, dass sie sowohl für Menschen als auch für Maschinen leicht lesbar und interpretierbar sind. Ein signifikanter Vorteil von ODF, insbesondere in seiner Ausprägung als ODT (Open Document Text), liegt in seiner Plattform- und Softwareunabhängigkeit. Im Unterschied zu proprietären Formaten, wie Microsofts DOC, das primär auf spezifische Anwendungen wie Microsoft Word ausgerichtet ist, gewährleistet ODF eine umfassende Kompatibilität mit einer Vielzahl von Textverarbeitungsprogrammen, darunter Apache OpenOffice Writer und LibreOffice [10, 11, 12].

ODT-Dateien, eine spezifische Ausprägung des ODF, sind nicht nur auf die Speicherung von Textinhalten beschränkt, sondern unterstützen auch eine breite Palette von Metadaten, Formatierungen, Bildern und Grafiken. Dies erweitert die Funktionalität und Anwendungsmöglichkeiten des Formats erheblich, indem es eine flexible und effiziente Plattform für die Erstellung, Bearbeitung und den Austausch von Dokumenten in verschiedenen Büroumgebungen bietet [10, 11, 12].

2.3 Information Retrieval

Im Kontext der sozialen Entwicklungen und des Fortschritts der Technologie entstand die Idee des Information Retrieval (IR) aufgrund des drängenden Bedarfs der Menschen, diese Informationen effektiv zu organisieren, um schnelleren und effizienteren Zugang zu ermöglichen. In der antiken Geschichte wurden Bücher und Bibliotheken zu wichtigen Quellen für die Speicherung von Wissen. Eine der großen Herausforderungen bestand jedoch darin, effektiv in diesen großen Bibliotheken zu suchen. Mit dem Fortschritt von Wissenschaft und Technologie im Mittelalter begann das Interesse an der Entwicklung von Methoden zur Organisation und Abruf von Informationen. Ideen über die Schaffung von Indizes und Klassifikationssystemen tauchten auf, um die Suche zu erleichtern. Mit der industriellen Revolution und der Erfindung des Drucks stieg das Volumen von Büchern und Dokumenten erheblich an. Es wurde offensichtlich, dass effektive Mittel zum Abrufen von Informationen erforderlich sind. In der modernen Ära, mit dem Fortschreiten der Informationstechnologie und dem Übergang zur digitalen Ära, wuchs die Notwendigkeit fortschrittlicher Informationssysteme. Suchmaschinen im Internet und automatisierte Informationssysteme entstanden, um diesen Bedarf zu decken. Im Allgemeinen besteht das ultimative Ziel dieses Prozesses darin, korrekte Informationen rechtzeitig bereitzustellen [13, 14].

Die Information Retrieval bezeichnet den Prozess des effektiven Suchens und Herausziehens gewünschter Informationen aus einer umfangreichen und vielfältigen Quellenpalette. Dieser Vorgang beinhaltet die Anwendung unterschiedlicher Prinzipien und Methoden zur Identifizierung und Bereitstellung von Informationen, die den Bedürfnissen des Nutzers entsprechen. Insbesondere im Bereich der Informationstechnologie wird der Begriff "Information Retrieval" genutzt, um die Abläufe der Suche in Datenbanken oder im Internet mithilfe von Suchmaschinen zu beschreiben. Das Ziel dieser Informationsgewinnung ist es, präzise und relevante Ergebnisse gemäß den Anforderungen des Nutzers zu liefern. Dabei werden Techniken wie Indexierung, Klassifizierung und Textanalyse eingesetzt. Insgesamt spielt die Information Retrieval eine zentrale Rolle, um den Zugang zu Informationen in Anbetracht der zunehmenden Menge und Vielfalt von Daten zu erleichtern und zu optimieren [13, 14].

2.3.1 Konzept

Information Retrieval, in *Abbildung 2* dargestellt, funktioniert im Allgemeinen durch die Anwendung von Methoden und Techniken, um relevante Informationen aus einer großen Menge von Daten zu identifizieren und abzurufen. Hier sind einige fundamentale Aspekte, die erklären, wie die Information Retrieval funktioniert [13, 14]:

1. **Indexierung:** Der erste Schritt besteht darin, die Dokumente zu indexieren, die man durchsuchen möchte. Dies bedeutet, dass relevante Begriffe, Schlüsselwörter oder Konzepte aus Dokumenten extrahiert und für einen schnelleren Zugriff in einem Index gespeichert werden.
2. **Anfrage:** Der Benutzer stellt eine Anfrage, die aus Schlüsselwörtern, Sätzen oder anderen Suchparametern besteht. Diese Anfrage wird dann mit dem erstellten Index abgeglichen.
3. **Suchalgorithmus:** Ein Suchalgorithmus wird verwendet, um die relevantesten Dokumente zu identifizieren. Es kann auf verschiedenen Methoden wie mathematischen Modellen, maschinellem Lernen oder statistischer Analyse basieren.
4. **Rangordnung der Ergebnisse:** Gefundene Dokumente werden nach ihrer Relevanz sortiert. Dabei werden verschiedene Algorithmen verwendet, um zu bewerten, ob eine Suchanfrage mit Dokumenten übereinstimmt.
5. **Ergebnisdarstellung:** Dem Nutzer werden die relevantesten Ergebnisse präsentiert. Dies kann in Form einer Linkliste, einer Zusammenfassung oder anderen relevanten Informationen erfolgen.
6. **Feedbackschleife:** Viele moderne IR-Systeme verfügen über eine Feedbackschleife, die das Benutzerverhalten analysiert, um die Suchergebnisse im Laufe der Zeit zu verbessern.

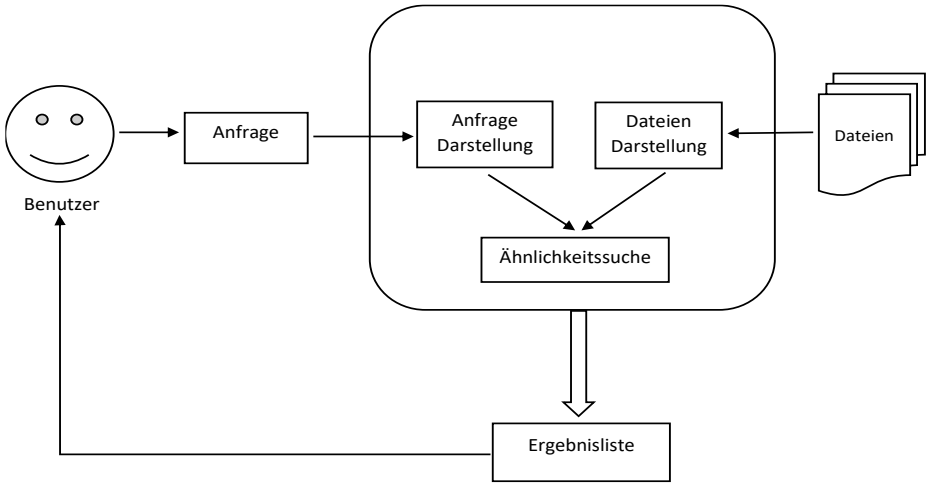

Abbildung 2: Prinzip der Information Retrieval

2.4 ETL-Prozess

Der ETL-Prozess (Extraktion, Transformation, Laden) ist ein wesentlicher Vorgang im Bereich der Datenengineering und Analyse. Dieser Prozess beinhaltet in seiner ersten Phase das Extrahieren von Daten aus mehreren unterschiedlichen Quellen. Darauf folgt eine Reihe komplexer Datentransformationen, bevor die Daten in ein Datenlager geladen werden. Dieser Prozess wird als eine integrierte und entscheidende Strategie im Datenmanagement angesehen, da er die Grundlage für die Umwandlung von rohen, primären Daten in wertvolle und bedeutungsvolle Informationen bildet [15].

2.4.1 Extraktion

Die Extraktionsphase im ETL-Prozess befasst sich mit den Herausforderungen der Datensammlung aus verschiedenen Quellen. Diese Quellen umfassen interne Systeme wie Aufzeichnungen und Datenbanken sowie externe Quellen wie verschiedene Textdateien (PDF, DOCX, TXT usw.). Diese Phase ist entscheidend, um genaue und zuverlässige Daten für die Verwendung in Datenanalysen und zur Unterstützung von Entscheidungsprozessen bereitzustellen. Es erfordert die effektive Verarbeitung und das Verständnis der vielfältigen Strukturen dieser Quellen und das effiziente Extrahieren der Daten, wobei der Schwerpunkt auf Genauigkeit und Effizienz in der Zusammenstellung und Vorbereitung für die nachfolgenden Phasen des Prozesses liegt [16].

Die Extraktion von Metadaten ist ein anschauliches Beispiel für die Schlüsselprozesse der Extraktionsphase innerhalb des ETL-Prozesses (Extract, Transform, Load). Diese Phase umfasst insbesondere:

1. **Datensammlung:** Das Sammeln von Daten aus vielfältigen Quellen, wobei PDF-Dateien ein typisches Beispiel darstellen.
2. **Metadatenextraktion:** In diesem Schritt erfolgt die Erfassung signifikanter Informationen, darunter Autor, Titel, Schlüsselwörter und das Erstellungsdatum.
3. **Datenpräparation:** Bei Bedarf findet eine sorgfältige Filterung der Daten statt, um sie für die anschließende Transformation optimal vorzubereiten und anzupassen.

Im Kontext der Metadatenextraktion aus Textdateien sind folgende Werkzeuge besonders hervorzuheben:

1. **Apache Tika:** Eine vielseitige Bibliothek, die spezielle Kommandos zur Extraktion von Metadaten aus unterschiedlichen Dateiformaten, einschließlich textbasierter Dokumente wie PDFs, verwendet [17].
   Die Metadaten, die aus Textdateien extrahiert werden, umfassen Folgendes: der Name des Autors, der verwendete Schrifttyp, der Titel, das Erstellungs- und Änderungsdatum, die Anzahl der Seiten und die Wortanzahl, die Schlüsselwörter und in einigen besonderen Fällen wird die Versionsnummer der Datei, die Abmessungen der Seiten und das Programm, das zur Erstellung der Datei verwendet wurde, einbezogen [18].
2. **Apache POI**: Dieses Tool ist besonders effektiv für das Extrahieren von Metadaten aus Microsoft Office-Dateien [19]. Es ist bekannt für seine hohe Effizienz im Umgang mit Dateien großen Umfangs und zeichnet sich durch seine Kompatibilität mit sowohl älteren als auch neueren Versionen dieser Dateitypen aus [19].

2.4.2 Transformation

In dieser Phase werden umfassende Verarbeitungsschritte an den Daten durchgeführt, die in der ersten Phase extrahiert wurden. Der Zweck dieser Verarbeitung ist es, die Daten so vorzubereiten, dass sie für die nachfolgende Analyse geeignet sind und effektiv in einem Datawarehouse gespeichert werden können. Zu den wesentlichen Aufgaben in dieser Phase gehören die Entfernung von redundanten Daten, die Korrektur von Fehlern, die Umwandlung der Daten in ein standardisiertes und einheitliches Format sowie die Berechnung von abgeleiteten Werten und andere wichtige Operationen. Der Stil und die Art der Konvertierung, die in dieser Phase gewählt wird, hängen stark von den spezifischen Anforderungen und Bedürfnissen jedes Projekts ab [15, 16].

2.4.3 Laden

Diese Phase stellt den abschließenden Schritt im ETL-Prozess dar, bei dem die extrahierten und transformierten Daten nach ihrer Bearbeitung und Filterung in das Datawarehouse geladen werden. Die Daten werden im Datawarehouse so organisiert, dass sie leicht zugänglich und analysierbar sind, was die Fähigkeit von Organisationen verbessert, wertvolle Einsichten und Informationen aus diesen Daten zu gewinnen [16].

3 Stand der Technik

Dieses Kapitel bietet einen Überblick über die Techniken und Werkzeuge, die in diesem Arbeitskontext gesucht und derzeit verfügbar sind. Es konzentriert sich auf die ausgewählten Techniken und Methoden und erläutert detailliert, warum sie ausgewählt wurden und welche Rolle sie bei der Erreichung des Ziels dieser Arbeit spielen.

3.1 Vorgehen bei der Suche

Um das gesetzte Ziel jeder Arbeit zu realisieren, liegt der Fokus anfänglich auf ein tiefgehendes Verständnis des Arbeitstitels sowie eine detaillierte Auseinandersetzung mit dem thematischen Kern, der dieser Arbeit zugrunde liegt. Dadurch soll ein präzises Verständnis für den Verlauf des Arbeitsprozesses geschaffen werden. Dieser Ansatz markierte den initialen Schritt im Rahmen der Bearbeitung dieser Arbeit. Intensives Verständnis des ETL-Prozesses wurde verfolgt, seiner Arbeitsmechanismen, der Implementierung sowie den Herausforderungen, die bei der Entwicklung eines ETL-Prozesses auftreten könnten. Die Entscheidung fiel darauf, die Arbeit auf die Grundlage von PDF-Dateien zu stellen, woraus sich die Idee ergab, Metadaten zu extrahieren und diese anschließend als Kennzahl einzusetzen. Weiterhin wurde eine Vertiefung in die Erforschung verschiedener Arten von Kennzahlen vorgenommen, die aus Texten extrahiert werden können, um sie in der Folge zur Erkennung von Plagiaten zu verwenden.

Basierend auf der bisherigen Betrachtung wurde die Entscheidung getroffen, die Arbeit in drei fundamentale Kategorien zu strukturieren. Diese bestehen aus der Extraktion von Metadaten, der Extraktion von Text aus PDF-Dateien und der Ermittlung der Kennzahlen aus den extrahierten Texten. Jede dieser Kategorien spielt eine wichtige Rolle für das umfassende Verstehen und die Zielsetzung dieser Arbeit und leistet einen entscheidenden Beitrag zum Forschungserfolg. Die Präsentation der Forschungsergebnisse wird in einer geordneten Folge vorgenommen, die der Gliederung in diese Abschnitte folgt, um einen methodisch fundierten Überblick und Einblick in die gewonnenen Erkenntnisse zu gewährleisten.

3.2 Erzielte Ergebnisse

Es existiert eine Vielzahl von Tools, die den Umgang mit PDF-Dateien erleichtern und optimieren. Jedes dieser Werkzeuge besitzt spezifische Eigenschaften, die es einzigartig machen und von anderen abheben. Die Auswahl und Anwendung dieser Tools richtet sich in der Regel nach der Programmiersprache, die im Rahmen einer Arbeit zum Einsatz kommt. In diesem Kontext soll eine Auswahl der bedeutendsten Tools vorgestellt werden.

3.2.1 Werkzeuge zur Metadatenextraktion

Hier werden die am häufigsten verwendete Werkzeuge zur Extraktion von Metadaten aus PDF-Dateien vorgestellt. Jedes Tool wird einzeln erklärt, wobei eine Vergleichstabelle am Ende erstellt wurde, um einen direkten Überblick über die jeweiligen Funktionen zu ermöglichen. Diese Tools sind unerlässlich für die Extraktion von Metadaten aus PDF-Dateien und spielen auch eine entscheidende Rolle bei der Textextraktion, auf die in Abschnitt „*3.2.2 Werkzeuge zur Textextraktion“* kurz eingegangen wird.

1. **PyPDF2:** ist eine freie und umfassende Python-Bibliothek, die speziell für die Bearbeitung von PDF-Dateien entwickelt wurde. Sie ermöglicht es, PDF-Dokumente zu lesen, Text und Metadaten zu extrahieren sowie PDF-Seiten zu teilen, zusammenzuführen, zu beschneiden und zu transformieren. Darüber hinaus unterstützt PyPDF2 das Hinzufügen von benutzerdefinierten Daten, Anzeigeoptionen und Passwörtern zu PDFs. Zusätzlich kann PyPDF2 wichtige Dokumenteninformationen wie Titel, Autor und Seitenanzahl extrahieren. Dank der Funktionen ‚**`PdfFileReader`**‘ und ‚**`PdfFileWriter`**‘ bietet PyPDF2 die essenziellen Werkzeuge für das Lesen und Schreiben von PDF-Dokumenten, was es zu einem idealen Werkzeug für die Entwicklung automatisierter Anwendungen macht, die mit PDF-Inhalten arbeiten [20, 21, 22].

Das folgende Beispiel[1], *Listing 1*, angewandt auf die PDF-Datei [21], veranschaulicht, wie PyPDF2 zur Extraktion von Metadaten aus einer PDF-Datei verwendet wird:

```
1. from PyPDF2 import PdfReader
2. reader = PdfReader("applsci-12-01352.pdf")
3. meta = reader.metadata
4. print("Seitenanzahl:",len(reader.pages))
5. print("------ Metadaten: ------")
6. for key, value in meta.items():
7.     print(f"------{key}: ------\n{value}")
```

Listing 1: Metadatenextraktion mit Hilfe von PyPDF2

Der Code lädt eine PDF-Datei und extrahiert deren Metadaten sowie die Anzahl der Seiten mit der Bibliothek **'PyPDF2'**. Nach dem Laden der Datei **'applsci-12-01352.pdf'** liest er die Metadaten und die Gesamtseitenzahl. Die Metadaten werden als Schlüssel-Wert-Paare ausgegeben, wobei jeder Metadatenschlüssel (z.B. Autor, Erstellungsdatum) und sein entsprechender Wert (z.B. Name des Autors, Datum) angezeigt werden. Die Seitenanzahl der PDF wird zu Beginn ausgegeben. *Listing 2* zeigt die Ergebnisse dieses Codes:

```
Seitenanzahl: 19
------ Metadaten: ------
------/Author: ------
Md. Saef Ullah Miah,Junaida Sulaiman, Talha Bin Sarwar, Ateeqa Naseer, Fasiha Ashraf,
Kamal Zuhairi Zamli and Rajan Jose
------/CreationDate: ------
D:20220127182535+08'00'
------/Creator: ------
LaTeX with hyperref
------/Keywords: ------
sentence boundary extraction; PDF to text conversion; NLP in material science; material
information system; material informatics; Materials 4.0; gensim; NLTK; Spacy
------/ModDate: ------
D:20220127112724+01'00'
------/Producer: ------
pdfTeX-1.40.21
------/Subject: ------
Given the growth of scientific literature on the web, particularly material science,
acquiring data precisely from the literature has become more significant. Material
information systems, or chemical information systems, play an essential role in
discovering data, materials, or synthesis processes using the existing scientific
literature. Processing and understanding the natural language of scientific literature is
the backbone of these systems, which depend heavily on appropriate textual content.
Appropriate textual content means a complete, meaningful sentence from a large chunk of
textual content. The process of detecting the beginning and end of a sentence and
extracting them as correct sentences is called sentence boundary extraction. The accurate
extraction of sentence boundaries from PDF documents is essential for readability and
natural language processing. Therefore, this study provides a comparative analysis of
different tools for extracting PDF documents into text, which are available as Python
libraries or packages and are widely used by the research community. The main objective is
to find the most suitable technique among the available techniques that can correctly
extract sentences from PDF files as text. The performance of the used techniques Pypdf2,
Pdfminer.six, Pymupdf, Pdftotext, Tika, and Grobid is presented in terms of precision,
recall, f-1 score, run time, and memory consumption. NLTK, Spacy, and Gensim Natural
Language Processing (NLP) tools are used to identify sentence boundaries. Of all the
techniques studied, the Grobid PDF extraction package using the NLP tool Spacy achieved
the highest f-1 score of 93% and consumed the least amount of memory at 46.13 MegaBytes.
```

[1] pypdf2.readthedocs. [Zugriff am 06 02 2024].

```
------/Title: ------
Sentence Boundary Extraction from Scientific Literature of Electric Double Layer Capacitor
Domain: Tools and Techniques
```

Listing 2: Ergebnisse des Codes aus Listing 1

2. **PDFMiner.six:** basiert auf der ursprünglichen Bibliothek PDFMiner, ein Werkzeug speziell für die Extraktion und Analyse von Informationen aus PDF-Dokumenten. Diese Bibliothek zeichnet sich durch ihre Fähigkeit aus, Text direkt aus dem Quellcode der PDF-Seiten zu extrahieren, wobei sie nicht nur auf den reinen Textinhalt abzielt, sondern auch präzise Angaben zu Standort, Schriftart und Farbe des Textes liefern kann. Durch ihren modularen Aufbau ermöglicht PDFMiner.six eine flexible Anpassung, indem einzelne Komponenten leicht ausgetauscht oder durch eigene Implementierungen ergänzt werden können, was über die bloße Textanalyse hinausgeht. Neben der Textextraktion bietet PDFMiner.six erweiterte Funktionen für die Gewinnung von Metadaten, Tabellen, interaktiven Formularen (Acroform), Bildern und getaggtem Inhalt, was die Bibliothek zu einem vielseitigen Werkzeug für eine Vielzahl von Anwendungsfällen macht [21, 23].
   Das folgende Beispiel[2], *Listing 3*, angewandt auf die PDF-Datei [21], veranschaulicht, wie PDFMiner.six zur Extraktion von Metadaten aus einer PDF-Datei verwendet wird:

```python
from pdfminer.pdfparser import PDFParser
from pdfminer.pdfdocument import PDFDocument
from pdfminer.high_level import extract_pages
print("Seitenanzahl: ",len(list(extract_pages('applsci-12-01352.pdf'))))
# Open PDF file
with open('applsci-12-01352.pdf', 'rb') as file:
    parser = PDFParser(file)
    document = PDFDocument(parser)
    # Access the metadata
    meta = document.info[0]
    print("------Metadaten: ------")
    for key, value in meta.items():
        print(f"-------{key}: ------\n{value.decode('utf-8')}")
```

Listing 3: Metadatenextraktion mit Hilfe von PDFMiner.six

   In diesem Code wird die **`PDFMiner.six`** Bibliothek verwendet, um eine PDF-Datei zu öffnen und die Metadaten auszulesen. Nach dem Öffnen der Datei **`'applsci-12-01352.pdf'`** erstellt der **`'PDFParser'`** ein Parser-Objekt, und **`'PDFDocument'`** verwendet dieses, um ein Dokument-Objekt zu erzeugen. Über **`'document.info[0]'`** werden die Metadaten des Dokuments abgerufen. Schließlich durchläuft der Code alle Schlüssel-Wert-Paare in den Metadaten und gibt sie formatiert aus, wobei **`'value.decode('utf-8')'`** verwendet wird, um

[2] stackoverflow. [Zugriff am 07 02 2024]. „und“ stackoverflow. [Zugriff am 22 03 2024]

sicherzustellen, dass die Werte korrekt als Strings dargestellt werden. *Listing 4* zeigt die Ergebnisse dieses Codes:

```
Seitenanzahl: 19
------Metadaten: ------
-------Author: ------
Md. Saef Ullah Miah,Junaida Sulaiman, Talha Bin Sarwar, Ateeqa Naseer, Fasiha Ashraf, Kamal
Zuhairi Zamli and Rajan Jose
-------CreationDate: ------
D:20220127182535+08'00'
-------Creator: ------
LaTeX with hyperref
-------Keywords: ------
sentence boundary extraction; PDF to text conversion; NLP in material science; material
information system; material informatics; Materials 4.0; gensim; NLTK; Spacy
-------ModDate: ------
D:20220127112724+01'00'
-------Producer: ------
pdfTeX-1.40.21
-------Subject: ------
Given the growth of scientific literature on the web, particularly material science,
acquiring data precisely from the literature has become more significant. Material
information systems, or chemical information systems, play an essential role in discovering
data, materials, or synthesis processes using the existing scientific literature. Processing
and understanding the natural language of scientific literature is the backbone of these
systems, which depend heavily on appropriate textual content. Appropriate textual content
means a complete, meaningful sentence from a large chunk of textual content. The process of
detecting the beginning and end of a sentence and extracting them as correct sentences is
called sentence boundary extraction. The accurate extraction of sentence boundaries from
PDF documents is essential for readability and natural language processing. Therefore, this
study provides a comparative analysis of different tools for extracting PDF documents into
text, which are available as Python libraries or packages and are widely used by the research
community. The main objective is to find the most suitable technique among the available
techniques that can correctly extract sentences from PDF files as text. The performance of
the used techniques Pypdf2, Pdfminer.six, Pymupdf, Pdftotext, Tika, and Grobid is presented
in terms of precision, recall, f-1 score, run time, and memory consumption. NLTK, Spacy,
and Gensim Natural Language Processing (NLP) tools are used to identify sentence boundaries.
Of all the techniques studied, the Grobid PDF extraction package using the NLP tool Spacy
achieved the highest f-1 score of 93% and consumed the least amount of memory at 46.13
MegaBytes.
-------Title: ------
Sentence Boundary Extraction from Scientific Literature of Electric Double Layer Capacitor
Domain: Tools and Techniques
```

Listing 4: Ergebnisse des Codes aus Listing 3

Beim Vergleich der Metadatenextraktionsergebnisse zwischen PyPDF2 und PDFMiner.six wurden keine Unterschiede festgestellt. Die Ergebnisse ähneln sich sogar in der Reihenfolge ihrer Darstellung.

3. **PyMuPDF:** ist eine leistungsstarke Python-Bibliothek, die als Wrapper für das MuPDF-Toolkit dient, einem effizienten Werkzeug zur Darstellung von PDFs, XPS und E-Books. Es zeichnet sich durch hohe Qualität und Geschwindigkeit bei der Verarbeitung von Dokumenten aus. PyMuPDF ermöglicht nicht nur die Extraktion von Text unter Beibehaltung des Dokumentlayouts, sondern bietet auch Zugriff auf Metadaten von PDF-Dokumenten. Diese Fähigkeit zur Metadatenextraktion erweitert die Möglichkeiten der Bibliothek erheblich, indem sie wichtige Dokumentinformationen wie Titel, Autor und Erstellungsdaten zugänglich macht. Darüber hinaus unterstützt PyMuPDF eine Vielzahl weiterer Funktionen für die Datenextraktion, Analyse, Konvertierung und Manipulation, was sie zu einem

wertvollen Tool für die Arbeit mit PDF- und anderen Dokumenten in Python macht [21, 24].

Das folgende Beispiel[3], *Listing 5*, angewandt auf die PDF-Datei [21], veranschaulicht, wie PyMuPDF zur Extraktion von Metadaten aus einer PDF-Datei verwendet wird:

```
1. import fitz  # PyMuPDF
2. # Open PDF file
3. with fitz.open('applsci-12-01352.pdf') as doc:
4.     meta = doc.metadata
5.     print("Seitenanzahl:", len(doc))
6.     print("------Metadaten: ------")
7.     for key, value in meta.items():
8.         print(f"-------{key}: ------\n{value}")
```

Listing 5: Metadatenextraktion mit Hilfe von PyMuPDF

In diesem Code wird die **PyMuPDF** Bibliothek verwendet, um eine PDF-Datei zu öffnen und deren Metadaten sowie die Gesamtanzahl der Seiten zu extrahieren. Nach dem Öffnen der Datei **'applsci-12-01352.pdf'** wird die Seitenanzahl mit **'len(doc)'** ermittelt. Anschließend werden die Metadaten ausgelesen, die Informationen wie Autor, Erstellungsdatum und ähnliches enthalten können. Jedes Schlüssel-Wert-Paar der Metadaten wird durchlaufen und formatiert ausgegeben. *Listing 6* zeigt die Ergebnisse dieses Codes.

```
Seitenanzahl: 19
------Metadaten: ------
-------format: ------
PDF 1.7
-------title: ------
Sentence Boundary Extraction from Scientific Literature of Electric Double Layer Capacitor Domain: Tools and Techniques
-------author: ------
Md. Saef Ullah Miah,Junaida Sulaiman, Talha Bin Sarwar, Ateeqa Naseer, Fasiha Ashraf, Kamal Zuhairi Zamli and Rajan Jose
-------subject: ------
Given the growth of scientific literature on the web, particularly material science, acquiring data precisely from the literature has become more significant. Material information systems, or chemical information systems, play an essential role in discovering data, materials, or synthesis processes using the existing scientific literature. Processing and understanding the natural language of scientific literature is the backbone of these systems, which depend heavily on appropriate textual content. Appropriate textual content means a complete, meaningful sentence from a large chunk of textual content. The process of detecting the beginning and end of a sentence and extracting them as correct sentences is called sentence boundary extraction. The accurate extraction of sentence boundaries from PDF documents is essential for readability and natural language processing. Therefore, this study provides a comparative analysis of different tools for extracting PDF documents into text, which are available as Python libraries or packages and are widely used by the research community. The main objective is to find the most suitable technique among the available techniques that can correctly extract sentences from PDF files as text. The performance of the used techniques Pypdf2, Pdfminer.six, Pymupdf, Pdftotext, Tika, and Grobid is presented in terms of precision, recall, f-1 score, run time, and memory consumption. NLTK, Spacy, and Gensim Natural Language Processing (NLP) tools are used to identify sentence boundaries. Of all the techniques studied, the Grobid PDF extraction package using the NLP tool Spacy achieved the highest f-1 score of 93% and consumed the least amount of memory at 46.13 MegaBytes.
-------keywords: ------
```

[3] pymupdf.readthedocs. [Zugriff am 08 02 2024].

```
sentence boundary extraction; PDF to text conversion; NLP in material science; material
information system; material informatics; Materials 4.0; gensim; NLTK; Spacy
-------creator: ------
LaTeX with hyperref
-------producer: ------
pdfTeX-1.40.21
-------creationDate: ------
D:20220127182535+08'00'
-------modDate: ------
D:20220127112724+01'00'
-------trapped: ------

-------encryption: ------
None
```

Listing 6: Ergebnisse des Codes aus Listing 5

Beim Vergleich der Metadatenextraktionsergebnisse zwischen PDFMiner.six, PyPDF2 und PyMuPDF zeigen sich deutliche Unterschiede. Zunächst ist die Reihenfolge der Metadaten bei PyMuPDF anders als bei den anderen beiden Tools. Darüber hinaus extrahiert PyMuPDF zusätzliche Informationen, wie das PDF-Format: **`'format: PDF 1.7'`**, was auf eine tiefere Analysefähigkeit hinweisen könnte. Während Daten zu **`'trapped'`** und **`'encryption'`** bei PyMuPDF auch erfasst werden, liefern sie in diesem Kontext keine weiterführenden, nutzbaren Ergebnisse.

4. **PDFPlumber:** ist eine leistungsstarke Python-Bibliothek, die speziell für die detaillierte Analyse von PDF-Dokumenten entwickelt wurde. Basierend auf PDFMiner.six, ermöglicht sie nicht nur das Extrahieren von Textzeichen, Seiten, Seitennummern, Matrizen und Linien, sondern auch von Tabellen. Ein weiterer Vorteil von PDFPlumber ist die Unterstützung visuellen Debuggings, was die Bibliothek besonders effektiv für maschinell erstellte PDFs macht. Neben der Umwandlung von PDF-Dateien in einfachere TXT-Dokumente und dem Bereitstellen umfangreicher Informationen über die strukturellen Elemente eines Dokuments, bietet PDFPlumber auch die Möglichkeit, Metadaten aus PDF-Dokumenten zu extrahieren. Diese Fähigkeit zur Metadatenextraktion erweitert das Einsatzspektrum von PDFPlumber erheblich, indem sie eine gründlichere Analyse und Verarbeitung der Dokumente ermöglicht und somit den Umgang mit unterschiedlichen Arbeitsanforderungen vereinfacht [25, 26].

   Das folgende Beispiel[4], *Listing 7*, angewandt auf die PDF-Datei [21], veranschaulicht, wie PDFPlumber zur Extraktion von Metadaten aus einer PDF-Datei verwendet wird:

```python
import pdfplumber
# Open PDF file
with pdfplumber.open("applsci-12-01352.pdf") as pdf:
    meta = pdf.metadata
    print("Seitenanzahl:", len(pdf.pages))
    print("------Metadaten: ------")
```

[4] github.pdfplumber. [Zugriff am 12 02 2024].

```
7.      for key, value in meta.items():
8.          print(f"-------{key}: ------\n{value}")
```

Listing 7: Metadatenextraktion mit Hilfe von PDFPlumber

In diesem Code wird die PDFPlumber Bibliothek verwendet, um eine PDF-Datei zu öffnen und ihre Metadaten sowie die Gesamtanzahl der Seiten zu extrahieren. Zuerst wird die PDF-Datei geöffnet und das PDF-Objekt erstellt. Dann werden die Metadaten des Dokuments ausgelesen und zusammen mit der Anzahl der Seiten in der Datei ausgegeben. Jedes Metadaten-Schlüssel-Wert-Paar wird in einer Schleife durchlaufen und formatiert dargestellt. Die Extraktionsergebnisse von Metadaten mit PDFPlumber sind ähnlich zu denen von PDFMiner und PyPDF2.

Die *Tabelle 1*, die sich auf vorab bereitgestellte Informationen und Quellen zu PyPDF2, PDFMiner.six, PyMuPDF und PDFPlumber stützt, stellt einen Vergleich zwischen den vier Bibliotheken PyPDF2, PDFMiner.six, PyMuPDF und PDFPlumber dar. Der Fokus dieses Vergleichs liegt ausschließlich auf der Fähigkeit zur Extraktion von Metadaten.

Funktion / Bibliothek	PyPDF2	PDFMiner.six	PyMuPDF	PDFPlumber
Metadatenextraktion	√	√	√	√
Einfachheit der Nutzung	Einfach	Mittel	Einfach	Einfach
Zugriff auf spezifische Metadaten	√	√	√	√
Benötigt externe Abhängigkeiten	X	X	Ja MuPDF[5]	X

Tabelle 1: Vergleich der Bibliotheken zur Metadatenextraktion

Die *Tabelle 1* beleuchtet folgende Punkte:

1. **Metadatenextraktion:** Alle vier Bibliotheken unterstützen die Extraktion von Metadaten aus PDF-Dateien.
2. **Einfachheit der Nutzung:** PyPDF2, PyMuPDF und PDFPlumber sind generell einfacher zu verwenden für die Metadatenextraktion, mit einer direkten und unkomplizierten Schnittstelle. PDFMiner.six bietet zwar auch direkten Zugriff, kann aber aufgrund seiner umfangreichen Analysefunktionen als etwas komplexer wahrgenommen werden.

[5] pymupdf.readthedocs. [Zugriff am 12 02 2024].

3. **Zugriff auf spezifische Metadaten:** Alle Bibliotheken ermöglichen den direkten Zugriff auf spezifische Metadatenfelder wie Autor, Titel usw.
4. **Benötigt externe Abhängigkeiten:** PyMuPDF benötigt MuPDF, eine externe Software, während die anderen Bibliotheken rein in Python implementiert sind.

3.2.2 Werkzeuge zur Textextraktion

Die zuvor erwähnten Bibliotheken, die für die Extraktion von Metadaten verwendet werden können, sind ebenfalls für die Extraktion von Texten aus PDF-Dateien geeignet. Diese Fähigkeit ist besonders relevant, da die vorliegende Arbeit als ein Folgeschritt einer vorherigen studentischen Arbeit [27] betrachtet wird, dessen Hauptziel die Textextraktion aus PDF-Dateien war. Aufgrund dieses direkten Bezugs und der Tatsache, dass eine detaillierte Untersuchung und Analyse die meisten dieser Bibliotheken bereits in der früheren Arbeit stattgefunden hat, wird in diesem Kontext auf eine erneute detaillierte Vorstellung oder eine tiefergehende Diskussion ihrer spezifischen Funktionen und Vorteile verzichtet. Stattdessen konzentrierte sich die aktuelle Arbeit auf die Nutzung dieser Tools im spezifischen Anwendungskontext der Metadatenextraktion.

3.2.3 Werkzeuge zur Textverarbeitung

Um die Effektivität der Textanalyse sicherzustellen und die gewünschten Ziele der extrahierten Texte zu erreichen, ist eine sorgfältige Vorbereitung der Texte notwendig. Dazu gehört die Entfernung von Wörtern mit geringer Bedeutung (Stoppwörter), das Extrahieren von Wortstämmen (Lemmatisierung) und weitere vorbereitende Schritte. In diesem Kontext werden die wichtigsten Bibliotheken und Tools im Bereich der Verarbeitung natürlicher Sprachen (NLP) vorgestellt, die in der Programmiersprache Python unterstützt werden, um einen effektiven und präzisen Umgang mit Texten zu ermöglichen.

1. **NLTK:** kurz für Natural Language Toolkit, ist eine umfassende und weit verbreitete Python-Bibliothek für die Verarbeitung natürlicher Sprachen (NLP). Sie bietet zahlreiche Funktionen zur Textanalyse, einschließlich Part-of-Speech-Tagging, Tokenisierung und Stemming. Darüber hinaus umfasst NLTK einen Modul zur Textzusammenfassung, der auf statistischen Methoden basiert, um wesentliche Sätze aus einem Text zu extrahieren. Diese Bibliothek beinhaltet diverse Algorithmen für NLP-Aufgaben wie Sentiment-Analyse und Themen-Segmentierung, und unterstützt die Analyse, Vorverarbeitung und das Verständnis von geschriebenem Text durch den Computer [28, 29, 30].
   Das folgende Beispiel[6], *Listing 8,* veranschaulicht wie NLTK zur Textverarbeitung verwendet werden kann:

```
1. import nltk
2. from nltk.corpus import stopwords
3. from nltk.tokenize import word_tokenize, sent_tokenize
4. nltk.download('punkt')
5. nltk.download('stopwords')
6. text = "Extraktion: In dieser Phase werden die Daten aus den
   verschiedenen Quellen extrahiert, sei es aus Datenbanken, Dateien oder
   anderen Datenquellen. Dieser Schritt beinhaltet die Gewinnung von
   Rohdaten in ihrer ursprünglichen Form. "
7. print("----- raw text: -----\n",text)
8. # Lower Casing
9. text_lower = text.lower()
10. print("----- Lower Casing: -----\n",text_lower)
11. # Sentence Tokenization
12. sentences = sent_tokenize(text_lower)
13. print("----- Sentence Tokenization: -----\n",sentences)
14. # Word Tokenization
15. words = word_tokenize(text_lower)
16. print("----- Word Tokenization: -----\n",words)
17. # Remove Stop Words
18. stop_words = set(stopwords.words('german'))
19. filtered_words = [word for word in words if word not in stop_words]
20. print("----- Remove Stop Words: -----\n",filtered_words)
```

Listing 8: einfaches Beispiel, wie NLTK zu Textverarbeitung verwendet werden kann

In diesem Codebeispiel wird die NLTK-Bibliothek zur Vorbereitung und Analyse eines Textes verwendet. Der Text durchläuft mehrere Schritte: Er wird zuerst in Kleinbuchstaben umgewandelt, anschließend in Sätze und Wörter aufgeteilt und schließlich werden unwichtige Stoppwörter entfernt. Dies bildet eine Grundlage für weiterführende Analysen. Es gibt allerdings noch viele weitere Schritte, die in der Textverarbeitung Anwendung finden können, wie z.B. Lemmatisierung, Stemming oder Named Entity Recognition, die hier nicht durchgeführt wurden. Diese Verarbeitung transformiert den Rohdatentext in eine strukturierte Form, die für

[6] becominghuman. [Zugriff am 14 02 2024].

weiterführende NLP-Analysen besser geeignet ist, indem sie eine konsistente Basis schafft und den Text von irrelevanten Elementen bereinigt.

Listing 9 zeigt die Ergebnisse dieses Codes:

```
----- raw text: -----
 Extraktion: In dieser Phase werden die Daten aus den verschiedenen Quellen extrahiert, sei es aus Datenbanken, Dateien oder anderen Datenquellen. Dieser Schritt beinhaltet die Gewinnung von Rohdaten in ihrer ursprünglichen Form.
----- Lower Casing: -----
 extraktion: in dieser phase werden die daten aus den verschiedenen quellen extrahiert, sei es aus datenbanken, dateien oder anderen datenquellen. dieser schritt beinhaltet die gewinnung von rohdaten in ihrer ursprünglichen form.
----- Sentence Tokenization: -----
 ['extraktion: in dieser phase werden die daten aus den verschiedenen quellen extrahiert, sei es aus datenbanken, dateien oder anderen datenquellen.', 'dieser schritt beinhaltet die gewinnung von rohdaten in ihrer ursprünglichen form.']
----- Word Tokenization: -----
 ['extraktion', ':', 'in', 'dieser', 'phase', 'werden', 'die', 'daten', 'aus', 'den', 'verschiedenen', 'quellen', 'extrahiert', ',', 'sei', 'es', 'aus', 'datenbanken', ',', 'dateien', 'oder', 'anderen', 'datenquellen', '.', 'dieser', 'schritt', 'beinhaltet', 'die', 'gewinnung', 'von', 'rohdaten', 'in', 'ihrer', 'ursprünglichen', 'form', '.']
----- Remove Stop Words: -----
 ['extraktion', ':', 'phase', 'daten', 'verschiedenen', 'quellen', 'extrahiert', ',', 'sei', 'datenbanken', ',', 'dateien', 'datenquellen', '.', 'schritt', 'beinhaltet', 'gewinnung', 'rohdaten', 'ursprünglichen', 'form', '.']
```

Listing 9: Ergebnisse des Codes aus Listing 8

2. **spaCy[7]:** ist eine umfassende Open-Source-Bibliothek für die Verarbeitung natürlicher Sprache (NLP) in Python. Sie bietet eine breite Palette von Funktionen, darunter Named Entity Recognition, Tokenisierung, Part-of-Speech-Tagging und einen Textzusammenfassungsmodul, das auf einem Graph basierten Verfahren beruht. Mit Unterstützung für über 75 Sprachen und 84 trainierte Pipelines für 25 Sprachen ermöglicht spaCy die Nutzung von vortrainierten Modellen und Vektoren sowie eine tiefe Lernanalyse mit Bibliotheken wie TensorFlow und PyTorch. Zu den Kernfunktionen gehören auch die linguistisch motivierte Tokenisierung, Dependency Parsing und visuelle Darstellungen für Syntax und NER. SpaCy ist auf Geschwindigkeit und Produktionsbereitschaft ausgelegt, bietet benutzerdefinierte Modelle und ist leicht erweiterbar [30, 31].

   Das Beispiel[8], *Listing 10,* veranschaulicht wie spaCy zur Textverarbeitung verwendet werden kann:

```python
import spacy
# Load the German model
nlp = spacy.load("de_core_news_sm")
text = "Extraktion: In dieser Phase werden die Daten aus den verschiedenen Quellen extrahiert, sei es aus Datenbanken, Dateien oder anderen Datenquellen. Dieser Schritt beinhaltet die Gewinnung von Rohdaten in ihrer ursprünglichen Form."
print("----- raw text: -----\n", text)
# Process the text
doc = nlp(text)
```

[7] pypi.spacy. [Zugriff am 15 02 2024].

[8] spacy.usage. „und“ spacy.token. [Zugriff am 15 02 2024].

```
# Lower Casing
print("----- Lower Casing: -----\n", text.lower())
# Sentence Tokenization
sentences = [sent.text for sent in doc.sents]
print("----- Sentence Tokenization: -----\n", sentences)
# Word Tokenization
words = [token.text for token in doc]
print("----- Word Tokenization: -----\n", words)
# Remove Stop Words
filtered_words = [word.text for word in doc if not word.is_stop]
print("----- Remove Stop Words: -----\n", filtered_words)
```

Listing 10: einfaches Beispiel, wie spaCy zu Textverarbeitung verwendet werden kann

In diesem Codebeispiel wird die spaCy Bibliothek verwendet, um einen deutschen Text für die Verarbeitung natürlicher Sprache vorzubereiten. Nach dem Laden des deutschen Modells wird der Text zuerst in Kleinbuchstaben umgewandelt. Anschließend erfolgt die Aufteilung in Sätze und Wörter (Tokenisierung). Abschließend werden Stoppwörter entfernt, um einen gefilterten Text zu erhalten, der für weitere Analysen geeignet ist. *Listing 11* zeigt die Ergebnisse des Codes:

```
----- raw text: -----
 Extraktion: In dieser Phase werden die Daten aus den verschiedenen Quellen extrahiert, sei es aus Datenbanken, Dateien oder anderen Datenquellen. Dieser Schritt beinhaltet die Gewinnung von Rohdaten in ihrer ursprünglichen Form.
----- Lower Casing: -----
 extraktion: in dieser phase werden die daten aus den verschiedenen quellen extrahiert, sei es aus datenbanken, dateien oder anderen datenquellen. dieser schritt beinhaltet die gewinnung von rohdaten in ihrer ursprünglichen form.
----- Sentence Tokenization: -----
 ['extraktion: in dieser phase werden die daten aus den verschiedenen quellen extrahiert, sei es aus datenbanken, dateien oder anderen datenquellen.', 'dieser schritt beinhaltet die gewinnung von rohdaten in ihrer ursprünglichen form.']
----- Word Tokenization: -----
 ['extraktion', ':', 'in', 'dieser', 'phase', 'werden', 'die', 'daten', 'aus', 'den', 'verschiedenen', 'quellen', 'extrahiert', ',', 'sei', 'es', 'aus', 'datenbanken', ',', 'dateien', 'oder', 'anderen', 'datenquellen', '.', 'dieser', 'schritt', 'beinhaltet', 'die', 'gewinnung', 'von', 'rohdaten', 'in', 'ihrer', 'ursprünglichen', 'form', '.']
----- Remove Stop Words: -----
 ['extraktion', ':', 'phase', 'daten', 'verschiedenen', 'quellen', 'extrahiert', ',', 'datenbanken', ',', 'dateien', 'datenquellen', '.', 'schritt', 'beinhaltet', 'gewinnung', 'rohdaten', 'ursprünglichen', 'form', '.']
```

Listing 11: Ergebnisse des Codes aus Listing 10

Beim Vergleich der Stoppwortbehandlung durch NLTK und spaCy wurde festgestellt, dass spaCy das Wort **`'sei'`** als Stoppwort erkennt und entfernt, während es bei NLTK erhalten bleibt. Hintergrund ist es, spaCy hat umfangreichere Liste[9] von Stoppwörtern, die dreimal so groß ist wie die von NLTK[10]. Diese Unterschiede in der Stoppwortbehandlung können entscheidende Auswirkungen auf die Textverarbeitung und Analyse haben.

[9] spaCy.stop_words. [Zugriff am 15 02 2024].

[10] nltk.data. [Zugriff am 15 02 2024].

3. **TextBlob:** ist eine benutzerfreundliche Python-Bibliothek für Textverarbeitung und natürliche Sprachverarbeitung (NLP), die auf NLTK aufbaut. Sie unterstützt zahlreiche NLP-Aufgaben wie Klassifikation, Sentiment-Analyse und Spracherkennung. TextBlob bietet eine einfache API und ist besonders für Einsteiger geeignet. TextBlob ermöglicht es, Emotionen in Texten zu erkennen, indem es zwei Schlüsselmaße verwendet: Polarität und Subjektivität. Die Polarität variiert zwischen -1 und +1, wobei negative Werte negative Emotionen und positive Werte positive Emotionen anzeigen. Die Subjektivität, die zwischen 0 und 1 skaliert, gibt an, wie sehr eine Aussage auf Fakten oder persönlichen Meinungen basiert, wobei höhere Werte auf eine subjektivere Perspektive hinweisen. Es eignet sich hervorragend für grundlegende NLP-Tätigkeiten und übertrifft in der Textanalyse teilweise NLTK [32, 33].

   Das folgende Beispiel[11], *Listing 12,* veranschaulicht wie TextBlob zur Textverarbeitung verwendet werden kann:

```
1. from textblob import TextBlob
2. text = "Extraktion: In dieser Phase werden die Daten aus den
   verschiedenen Quellen extrahiert, sei es aus Datenbanken, Dateien oder
   anderen Datenquellen. Dieser Schritt beinhaltet die Gewinnung von
   Rohdaten in ihrer ursprünglichen Form."
3. print("----- raw text: -----\n", text)
4. # Creating a TextBlob object
5. blob = TextBlob(text)
6. # Lower Casing
7. text_lower = blob.lower()
8. print("----- Lower Casing: -----\n", text_lower)
9. # Sentence Tokenization
10. sentences = [sentence for sentence in text_lower.sentences]
11. print("----- Sentence Tokenization: -----\n", sentences)
12. # Word Tokenization
13. words = [word for word in text_lower.words]
14. print("----- Word Tokenization: -----\n", words)
```

Listing 12: einfaches Beispiel, wie TextBlob zu Textverarbeitung verwendet werden kann

In diesem Code wird die TextBlob Bibliothek verwendet, um einen gegebenen Text zu verarbeiten. Zunächst wird der Text in Kleinbuchstaben umgewandelt, um eine einheitliche Basis für die weitere Verarbeitung zu schaffen. Anschließend wird der Text in Sätze und Wörter aufgeteilt (Tokenisierung). Dieser Prozess hilft bei der Strukturierung des Textes für Analysen. Obwohl TextBlob für viele NLP-Aufgaben genutzt werden kann, bietet es keine eingebaute Funktion zur Entfernung von Stoppwörtern. Für diese spezielle Aufgabe wird häufig auf eine andere Bibliothek wie NLTK zurückgegriffen, die eine umfangreiche Liste von Stoppwörtern bereitstellt und

[11] readthedocs.textblob. [Zugriff am 17 02 2024].

eine einfache Integration mit TextBlob ermöglicht. *Listing 13* zeigt die Ergebnisse dieses Codes:

```
----- raw text: -----
 Extraktion: In dieser Phase werden die Daten aus den verschiedenen Quellen extrahiert, sei es aus Datenbanken, Dateien oder anderen Datenquellen. Dieser Schritt beinhaltet die Gewinnung von Rohdaten in ihrer ursprünglichen Form.
----- Lower Casing: -----
 extraktion: in dieser phase werden die daten aus den verschiedenen quellen extrahiert, sei es aus datenbanken, dateien oder anderen datenquellen. dieser schritt beinhaltet die gewinnung von rohdaten in ihrer ursprünglichen form.
----- Sentence Tokenization: -----
 [Sentence("extraktion: in dieser phase werden die daten aus den verschiedenen quellen extrahiert, sei es aus datenbanken, dateien oder anderen datenquellen."), Sentence("dieser schritt beinhaltet die gewinnung von rohdaten in ihrer ursprünglichen form.")]
----- Word Tokenization: -----
 ['extraktion', 'in', 'dieser', 'phase', 'werden', 'die', 'daten', 'aus', 'den', 'verschiedenen', 'quellen', 'extrahiert', 'sei', 'es', 'aus', 'datenbanken', 'dateien', 'oder', 'anderen', 'datenquellen', 'dieser', 'schritt', 'beinhaltet', 'die', 'gewinnung', 'von', 'rohdaten', 'in', 'ihrer', 'ursprünglichen', 'form']
```

Listing 13: Ergebnisse des Codes aus Listing 12

Die folgende Tabelle 2[12] stellt einen Vergleich zwischen den drei Bibliotheken NLTK, spaCy und TextBlob dar. Der Fokus dieses Vergleichs liegt auf den Stärken und Schwächen jeder Bibliothek.

Vergleich Bibliothek	Vorteile	Nachteile
NLTK	1. Die bekannteste und umfangreichste NLP-Bibliothek mit vielen Erweiterungen von Drittanbietern 2. die Unterstützung der größten Anzahl an Sprachen im Vergleich zu anderen Bibliotheken.	1. schwierig zu erlernen und zu verwenden 2. langsam 3. teilt Text nur nach Sätzen auf 4. ohne die semantische Struktur zu analysieren, keine neuronalen Netzwerkmodelle.
spaCy	1. schnell 2. leicht zu erlernen und zu verwenden 3. nutzt neuronale Netzwerke für das Training von Modellen	weniger flexibel im Vergleich zu NLTK
TextBlob	1. einfach zu verwenden und intuitive Schnittstelle zu NLTK 2. bietet Sprachübersetzung und -erkennung, die durch Google Translate ermöglicht wird	1. langsam 2. keine neuronalen Netzwerkmodelle 3. keine integrierten Wortvektoren

Tabelle 2: Vergleich der Bibliotheken: NLTK, spaCy und TextBlob

[12] nlp.libraries.pros.and.cons. [Zugriff am 17 02 2024].

3.2.4 Datenspeicherung

Die Phase der Datenspeicherung ist ebenso wichtig wie die Extraktion von Kennzahlen oder die Textverarbeitung. Eine strukturierte und geordnete Datenspeicherung ist stets gefragt, denn nur so kann jederzeit effizient und schnell auf die Daten zugegriffen werden. Hier sind die wichtigsten Ergebnisse der Suche zusammengefasst:

3.2.4.1 Datenbankmanagementsysteme

Ein Datenbankmanagementsystem (DBMS) ist spezialisierte Software zur Verwaltung eines Datenbanksystems, bestimmt das Datenbankmodell und beeinflusst maßgeblich die Systemleistung[13]. DBMS sind komplex und sichern wichtige Anforderungen, einige davon sind:

1. **SQLite**[14] ist eine öffentlich zugängliche Datenbank-Engine, die für ihre Kompaktheit, Effizienz und Zuverlässigkeit bekannt ist. Ursprünglich im Jahr 2000 veröffentlicht, bietet sie eine eingebettete SQL-Datenbanklösung, die direkt in Anwendungen integriert wird, wodurch der Bedarf an separaten Datenbankservern entfällt. SQLite zeichnet sich durch seine plattformübergreifende Stabilität aus und wird aufgrund seiner einfachen Handhabung und hohen Portabilität in einer Vielzahl von Geräten und Anwendungen eingesetzt, von Mobiltelefonen bis zu großen Software-Systemen. Mit der Garantie der Entwickler, die Kompatibilität des Dateiformats bis zum Jahr 2050 beizubehalten, hat sich SQLite als eine langfristige Lösung für die Datenspeicherung etabliert [34, 35]. SQLite zeichnet sich durch eine Vielzahl an Merkmalen aus, einschließlich:
   a. **Flexibilität:** SQLite zeichnet sich durch seine Vielseitigkeit als eingebettete Datenbank aus. Sie kombiniert die Vorteile einer relationalen Datenbank mit der Einfachheit einer B-Baum-Struktur. SQLite erfordert keine komplexen Serverkonfigurationen, eliminiert Verbindungsprobleme, ist plattformunabhängig und verursacht keine Lizenzgebühren. Es bietet eine einfache Integration von SQL-Unterstützung direkt in Anwendungen [34].
   b. **Zuverlässigkeit:** SQLite überzeugt durch seine hochwertige, gut kommentierte Codebasis aus ANSI C, leicht verständlich und anpassbar für C-Programmierer. Die API unterstützt Erweiterungen durch benutzerdefinierte Funktionen. Die

[13] Wikipedia: Datenbankmanagementsystem. [Zugriff am 30 03 2024].
[14] sqlite. [Zugriff am 30 03 2024].

außergewöhnliche Zuverlässigkeit von SQLite basiert auf einer umfangreichen Testabdeckung mit über 600 Testzeilen pro Codezeile, die 100% des Codes abdecken, einschließlich diverser Tests für kritische Fehlerbedingungen [34, 35].

c. **Schnelligkeit[15]:** SQLite zeigt herausragende Leistungsfähigkeit in der Datenverarbeitung, mit der Fähigkeit, zehntausende Transaktionen pro Sekunde zu unterstützen. Es verarbeitet Blob-Daten schneller und effizienter als herkömmliche Dateisysteme, was zu einer 35% schnelleren Lese- und Schreibgeschwindigkeit sowie 20% geringerem Speicherplatzbedarf führt. Diese Effizienz resultiert aus dem optimierten Umgang mit Datenbankoperationen, welche die häufigen Öffnungs- und Schließvorgänge von Dateien minimieren. Zukünftige Versionen von SQLite versprechen sogar noch bessere Leistungen [35].

d. **Keine Konfiguration[16]:** SQLite ist von Anfang an so konzipiert worden, dass es einfach ohne Datenbankadministrator integriert und verwendet werden kann. Es ist unkompliziert in der Konfiguration und Verwaltung, benötigt keinen Installations- oder Einrichtungsprozess und kommt ohne einen Serverprozess aus. SQLite verlangt keine Konfigurationsdateien, keine Datenbank-Instanz-Erstellung oder Benutzerberechtigungen und erfordert keine Wiederherstellungsmaßnahmen nach Systemausfällen. Seine Einfachheit und Selbstständigkeit minimieren den administrativen Aufwand und Speicherbedarf [34].

e. **Vollständige SQL-Implementierung mit fortgeschrittenen Funktionen[15]**, wie partielle Indizes, Ausdrucksindizes, JSON, gemeinsame Tabellenausdrücke und Fensterfunktionen.

2. **MySQL:** MySQL ist ein leistungsstarker, mehrbenutzerfähiger und robuster SQL-Datenbankserver, der für hochbelastete Produktionssysteme sowie für die Einbindung in weit verbreitete Software konzipiert ist. MySQL unterliegt einer Dualen Lizenzierung, wobei Nutzer die Wahl haben, es unter den Bedingungen der GNU General Public License als Open-Source-Produkt zu verwenden. Oracle und MySQL sind eingetragene Marken von Oracle Corporation [36, 37]. MySQL zeichnet sich durch eine Vielzahl an Merkmalen aus, einschließlich:

   a. **Vielfalt an Datentypen:** MySQL bietet eine vielfältige Auswahl an Datentypen, darunter ganze Zahlen (sowohl vorzeichenbehaftet als auch vorzeichenlos), Gleitkommazahlen, Zeichenketten, Datum- und Zeitangaben sowie spezialisierte

[15] sqlite.features. [Zugriff am 30 03 2024].
[16] sqlite.zeroconf. [Zugriff am 30 03 2024].

Typen wie ENUM und SET für Listen fester Werte. Diese Vielfalt ermöglicht eine flexible Datenstrukturierung für unterschiedlichste Anwendungsfälle [36, 37].

b. **Client-Server-Architektur** wird von MySQL unterstützt. Bei dieser Architektur wird der MySQL-Datenbankserver als eigenständiges Programm in einer vernetzten Umgebung eingesetzt. Der Server kann sowohl auf demselben Rechner als auch auf einem entfernten Rechner innerhalb eines Netzwerks oder über das Internet betrieben werden, wobei die MySQL-Clients mit dem Server kommunizieren [36, 37].
c. **Hohe Leistungsfähigkeit:** MySQL zeichnet sich durch seine hohe Leistungsfähigkeit aus, die es ermöglicht, mit dem zunehmenden Datenvolumen Schritt zu halten. Dank seiner effizienten Ladewerkzeuge, einem dedizierten Speicher-Cache und weiteren Leistungssteigerungsmechanismen kann MySQL große Datenmengen effektiv verarbeiten [37].
d. **Zeichensätze:** MySQL bietet vollständige Unterstützung für zahlreiche Zeichensätze, einschließlich „latin1 (cp1252)“, „big5“, „ujis“ und verschiedene Unicode-Zeichensätze. So sind beispielsweise skandinavische Zeichen wie „å“, „ä“ und „ö“ auch in Tabellen- und Spaltennamen zulässig. Diese umfassende Kompatibilität ermöglicht eine flexible Anwendung in verschiedenen Sprach- und Datenkontexten [36].
e. **SQL-Funktionen** werden mithilfe einer hoch optimierten Klassenbibliothek implementiert, die höchstmögliche Geschwindigkeit gewährleisten soll. Typischerweise findet nach der Initialisierung einer Abfrage überhaupt keine Speicherzuweisung mehr statt. Diese Effizienz trägt zur Steigerung der Gesamtleistung bei und optimiert die Verarbeitung von Abfragen [36].

Die *Tabelle 3* basiert auf den vorigen Abschnitten, *1 SQLite und 2 MySQL*, und stellt die wesentlichen Unterschiede sowie charakteristischen Merkmale von SQLite und MySQL dar:

Merkmale / DBMS	SQLite	MySQL
Einführung & Verwendung	Öffentlich zugängliche, eingebettete SQL-Datenbanklösung ohne separaten Datenbankserver, direkt in Anwendungen integriert.	Leistungsstarker, mehrbenutzerfähiger SQL-Datenbankserver für hochbelastete Produktionssysteme und Softwareintegration.
Flexibilität	Vielseitigkeit durch Einbettung direkt in Anwendungen, keine komplexen Serverkonfigurationen erforderlich.	Unterstützt eine Vielzahl an Datentypen, ermöglicht flexible Datenstrukturierung.

Zuverlässigkeit	Hochwertige, gut kommentierte Codebasis aus ANSI C. Außergewöhnliche Zuverlässigkeit durch umfangreiche Testabdeckung.	Client-Server-Architektur für vernetzte Umgebungen, ermöglicht Fernzugriff über Netzwerke oder Internet.
Schnelligkeit	Unterstützt zehntausende Transaktionen pro Sekunde mit optimiertem Umgang mit Datenbankoperationen.	Zeichnet sich durch hohe Leistungsfähigkeit aus, effektive Verarbeitung großer Datenmengen durch effiziente Werkzeuge und Speicher-Cache.
Konfiguration	Keine Konfiguration erforderlich, unkompliziert in der Verwaltung und Nutzung.	Bietet Unterstützung für zahlreiche Zeichensätze, erlaubt flexible Anwendung in verschiedenen Sprach- und Datenkontexten.
SQL-Implementierung	Vollständige SQL-Implementierung mit fortgeschrittenen Funktionen wie partielle Indizes und Fensterfunktionen.	Implementiert SQL-Funktionen mit einer hoch optimierten Klassenbibliothek für höchstmögliche Geschwindigkeit.

Tabelle 3: Vergleich zwischen SQLite und MySQL

3.3 Auswahl der Hauptergebnisse

In diesem Abschnitt wird dargelegt, welche Auswahl für die Werkzeuge zur Textverarbeitung und Metadatenextraktion getroffen wurde. Es wird diskutiert, warum speziell PyPDF2 und NLTK für das geplante Vorhaben ausgewählt wurden und inwiefern sie zur Erreichung der gesetzten Ziele der Arbeit beitragen. Die Gründe für die Auswahl orientieren sich an der Kompatibilität mit bestehenden Strukturen, der Einfachheit der Anwendung und der reichhaltigen Funktionsvielfalt, die eine effiziente Umsetzung der Projektziele unterstützen.

3.3.1 PyPDF2

Kompatibilität mit Vorarbeiten: Die Entscheidung, PyPDF2 zu wählen, basiert auf der erfolgreichen Anwendung von PDFMiner in früheren Arbeiten von Alkhamis. Obwohl PDFMiner detaillierte Ergebnisse lieferte, bietet PyPDF2 ähnliche Funktionalitäten mit einer einfacheren Schnittstelle und ist für die Extraktion ausreichend, vor allem, wenn die Performance-Anforderungen nicht so hoch sind wie bei größeren Datensätzen.

Einfachheit der Anwendung: PyPDF2 ist bekannt für seine einfache und unkomplizierte Anwendung. Die Benutzerfreundlichkeit und die Fähigkeit, schnell integriert zu werden, sind bei der Fortführung von Projektarbeiten von Vorteil, da sie die Lernkurve reduzieren und auf vorhandenem Wissen aufbauen.

3.3.2 NLTK

Vielseitigkeit und Ressourcenreichtum[17]: NLTK bietet eine umfangreiche Sammlung von Ressourcen, Tools und vorverarbeiteten Korpora, die es ermöglicht, vielfältige Aufgaben der Sprachverarbeitung zu bewältigen. Diese Vielfalt macht NLTK zu einer bevorzugten Wahl für Forschung und Entwicklung, wo Anpassungsfähigkeit und eine breite Funktionspalette erforderlich sind.

Etablierte Gemeinschaft und Unterstützung[12]: NLTK verfügt über eine große Gemeinschaft, die eine umfangreiche Dokumentation und viele Diskussionsforen für Support bietet. Diese Aspekte sind besonders für wissenschaftliche Arbeiten und Lehrzwecke wertvoll, wo Zuverlässigkeit und Unterstützung kritisch sein können.

Bildungsorientiert[12]: NLTK ist besonders in Bildungsumgebungen beliebt, da es eine breite Palette an Funktionalitäten in einer Lernumgebung bietet, was es ideal für akademische Projekte macht, die Wert auf das Verständnis der zugrunde liegenden Prozesse und Algorithmen legen.

[17] nltk.org. [Zugriff am 18 02 2024].

3.3.3 DBMS

Angesichts der Zielsetzung dieser Arbeit, eine umfangreiche Menge an Merkmalen – darunter numerische Werte, textuelle Informationen und Metadaten – aus Textdokumenten zu extrahieren und strukturiert in separaten Tabellen zu speichern, wurde MySQL als Datenbankmanagementsystem ausgewählt, Diese Begründungen basieren auf zuvor genannten Informationen, *siehe Seite 32 Punkt 2 MySQL*. MySQL bietet eine reiche Vielfalt an Datentypen und fortgeschrittene Funktionen für eine flexible Datenmodellierung, die es ermöglichen, die unterschiedlichen Arten von extrahierten Merkmalen entsprechend zu repräsentieren und zu speichern. Insbesondere erlaubt MySQL die Bildung relationaler Verknüpfungen zwischen den Tabellen, was eine essenzielle Anforderung für die strukturierte Speicherung der extrahierten Daten in separaten, aber miteinander in Beziehung stehenden Tabellen darstellt. Diese Eigenschaft ist entscheidend für das Ziel, eine klare und effiziente Organisation der Daten zu gewährleisten, die wiederum schnellen Zugriff und tiefgehende Analysen ermöglicht. Zudem bietet MySQL herausragende Leistungsfähigkeit und Skalierbarkeit, die sicherstellen, dass auch bei großen Datenmengen und komplexen Abfragen die Systemperformance erhalten bleibt. Diese Aspekte, zusammen mit der Verfügbarkeit umfassender Dokumentation, machen MySQL zur optimalen Wahl für die Datenverwaltung in dieser Arbeit.

4 Konzeption des Verfahrens

In diesem Kapital wird das Konzept detailliert vorgestellt, das in fortgeschrittenen Phasen des Projekts umgesetzt wird. Ziel ist es, ein tiefgehendes Verständnis für die Absichten und Erwartungen hinsichtlich der angestrebten Ergebnisse zu fördern. Jede Phase wird einzeln analysiert, wobei der Fokus auf der Erläuterung ihrer Bedeutung im Gesamtrahmen des Projekts liegt. Dieser Ansatz zielt darauf ab, Transparenz in den Prozess der strategischen Planung und die dahinterstehenden Ziele zu bringen, was die lebenswichtige Rolle jedes Schritts im Entwicklungs- und Implementierungsprozess hervorhebt. Durch diese tiefgreifende Herangehensweise wird danach gestrebt, eine klare und detaillierte Vision zu präsentieren, die zu einem besseren Verständnis darüber beiträgt, wie die gesetzten Ziele erreicht werden können. Dies geschieht durch eine umfassende Erläuterung der verschiedenen Phasen, mit einem besonderen Augenmerk auf die Verbindung zwischen ihnen und den erwarteten Ergebnissen.

Die Pipeline, in der untenstehenden *Abbildung 3* dargestellt, beginnt mit dem Extrahieren von Text aus PDF-Dokumenten. Dieser wichtige erste Schritt wird es ermöglichen, den Rohinhalt für die nachfolgenden Verarbeitungsphasen verfügbar zu machen. Nach der Extraktion wird der Text aufgeteilt, um eine strukturierte Aufteilung in kleinere Abschnitte zu gewährleisten. Im nächsten Schritt der Vorverarbeitung wird der Text gereinigt und gefiltert. Dies beinhaltet die Entfernung von irrelevanten Zeichen und die Anwendung von anderen Techniken. Die Vorverarbeitung ist entscheidend, um die Qualität der Daten zu sichern und die Effizienz der Merkmalsextraktion zu verbessern. Die darauffolgende Merkmalsextraktion nutzt bestimmte Methoden, um aus dem vorverarbeiteten Text bedeutungsvolle Attribute zu identifizieren. Diese Merkmale könnten textuelle Muster oder statistische Eigenschaften sein, die für die spätere Analyse von Bedeutung sind. Abschließend werden die extrahierten Merkmale systematisch gespeichert. Die strukturierte Speicherung ermöglicht eine schnelle Wiederherstellung und einfache Integration in verschiedene Anwendungen, die auf maschinellem Lernen oder Datenanalyse basieren.

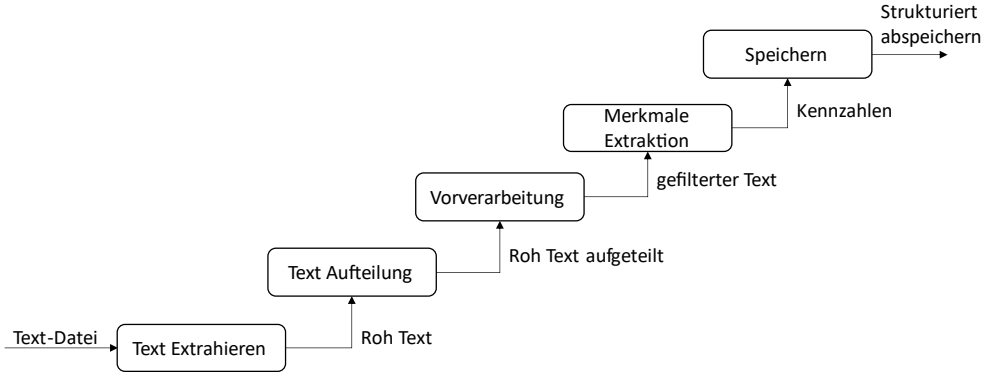

Abbildung 3: Pipeline für Extraktion der Kennzahlen aus PDF-Dateien

4.1 Textextraktion und Aufteilung

Die Textextraktion ist der unerlässliche erste Schritt in einer Textanalyse-Pipeline. Dieser Prozess befasst sich mit dem Einlesen und Laden von Textdaten aus einer PDF-Datei. Ziel ist es dabei, die ursprünglichen Daten zu erfassen und sie in eine bearbeitbare und analysierbare Form zu überführen. Es wird darauf geachtet, dass alle relevanten Informationen erhalten bleiben und in ein standardisiertes Format konvertiert werden, das eine kompatible Grundlage für die nachfolgenden Verarbeitungsschritte bildet. Die extrahierten Texte werden dann bereinigt und normalisiert, um sie für komplexe Analysen wie die Merkmalsextraktion, semantische Interpretation und das maschinelle Lernen vorzubereiten.

Die Textaufteilung ist ein wichtiger Schritt, bei dem der extrahierte Text in einzelne Absätze oder Segmente gegliedert wird. Diese Aufteilung ermöglicht es, jeden Abschnitt einzeln zu bearbeiten und zu analysieren. Ziel dieses Prozesses ist es, eine strukturierte Basis für die nachfolgende inhaltliche Analyse zu schaffen. Diese Gliederung des Textes ist besonders nützlich, um spezifische Aspekte des Textes detailliert untersuchen zu können. Durch die separate Verarbeitung jedes Absatzes wird die Effektivität der Analysewerkzeuge erhöht, was zu präziseren Ergebnissen führt und die Grundlage für fortgeschrittene Textanalysen bildet.

Die *Abbildung 4* veranschaulicht den Prozessablauf, beginnend mit der Textextraktion aus einer PDF-Datei, gefolgt von der anschließenden Textaufteilung.

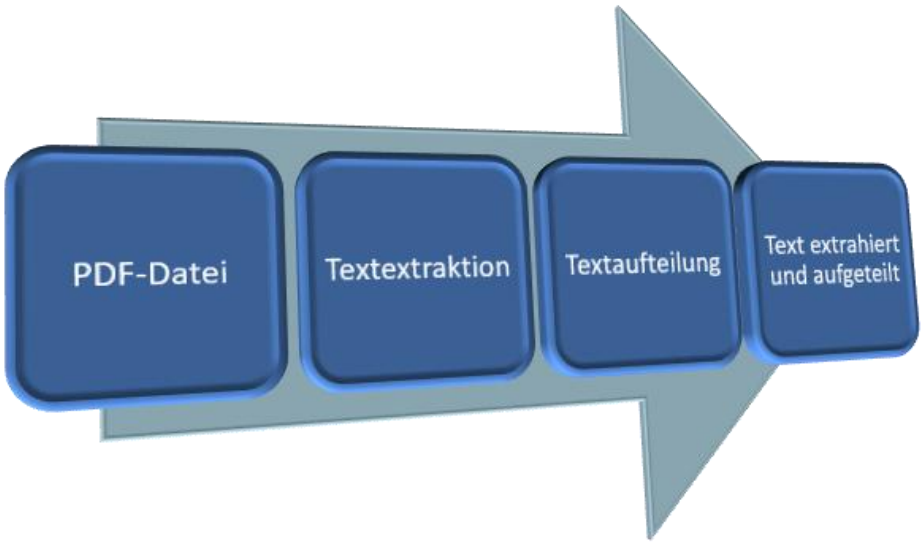

Abbildung 4: Text aus PDF-Datei extrahieren und aufteilen

4.2 Textvorverarbeitung

Die Vorverarbeitung des Textes ist ein entscheidender Schritt, der darauf abzielt, den Text zu reinigen und von irrelevanten Inhalten zu befreien, um eine einheitliche und analysierbare Datenbasis zu schaffen. Innerhalb dieses Prozesses werden verschiedene Techniken der Verarbeitung natürlicher Sprache angewendet, um die Qualität und Konsistenz der Daten zu verbessern. Dieser Schritt, in *Abbildung 5* dargestellt, beinhaltet fünf Techniken der Vorverarbeitung natürlicher Sprache: das Umwandeln in Kleinbuchstaben, die Tokenisierung, das Entfernen von Stoppwörtern, das Entfernen von Sonderzeichen und die Lemmatisierung.

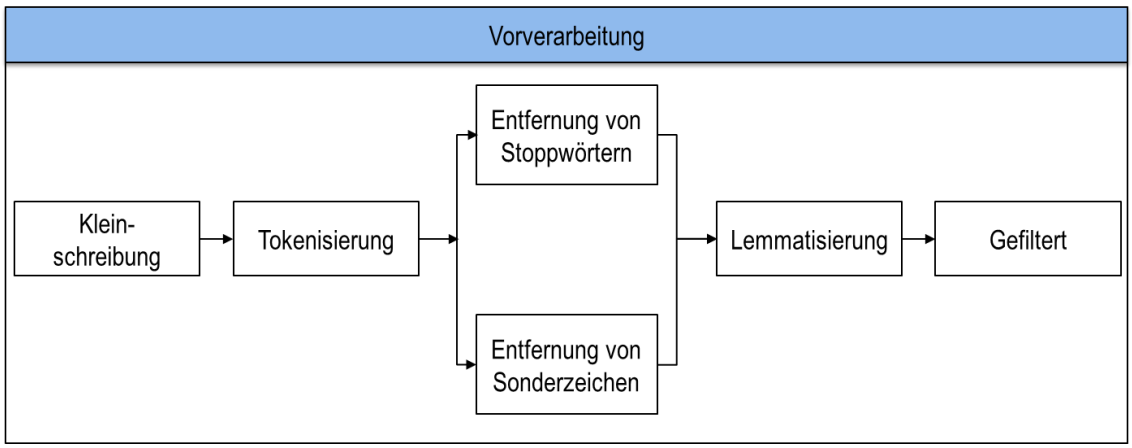

Abbildung 5: Ablauf der Vorverarbeitung des Textes

4.2.1 Kleinschreibung

Das Umwandeln aller Zeichen in Kleinbuchstaben ist ein einfacher, aber effektiver Schritt in der Vorverarbeitung von Textdaten. Durch diese Transformation werden sämtliche Großbuchstaben im Text in ihre entsprechenden Kleinbuchstaben umgewandelt. Diese Methode trägt dazu bei, die Komplexität der Datenanalyse zu reduzieren. Insbesondere wird durch die Vereinheitlichung der Schreibweise die Unterscheidung zwischen Wörtern, die am Satzanfang stehen, und solchen, die mitten im Satz vorkommen, aufgehoben.

4.2.2 Tokenisierung

Die Tokenisierung ist ein grundlegender Schritt in der Textverarbeitung, der darauf abzielt, Textabsätze in individuelle Einheiten zu zerlegen, die als Unigramme bezeichnet werden. Ein Unigramm repräsentiert in diesem Kontext ein einzelnes Wort oder Token. Dieser Prozess ist entscheidend für viele Anwendungen der natürlichen Sprachverarbeitung (NLP), da er die Basis für das Verständnis und die Analyse der Struktur und Bedeutung von Texten bildet. Durch die Tokenisierung wird der Text in eine Sequenz von Tokens umgewandelt, wobei jedes Token einem Wort oder einer Wortgruppe entspricht. Diese Methode ermöglicht es, den Text auf einer gründlichen Ebene zu untersuchen und zu verarbeiten. Es erleichtert nicht nur die Identifizierung und Analyse von Schlüsselwörtern und Phrasen, sondern auch die Anwendung weiterer Verarbeitungsschritte wie das Entfernen von Stoppwörtern und die Lemmatisierung.

4.2.3 Entfernung von Stoppwörtern

In der dritten Technik der Textvorverarbeitung, das Entfernen von Stoppwörtern, werden häufig vorkommende Wörter wie „my“, „is“, „are“ und „the“ aus dem Text eliminiert. Stoppwörter sind Wörter, die in der Sprache häufig auftreten, aber in der Regel wenig bis keinen Beitrag zum Verständnis des eigentlichen Inhalts oder Kontexts eines Textes leisten. Ihre Entfernung zielt darauf ab, den Text auf die wesentlichen, bedeutungstragenden Elemente zu reduzieren, um die Effizienz von Textanalysen zu steigern. Das Aussortieren dieser Wörter erleichtert die Fokussierung auf die für die Analyse relevanten Begriffe und reduziert die Größe der Daten, was wiederum die Verarbeitungsgeschwindigkeit und -genauigkeit bei Suchanfragen, maschinellem Lernen und anderen Formen der Datenanalyse verbessert. Durch die Reduktion auf die Kerninhalte wird es einfacher, Muster zu erkennen, Schlüsselbegriffe zu identifizieren und die zugrundeliegende Bedeutung eines Textes zu extrahieren.

4.2.4 Entfernung von Sonderzeichen

In der vierten Technik der Textvorverarbeitung werden Sonderzeichen und Zahlen, die für die Analyse von Textdaten oft irrelevant sind, systematisch entfernt. Dies umfasst eine Vielzahl von Symbolen wie ($, @, &,!, [, }, >) und ähnliche Zeichen, die keine bedeutungstragende Rolle spielen. Die Ausnahme bilden hierbei spezifische Satzzeichen wie (, . ; :), da diese Elemente wichtige Indikatoren für den Schreibstil einer Person sein können und unter Umständen Einfluss auf die Interpretation des Textes haben. Die Entfernung dieser Zeichen trägt dazu bei, den Text von potenziell störenden Elementen zu bereinigen und somit die Klarheit und Qualität der Daten für die nachfolgende Analyse zu verbessern. Indem irrelevante Symbole und Ziffern eliminiert werden, wird der Fokus stärker auf den eigentlichen Textinhalt gelegt, was die Effektivität von Algorithmen zur Textanalyse und das maschinelle Lernen positiv beeinflusst. Diese Technik vereinfacht nicht nur die Datenstruktur, sondern hilft auch dabei, Missverständnisse und Fehlinterpretationen zu vermeiden, die durch die Präsenz nicht-textlicher Zeichen entstehen könnten.

4.2.5 Lemmatisierung

Zum Abschluss des Vorverarbeitungsprozesses führt die Lemmatisierung eine tiefgehende Analyse der Wortformen durch, um deren Grundform zu ermitteln. Diese Technik berücksichtigt den Kontext, in dem ein Wort verwendet wird, sowie seine grammatische Funktion, um es auf seine Ursprungsform zurückzuführen. Das bedeutet, dass beispielsweise verschiedene konjugierte Formen eines Verbs wie "läuft", "lief" und "gelaufen" auf das Lemma "laufen" reduziert werden. Diese Methode unterscheidet sich von der einfacheren Stemming-Technik, indem sie eine präzisere und kontextbezogene Reduktion der Wörter auf ihre Basisformen ermöglicht. Lemmatisierung erfordert ein tieferes linguistisches Verständnis der Sprache, einschließlich der Kenntnis von Wortarten (wie Nomen, Verben, Adjektive) und deren Beziehungen im Satz. Der Einsatz der Lemmatisierung in der Textvorverarbeitung verbessert die Qualität der Daten für die anschließende Analyse, indem er die Vielfalt der Wortformen reduziert. Dies erleichtert die Erkennung von Mustern, die Zuordnung von Bedeutungen und die Durchführung von präzisen Textanalysen, da Wörter unabhängig von ihrer spezifischen Form oder Konjugation korrekt identifiziert und zugeordnet werden können.

4.2.6 Gefilterter Text

Nach der Anwendung der fünf Techniken der Vorverarbeitung natürlicher Sprache – Umwandlung in Kleinbuchstaben, Tokenisierung, Entfernen von Stoppwörtern, Entfernen von Sonderzeichen und Lemmatisierung – resultiert als Endprodukt ein sorgfältig gefilterter Text. Dieser Text ist von überflüssigen Elementen befreit und auf seine wesentlichen Inhalte reduziert, wodurch er für die weiterführende Analyse und Verarbeitung optimiert ist. Die Umwandlung aller Buchstaben in Kleinbuchstaben und die Tokenisierung erleichtern die strukturelle Analyse und die Identifikation von Schlüsselbegriffen. Das Entfernen von Stoppwörtern und Sonderzeichen trägt dazu bei, den Hintergrundrauschen im Text zu minimieren und die Relevanz der verbleibenden Daten zu maximieren. Die Lemmatisierung verdichtet den Text weiter, indem sie Wörter auf ihre Grundformen reduziert, was eine präzisere und bedeutungsvollere Analyse ermöglicht. Die Kollaboration dieser Techniken verbessert signifikant die Qualität der Textdaten und bildet eine solide Basis für anspruchsvolle Anwendungen in der natürlichen Sprachverarbeitung, von der semantischen Analyse bis hin zum maschinellen Lernen.

4.3 Merkmalextraktion

Im Kontext der Merkmalsextraktion aus Textdaten lassen sich Informationen in drei Hauptkategorien, in *Abbildung 6* dargestellt, unterteilen: Metadatenmerkmale, die den Kontext und die Herkunft der Textsammlungen widerspiegeln, textuelle Merkmale, die verschiedene Arten von strukturellen und inhaltsbasierten Informationen repräsentieren, die von den Texten getragen werden, und numerische (statistische) Merkmale, die verschiedene Aspekte der Textdaten quantifizieren. Diese Kategorisierung bildet die Grundlage für eine tiefe Analyse und Interpretation von Textdaten, wobei ihre synergetische Integration das Beantworten komplexer Probleme mit verschiedenen Einsichten ermöglicht. Dieser methodologische Ansatz in der Textdatenanalyse ist besonders wichtig für fortgeschrittene Anwendungen in den Bereichen der Datenwissenschaft, des maschinellen Lernens, da er die Grundlage für die Entwicklung präziser Analysemodelle und die Generierung fundierter Einsichten legt.

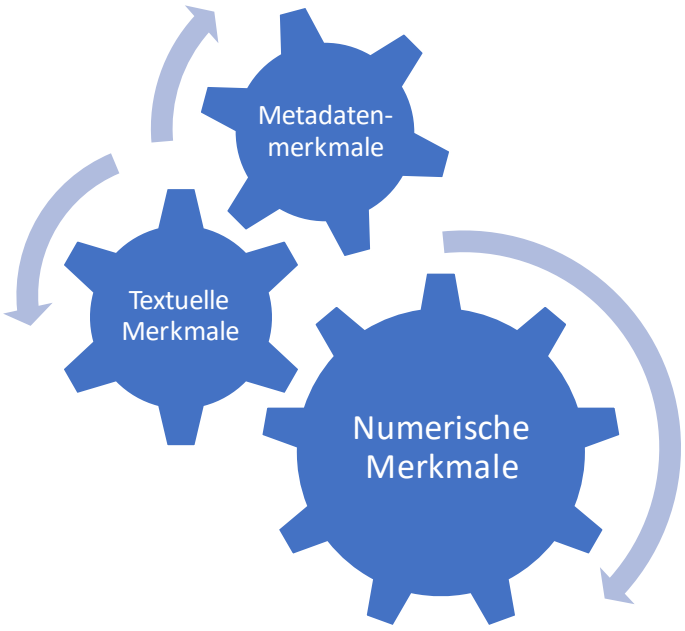

Abbildung 6: die drei Hauptkategorien der Merkmalextraktion

4.3.1 Metadatenmerkmale

Metadaten, einige davon sind in *Abbildung 7* dargestellt, eines Textes sind Informationen, die bestimmte Eigenschaften bezüglich des Textes beschreiben, die jedoch nicht direkt im Textinhalt vorhanden sind, wie der Name des Autors, das Erstellungsdatum, Schlüsselwörter, der Titel und vieles mehr. Diese Daten sind entscheidend, um den Rahmen zu verstehen, in dem der Text erstellt wurde, und tragen dazu bei, die Fähigkeit zur Informationswiedergewinnung und genauen Analyse in akademischen und Forschungskontexten zu verbessern.

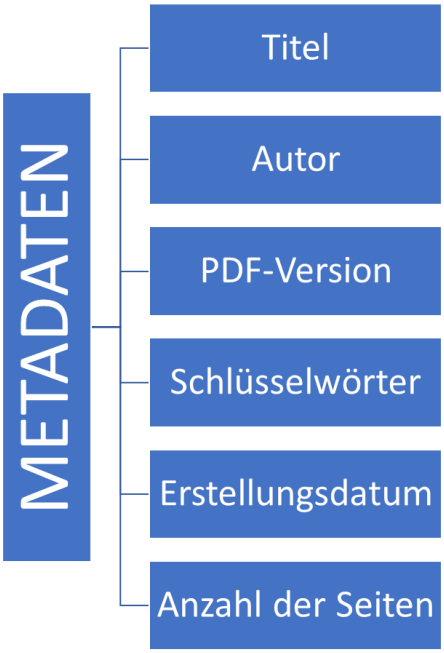

Abbildung 7: mögliche Metadaten, die extrahiert werden können

Die Nutzung der Informationen, die die Metadaten über Texte bereitstellen, stellt ein wichtiges Kriterium dar, das bei der Feststellung der Möglichkeit eines Plagiats des untersuchten Textes hilft. Beispielsweise:

1. Eine enge Ähnlichkeit der Titel zwischen zwei verschiedenen Texten erfordert eine umfassende detaillierte Analyse des Textes, um die Möglichkeit eines Plagiats zu evaluieren.
2. Des Weiteren ermöglicht ein Vergleich der Erstellungs- oder Veröffentlichungsdatum eine Einschätzung darüber, ob der Text auf der Basis eines zuvor publizierten Werkes konzipiert wurde, ohne ein explizites Referenzieren des ursprünglich herangezogenen oder zitierten Textes.

4.3.2 Textuelle Merkmale

Die Analyse textbasierter Daten können die Untersuchung verschiedener Elemente umfassen – mögliche Elemente sind in dargestellt *Abbildung 8* – die für das Verständnis und die Interpretation des Textes wesentlich sind. Zu diesen möglichen Elementen gehören die Wörter, die am häufigsten sowie die, die am seltensten im Text auftreten, was wichtige Einblicke in die zentralen Themen und Inhalte ermöglichen könnte. Darüber hinaus sind N-Gramme – Sequenzen aus n direkt aufeinanderfolgenden Wörtern – sowohl in ihrer häufigsten als auch seltensten Form bedeutend, da sie Aufschluss über charakteristische Sprachstrukturen oder typische Ausdrucksweisen geben können. Ein weiterer möglicher Aspekt sind die spezifischen Entitäten, die aus dem Text gewonnen werden, wie zum Beispiel die Namen von Personen, Unternehmen oder geografischen Orten. Diese extrahierten Entitäten tragen wesentlich zum Verständnis bei, indem sie präzise Informationen bereitstellen, die essenziell für das Erkennen von Beziehungen, das Durchführen thematischer Analysen oder das Identifizieren von Entwicklungstendenzen sind. In ihrer Gesamtheit stellen diese textuellen Merkmale eine umfassende Quelle für die Analyse dar und fördern eine genaue Bewertung sowie Interpretation der Textdaten.

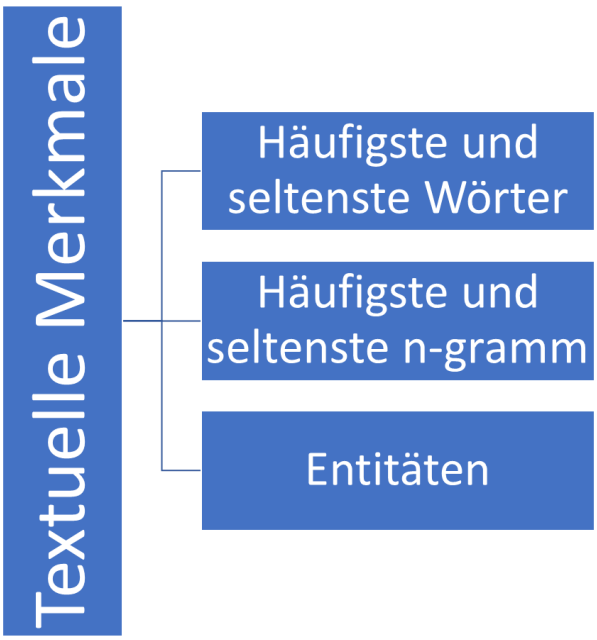

Abbildung 8: mögliche Textuelle Merkmale, die extrahiert werden können

4.3.3 Numerische Merkmale

Numerischer Merkmale, die in *Abbildung 9* dargestellt sind, liefern eine tiefgehende Einschätzung des Textes durch eine Reihe von statistischen Angaben. Diese Angaben beleuchten strukturelle Merkmale des Textes, wie beispielsweise die Anzahl der Wörter, welche die Gesamtmenge der im Text verwendeten Wörter widerspiegelt. Außerdem bietet die Erfassung der Anzahl der Sätze Einblicke in die strukturelle Aufteilung und die Komplexität des Textes. Des Weiteren liefert die Anzahl der Satzzeichen im Text Einblicke in dessen Interpunktion und kann Aufschluss über den Schreibstil sowie die Strukturierung der Inhalte geben und die Zeichenanzahl, einschließlich der Leerzeichen, spiegelt den gesamten Textumfang wider. Darüber hinaus ist die Anzahl der einzigartigen Wörter von Bedeutung, da sie die lexikalische Vielfalt innerhalb des Textes anzeigt. Zusätzlich sind die durchschnittliche Wortlänge sowie die durchschnittliche Satzlänge, gemessen an der Anzahl der Wörter, weitere wesentliche Merkmale. Diese Durchschnittswerte geben Aufschluss über die Komplexität und den Detailgrad des Textes. Insgesamt ermöglichen diese numerischen Merkmale eine tiefgreifende Einsicht in die strukturellen und stilistischen Eigenschaften des Textes, was für verschiedene Anwendungen in der Textanalyse von großer Bedeutung ist.

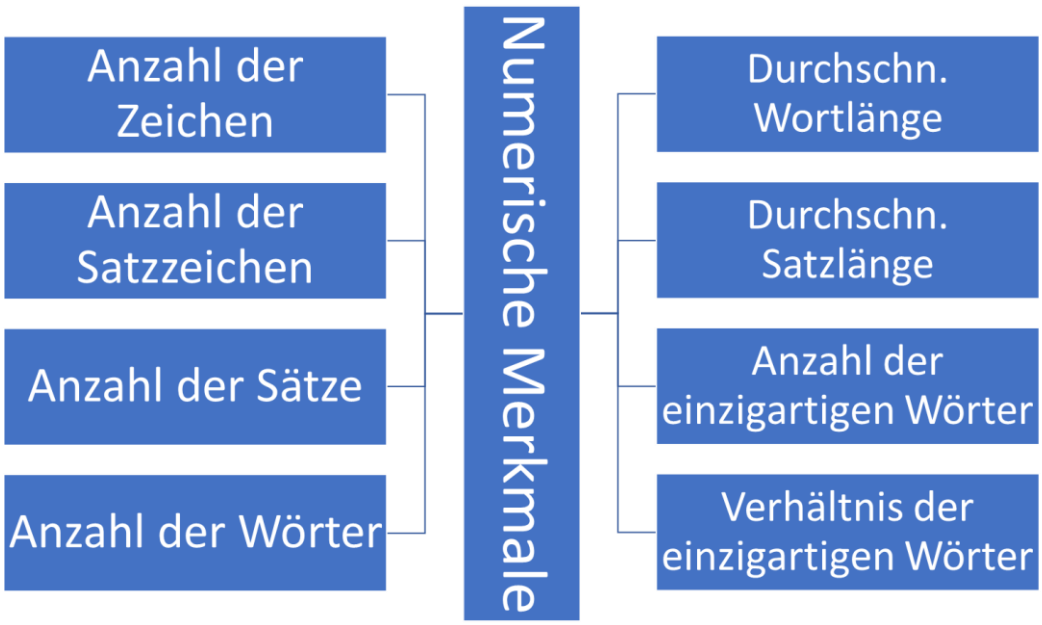

Abbildung 9: numerische Kennzahlen, die extrahiert werden können

4.3.4 Gesamtübersicht der Merkmalsextraktion

Es wird zurückgeblickt auf die ausführliche Auseinandersetzung mit den drei Hauptkategorien der Merkmalsextraktion aus Textdaten, die in *Abbildung 10* visualisiert sind. Anfangs wurde der Fokus auf die Metadatenmerkmale gelegt, welche essenzielle Informationen über den Ursprung und den Kontext der Textkollektionen bereitstellen. Darauffolgend richtete sich das Augenmerk auf die textuellen Merkmale, die ein breites Spektrum an strukturellen und inhaltlichen Informationen umfassen. Abschließend wurde sich den numerischen Merkmalen befasst, durch die die quantifizierbaren Eigenschaften der Textdaten erschlossen werden. Diese Analyse verschafft Einblick in die konkreten strukturellen Merkmale der Texte, einschließlich der Längen von Wörtern und Sätzen, und ermöglicht es, ein quantitatives Verständnis der Textbeschaffenheit zu erlangen. Diese differenzierte Betrachtung der verschiedenen Merkmalskategorien hat es ermöglicht, ein umfassendes Verständnis für die Komplexität und Reichhaltigkeit der Textdaten zu entwickeln.

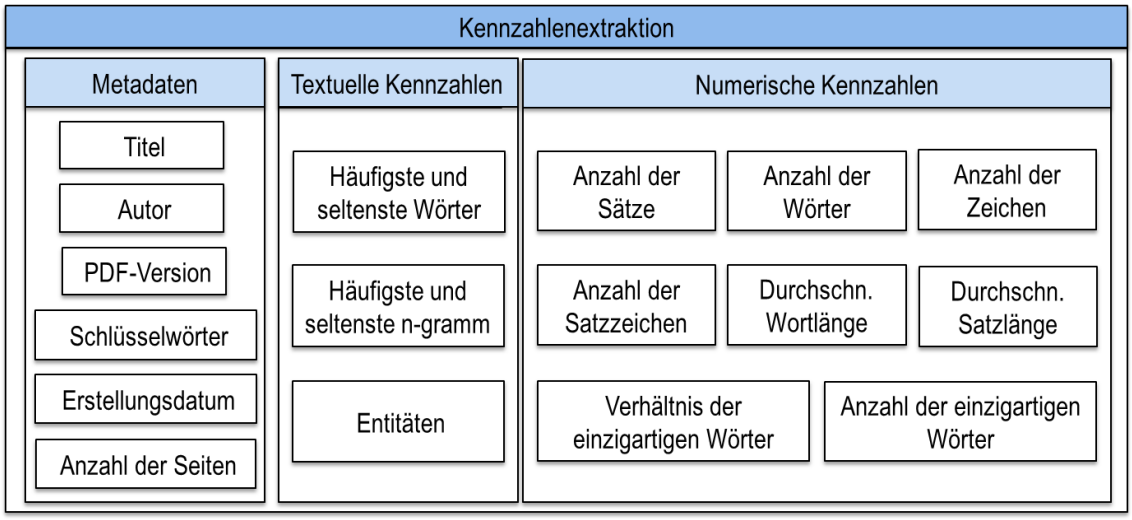

Abbildung 10: Gesamtübersicht der möglichen Merkmale, die extrahiert werden können

4.4 Speicherung der Daten

Die Speicherung der aufbereiteten Daten in einem gut organisierten Datawarehouse ermöglicht sowohl effiziente Abfragen und Analysen als auch eine klar strukturierte Datenaufbewahrung. Für diesen Zweck wurde in dieser Arbeit MySQL ausgewählt, um

einen geordneten Datenspeicher zu schaffen, der die extrahierten Daten zugänglich und einfach analysierbar macht. Zur Veranschaulichung der Datenstrukturierung und der Beziehungen zwischen den verschiedenen Entitäten innerhalb der Datenbank, wird das nachfolgend präsentierte Entity-Relationship-Modell (ERM) herangezogen.

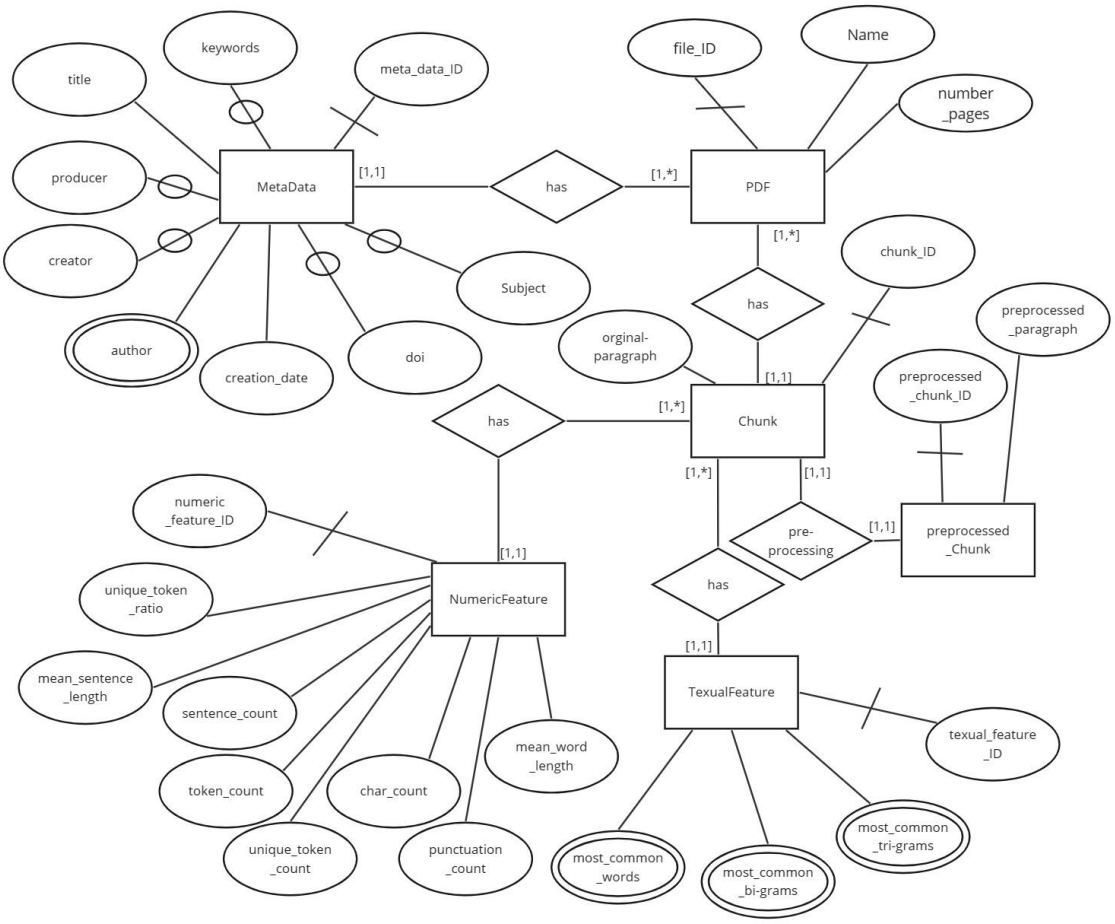

Abbildung 11: ERM der Datenbank für Merkmalsextraktion

In diesem ERM wird die Struktur einer Datenbank abgebildet, die Informationen zu PDF-Dokumenten und ihren zugehörigen Daten speichert. Hier ist eine detaillierte Beschreibung des ERMs:

1. **Entität 'PDF'** repräsentiert ein PDF-Dokument in der Datenbank. Der Primärschlüssel ist 'file_ID', der jedes PDF-Dokument eindeutig identifiziert. Zusätzliche Attribute sind 'Name' für den Namen des Dokuments und 'number_pages' für die Anzahl der Seiten im PDF.
2. **Beziehung 'has' zu 'Metadata':** Jedes PDF-Dokument ist mit Metadaten verbunden, die Informationen wie Titel, Autor, Erstellungsdatum usw. umfassen.

'meta_data_ID' ist der Schlüssel, der die Metadaten eindeutig identifiziert. Es ist zu beachten, dass nicht alle möglichen Metadaten-Attribute in diesem ERM oder in dieser Beschreibung aufgrund ihrer großen Anzahl aufgeführt sind.

3. **Entität 'Chunk':** Ein 'Chunk' ist ein Abschnitt des PDF-Dokuments, identifiziert durch 'chunk_ID'. Es enthält den 'original-paragraph', der den ursprünglichen Textabschnitt des Dokuments darstellt.
4. **Beziehung 'preprocessing' zu 'preprocessed_Chunk':** Der Inhalt jedes Chunks wird einer Vorverarbeitung unterzogen, die in der Entität 'preprocessed_Chunk' festgehalten wird. Diese Entität hat einen Primärschlüssel 'preprocessed_chunk_ID' und ein Attribut 'preprocessed_paragraph', das den vorverarbeiteten Textabschnitt enthält.
5. **Beziehung 'has' zu 'TextualFeature':** Die Entität 'Chunk' ist auch mit textuellen Merkmalen verbunden, die in der Entität 'TextualFeature' gespeichert sind. Diese enthält Attribute wie 'textual_feature_ID', 'most_common_tri-grams', 'most_common_bi-grams' und 'most_common_words', welche die häufigsten Wortkombinationen im Textsegment beschreiben.
6. **Beziehung 'has' zu 'NumericFeature':** Schließlich ist 'Chunk' auch mit numerischen Merkmalen verknüpft, die in 'NumericFeature' gespeichert sind. Diese Entität umfasst neben dem Schlüsselattribut 'numeric_feature_ID' auch Attribute wie 'token_count' (Anzahl der Wörter), 'sentence_count' (Anzahl der Sätze), 'unique_token_ratio' (Verhältnis einzigartiger Wörter) und 'mean_word_length' (durchschnittliche Wortlänge), welche numerische Analysen des Textinhalts ermöglichen.

Das beschriebene ERM ermöglicht die strukturierte Speicherung und Analyse von Textdaten aus PDF-Dokumenten. Es illustriert, wie aus den Inhalten eines PDF-Dokuments verschiedene Datenelemente extrahiert und in relationale Strukturen überführt werden können, um tiefergehende Analysen und Informationsrückgewinnung zu ermöglichen.

5 Implementierung

In diesem Kapitel wird aufgezeigt, wie die Konzeption zur Merkmalsextraktion, die im Kapitel 4 eingeführt wurde, mithilfe der Programmiersprache Python praktisch angewendet werden kann. Es werden alle notwendigen Schritte und die grundlegenden Programmiercodes behandelt, die für die erfolgreiche Durchführung des Extraktionsprozesses erforderlich sind.

5.1 Erforderliche Anforderungen

Für die Implementierung dieser Arbeit werden spezifische Softwareversionen und Bibliotheken benötigt. Als Grundlage dient Python in der Version 3.12.2, eine vielseitige Programmiersprache, die den Kern der Entwicklungsumgebung bildet. Zur effektiven Verarbeitung von PDF-Dokumenten wird die Bibliothek PyPDF2 in der Version 3.0.1 eingesetzt, welche umfassende Funktionen zum Lesen und Bearbeiten von PDF-Dateien bietet. Für Aufgaben im Bereich der Verarbeitung natürlicher Sprache wird auf NLTK in der Version 3.8.1 gesetzt, ein leistungsfähiges Toolkit, das eine umfassende Auswahl an Werkzeugen und Ressourcen für die Analyse bereitstellt. Die Bibliothek Regex, hier in der Version 2023.12.25, wird für die Anwendung von regulären Ausdrücken genutzt, ein Werkzeug zur Textsuche und -manipulation, Pandas in der Version 2.2.1 kommt zur Anwendung, um Daten effizient in Dataframes umzuwandeln. Zusätzlich zur den Bibliotheken ist es wichtig, dass für die Installation und Verwaltung dieser Komponenten der Paketmanager pip wie in *Listing 14* verwendet wird. Pip erleichtert die Installation und das Update der benötigten Pakete und sorgt für eine reibungslose Integration in die Projektumgebung.

```
1. pip install pypdf2 nltk regex pandas
```

Listing 14: pip Befehle zur Installierung der erforderlichen Anforderungen

Zusätzlich zu den bereits genannten Bibliotheken und Werkzeugen sind für die erfolgreiche Implementierung des Projekts weitere Module notwendig: **`'string'`** für den Zugriff auf nützliche Zeichenkettenoperationen einschließlich Satzzeichen (**`'punctuation'`**), **`'json'`** für das Arbeiten mit JSON-Daten. Diese Module erweitern

die Funktionalität der Anwendung, indem sie wichtige Hilfsmittel für Textverarbeitung, Datenformatierung und Datenpersistenz bereitstellen.

5.2 Extraktion

In diesem Abschnitt wird eine eingehende Betrachtung und Erläuterung von Codeausschnitten aus vier spezifischen Bereichen vorgenommen. Diese demonstrieren, auf welche Weise sowohl Texte als auch dazugehörige Metadaten aus PDF-Dokumenten entnommen werden können. Besonderes Augenmerk liegt dabei auf der Extraktion von textuellen und numerischen Kennzahlen. Die bereitgestellten Codebeispiele illustrieren detailliert die erforderlichen Schritte und Techniken, um diese Daten effizient zu extrahieren.

5.2.1 Textextraktion

Um Texte aus einer PDF-Datei zu extrahieren, wurde, wie bereits erwähnt, die Bibliothek PyPDF2 verwendet. Anschließend wird der extrahierte Text in Abschnitte unterteilt. Der folgende Code, *Listing 15*, demonstriert den Einsatz von PyPDF2 für diesen Zweck:

```
1. # read pdf file
2. pdf_file = PdfReader("test.pdf")
3.
4. # extract and splitting DATA
5.
6. # extract content to string
7. all_data = ' '.join(
8.       [
9.           page.extract_text()
10.          for page in pdf_file.pages
11.      ]
12.    )
13.
14.# text splitting into chunks
15.def split_text_into_chunks(text, chunk_size=1000, overlap=100):
16.    chunks = []
17.    start = 0
18.    while start < len(text):
```

```
19.         end = start + chunk_size
20.         chunks.append(text[start:end])
21.         start = end - overlap
22.     return chunks
23.
24. all_data = split_text_into_chunks(all_data, 1000, 100)
```

Listing 15: Lesen, Extrahieren und Aufteilung der Texte aus einer PDF-Datei

Zunächst wird die PDF-Datei mit dem Namen **'test.pdf'** mithilfe der Funktion **'PdfReader'** aus der PyPDF2 Bibliothek gelesen. Der Inhalt jeder Seite der PDF-Datei wird dann zu einem einzigen String zusammengeführt, indem der Text jeder Seite extrahiert und diese Texte in einer Liste gesammelt werden. Anschließend werden alle Elemente der Liste mit einem Leerzeichen als Trennzeichen zu einem großen String zusammengefügt.

Nach der Extraktion des Textes aus der PDF wird der gesamte Textinhalt in kleinere Abschnitte oder **'Chunks'** unterteilt. Dies geschieht in der Funktion **'split_text_into_chunks'**, welche den Text als Eingabe erhält sowie die Größe jedes Abschnitts 1000 Zeichen und das Überlappungsmaß zwischen aufeinanderfolgenden Abschnitten 100 Zeichen. Die Funktion arbeitet, indem sie bei der ersten Zeichenposition beginnt und einen Abschnitt von der definierten Größe ausschneidet. Für jeden folgenden Abschnitt beginnt sie dann am Ende des vorherigen Abschnitts minus der Überlappung, um sicherzustellen, dass keine Information verloren geht. Dieser Prozess wiederholt sich, bis das Ende des Textes erreicht ist, und jeder dieser Abschnitte wird in einer Liste gesammelt.

Zusammenfassend wandelt dieser Code eine PDF-Datei in eine Liste von Textabschnitten um, wobei jeder Abschnitt eine Teilmenge des gesamten Textinhalts der PDF ist. Dies kann besonders nützlich sein, um große Textmengen verwaltbar zu machen oder um spezifische Textverarbeitungsaufgaben auf kleinere Texteinheiten anzuwenden.

5.2.2 Metadatenextraktion

Metadaten stellen wichtige Kennzahlen dar, die mit Hilfe von PyPDF2 einfach extrahiert werden können. Der folgende Codeausschnitt, *Listing 16*, veranschaulicht, wie die Extraktion von Metadaten durchgeführt wurde.

```
# number of pages
len(pdf_file.pages)

# extract meta data
pdf_file.metadata
```

Listing 16: Seitenanzahl- und Metadatenextraktion

Der Code führt zwei Operationen aus, um Informationen aus einer PDF-Datei zu extrahieren, die zuvor mit der PyPDF2-Bibliothek geöffnet wurde. Zunächst wird die Gesamtanzahl der Seiten der PDF-Datei bestimmt, indem die Länge der Seitenliste mit **'len(pdf_file.pages)'** ermittelt wird. Dieser Befehl zählt effektiv, wie viele Seiten sich in der PDF befinden, und liefert somit einen Überblick über den Umfang des Dokuments. Im nächsten Schritt fokussiert sich der Code auf die Extraktion der Metadaten aus der PDF-Datei durch den Zugriff auf **'pdf_file.metadata'**. Metadaten sind zusätzliche Informationen über die PDF, wie beispielsweise den Autor, den Titel und das Erstellungsdatum. Diese Operation extrahiert diese Informationen und stellt sie in einer für den Benutzer zugänglichen Form dar, in dem Schlüssel (wie z.B. "Autor" oder "Titel") den jeweiligen Werten zugeordnet sind.

5.2.3 Extraktion der textuellen Kennzahlen

Für die Extraktion textueller Kennzahlen aus bereits extrahierten und vorverarbeiteten Texten kommt die Ngrams aus der NLTK-Bibliothek zum Einsatz. Der folgende Codeausschnitt, *Listing 17*, zeigt, wie diese Extraktion umgesetzt wurde:

```
from nltk import ngrams
# textual features
def get_ngram_frequencies(sample, n, m, most=True):
    sample = [token for sent in sample for token in sent]
    grams = ngrams(sample, n)
    gram_counts = {}
    for gram in grams:
        gram = " ".join(gram)
        gram_counts[gram] = gram_counts.get(gram, 0) + 1
    # Sort by frequency
    sorted_grams = sorted(gram_counts.items(), key=lambda x: x[1], reverse=most)
    return [gram[0] for gram in sorted_grams[:m]]
```

Listing 17: Extraktion der textuellen Kennzahlen

Der vorgestellte Code, *Listing 17*, beginnt mit dem Import der **'ngrams'** Funktion aus der NLTK-Bibliothek, einem Werkzeug für die Verarbeitung natürlicher Sprachen, das für die Analyse und das Verständnis von Texten verwendet wird. Er führt dann eine Funktion namens **'get_ngram_frequencies'** ein, die dazu dient, die Häufigkeiten bestimmter Wortsequenzen – bekannt als N-Gramme – in einem Text zu ermitteln. Diese Funktion nimmt einen Text in Form einer Liste von Sätzen entgegen, wobei jeder Satz selbst eine Liste von Wörtern (Token) ist. Zusätzlich werden zwei Zahlen übergeben: **'n'**, die die Länge der Wortsequenzen angibt, und **'m'**, die Anzahl der am häufigsten oder seltensten vorkommenden Sequenzen, die die Funktion zurückgeben soll. Ein optionaler Parameter **'most'** legt fest, ob die häufigsten **'True'** oder die seltensten **'False'** N-Gramme zurückgegeben werden. Innerhalb der Funktion wird der Text zunächst zu einer einzigen Liste von Wörtern zusammengefasst. Anschließend wird für jede mögliche Sequenz von **'n'** aufeinanderfolgenden Wörtern im Text ein N-Gramm generiert. Für jedes dieser N-Gramme führt die Funktion eine Zählung durch, indem sie jedes N-Gramm in einen lesbaren String umwandelt und seine Häufigkeit in einem Wörterbuch speichert. Nachdem alle N-Gramme erfasst sind, werden diese nach ihrer Häufigkeit sortiert, sodass die am häufigsten vorkommenden N-Gramme an erster Stelle stehen, es sei denn, der Parameter **'most'** ist auf **'False'** gesetzt, in welchem Fall die seltensten N-Gramme priorisiert werden. Abschließend gibt die Funktion eine Liste der **'m'** häufigsten oder seltensten N-Gramme zurück, basierend auf der vorgegebenen Sortierung.

5.2.4 Extraktion der nummerischen Kennzahlen

Für die Extraktion numerischer Kennzahlen aus den bereits extrahierten und vorverarbeiteten Texten kommen verschiedene Ansätze zur Anwendung, wobei die Datenberechnung mithilfe der Pandas-Bibliothek erfolgt und die Ergebnisse in einem DataFrame dargestellt werden. Diese Methoden werden im folgenden Codeausschnitt, *Listing 18*, dargestellt und anschließend ausführlich erläutert.

```
1. # convert data to dataframe
2. data_df = pd.Series(all_data).to_frame(name="chunk")
3. data_df
4.
5. from string import punctuation
6.
```

```
7.  data_df["sentence_count"] = data_df['chunk'].apply(len)
8.  data_df["token_count"] = data_df['chunk'].apply(lambda x: len([token
    for sent in x for token in sent]))
9.  data_df["unique_token_count"] = data_df['chunk'].apply(lambda x:
    len(set(token for sent in x for token in sent)))
10. data_df["char_count"] = data_df['chunk'].apply(lambda x: len([char for
    sent in x for token in sent for char in token]))
11. data_df["punctuation_count"] = data_df['chunk'].apply(lambda x:
    len([char for sent in x for token in sent for char in token if char in
    punctuation]))
12. data_df["mean_word_length"] = data_df['char_count'] /
    data_df['token_count']
13. data_df["mean_sentence_length"] = data_df['token_count'] /
    data_df['sentence_count']
14. data_df["unique_token_ratio"] = data_df['unique_token_count'] /
    data_df['token_count']
```

Listing 18: Extraktion der textuellen Kennzahlen

Dieser Code, *Listing 18*, demonstriert, wie man numerische Kennzahlen aus Textdaten berechnet und die Ergebnisse in einem Pandas DataFrame organisiert. Zuerst wird eine Reihe von Textabschnitten, **`'all_data'`**, in einen DataFrame **`'data_df'`** umgewandelt, wobei jeder Textabschnitt als eine Zeile repräsentiert wird. Jeder dieser Abschnitte wird in der Spalte **`'chunk'`** gespeichert. Nach der Initialisierung des DataFrames werden verschiedene textuelle Kennzahlen für jeden Textabschnitt berechnet und als neue Spalten zum DataFrame hinzugefügt:

1. **Zeile 7 `'sentence_count'`:** Hier wird einfach die Länge jedes Abschnitts gemessen. die Implementierung zählt tatsächlich, wie viele Sätze der Abschnitt enthält.
2. **Zeile 8 `'token_count'`:** Die Anzahl der Wörter (Tokens) in jedem Abschnitt. Dazu werden die Sätze in Tokens zerlegt und deren Anzahl gezählt.
3. **Zeile 9 `'unique_token_count'`:** Ähnlich wie bei der Token-Anzahl, aber hier wird die Menge der einzigartigen Tokens in jedem Abschnitt ermittelt, was Duplikate eliminiert.
4. **Zeile 10 `'char_count'`:** Die Gesamtzahl der Zeichen in jedem Abschnitt, einschließlich Leerzeichen und Satzzeichen.
5. **Zeile 11 `'punctuation_count'`:** Dies zählt, wie viele Satzzeichen in jedem Abschnitt vorhanden sind, indem überprüft wird, ob jedes Zeichen in der **`'punctuation'`**-Liste aus dem **`'string'`**-Modul enthalten ist.
6. **Zeile 12 `'mean_word_length'`:** Die durchschnittliche Länge der Wörter in jedem Abschnitt, berechnet durch Division der Gesamtanzahl der Zeichen durch die Anzahl der Tokens.

7. **Zeile 13 `'mean_sentence_length'`:** Die durchschnittliche Anzahl von Wörtern pro Satz in jedem Abschnitt, berechnet durch Division der Token-Anzahl durch die Anzahl der Sätze.
8. **Zeile 14 `'unique_token_ratio'`:** Das Verhältnis der Anzahl einzigartiger Tokens zur Gesamtzahl der Tokens, was ein Maß für die Diversität der Wortverwendung in jedem Abschnitt ist.

Jede dieser Kennzahlen bietet einen Einblick in verschiedene Aspekte der Textdaten, von der grundlegenden Struktur bis hin zur Komplexität und Vielfalt des Wortschatzes.

5.3 Transformation

Um Textinhalte effizient verarbeiten zu können, war es notwendig, die Texte in einen bearbeitbaren und für die Analyse geeigneten Zustand zu transformieren. In diesem Prozess spielte die Natural Language Toolkit (NLTK)-Bibliothek eine zentrale Rolle. Durch die Anwendung von NLTK-Funktionen konnten die Texte in eine strukturierte Form gebracht werden, die Tokenisierung, das Entfernen von Stoppwörtern, das Lemmatisieren umfasste. Diese Schritte waren entscheidend, um die Texte nicht nur für maschinelle Lernmodelle zugänglich zu machen, sondern auch um tiefergehende linguistische Analysen durchzuführen. Durch die Verwendung von NLTK wurde somit eine solide Grundlage für die weiterführende Textverarbeitung und -analyse geschaffen.

5.3.1 Textvorverarbeitung

Die Verarbeitung der Textinhalte in dieser Arbeit erfolgte in klar definierten Schritten, die hier schrittweise dargestellt sind:

1. **Kleinschreibung:** Der erste Schritt besteht darin, den Text in Kleinbuchstaben umzuwandeln. Der folgende Code, *Listing 19*, demonstriert, wie dies umgesetzt wurde.

```
1. # lower casing
2. all_data = list(map(str.lower, all_data))
```

Listing 19: Text in Kleinschreibung umwandeln

Dieser Code, *Listing 19*, wandelt alle Texte in der Liste **'all_data'** mithilfe der Funktion **'map()'** in Kleinbuchstaben um. Die **'str.lower'** Methode wird dabei auf jedes Element der Liste angewendet, und das Ergebnis wird zurück in eine Liste konvertiert und in **'all_data'** gespeichert. Dieser Schritt standardisiert die Textdaten für die nachfolgende Analyse.

2. **Tokenisierung:** Als nächster Schritt und mit Unterstützung von NLTK wird die Tokenisierung, *Listing 20*, durchgeführt, die zunächst auf Satzebene und anschließend auf Wortebene erfolgt.

```
1. # Tokenization
2.
3. # sentence tokenizer
4. all_data = list(map(sent_tokenize, all_data))
5.
6. # word tokenizer
7. for index_chunk in range(len(all_data)):
8.     all_data[index_chunk]          =          list(map(word_tokenize,
   all_data[index_chunk]))
```

Listing 20: Text-Tokenisierung

In diesem Codeabschnitt wird die Tokenisierung eines Textes in zwei Hauptphasen durchgeführt, wobei die NLTK-Bibliothek zum Einsatz kommt. Zuerst erfolgt die Satz-Tokenisierung, bei der der gesamte Textinhalt, gespeichert in der Variablen **'all_data'**, mithilfe der **'sent_tokenize'**-Funktion in einzelne Sätze unterteilt wird. Dieser Vorgang wandelt **'all_data'** in eine Liste um, in der jedes Element eine Liste von Sätzen eines bestimmten Textabschnitts darstellt.
Anschließend wird die Wort-Tokenisierung durchgeführt. Hierbei durchläuft der Code jede der zuvor erstellten Listen von Sätzen und zerlegt diese weiter in ihre einzelnen Wörter und Satzzeichen, indem er die **'word_tokenize'**-Funktion auf jeden Satz anwendet. Das Ergebnis ist eine noch detailliertere Struktur, in der **'all_data'** nun als eine Liste von Listen von Listen repräsentiert wird, wobei jede innerste Liste die individuellen Wörter eines Satzes enthält.

3. **Entfernung der Stoppwörter:** Nun erfolgt die Entfernung der Stoppwörter aus dem Text, wobei ebenfalls die NLTK-Bibliothek zum Einsatz kommt, wie folgt:

```
# get english stop words
stop_words = set(stopwords.words('english'))

all_data = [
      [
        [
          token for token in sentence if not token in stop_words
        ] for sentence in chunk
      ] for chunk in all_data
    ]
```

Listing 21: Entfernung der Stoppwörter

In diesem Codeabschnitt wird zunächst mittels **'nltk.download('stopwords')'** eine Liste englischer Stoppwörter aus der NLTK-Bibliothek heruntergeladen und anschließend als Set mit **'stopwords.words('english')'** für die Textbereinigung verwendet. Dieses Set dient dazu, den Text von häufigen, aber oft analytisch irrelevanten Wörtern zu befreien. Der Prozess zielt darauf ab, den Text vorzubereiten, indem irrelevante Elemente entfernt werden.

4. **Entfernung der Sonderzeichen:** Anschließend werden die Sonderzeichen aus dem Text entfernt. Der folgende Code, *Listing 22*, demonstriert, wie dieser Schritt umgesetzt wurde.

```
# Remove special character
def remove_special_characters(text):
  return re.sub('[^A-Za-z0-9 ,:;.]+', '', text)

all_data = [
      [
        remove_special_characters('  '.join(sentence)).split('  ') for sentence in chunk
      ] for chunk in all_data
    ]
```

Listing 22: Entfernung der Sonderzeichen

In diesem Code wird eine Funktion **'remove_special_characters'** definiert, die mithilfe eines regulären Ausdrucks alle Sonderzeichen aus dem Text entfernt, außer Buchstaben, Zahlen und einigen Satzzeichen (Komma, Doppelpunkt, Semikolon, Punkt). Diese Funktion wird dann auf jeden Satz in **'all_data'** angewendet, nachdem die Sätze zuerst zu einem String zusammengeführt und anschließend die Sonderzeichen entfernt wurden. Danach wird der bereinigte String wieder in einzelne Wörter aufgeteilt.

5. **Lemmatisierung:** Im Schritt der Lemmatisierung werden alle Wörter auf ihr Lemma zurückgeführt. Der folgende Code, *Listing 23*, veranschaulicht, wie dies umgesetzt wurde.

```
# Lemmatization
# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

all_data = [
      [
        [
          lemmatizer.lemmatize(token) for token in sentence
        ] for sentence in chunk
      ] for chunk in all_data
    ]
```

Listing 23: Lemmatisierung

In diesem Codeabschnitt wird zunächst die Anzahl einzigartiger Wörter im Datensatz **`'all_data'`** ermittelt, bevor der **`'WordNetLemmatizer'`** aus der NLTK-Bibliothek initialisiert wird, um die Lemmatisierung durchzuführen. Diese Technik reduziert jedes Wort auf sein Lemma, wodurch die Wortformenvielfalt verringert wird. Die Lemmatisierung wird auf jedes Wort in jedem Satz und jedem Textabschnitt angewendet. Der Vorgang standardisiert die Textdaten, indem er Wortvarianten auf ihre Grundformen zurückführt.

5.3.2 Textfilterung

Nach den Schritten der Textvorverarbeitung, wie der Umwandlung in Kleinschreibung, Tokenisierung, Entfernung von Stoppwörtern, Entfernung von Sonderzeichen und Lemmatisierung, folgt die Textfilterung. Diese umfasst die Entfernung numerischer Tokens, das Entfernen von Leerzeichen, die Entfernung von Tokens mit weniger als zwei Zeichen und die Entfernung von Sätzen, die ein oder weniger Tokens enthalten, wie folgt durchgeführt:

1. **Entfernung der nummerische Tokens:** Der erste Schritt umfasst die Entfernung numerischer Tokens. Zahlen und numerische Daten, die für die bevorstehende Textanalyse möglicherweise keinen Mehrwert bieten, werden aus dem Text entfernt. Der folgende Codeausschnitt, *Listing 24*, veranschaulicht, wie dieser Schritt umgesetzt wurde.

```
# remove numeric tokens

# Define a function to remove numeric tokens
def remove_numeric_tokens(token):
    return   re.sub(r'\b(\d+(\.\d*)?|\.\d+)([eE][-+]?[0-9]+)?\b',   '', token)

all_data = [
      [
        remove_numeric_tokens('   '.join(sentence)).split('   ')   for sentence in chunk
      ] for chunk in all_data
    ]
```

Listing 24: Entfernung der numerischen Tokens

In diesem Code wird eine Funktion **'remove_numeric_tokens'** definiert, um numerische Tokens aus Texten zu entfernen. Dies geschieht mit einem regulären Ausdruck, der Zahlen im Text sucht und sie entfernt. Die Funktion wird auf jeden Satz angewendet, indem die Sätze zuvor zu einem String verbunden und nach der Entfernung der Zahlen wieder in Wörter aufgeteilt werden.

2. **Entfernung der Leerzeichen:** Es wurde festgestellt, dass einige leere Tokens vorhanden sind, da diese Tokens spezielle Codes waren, die gelöscht wurden. Daher müssen wir die leeren Tokens entfernen, wie in diesem Code, *Listing 25*, veranschaulicht wurde.

```
# remove empty token
all_data = [
      [
        [
          token for token in sentence if token != ''
        ] for sentence in chunk
      ] for chunk in all_data
    ]
```

Listing 25: Entfernung der Leerzeichen

'all_data' ist eine verschachtelte Struktur aus Textabschnitten, die Sätze mit einzelnen Wörtern (Tokens) enthält. Leere Tokens, die durch die Entfernung spezieller Codes oder Zeichen entstehen, werden durch Durchlaufen aller Ebenen des Texts und Ausschließen der leeren Tokens identifiziert und entfernt.

3. **Entfernungen der Token, die weniger als 4 Zeichen enthalten:** Nun werden Tokens, die weniger als vier Zeichen enthalten, entfernt. Der folgende Code, *Listing 26*, veranschaulicht, wie dieser Schritt durchgeführt wurde.

```
# remove token with length less than 4 char
all_data = [
       [
         [
           token for token in sentence if len(token) > 3 or token in ",:;."
         ] for sentence in chunk
       ] for chunk in all_data
     ]
```

Listing 26: Entfernungen der Token, die weniger als 4 Zeichen enthalten

In diesem Codeabschnitt werden Tokens, die weniger als vier Zeichen enthalten, aus dem Text entfernt, es sei denn, es handelt sich um spezifische Satzzeichen (Komma, Doppelpunkt, Semikolon, Punkt). Es wird durch alle Textebenen navigiert – durch Chunks, Sätze und Tokens –, und für jedes Token wird überprüft, ob es die Längenbedingung erfüllt oder eines der ausgewählten Satzzeichen ist.

4. **Entfernung der Sätze, die ein Token oder weniger enthalten:** Zum Abschluss werden Sätze, die ein Token oder weniger enthalten, entfernt. Der folgende Code veranschaulicht, wie dieser Schritt umgesetzt wurde.

```
# remove sentences with length less than or equal one token
all_data = [
       [
         sentence for sentence in chunk if len(sentence) > 1
       ] for chunk in all_data
     ]
```

Listing 27: Entfernung der Sätze, die ein Token oder weniger enthalten

In diesem Codeabschnitt werden Sätze, die ein oder weniger Tokens enthalten, aus dem Datensatz **`'all_data'`** entfernt. Es wird durch alle Textabschnitte (Chunks) navigiert, und es werden nur die Sätze beibehalten, die mehr als ein Token enthalten.

5.4 Laden

Die Phase der Datenspeicherung ist von kritischer Bedeutung, vergleichbar mit der Extraktion von Kennzahlen oder der Textverarbeitung. Eine systematische und strukturierte Datenspeicherung ist unerlässlich, um einen effizienten und schnellen Zugriff auf die Daten zu gewährleisten. Für die Zwecke dieser Arbeit wurde das Star Schema ausgewählt, dessen strukturierte und fokussierte Speicherform die Basis für

effiziente Datenabfragen und -analysen legt. Im Folgenden werden die angewandte Schemaarchitektur dargestellt:

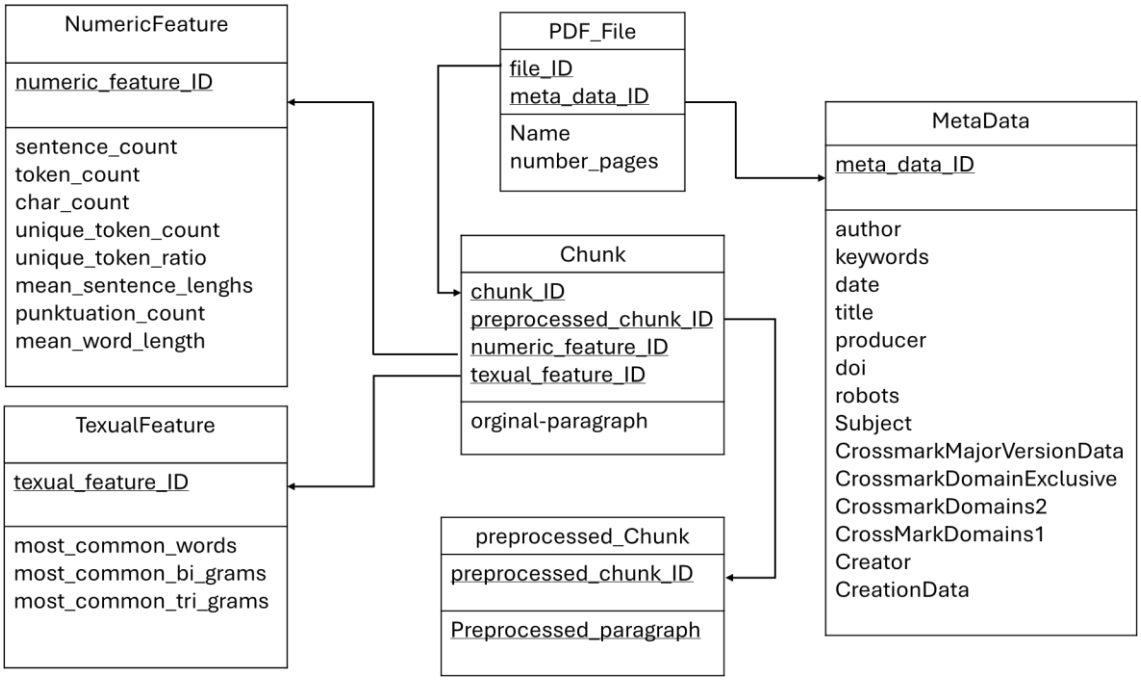

Abbildung 12: Star-Schemaarchitektur

Hier eine Erklärung für jedes Element des Schemas:

1. **MetaData:** Diese Tabelle enthält Metadaten, die allgemeine Informationen über PDF-Dokumente bereitstellen. Attribute wie Autor, Schlüsselwörter, Datum, Titel und Produzent sind in dieser Tabelle enthalten. Zusätzliche Metadatenfelder wie DOI (Digital Object Identifier) und verschiedene andere Identifikatoren und Klassifizierungsinformationen sind ebenfalls Teil dieser Tabelle.
2. **PDF_File:** Diese Tabelle beinhaltet grundlegende Informationen zu den einzelnen PDF-Dateien, einschließlich einer eindeutigen ID, einer Verbindung zur Metadaten-ID, dem Namen der Datei und der Anzahl der Seiten.
3. **Chunk:** Hier werden einzelne Abschnitte (Chunks) der PDF-Dateien gespeichert. Diese Tabelle verknüpft jeden Chunk mit einer spezifischen PDF-Datei und referenziert zudem vorverarbeitete Inhalte sowie numerische und textuelle Merkmale.
4. **preprocessed_Chunk:** Diese Tabelle enthält die vorverarbeiteten Chunks. Hierzu zählen transformierte oder gereinigte Textpassagen, die für weitere Analysen vorbereitet sind.
5. **NumericFeature:** In dieser Tabelle werden numerische Merkmale jedes Chunks gespeichert, wie beispielsweise die Anzahl der Sätze, durchschnittliche

Satzlänge, Anzahl der Satzzeichen, Anzahl einzigartiger Token, Gesamtzeichenanzahl, durchschnittliche Wortlänge und Gesamtanzahl der Token.

6. **TexualFeature:** Diese Tabelle speichert textuelle Merkmale der Chunks, wie die am häufigsten vorkommenden Wörter, Bi-Gramme und Tri-Gramme.

Jede dieser Tabellen ist über Schlüssel miteinander verbunden, was eine relationale Struktur ermöglicht. So verweist zum Beispiel die '**PDF_File**' Tabelle auf die **'MetaData'** Tabelle über das Attribut **'meta_data_ID'**, was bedeutet, dass jede PDF-Datei zugehörige Metadaten hat. Ebenso verbindet die **'Chunk'** Tabelle Informationen über die PDF-Dateien, ihre vorverarbeiteten Inhalte und zugehörigen Merkmale.

In einer solchen Struktur können komplexe Abfragen durchgeführt werden, die die Verknüpfung mehrerer Tabellen erfordern, um tiefgehende Analysen über die Inhalte und Metadaten von PDF-Dokumenten zu ermöglichen.

Um die strukturierte Speicherung der Daten innerhalb der Datenbank zu gewährleisten, wird im nächsten Schritt des Projektes das zuvor konzipierte Sternschema mithilfe der **'CREATE TABLE'** Anweisungen in SQL umgesetzt. Diese Anweisungen definieren die Tabellen und Beziehungen, die erforderlich sind, um die im Schema festgelegten Datenstrukturen in der Datenbank zu realisieren. Dadurch wird die Basis für ein effektives Datenmanagement geschaffen. Hier sind die Anweisungen zur Erstellung der Datenbanktabellen:

1. PDF_File-Tabelle

```
1. CREATE TABLE PDF_File (
2.     file_ID INT PRIMARY KEY NOT NULL,
3.     meta_data_ID INT NOT NULL,
4.     Name VARCHAR(255),
5.     number_pages INT NOT NULL,
6.     FOREIGN KEY (meta_data_ID) REFERENCES MetaData(meta_data_ID)
7. );
```

Listing 28: PDF-Tabelle erstellen

Diese Anweisung erstellt die Tabelle **'PDF_File'** mit Informationen zu PDF-Dateien, einschließlich einer eindeutigen ID, dem Namen und der Seitenanzahl. Über einen Fremdschlüssel **'meta_data_ID'** wird eine Beziehung zur Tabelle **'MetaData'** hergestellt.

2. MetaData-Tabelle

```
1. CREATE TABLE MetaData (
2.     meta_data_ID INT PRIMARY KEY NOT NULL,
3.     keywords VARCHAR(255),
4.     date VARCHAR(20),
5.     title VARCHAR(255) NOT NULL,
6.     producer VARCHAR(255),
7.     doi VARCHAR(255),
8.     robots VARCHAR(255),
9.     Subject VARCHAR(255),
10.    CrossmarkMajorVersionData DATE,
11.    CrossmarkDomainExclusive VARCHAR(255),
12.    CrossMarkDomains1 VARCHAR(255),
13.    CrossMarkDomains2 VARCHAR(255),
14.    Creator VARCHAR(255),
15.    CreationData VARCHAR(255) NOT NULL
16. );
```

Listing 29: Metadaten-Tabelle erstellen

Diese Anweisung erstellt die Tabelle **'MetaData'**, die Metadaten mit Attributen wie Schlüsselwörter, Titel und Erstellungsdatum speichert. **'meta_data_ID'** dient als eindeutiger Primärschlüssel.
Da **'author'** als Gruppierungen von Attributen behandelt wird, ist es sinnvoll, diese in separate Tabellen auszulagern.

```
1. CREATE TABLE author (
2.     meta_data_ID INT NOT NULL,
3.     author VARCHAR(255) NOT NULL,
4.     PRIMARY KEY (meta_data_ID, author),
5.     FOREIGN KEY (meta_data_ID) REFERENCES MetaData(meta_data_ID)
6. );
```

Listing 30: Autor-Tabelle erstellen

Diese SQL-Anweisung erstellt eine Tabelle **'author'** für die Speicherung von Autorennamen, wobei jeder Eintrag eindeutig durch eine Kombination aus **'meta_data_ID'** und **'author'** gekennzeichnet ist. Die **'meta_data_ID'** fungiert als Verbindung zur Tabelle **'MetaData'**, ermöglicht also die Zuordnung mehrerer Autoren zu einem Metadatensatz.

3. Chunk-Tabelle

```
1. CREATE TABLE Chunk (
2.     chunk_ID INT PRIMARY KEY NOT NULL,
3.     preprocessed_chunk_ID INT NOT NULL,
4.     numeric_feature_ID INT NOT NULL,
5.     texual_feature_ID INT NOT NULL,
6.     orginal_paragraph VARCHAR(255) NOT NULL,
7.     FOREIGN KEY (preprocessed_chunk_ID)
```

```
8.                 REFERENCES PreprocessedChunk(preprocessed_chunk_ID),
9.     FOREIGN KEY (numeric_feature_ID)
10.                    REFERENCES NumericFeature(numeric_feature_ID),
11.    FOREIGN KEY (texual_feature_ID)
12.                       REFERENCES TexualFeature(texual_feature_ID)
13. );
```

Listing 31: Chunk-Tabelle erstellen

Diese Anweisung erstellt die Tabelle **'Chunk'** für die Speicherung von Textblöcken aus PDF-Dateien. Jeder **'Chunk'** wird durch eine eindeutige ID identifiziert und enthält den Originaltext sowie Verweise auf seine vorverarbeitete Form, numerische und textuelle Merkmale über Fremdschlüssel. Die Beziehungen zu den Tabellen **'preprocessed_Chunk'**, **'NumericFeature'** und **'TexualFeature'** werden durch diese Fremdschlüssel hergestellt.

4. Preprocessed_Chunk-Tabelle

```
1. CREATE TABLE preprocessed_Chunk (
2.     preprocessed_chunk_ID INT PRIMARY KEY NOT NULL,
3.     Preprocessed_paragraph JSON NOT NULL
4. );
```

Listing 32: Preprocessed-Chunk-Tabelle erstellen

Diese Anweisung erstellt die Tabelle **'preprocessed_Chunk'**, in der vorverarbeitete Textabschnitte als JSON-Datenstruktur gespeichert werden. Jeder **'preprocessed_Chunk'** wird durch eine eindeutige ID identifiziert.

5. NumericFeature-Tabelle

```
1. CREATE TABLE NumericFeature (
2.     numeric_feature_ID INT PRIMARY KEY NOT NULL,
3.     sentence_count INT NOT NULL,
4.     token_count INT NOT NULL,
5.     char_count INT NOT NULL,
6.     unique_token_count INT NOT NULL,
7.     unique_token_ratio FLOAT NOT NULL,
8.     mean_sentence_lenghs FLOAT NOT NULL,
9.     punktuation_count INT NOT NULL,
10.    mean_word_length FLOAT NOT NULL
11. );
```

Listing 33: Tabelle für nummerische Merkmale erstellen

Die Anweisung erstellt die Tabelle **'NumericFeature'** mit einem eindeutigen ID und verschiedenen numerischen Merkmalen wie Anzahl der Sätze, Tokens, Zeichen, eindeutigen Tokens, dem Verhältnis eindeutiger Tokens,

durchschnittlicher Satzlänge, Anzahl der Satzzeichen und durchschnittlicher Wortlänge.

6. TexualFeature-Tabelle

```
1. CREATE TABLE TexualFeature (
2.     texual_feature_ID INT PRIMARY KEY NOT NULL,
3.     most_common_words VARCHAR(255) NOT NULL
4. );
```

Listing 34: Tabelle für textuelle Merkmale erstellen

Diese Anweisung erstellt die Tabelle **'TexualFeature'** mit einem eindeutigen ID und Feld für die häufigsten Wörter zu speichern.
Da **'most_common_bi_grams'** und **'most_common_tri_grams'** als Gruppierungen von Attributen behandelt werden, ist es sinnvoll, sie in separate Tabellen auszulagern.

```
1. CREATE TABLE most_common_bi_grams (
2.     texual_feature_ID INT NOT NULL,
3.     most_common_bi_grams VARCHAR(255) NOT NULL,
4.     PRIMARY KEY (texual_feature_ID, most_common_bi_grams),
5.     FOREIGN KEY (texual_feature_ID)
6.                         REFERENCES TexualFeature(texual_feature_ID)
7. );
```

Listing 35: most-common-bi-grams-Tabelle erstellen

Diese Anweisung erstellt die Tabelle **'most_common_bi_grams'**, die Bigramme speichert, die am häufigsten in Textdaten auftreten. Jeder Eintrag verknüpft ein Bigramm **'most_common_bi_grams'** mit einer ID **'texual_feature_ID'** aus der **'TexualFeature'** Tabelle. Die Kombination aus **'texual_feature_ID'** und **'most_common_bi_grams'** dient als eindeutiger Schlüssel, was sicherstellt, dass die Bigramme eindeutig den Textmerkmalen zugeordnet werden.
Das gleiche Prinzip wird auch für **'most_common_tri_grams'** angewendet, die in einer separaten Tabelle gespeichert werden, um Trigramme zu erfassen.

```
1. CREATE TABLE most_common_tri_grams (
2.     texual_feature_ID INT NOT NULL,
3.     most_common_tri_grams VARCHAR(255) NOT NULL,
4.     PRIMARY KEY (texual_feature_ID, most_common_tri_grams),
5.     FOREIGN KEY (texual_feature_ID)
6.                         REFERENCES TexualFeature(texual_feature_ID)
7. );
```

Listing 36: most-common-tri-grams-Tabelle erstellen

6 Schlussbetrachtung

Das abschließende Kapitel fasst die Bachelorarbeit zusammen und bietet einen Ausblick auf mögliche Erweiterungen und zukünftige Forschungsrichtungen. Es schließt die Untersuchung ab, indem es aufzeigt, wie die bestehenden Methoden zur Kennzahlenextraktion aus PDF-Dokumenten vertieft und durch neue Technologien und Analyseverfahren erweitert werden können.

6.1 Zusammenfassung

Der Kern dieser Bachelorarbeit konzentriert sich auf die Entwicklung eines ETL (Extract, Transform, Load)-Prozesses, der auf die Extraktion von textuellen und numerischen Kennzahlen sowie Metadaten aus PDF-Dokumenten ausgerichtet ist. Die Extraktion erfolgte unter Einsatz der PyPDF2-Bibliothek in der Programmiersprache Python. Diese Metadaten, zusammen mit den textuellen und numerischen Kennzahlen, dienen als grundlegende Indikatoren, die für weiterführende Analysen, insbesondere zur Plagiatserkennung in studentischen Arbeiten, herangezogen werden können.

Die vorliegende Arbeit legt dar, wie durch den gezielten Einsatz von Programmbibliotheken wie PyPDF2 und NLTK nicht nur die Extraktion von Inhalten aus PDF-Dokumenten optimiert, sondern auch die Analyse dieser Daten effizient gestaltet werden kann. Die Untersuchung betont die Bedeutung einer sorgfältigen Auswahl und Anwendung dieser Werkzeuge, um die Genauigkeit und Zuverlässigkeit der extrahierten Informationen zu gewährleisten.

Ein wesentlicher Aspekt der Arbeit ist die Erkenntnis, dass die Extraktion und Analyse von Metadaten sowie textuellen und numerischen Kennzahlen als Vorbereitungsschritte dienen können, um Mechanismen zur Plagiatserkennung zu entwickeln. Diese Vorgehensweise ermöglicht einen neuen Blick auf die Verwendung von extrahierten Daten zur Förderung der Authentizität und zur Vermeidung von Betrug in der akademischen Welt.

Abschließend betrachtet, repräsentiert diese Arbeit einen signifikanten Schritt in der Erforschung und Anwendung von Datenextraktionsprozessen aus PDF-Dokumenten.

Sie dient als Fundament für die Entwicklung fortschrittlicher Analysemethoden, die in der Lage sind, die wachsenden Herausforderungen im Bereich der Textanalyse und Plagiatserkennung zu bewältigen.

6.2 Ausblick

Die vorliegende Arbeit legt ein solides Fundament für die Extraktion und Analyse von Daten aus PDF-Dokumenten und bietet vielfältige Ansatzpunkte für zukünftige Erweiterungen. Wesentlich ist die Integration zusätzlicher Merkmale wie die Bestimmung der Dokumentsprache, die für die Analyse multilingualer Texte (beispielsweise studentische Arbeiten, die in deutscher Sprache verfasst sind, jedoch Codeausschnitte auf Englisch enthalten) unerlässlich ist. Der Lesbarkeitsindex und die Sentiment-Analyse könnten neue Dimensionen eröffnen, indem sie die Zugänglichkeit und den Ton eines Textes bewerten und somit tiefere Einblicke in die Kommunikationsstrategien der Autoren ermöglichen. Eine Layout-Analyse könnte zur besseren Verständlichkeit der visuellen Strukturierung von Informationen führen.

Die Verarbeitung komplexer PDF-Layouts stellt eine besondere Herausforderung dar. Die Integration von OCR-Technologien und maschinellem Lernen könnte hier die Texterkennung und das Verständnis von Layouts verbessern, wobei eine adaptive Erkennungslogik die Effizienz und Genauigkeit der Merkmalenextraktion erheblich steigern könnte. Zudem bietet die Erweiterung der Datenquellen über PDFs hinaus, um Formate wie DOCX, ODT und TXT einzuschließen, das Potential, die Anwendbarkeit der Forschung zu verbreitern und zu einer umfassenderen Analyse beizutragen. Diese zukunftsorientierten Entwicklungen würden die Grundlage für robustere und flexiblere Analysemethoden bilden und die Grenzen der aktuellen Extraktions- und Analyseprozesse erweitern.

Meine Meinung zur Zukunft der Extraktion aus PDF-Dokumenten und Textanalyse im Rahmen der heutigen Informations- und digitalen Explosion ist, dass wir an einem Wendepunkt stehen. Die rapide Zunahme digitaler Inhalte und die Notwendigkeit, diese effizient zu verarbeiten, zu analysieren und zu verstehen, treiben die Innovation in der Text- und Datenanalyse voran. Künstliche Intelligenz und maschinelles Lernen werden zunehmend in den Vordergrund rücken, um automatisierte Lösungen für die Extraktion und Analyse von Daten zu bieten. Diese Entwicklungen ermöglichen nicht nur eine

tiefere Einsicht in große Datenmengen, sondern eröffnen auch neue Wege zur Bekämpfung von Plagiaten und zur Sicherung der wissenschaftlichen Redlichkeit. Die Zukunft liegt in der Entwicklung intelligenter, adaptiver Systeme, die in der Lage sind, mit der Komplexität und dem Volumen der Daten Schritt zu halten, und dabei helfen, die Qualität und Integrität akademischer und anderweitiger Publikationen zu gewährleisten.

Literaturverzeichnis

[1] J. Sreemathy, K. N. Durai, E. L. Priya, R. Deebika, K. Suganthi und P. Aisshwarya, „Data Integration and ETL: A Theoretical Perspective," in *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2021.

[2] E. Farrier, „Avast," 12 10 2023. [Online]. Available: https://www.avast.com/c-what-is-metadata. [Zugriff am 02 01 2024].

[3] Talend, [Online]. Available: https://www.talend.com/de/resources/was-sind-metadaten/. [Zugriff am 02 01 2024].

[4] W. Hamza, 22 11 2016. [Online]. Available: http://techno3annes.blogspot.com/2016/11/blog-post.html. [Zugriff am 02 01 2024].

[5] Adobe, [Online]. Available: https://www.adobe.com/de/acrobat/about-adobe-pdf.html. [Zugriff am 19 12 2023].

[6] C. Schöch, „Ein digitales Textformat für die Literaturwissenschaften," *Romanische Studien,* Bd. 4, pp. 325-364, 2016.

[7] Adobe, [Online]. Available: https://www.adobe.com/de/acrobat/resources/document-files/text-files/docx-file.html. [Zugriff am 19 12 2023].

[8] A. M. Naser, M. H. Btoush und A. H. Hadi, „Analyzing and detecting malicious content: DOCX files," *International Journal of Computer Science and Information Security,* Bd. 14, Nr. 8, pp. 404-412, 2016.

[9] Adobe, [Online]. Available: https://www.adobe.com/de/acrobat/resources/document-files/text-files/txt-file.html. [Zugriff am 20 12 2023].

[10] Adobe, [Online]. Available: https://www.adobe.com/de/acrobat/resources/document-files/open-doc/odt-file.html. [Zugriff am 20 12 2023].

[11] R. Weir, „OpenDocument Format: The Standard for Office Documents," *IEEE Internet Computing,* Bd. 12, Nr. 2, pp. 83-87, 2009.

[12] file-extensions.com, [Online]. Available: https://file-extensions.com/docs/odt. [Zugriff am 20 12 2023].

[13] K. S. Jones und P. Willett, Information Retrieval, USA: Elsevier Science, 1997.

[14] G. G. Chowdhury, Introduction to modern Information Retrieval, London: Facet Publishing, 2010.

[15] I.-Y. Song, S. W. Liddle, T. W. Ling und P. S. (Eds.), Conceptual Modeling– ER 2003, Chicago: Springer, 2003.

[16] L. Wuttke, „datasolut," datasolut, 04 07 2022. [Online]. Available: https://datasolut.com/wiki/etl-prozess/. [Zugriff am 17 01 2024].

[17] Apache Tika, „Apache Tika," The Apache Software Foundation, [Online]. Available: https://tika.apache.org/. [Zugriff am 24 01 2024].

[18] T. Allison, „Confluence," 28 11 2022. [Online]. Available: https://cwiki.apache.org/confluence/display/TIKA/Metadata+Overview. [Zugriff am 31 01 2024].

[19] Apache POI, „Apache POI," Apache, [Online]. Available: https://poi.apache.org/text-extraction.html. [Zugriff am 24 01 2024].

[20] H. Seetha, V. Tiwari, K. R. Anugu, S. Makka und R. Karnati, „A GUI Based Application for PDF Processing Tools Using Python & CustomTkinter," *International Journal for Research in Applied Science & Engineering Technology,* Bd. 11, 01 2023.

[21] M. S. U. Miah, J. Sulaiman, T. B. Sarwar, A. Naseer, F. Ashraf, K. Z. Zamli und R. Jose, „Sentence boundary extraction from scientific literature of electric double layer capacitor domain: tools and techniques," *Applied Sciences,* Bd. 12, p. 1352, 2022.

[22] M. Fenniak, „Python Package Index (PyPI),“ PyPDF2, [Online]. Available: https://pypi.org/project/PyPDF2/. [Zugriff am 06 02 2024].

[23] Y. Shinyama und P. Guglielmetti, „Python Package Index,“ PDFMiner.six, 28 12 2023. [Online]. Available: https://pypi.org/project/pdfminer.six/. [Zugriff am 07 02 2024].

[24] Artifex, „Python Package Index,“ Artifex, 01 02 2024. [Online]. Available: https://pypi.org/project/PyMuPDF/1.23.21/. [Zugriff am 08 02 2024].

[25] X. Chen, „Introduction and Analysis of Python Software,“ *Frontiers in Computing and Intelligent Systems,* Bd. 5, pp. 41 - 43, 2023.

[26] J. Singer-Vine, „Python Package Index,“ MIT License, 11 02 2024. [Online]. Available: https://pypi.org/project/pdfplumber/0.10.4/. [Zugriff am 12 02 2024].

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

[28] B. Khemani und A. Adgaonkar, „A review on reddit news headlines with nltk tool,“ in *Proceedings of the International Conference on Innovative Computing & Communication (ICICC)*, India, 2021.

[29] S. Bird, „NLTK: The Natural Language Toolkit,“ in *Proceedings of the Interactive Presentation Sessions*, Sydney, Australia, 2006.

[30] A. Sharma, R. Aggarwal und R. Alawadhi, „A Comparative Study of Text Summarization using Gensim, NLTK, Spacy, and Sumy Libraries,“ *Journal of Xi'an Shiyou University, Natural Science Edition,* Bd. 19, pp. 1038 - 1043, 04 04 2023.

[31] A. Fantechi, S. Gnesi, S. Livi und L. Semini, „A spaCy-based tool for extracting variability from NL requirements,“ in *Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume B*, Leicester, United Kindom, 2021.

[32] M. Pandey, R. Williams, N. Jindal und A. Batra, „Sentiment analysis using lexicon based approach,“ *IITM Journal of Management and IT,* Bd. 10, pp. 68 - 76, 2019.

[33] A. A. Chaudhri, S. Saranya und S. Dubey, „Implementation Paper on Analyzing COVID-19 Vaccines on Twitter Dataset Using Tweepy and TextBlob," *Annals of RSCB,* Bd. 25, pp. 8393 - 8396, 04 2021.

[34] G. Allen und M. Owens, The Definitive Guide to SQLite, Apress LP New York, 2010.

[35] K. P. Gaffney, M. Prammer, L. Brasfield, D. R. Hipp, D. Kennedy und J. M. Patel, „SQLite: Past, Present, and Future," *Proceedings of the VLDB Endowment,* Bd. 15, Nr. 12, pp. 3535-3547, 01 08 2022.

[36] MySQL, *MySQL 8.0 Reference Manual Including MySQL NDB Cluster 8.0,* 2024.

[37] J. Wahyudi, M. Asbari, I. Sasono, T. Pramono und D. Novitasari, „Database Management Education in MYSQL," *Edumaspul - Jurnal Pendidikan,* Bd. 6, Nr. 2, pp. 2413-2417, 10 2022.

[38] T. Sharma, A. Ashri, N. Kumar, S. Pal und Rajat, „Evaluation of Python Text Summarization Libraries," *INTERNATIONAL JOURNAL OF ENGINEERING DEVELOPMENT AND RESEARCH,* Bd. 9, pp. 159 - 164, 2021.

Abbildungsverzeichnis

Tabellenverzeichnis

Listing-Verzeichnis

Selbständigkeitserklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben. Ich erkläre weiterhin, dass die vorliegende Arbeit in gleicher oder ähnlicher Form noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Zur Unterstützung bei der Korrektur und zur Verbesserung der sprachlichen sowie grammatikalischen Qualität dieser Arbeit wurde das KI-gestützte Tool ChatGPT von OpenAI verwendet. Diese Nutzung beschränkte sich ausschließlich auf die Überprüfung der Sprache und Grammatik. Alle inhaltlichen Beiträge und die wissenschaftliche Auseinandersetzung mit dem Thema erfolgten eigenständig und ohne direkte Einflussnahme durch ChatGPT.

Rostock, den 03.04.2024 **Carla Kalaf**

Carla Kalaf