Extraktion textueller Informationen aus heterogenen PDF-Dokumenten

Bachelorarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Bereich Informatik

vorgelegt von: Mahmoud Ahmad Alkhamis
Geboren am: 15.09.1994 in Al-Raqqa
Matrikelnummer: 217204255
Betreuer: Dr.-Ing. Hannes Grunert
Zweitgutachter: Dr.-Ing. Holger Meyer
Abgabedatum: 15. März 2023

Zusammenfassung

Diese Bachelorarbeit untersucht den Prozess der Extraktion von Text aus Absätzen, Überschriften und Tabellen, die in PDF-Dokumenten enthalten sind, mithilfe des Python-Moduls PDFMiner. In dieser Arbeit wird ein Überblick über viele Python-Bibliotheken sowie ein Vergleich ihrer Funktionen und Eigenschaften präsentiert, mit der Schlussfolgerung, dass PDFMiner die überlegene Option für das Ziel der Textextraktion ist. Die Verarbeitungs- und Implementierungsmethode von PDFMiner wird detailliert aufgeschlüsselt und die Ausgabeergebnisse der Textextraktion werden untersucht, bewertet und auf ihre Korrektheit und Verlässlichkeit geprüft. Die Arbeit schließt mit einigen Vorschlägen für weitere Forschungen in diesem Bereich sowie mit einigen möglichen Anwendungen von PDFMiner in anderen Studienbereichen. Generell zeigen die Ergebnisse der Studie, dass die effiziente Extraktion von Text aus PDF-Dateien das Potenzial hat, in einer Vielzahl von Bereichen wie Bildung, Forschung und Datenanalyse eingesetzt zu werden.

Abstract

This bachelor thesis investigates the process of extracting text from paragraphs, headings and tables contained in PDF documents using the Python module PDFMiner. This thesis presents an overview of many Python libraries and a comparison of their functions and features, concluding that PDFMiner is the superior option for the goal of text extraction. The processing and implementation methodology of PDFMiner is broken down in detail and the output results of text extraction are examined, evaluated and tested for correctness and reliability. The paper concludes with some suggestions for further research in this area as well as some possible applications of PDFMiner in other areas of study. In general, the results of the study show that efficient extraction of text from PDF files has the potential to be used in a variety of fields such as education, research and data analysis.

Inhaltsverzeichnis

Kapitel 1

Einführung

In unserem täglichen Leben verwenden wir oft Dateien im **PDF-Format** für viele Dinge. Als Beispiel dafür ist, wenn wir ein Buch lesen oder bestimmte Dateien per E-Mail von jemandem erhalten, wie z. B. einen Mietvertrag für ein Haus oder eine Immatrikulationsbescheinigung einer Universität usw. In manchen anderen Fällen brauchen wir Daten über uns als Einzelpersonen in Form von PDF-Dateien versenden, wie ein Lebenslauf oder eine Bewerbung, sodass es schwierig geworden ist, auf diese Art von Dateien zu verzichten. Es kann vorkommen, dass wir Textinformationen aus diesen Dateien kopieren müssen, um sie in eine andere Datei einzufügen oder um z. B. bei Google nach anderen Informationen zu suchen. Wenn es sich um eine Datei oder eine kleine Anzahl von PDF-Dateien handelt, bei denen wir die Informationen direkt aus den Dateien kopieren können, braucht es nicht viel Zeit. Das gilt jedoch nicht, wenn wir große Mengen von Texten oder Textinformationen aus einer großen Anzahl dieser Dateien, die wir im PDF-Format haben, extrahieren müssen, dann brauchen wir sicherlich eine andere, bessere und effektivere Methode, um die notwendigen Informationen zu extrahieren, die der Benutzer benötigt.

Im Bereich der wissenschaftlichen und akademischen Forschung führen Experten häufig viele dieser Operationen (**Textextraktionsprozesse**) für viele Zwecke durch. Beispielsweise um bestimmte Ergebnisse zu erhalten oder um diese Texte neu zu analysieren, nachdem sie von vielen nutzlosen Informationen gesäubert und alle störenden Elemente aus ihnen entfernt wurden, um sie dann in anderen Dateiformaten zu speichern oder um diese Informationen mit dem Ziel zu verarbeiten, sie in separaten Analyse- oder Suchvorgängen erneut zu verwenden. So ist es einfach, korrekte und nützliche Ergebnisse zu erhalten, nachdem eine Reihe von Operationen an den aus den verfügbaren Dateien extrahierten Informationen durchgeführt wurden.

1.1 Motivation

Die **Textextraktion** beschreibt als allgemeiner Begriff die gezielte Analyse von Dateien und Dokumenten, die eine Reihe von Techniken umfasst, deren Aufgabe in erster Linie darin besteht, Sammlungen von Dateien zu analysieren, die entweder viele oder wenige Texte enthalten, und nach der Durchführung der notwendigen Analyse kommt die zweite Aufgabe. Diese Aufgabe besteht darin, diese Texte zu extrahieren und die erforderlichen und gewünschten Informationen zu gewinnen. Die Zahl der wissenschaftlichen Forschungsarbeiten, die in bestimmten Formaten wie Artikeln, Büchern oder Forschungspapieren elektronisch veröffentlicht werden, hat in der heutigen Zeit dramatisch zugenommen. Das vorherrschende Format bei der Verbreitung von Dateien im Allgemeinen, sei es im normalen Leben oder in den wissenschaftlichen Bereichen (insbesondere in der Welt der Informatik), ist das PDF-Format, von dem allein Google mehr als drei Milliarden Dateien und Dokumente enthält [1]. Es wurde 1993 von Adobe Systems entwickelt [2]. Das Format ist plattformunabhängig und ermöglicht es, Angaben zu Schriftfarben, -arten und -größen sowie Tabellen und Bilder in Dokumente einzubinden. Aus diesem Grund verwenden viele Autoren und Organisationen das Format, um ihre Texte und verschiedene Informationen in digitaler Form zu veröffentlichen. Aus den obigen Ausführungen können wir uns verschiedene Szenarien für die Extraktion von Texten aus PDF-Dokumenten durch viele Anwendungen unterschiedlicher Qualität vorstellen, die bei der Extraktion von Texten auf viele Schwierigkeiten stoßen, insbesondere wenn es sich um Grafiken, Bilder oder Tabellen handelt, die ebenfalls gewisse Textmengen enthalten. Dies ist zusätzlich zu anderen Herausforderungen. Daher müssen wir bei der Textextraktion die richtigen Schritte unternehmen, um den reinen Text und die erforderlichen Elemente zu extrahieren, was uns die Suche nach bestimmten Informationen oder das Umschreiben in andere Formate erleichtern wird.

1.2 Problemstellung und Zielsetzung

Die Extraktion von Texten hat einen großen Einfluss im Bereich der modernen Informationsverarbeitung, da die schnelle Suche nach Informationen im Internet oder die Extraktion bestimmter Informationen aus bestimmten Dokumenten eine entscheidende Rolle in diesem Bereich spielt. Das Extrahieren von Informationen aus Dokumenten wie beispielsweise PDF-Dokumenten ist nicht so einfach, wie wir uns das vorstellen, und wir

stoßen oft auf viele Herausforderungen und Hindernisse, weswegen wir nicht das gewünschte Ergebnis des Extraktionsprozesses erhalten. Diese Probleme sind auf das Vorhandensein von Elementen zurückzuführen, die den Extraktionsprozess behindern, wie z. B. Grafiken und Tabellen, die auch Textteile enthalten. Wir haben auch Bilder, die manchmal als Trennwand zwischen Absätzen oder Sätzen fungieren und durch die Sätze zerrissen werden und ihren Sinn verlieren. Weitere Schwierigkeiten sind das Vorhandensein von Textkommentaren unter oder neben den Bildern sowie das Vorhandensein von Randbemerkungen und Kommentaren am unteren Rand der Seite. Hinzu kommen Kopf- und Fußzeilen sowie Kapitelüberschriften, die ebenfalls Informationen enthalten, die sich auf die Seite, den Namen des Autors oder den Namen des Dokuments beziehen. Es gibt einige Programme, mit denen man einfach Texte aus Dokumenten extrahieren kann, aber leider liefern sie völlig ungenaue Ergebnisse, da, wie bereits erwähnt, Sätze durch das Vorhandensein von Tabellen, Grafiken oder Bildern aus dem Kontext gerissen werden und somit irrelevante Elemente im Ausgabeergebnis enthalten sind. Dementsprechend werde ich im Rahmen dieser Arbeit versuchen, ein System oder Verfahren zu entwickeln, mit dem ich Dokumente im PDF-Format verarbeiten und Texte daraus extrahieren könnte, sodass ich die bestmöglichen Ergebnisse erhalten könnte, die wir später in vielen Anwendungen einsetzen können.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit ist in sechs Kapitel unterteilt. Im Kapitel 2 wird über die Informationen- und Textextraktion und auch über Begriffe, die dazu gehören, erklärt. Das Kapitel 3 bespricht die aktuellen Python-Bibliotheken, die eine große Rolle in der Textextraktion spielen. Das Kapitel 4 erklärt die prinzipielle Konzeption für die Textextraktion mithilfe der Python-Bibliotheken und das Kapitel 5 enthält die praktische Implementierung dafür. Die Arbeit schließt mit einer Diskussion ab 6.

Kapitel 2

Grundlagen

Wenn man sich mit dem Extrahieren von Texten aus **PDF-Dateien** beschäftigt, muss man zwei grundlegende Begriffe hervorheben, die in diesem Bereich von Bedeutung sind, nämlich das Extrahieren von Informationen und die Rückgabe von Informationen. In dieser Lektion werden wir die **PDF-Dateien**, ihre Zusammensetzung und ihre Bedeutung erklären und verdeutlichen. Darüber hinaus werden wir den Prozess der Extraktion von Texten aus diesen Dateien näher erläutern und einen kurzen Blick auf die beiden bereits erwähnten Begriffe **Informationsextraktion (IE)** und **(Information-Retrieval) IR** werfen

2.1 Informationsextraktion

Da die meisten Informationen, die heute zugänglich sind, nur in Form von Texten und anderen unstrukturierten Medien vorliegen, ist es für Computer unmöglich, diese Informationen direkt zu erfassen. Viele verschiedene Analyseverfahren benötigen jedoch Zugang zu diesen undurchsichtigen Informationen, die unter Umständen nur schwer auffindbar sind. Der Prozess der **Informationsextraktion (IE)** beginnt mit diesem Schritt: Der Prozess der Gewinnung strukturierter Informationen aus unstrukturiertem oder nur teilweise organisiertem Text. Es handelt sich dabei um eine wesentliche Aufgabe im Text Mining, die in einer Reihe von Studienbereichen wie der Verarbeitung natürlicher Sprache, dem Information Retrieval und dem Web Mining ausgiebig erforscht worden ist. Es befasst sich mit dem Prozess der Extraktion vorbestimmter Informationskategorien wie Entitäten und Beziehungen aus Texten, die für Maschinen zugänglich sind. Die Informationsextraktion ist nicht dasselbe wie das Information Retrieval, bei dem es darum geht, Dokumente zu finden, die bestimmten Suchanfragen entsprechen. Die Informationsextraktion ist eine wichtige Technologie für viele moderne Informationsverarbeitungssysteme, da sie als Schnittstelle zwischen strukturierten und unstrukturierten Daten fungiert [3, 4].

Das Hauptziel der **IE** besteht darin, Datenbanken mit extrahierten Informationen zu füllen. Diese Datenbanken werden dann für die weitere Verarbeitung zugänglich gemacht, z. B. für Textmining, Meinungsforschung, Textzusammenfassung und viele andere Möglichkeiten. Die Bearbeitung von textuellen Kundenanfragen und -problemen im Service-Support, die Analyse von Dokumenten in finanz-, rechts-, unternehmens- und sicherheitsrelevanten Bereichen, die Extraktion von formalem Wissen aus Büchern sind nur einige der vielen möglichen Anwendungsbereiche. Weitere Beispiele sind die Analyse von medizinischen Berichten und Arztbriefen für die evidenzbasierte Medizin, die Bearbeitung von textuellen Kundenanfragen und -problemen im Service-Support, die Analyse von Dokumenten in finanziellen, juristischen, unternehmensbezogenen und sicherheitsrelevanten Bereichen[3, 4].

Der Prozess der Informationsextraktion kann in einer sehr breiten Palette von Bereichen nützlich sein. Im Folgenden sind einige Anwendungen aufgeführt, die den Einsatz der **Informationsextraktion** veranschaulichen [5]:

1. Nachrichtendienstliche Analysten durchforsten riesige Mengen an Material, um nach Informationen wie den Namen von Personen, die an terroristischen Aktivitäten beteiligt waren, der Art der verwendeten Waffen und den Orten, an denen die Anschläge verübt wurden, zu suchen. Obwohl Information-Retrieval-Technologien eingesetzt werden können, um schnell Dokumente zu finden, die Vorfälle im Zusammenhang mit Terrorismus beschreiben, sind Technologien zur Informationsextraktion erforderlich, um die genauen Informationseinheiten, die in diesen Dokumenten enthalten sind, genauer zu bestimmen [5].

2. Durch die rasche Ausbreitung des Internets sind Suchmaschinen zu einem wesentlichen Bestandteil des Lebens der Menschen geworden, und die Verhaltensmuster, die die Benutzer bei der Suche anwenden, sind nun viel besser bekannt. Es ist nicht mehr möglich, mit einer Suche, die auf einer *Bag-of-Words-Darstellung* von Dokumenten basiert, gute Ergebnisse zu erzielen. Die Sucherfahrung für die Benutzer kann durch den Einsatz komplexerer Suchtechniken verbessert werden, z. B. durch die Suche nach Entitäten, die strukturierte Suche und die Beantwortung von Fragen [5].

3. Forscher im Bereich der Biomedizin müssen manchmal eine riesige Menge wissenschaftlicher Artikel durchforsten, um Ergebnisse zu finden, die mit bestimmten Genen, Proteinen oder anderen biologischen Einheiten in Verbindung stehen. Da biologische Dinge oft Synonyme und verwirrende Namen haben, kann es schwierig sein, mit einer einfachen Suche, die auf passenden Schlüsselwörtern basiert, relevante Artikel zu finden. Eine einfache Suche auf der Grundlage von Schlüsselwörtern reicht daher möglicherweise nicht aus, um dieses Vorhaben zu unterstützen. Eine der wichtigsten Aufgaben im biomedizinischen Literatur-Mining ist daher die automatische Erkennung von Verweisen auf medizinische Entitäten im Text und die Verknüpfung dieser Erwähnungen mit den entsprechenden Einträgen in bestehenden Wissensdatenbanken [5].

4. Wer in der Finanzbranche arbeitet, muss bei seinen täglichen Entscheidungen oft bestimmte Informationen aus mehreren Nachrichten finden. Ein Finanzdienstleister muss beispielsweise alle Unternehmensübernahmen in einem bestimmten Zeitraum sowie die Einzelheiten jedes einzelnen Kaufs kennen. Standard-Informationsextraktionsmethoden wie *Named-Entity-Identifikation* und Extraktion von Kontaktdaten sind erforderlich, um diese Art von Informationen schnell und automatisch im Text zu finden [5].

2.2 Information Retrieval

Der Begriff **Information Retrieval** beschreibt die Maßnahmen, die ergriffen werden, um Daten und Informationen, die in einer computerisierten Datei oder Datenbank gespeichert sind, zu suchen und abzurufen. Die Suche in Volltextdatenbanken, die Nutzung bibliographischer Datenbanken zum Auffinden bestimmter Dinge und die Bereitstellung von Dokumenten über ein Netzwerk sind alles Methoden, die in heutigen Bibliotheken und Archiven zur Informationsbeschaffung eingesetzt werden [6, 7].

Im Kontext eines Informationssystems ist **Information Retrieval** *(IR)* der Prozess der Identifizierung und Auswahl jener Ressourcen aus einer Datenbank, die einen bestimmten Informationsbedarf am besten decken. Die Indizierung von Material, entweder im Volltext oder in einer anderen Form, kann zur Durchführung von Suchvorgängen verwendet werden. Die Suche nach Informationen in einem Dokument, die Suche nach Dokumenten, die Suche nach Metadaten, die Daten charakterisieren, und die Suche nach Datenbanken, die Texte, Bilder oder Töne enthalten, fallen alle unter den Begriff **Information Retrieval**, das heißt die Untersuchung, wie man Informationen in diesen Formaten findet [7, 8].

Die Entwicklung von Suchmaschinen im Internet ist ein Beispiel für die Verschmelzung von natürlicher Sprachverarbeitung, Hypertext-Verknüpfungen und Schlagwortindexierung bei der Informationsbeschaffung. Forscher auf dem Gebiet der künstlichen Intelligenz untersuchen andere Methoden, die auf eine höhere Abrufgenauigkeit abzielen.

Die Forschung in diesem Bereich umfasst die folgenden Aktivitäten [9]:

1. Dokumente und Textabschnitte werden nach ihrer Kategorie sortiert.
2. Dokumentensammlung werden mithilfe eines Clustering-Algorithmus gruppiert.
3. Dokumente und Sammlungen werden analysiert, um den thematischen Inhalt zu identifizieren.
4. Vergleichende Analyse von linguistischen Merkmalen wie Worthäufigkeit, Wortbedeutung und Wortverteilung.
5. Verlinkung der Dokumente.

Unter der Annahme, dass die Definitionen der beiden vorangegangenen Begriffe für den Leser dargelegt wurden, ist es nun möglich, zum nächsten Absatz überzugehen und eine Erzählung über das PDF zu beginnen, beginnend mit seiner Geschichte, seinen Ursprüngen, seiner Bedeutung und den Gründen, aus denen es verwendet wird.

2.3 PDF-Dokumente

Es ist ziemlich unwahrscheinlich, dass jemand, der regelmäßig das Internet nutzt, das als PDF bekannte Dokumentenformat nicht kennt. Der Name dieser Datei namens Portable Document Format (PDF) setzt sich aus den Anfangsbuchstaben der einzelnen Wörter zusammen, die etwas weiter oben erwähnt wurden. PDF ist eines der am weitesten verbreiteten Formate der Welt und bietet eine Vielzahl unterschiedlicher Möglichkeiten.

2.3.1 Was ist PDF?

Das Adobe Portable Document Format (PDF), das Mitte 1993 erstmals der Öffentlichkeit zugänglich gemacht wurde, hat sich zu einem Standard für die weltweite Verbreitung elektronischer Dokumente in verschiedenen institutionellen Kontexten entwickelt. Seine Fähigkeit, sowohl den Text als auch das visuelle Erscheinungsbild von Quelldokumenten, einschließlich der Schriftarten, Formatierungen, Farben und Grafiken korrekt zu kodieren, hat wesentlich zu seiner weiten Verbreitung und Beliebtheit beigetragen. PDF-Dokumente können mit einem kostenlosen Programm namens Adobe Acrobat Reader, das auf allen wichtigen Computerplattformen verfügbar ist, angezeigt, durchsucht und gedruckt werden. PDF ist ein vielseitiges Format, das häufig zu Veröffentlichungszwecken in einer Vielzahl von Branchen verwendet wird, darunter der öffentliche Sektor, akademische Einrichtungen und Regierungen. Das PDF-Dateiformat wird für die Veröffentlichung eines großen Teils der elektronischen Zeitschriften und anderer digitaler Materialien verwendet, die Bibliotheken gesammelt haben [10, 11].

Wenn es online nach PDF-Dateien gesucht wird, könnte der Suchender auf jedem Computer mehr als 1,5 Milliarden Dokumente herunterladen (Suchmaschine: Yahoo). Dies ist nur ein winziger Teil aller gespeicherten und versendeten Dateien. Das PDF-Format ist aufgrund seiner Zugänglichkeit, Erschwinglichkeit, Nützlichkeit, geringen Dateigröße und hervorragenden Sicherheit beliebt. Der Adobe Reader kann kostenlos auf Linux-, macOS- und Windows-Systemen heruntergeladen werden, um PDF-Dateien zu lesen [12].

Adobes Postscript-Sprache diente als Grundlage für die Entwicklung des plattformunabhängigen PDF-Dokumentenformats, das das Unternehmen später herausbrachte (und das heute fast überall in grafikfähigen Druckern verwendet wird). PDF wurde entwickelt, um neben dem Druck auch die sichere, interaktive Online-Nutzung zu ermöglichen, im Gegensatz zu Postscript, das nur für den Druck entwickelt wurde.

PDF ist ein Format, das Aspekte von strukturierten Textformaten mit konventionellen Bildformaten kombiniert. Dadurch kann ein Dokument sowohl inhaltlich als auch optisch auf dem Bildschirm oder auf der Seite genutzt werden. PDF ist das am weitesten verbreitete Format für den elektronischen Dokumentenaustausch. So können PDF-Leser Dokumente nicht nur anzeigen und drucken, sondern auch textorientierte Aktionen wie die Suche nach Textstrings, Ausschneiden und Kopieren usw. durchführen. Da PDF im Kern eine Seitenbeschreibungssprache ist, ähnelt es in dieser Hinsicht grundsätzlich Postscript. Andererseits ist es im Gegensatz zu Postscript eher eine deklarative als eine prozedurale Sprache (d. h. die Daten sind nicht wie in Postscript in Computercode verpackt). Aus diesem Grund sind PDF-Dateien einfacher zu ändern und leichter zu analysieren als Postscript-Dateien [10, 11].

Eine Standard-PDF-Datei besteht aus einer Reihe von Seiten, von denen jede Daten enthält, die angeben, was auf der jeweiligen Seite gezeichnet werden soll. Diese Informationen bestehen aus Text, Schriftarten, Rändern, Layout, grafischen Komponenten und den Farben des Hintergrunds und des Textes. Eine PDF-Datei enthält Metadaten über das Dokument, die u. a. die Seitengröße, das Seitennummerierungssystem und ein strukturiertes Inhaltsverzeichnis angeben. Zusätzliche Add-ons, die dynamische Anzeigen oder die Benutzeroberfläche steuern, können ebenfalls eingefügt werden. Diese Zusätze können in Form von Hyperlinks, Formularen oder sogar in Java oder Postscript geschriebenem Skriptcode vorliegen [10, 11].

Einige PDFs kodieren den Dokumentinhalt nicht direkt. PDFs können Seitenbilder enthalten. Einige Scanner erzeugen uninterpretierte PDF-Dateien. Uninterpretierte PDF-Dateien sind größer als PDF-Dateien, die Text direkt als Zeichen kodieren, und erlauben keine textorientierten Funktionen wie Suchen oder Kopieren und Einfügen. Andere Software kann versuchen, Text in einer gescannten PDF-Datei zu erkennen und die Bilder durch eine kompaktere Kodierung der Buchstaben und Schriftarten zu ersetzen. Diese Anwendungen erzeugen hybride PDF-Dateien, indem sie Textstrings in Seitengrafiken kodieren und unbekannte Zeichen als Bilder beibehalten. Diese PDF-Dateien sind zwar kleiner als solche mit uninterpretierten Seitenbildern, aber ihr Inhalt ist manchmal unvollständig und abgehackt kodiert. Erkennungswerkzeuge können nicht 100 Prozent des Textes einer gescannten Seite identifizieren. Nicht erkannte Zeichen können zu Problemen mit Hybriddokumenten führen [10].

PDF-Dateien können den Inhalt verschlüsseln und das Drucken verbieten, wenn der Benutzer dies wünscht. Eine Datei kann so verschlüsselt sein, dass sie ohne das richtige Kennwort nicht entschlüsselt werden kann. Wenn ein Dokument entschlüsselt ist, kann es frei kopiert, gelesen und gedruckt werden [10, 11].

2.3.2 PDF-Dateiformate

Die PDF-Standardfamilie umfasst neben dem traditionellen PDF-Format auch die folgenden Dateitypen: PDF/A, PDF/UA, PDF/X und PDF/VT. Auch PDF/E ist ein Mitglied der Familie. Darüber hinaus ist das Verfahren entsprechend den verschiedenen Dokumentenformaten in Stufen aufgeteilt. Wir stellen die Familie der PDF-Standards in komprimierter, präziser und zukunftsorientierter Form zur Verfügung [13, 14].

1. Diese Version von PDF, PDF/A genannt, wurde speziell für Archivare und Aufbewahrer entwickelt. Der PDF/A-Standard hat im Gegensatz zum Standard-PDF einen eingeschränkteren Funktionsumfang. So werden beispielsweise Audio- und Videodateien blockiert, Verschlüsselungen verwendet und die verwendbaren Schriftarten eingeschränkt. Der Grund dafür ist die Möglichkeit, dass diese Funktionen in Zukunft nicht mehr unterstützt werden oder von demjenigen, der das Dokument betrachtet, nicht mehr zugänglich sind [13, 14].

2. Der Zugang zu Dokumenten wird für Menschen mit Behinderungen durch einen Standard erleichtert, der als PDF/UA bekannt ist, was eine Abkürzung für *Universal Access* ist. PDF/UA-Dokumente können, wenn sie mit der erforderlichen Software geöffnet werden, mit Hilfe von Hilfsmitteln wie Bildschirmlesegeräten, Bildschirmlupen, Joysticks und anderen Geräten gelesen werden, die es sehbehinderten Personen ermöglichen, auf den Inhalt des Dokuments zuzugreifen [13, 14].

3. PDF/VT: Der PDF/VT-Standard unterstützt, ähnlich wie der PDF/X-Standard, Farbprofile, Ebenen und durchscheinende Grafiken. VT (Variable und Transactional) ist eine Abkürzung, die für Dokumente wie Kontoauszüge, Firmenrechnungen und andere variable Dokumente verwendet wird, die aus geschäftlichen Gründen gedruckt werden müssen [13, 14].

4. PDF/E ist eine Variante von PDF, die die Anforderungen der Bau- und Fertigungsindustrie sowie interaktive Medien, Animationen und 3D-Grafiken unterstützt. Aus diesem Grund ist es vor allem für Produktdesign-Teams sowie für Architektur- und Ingenieurbüros hilfreich [13, 14].

5. PDF/X: Wenn Sie ein Grafikdesigner oder ein kommerzieller Drucker sind, bietet der PDF/X-Standard hervorragende Unterstützung für Grafiken während des gesamten Freigabe- und Druckprozesses. Im Gegensatz zum normalen PDF-Format enthält dieses Format zusätzliche Funktionen, die es ermöglichen, Papiere auf hohem professionellem Niveau zu drucken, ohne die Qualität des Dokuments zu verschlechtern oder Verzerrungen zu verursachen. PDF/X bettet Schriftarten, Fotos, Farbprofile und andere Informationen so ein, dass sie beim Druck nicht verändert werden können. Dadurch wird verhindert, dass Bilder falsch gedruckt werden und sichergestellt, dass auch andere Informationen korrekt gedruckt werden [13, 14].

PDF ist ein Dokumentenformat, das eine breite Palette von Anwendungen bietet, sehr effektiv ist und von einer Vielzahl von Herstellern und Softwareprogrammen unterstützt wird. Sie werden also nie um eine Lösung verlegen sein, die trotz der vielen Kriterien sowohl individuell als auch kostengünstig ist. Im nächsten Abschnitt wird erläutert, wie sich PDF-Dateien mithilfe bestimmter Bibliotheken in der Programmiersprache *Python* bearbeiten lassen und wie effektiv sie bei der Extraktion und Änderung von textuellen und nicht textuellen Informationen sind.

2.3.3 PDF-Struktur

Die Interpretation eines Dokuments hängt von der eigenen Interpretation des Lesers ab, wobei ein Dokument strukturiert, halbstrukturiert oder unstrukturiert sein kann. In den meisten Fällen hat ein Dokument, das von Menschen gelesen werden kann, sowohl eine physische Form als auch eine logische Struktur. Ein Dokument hat Teile. Ein Abschnitt kann einen Titel, einen Abschnittsteil oder eine darin verschachtelte Struktur haben. Ein Abschnittswechsel kann in Form von zusätzlichem Abstand, einer oder mehreren Leerzeilen oder einer Abschnittsüberschrift für die Komponente, die nach dem Abschnitt kommt, erfolgen. Abschnitte sind die Komponenten, die visuell voneinander getrennt sind [15]. Die Abbildung 2.1 zeigt ein PDF-Dokument-Modell[1] als eine Sammlung von Teilen, Unterabschnitten usw.

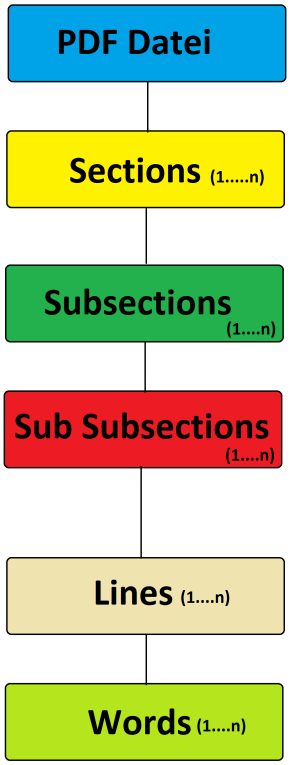


Abbildung 2.1: Ein PDF-Dokument-Modell [15]

Verschiedene Anwendungen zur Informationsextraktion und -abfrage, zur Kategorisierung und Zusammenfassung von Dokumenten und zur Textanalyse können alle von der Identifizierung der logischen Abschnitte eines Dokuments und der Organisation dieser Abschnitte in einer Standardstruktur profitieren [15].

[1] `https://ebiquity.umbc.edu/_file_directory_/papers/857.pdf`, zuletzt aufgerufen am 20.01.2023

2.3.4 Logische und physische Struktur der PDF-Datei

Um zu wissen, wie ein PDF-Dokument aufgebaut ist, ist es erforderlich, zwischen der logischen Dokumentenstruktur und der physischen Dokumentenstruktur zu unterscheiden.

Logische Dokument-Struktur

Da der Inhalt des Dokuments in Form von hierarchischen Zusammenhängen organisiert werden kann, lässt sich daraus schließen, dass das Dokument eine logische Struktur [16] aufweist und wird dieses als logisches Dokument bezeichnet. Die logische Struktur des Dokuments verbindet den semantischen Inhalt des Dokuments mit der tatsächlichen physischen Erscheinungsform des Dokuments. Sie spiegelt die Art und Weise wider, in der Informationen in Form von logischen Objekten wie Kapiteln, Titeln, Absätzen, Abbildungen, Überschriften usw. strukturiert sind. Die hierarchischen Beziehungen sind die üblichen Formen der Verbindungen zwischen den zahlreichen logischen Objekten, aus denen die logische Struktur besteht. Ein Dokument hat zum Beispiel einen Titel, eine Übersicht und eine Reihe von Kapiteln. Jedes dieser Kapitel hat seinen eigenen Titel sowie seine eigene Abfolge von Teilen usw. Die logische Struktur wird oft als Baumstruktur dargestellt, um die hierarchischen Verbindungen zwischen den verschiedenen logischen Elementen zu erfassen. Die logischen Strukturen der verschiedenen Dokumente können gleich oder unterschiedlich sein. So unterscheidet sich beispielsweise die logische Struktur eines juristischen Dokuments erheblich von der einer Geschäftsstrategie oder einer wissenschaftlichen Untersuchung. Die logische Struktur des PDF-Dokuments kann als Baum dargestellt werden wie in der Abbildung 2.2 [16].

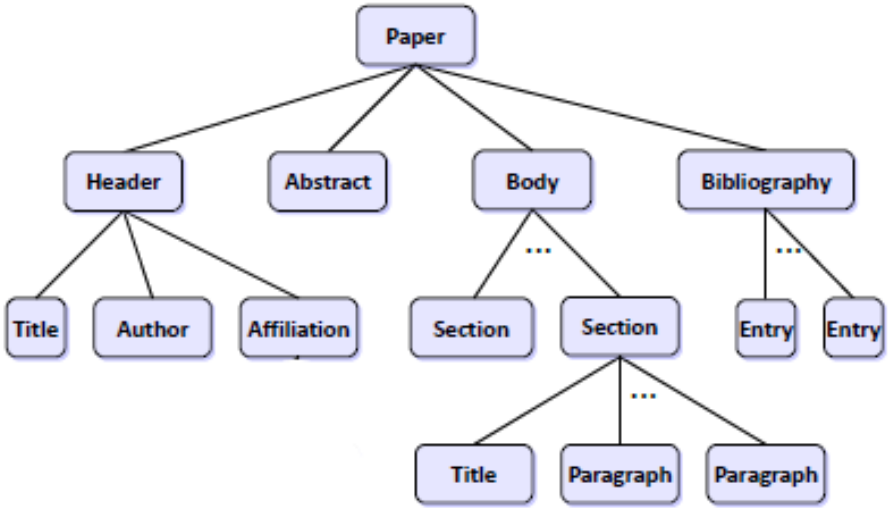


Abbildung 2.2: Der logische Aufbau eines Dokuments [16].

Physische Dokument-Struktur

Die logische Struktur eines physischen Dokuments wird nicht mehr direkt festgelegt, sondern lässt sich aus den typografischen Merkmalen und dem Layout des Dokuments ableiten, die zusammen die physische Struktur des Dokuments ausmachen. Die physische Struktur einer PDF-Datei wird durch eine Reihe von Standarddatenstrukturen und -typen definiert, die verwendet werden, um die verschiedenen Elemente des Dokuments darzustellen. Bilder, Grafiken und Textblöcke, die weiter in Textzeilen, Wörter und Zeichen unterteilt werden können, beziehen sich auf die physische Struktur eines Dokuments in Bezug auf die Organisation der Seite [16]. Auf diese Weise spiegelt das äußere Erscheinungsbild einer Seite lediglich die zugrunde liegende physische Struktur des Textes wider. Die physische Struktur wird oft als Baumstruktur dargestellt, ähnlich wie die logische Struktur, um die hierarchischen Verbindungen zwischen den vielen physischen Objekten wiederzugeben. Das in Abbildung 2.3 dargestellte Dokument wird in seine Bestandteile zerlegt und in Abbildung 2.4 dargestellt [16]. Insgesamt ist die physikalische Struktur einer PDF-Datei so ausgelegt, dass sie sowohl flexibel als auch effizient ist und eine breite Palette von Inhalten darstellen kann, um leicht übertragen und gespeichert werden zu können.

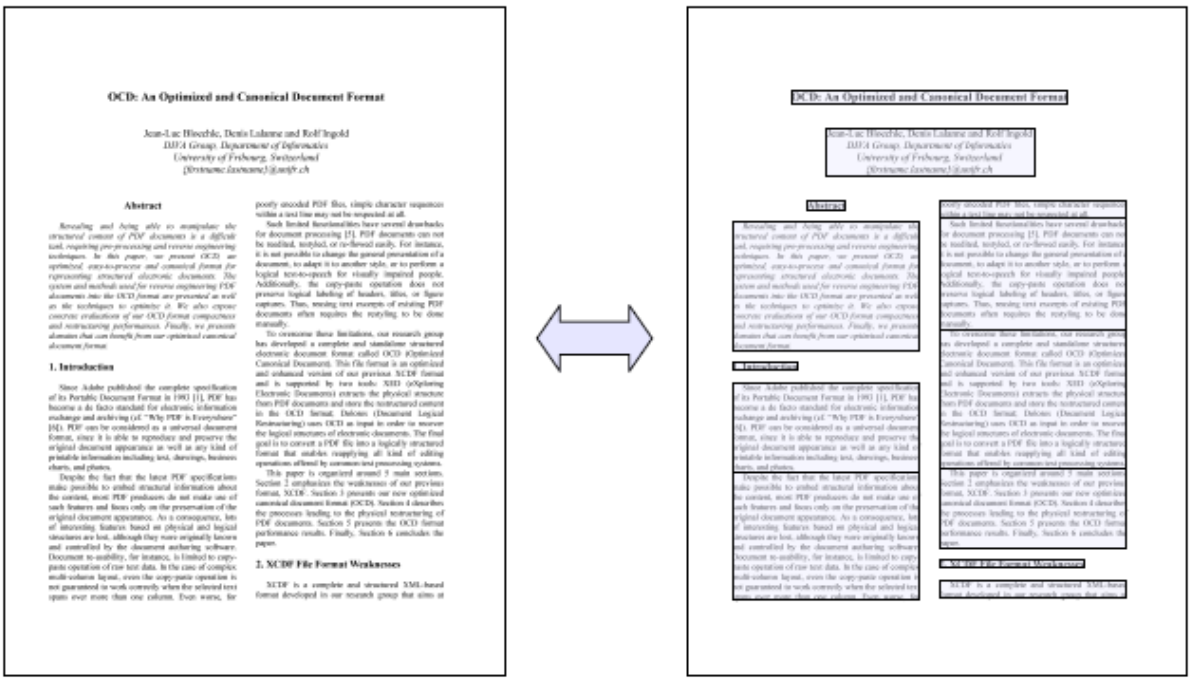


Abbildung 2.3: Links ist Seite eines Dokuments, rechts Klassifizierung ihrer Textblöcke [16].

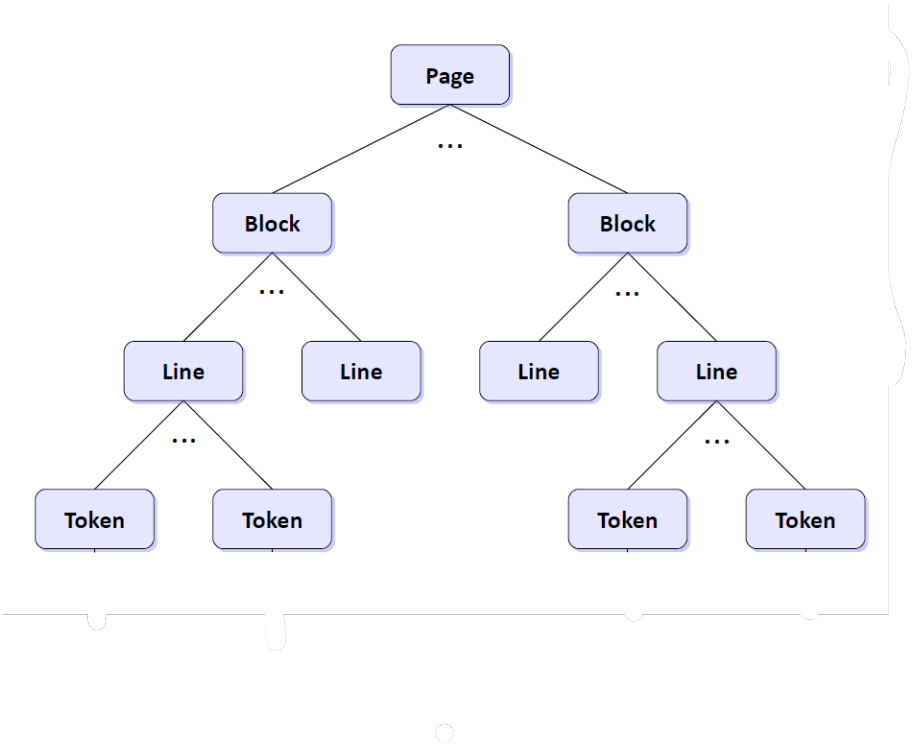


Abbildung 2.4: Darstellung der physischen Struktur des Dokuments als Baumstruktur [16].

2.3.5 Aufbau der PDF-Seite

Eine PDF-Seite ist ein Teil eines PDF-Dokuments, der den Inhalt und das Layout einer Seite in einem Druck- oder digitalen Dokument darstellt. Eine PDF-Seite kann Text, Bilder, Grafiken, Anmerkungen und andere Inhalte enthalten, die auf der Seite angezeigt werden (siehe Abbildung 2.3) [17].
Einige der Hauptkomponenten einer PDF-Seite können sein:

1. Text: Der Hauptteil des Dokuments besteht in der Regel aus Text, der mit verschiedenen Schriftarten, Größen und Stilen formatiert werden kann.

2. Bilder: PDFs können statische Bilder wie JPEG- oder PNG-Dateien enthalten, aber auch interaktive Elemente wie Schaltflächen und Formularfelder.

3. Grafiken: PDFs können auch Vektorgrafiken wie Linien, Formen und Kurven enthalten, die innerhalb des PDF selbst bearbeitet und modifiziert werden können.

4. Anmerkungen: PDFs können Anmerkungen wie Notizen, Hervorhebungen und Kommentare enthalten, die von Benutzern hinzugefügt werden können, um zusätzlichen Kontext oder Informationen zum Dokument bereitzustellen.

5. Links: PDFs können Hyperlinks enthalten, die es Benutzern ermöglichen, auf ein bestimmtes Wort oder eine bestimmte Phrase zu klicken, um zu einem anderen Teil des Dokuments oder zu einer externen Website zu navigieren.

6. Tabelle: In einer PDF (Portable Document Format) -Datei können Tabellen verwendet werden, um Daten in einer strukturierten Form darzustellen. Eine Tabelle besteht aus Zeilen und Spalten, die Daten in einer matrixartigen Struktur organisieren. Jede Zelle in der Tabelle kann Text oder andere Inhalte enthalten, wie zum Beispiel Zahlen oder Bilder.

Insgesamt Ein PDF-Dokument kann aus einer einzelnen Seite oder aus mehreren Seiten bestehen, je nachdem, wie viel Inhalt das Dokument enthält. Benutzer können in der Regel zwischen den Seiten eines PDF-Dokuments navigieren, indem sie die Pfeiltasten oder die Schieberegler in der Anwendung verwenden, die das PDF anzeigt [17].

Kapitel 3

Stand der Technik und verwandte Arbeiten

3.1 Extraktion von Daten/Informationen aus PDF-Formaten

Das Extrahieren von Informationen aus einer Vielzahl von Formaten und Quellen (einschließlich Adobe PDF-Dateien, CSV-Dateien, dem Internet usw.) wird als Datenextraktion bezeichnet. Während bei bestimmten Formaten, wie CSV, die Datenextraktion unkompliziert ist, sind bei unstrukturierten PDF-Dokumenten weitere Bearbeitungsschritte vorzunehmen [18]. Das einfache Kopieren und Einfügen der relevanten Informationen aus einer kleinen Anzahl von PDF-Dateien kann Zeit sparen. Der benötigte Text kann beispielsweise leicht in einer Excel-Tabelle zusammengestellt werden, indem jedes Quelldokument geöffnet wird, und dann der benötigte Text wird markieret, kopiert und dann eingefügt. Dieser Ansatz eignet sich jedoch am besten, wenn es keine große Anzahl von Dokumenten zum Verwalten geben würde. Mithilfe spezieller Technologien kann der zeitaufwendige Prozess der Extraktion von Daten aus mehreren Dokumenten vereinfacht werden [19, 20]. Der am häufigsten verwendete digitale Ersatz für Papierdokumente ist das PDF-Dateiformat. Da es mit den meisten Betriebssystemen und Geräten kompatibel ist, stellt es eine einfache und praktische Lösung für alle dar, die die Informationen an eine große Anzahl von Personen weitergeben möchten. Forscher versuchen oft, so viele nützliche Daten wie möglich aus wissenschaftlichen Veröffentlichungen zu extrahieren. Leider erschweren viele der heute gebräuchlichen digitalen Dokumentenformate (vor allem PDF) die Extraktion von Informationen aus digitalen Veröffentlichungen, da sie für den Druck und nicht für die Computerverarbeitung konzipiert wurden. Es gibt zwar eine Reihe von Tools, mit denen die Informationen von einem PDF-Dokument genau extrahiert werden können, aber die Forscher stellen oftmals fest, dass die resultierende Ausgabe ihren Anforderungen nicht genügt, da diese Ausgabe nicht angemessen beschriftet ist und kein maschinenlesbares Format hat. Die Herausforderun-

gen bei der Informationsextraktion aus PDF-Dateien werden durch die Einschränkungen herkömmlicher PDF-Extraktionsmethoden und der auf diesen Strategien basierenden Bewertungswerkzeuge noch verschärft. Probleme mit Kopf- und Fußzeilen, Abbildungen und Tabellen sind allesamt potenzielle Schwachstellen [21].

Bei PDF handelt es sich um ein layout-basiertes Format, d. h, das PDF enthält nur Informationen darüber, welche Buchstaben in welcher Schriftart an welcher Stelle im Text stehen, deshalb ist es nicht einfach, bestimmte Informationen aus einem solchen Dokument zu extrahieren. In den meisten Fällen ist der strukturelle Zusammenhang zwischen den Zeichen jedoch unverständlich. Dies ist aber eine entscheidende Information für eine textuelle Inhaltssuche. Inzwischen gibt es eine Vielzahl von Anwendungen, die rudimentäre Versuche unternehmen, nützlichen Text aus PDF zu extrahieren. Der Prozess ist jedoch unvollständig, da viele unwichtige Details übrigbleiben, nachdem die Sätze durch Tabellen und/oder Abbildungen zerlegt wurden [22].

Die Textextraktion aus Dokumenten kann manchmal schwierig sein, da das Portable Document Format (PDF) nur für die Anzeige von Dokumenten und nicht für die organisierte Speicherung von Dokumentinformationen konzipiert wurde. Die meisten Herausforderungen, die sich bei den hier besprochenen wissenschaftlichen Veröffentlichungen stellen, sind auf die Verwendung von Layouts mit vielen Spalten zurückzuführen. Tabellen, Grafiken, Algorithmen und andere vergleichbare Blöcke, die Text enthalten, können ebenfalls eine Quelle von Schwierigkeiten darstellen. Sie können nicht in die Extraktion einbezogen werden, da sie eine Unterbrechung im übrigen Text darstellen. Es ist möglich, einen Satz so zu zerlegen, dass er seinen beabsichtigten Sinn nicht mehr vermittelt. Zusätzlich zu Überschriften oder Fußnoten enthalten diese fehlerhaften Blöcke eine Reihe von anderen Blöcken. Die Position von Fußnoten am Ende einer Seite führt dazu, dass Sätze unzusammenhängend werden, obwohl die Fußnoten selbst Informationen liefern. Bindestriche, Worttrenner und Fußnotenmarkierungen, die sich alle innerhalb des Textes befinden, dienen dazu, Sätze zu unterbrechen. Aus diesem Grund müssen auch sie aufgelöst werden. In wissenschaftlichen Fachzeitschriften sind Quellenangaben zwar vorgeschrieben, aber weil sie für die Textextraktion keine für die semantische Suche nutzbaren Informationen liefern, werden sie weggelassen. Darüber hinaus stellen Kopf- und Fußzeilen zusätzliche Herausforderungen für Systeme zur Textextraktion dar. Manchmal sind die einzigen Informationen, die sie enthalten, ein Wasserzeichen, die Überschrift des Kapitels, die Seitenzahl, der Autorenname oder der Titel der Arbeit. Der Autorenname und der Titel des Artikels sind ebenfalls Beispiele für andere Textbereiche, die Metadaten enthalten, die verwendet werden können. Dem Leser werden große Textabschnitte zur Navigation angeboten, die durch Metainformationen getrennt sind, die auch durchgesucht werden können [23].

3.2 Python-Bibliotheken für die Extraktion von PDF-Inhalten

Da Python derzeit als eine der leistungsstärksten Programmiersprachen im Bereich der künstlichen Intelligenz im Allgemeinen gilt [24], liegt der Schwerpunkt dieses Kapitels auf der praktischen Anwendung des Abrufs von Daten aus einer PDF-Datei und dem anschließenden Export in ein anderes Format mit Python. In den Bereichen der Verarbeitung natürlicher Sprache, der Textverarbeitung und des Data Mining gibt es jeweils eine Fülle von Bibliotheken, die in Python zur Verfügung stehen, um die verschiedenen Aufgaben, die in jedem dieser Bereiche durchgeführt werden können, zu erleichtern. Da sie für die grundlegenden Aufgaben in der Textverarbeitung repräsentativ sind, werden die Bibliotheken und spezialisierten Pakete, die in diesem Kapitel behandelt werden, im Mittelpunkt der Aufmerksamkeit stehen. Leider gibt es nicht viele Python-Bibliotheken, die in der Lage sind, Daten auf zufriedenstellende Weise zu extrahieren. In diesem Kapitel wird eine Reihe von verschiedenen Paketen von Python diskutiert, die für die Textextraktion verwendet werden können. Jedes dieser Pakete wird kurz auf seine Bedeutung sowie auf seine Stärken und Schwächen eingehen.

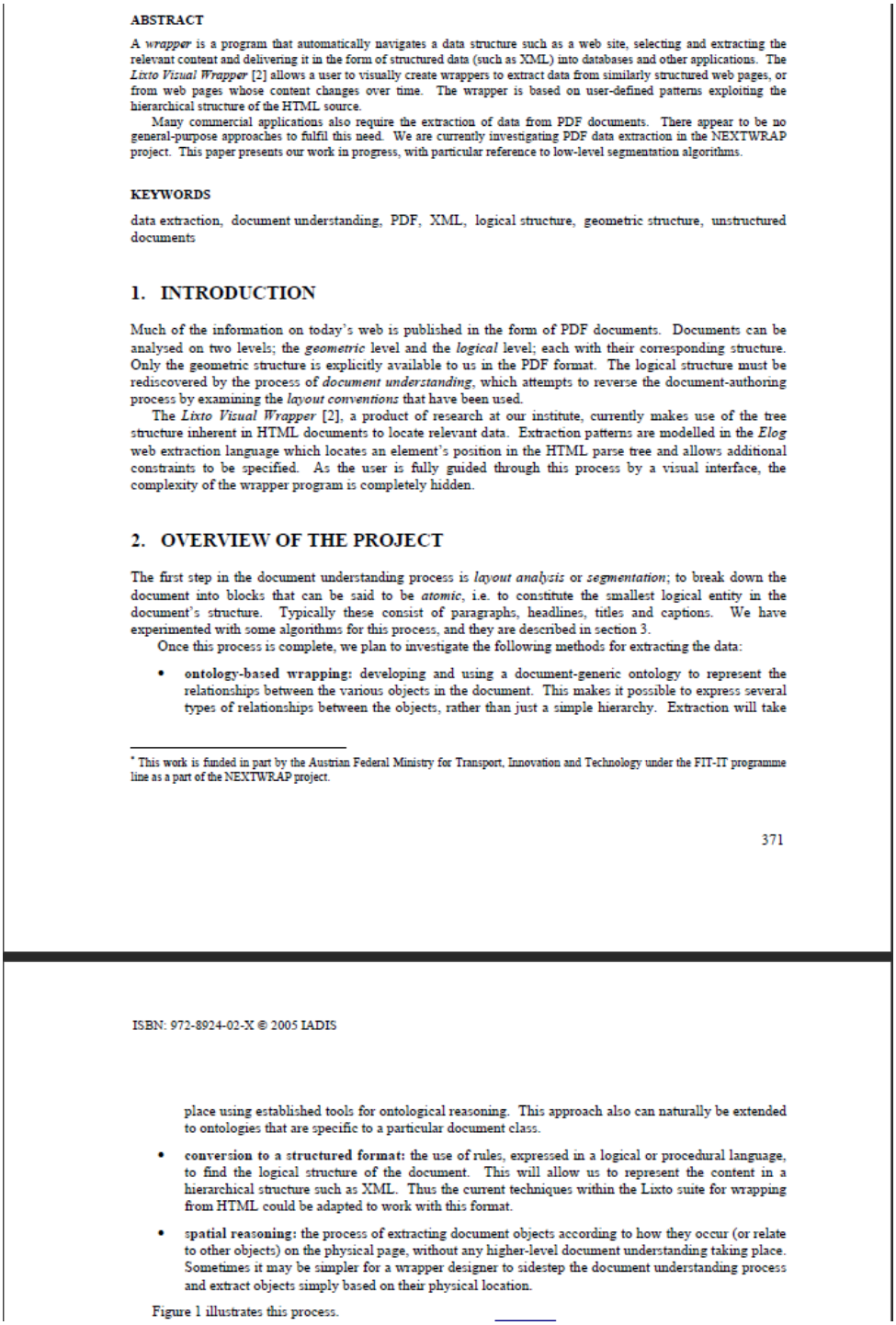
3.2.1 pypdf2

PyPDF2 ist eine der beliebtesten Softwarebibliotheken für den Umgang mit PDF-Dateien. Diese Bibliothek zeichnet sich dadurch aus, dass sie vollständig in der Programmiersprache Python geschrieben wurde, Open Source ist und völlig kostenlos genutzt werden kann. Der Hauptzweck der Erstellung dieser Bibliothek besteht darin, mit PDF-Dateien umzugehen und ihre Seiten auf verschiedene Arten zu verarbeiten, einschließlich Zuschneiden, Teilen und Zusammenführen. PyPDF2 ist in der Lage, Text und Informationen aus PDF-Dateien zu extrahieren, z.B. den Autor, den Herausgeber und das Erstellungsdatum des Dokuments sowie weitere Informationen [25, 26, 27].
Diese Bibliothek extrahiert alle Textinhalte aus der PDF-Datei, mit der sie umgeht, beginnend mit dem Dateinamen und endend mit dem Index, wenn die Datei ein Inhaltsverzeichnis hätte. Dabei untersucht die Bibliothek jede Seite der Datei, liest sie und extrahiert den Text daraus unabhängig davon, ob der Text aus einem Absatz, einer Tabelle, einer Erklärung eines Bildes oder einer Grafik stammt. Die Wahrscheinlichkeit, dass das Extraktionsergebnis genau ist, ist größer, wenn die Datei kleiner ist. Wenn die Datei jedoch eine große Anzahl von Seiten und verschiedene Komponenten wie Texte, Fotos und Tabellen enthält, ist das Ergebnis eher fehlerhaft.

```
1  !pip install PyPDF2
2
3
4  from PyPDF2 import PdfFileReader, PdfFileWriter
5
6  file_path = '/content/Intelligent_Text_Extraction_from_PDF.pdf'
7  pdf = PdfFileReader(file_path)
8
9  with open('Lecture Note.txt', 'w') as f:
10     for page_num in range(pdf.numPages):
11         pageObj = pdf.getPage(page_num)
12
13         try:
14             txt = pageObj.extractText()
15             print(''.center(100, '-'))
16         except:
17             pass
18         else:
19             f.write('Page {0}\n'.format(page_num+1))
20             f.write(''.center(100, '-'))
21             f.write(txt)
22
23     f.close()
24
25 with open('/content/Lecture Note.txt') as f:
26     contents = f.read()
27     print(contents)
```

Listing 3.1: Python-Beispiel für das Package Pypdf2

Listing 3.1 zeigt ein Codebeispiel zum Aufrufen und Verwenden des Packages Pypdf2, wo in der Zeile 1 wurde die Bibliothek installiert zum Framework, um deren Klassen zu benutzen. In der Zeile 4 wurden **PdfFileReader** und **PdfFileWriter** aufgerufen, um die Originaldatei zu lesen und in der neuen Datei die Ausgaben zu schreiben. In der Zeile 6 wurde die Originaldatei (siehe Abbildung 3.1) aufgerufen und in der Zeile 7 gelesen durch PdfFileReader. In der Zeile 9 wurde eine neue Datei erstellt und durch eine **for**-Schleife wurde den Text von der Originaldatei extrahiert und in der neuen Datei aufgeschrieben. In der Zeile 25 wurde die Ausgabe-Datei geöffnet und durch die Anweisung **print** in der Zeile 27 gedruckt und gezeigt wie in Listing 3.2.



ABSTRACT

A *wrapper* is a program that automatically navigates a data structure such as a web site, selecting and extracting the relevant content and delivering it in the form of structured data (such as XML) into databases and other applications. The *Lixto Visual Wrapper* [2] allows a user to visually create wrappers to extract data from similarly structured web pages, or from web pages whose content changes over time. The wrapper is based on user-defined patterns exploiting the hierarchical structure of the HTML source.

Many commercial applications also require the extraction of data from PDF documents. There appear to be no general-purpose approaches to fulfil this need. We are currently investigating PDF data extraction in the NEXTWRAP project. This paper presents our work in progress, with particular reference to low-level segmentation algorithms.

KEYWORDS

data extraction, document understanding, PDF, XML, logical structure, geometric structure, unstructured documents

1. INTRODUCTION

Much of the information on today's web is published in the form of PDF documents. Documents can be analysed on two levels; the *geometric* level and the *logical* level; each with their corresponding structure. Only the geometric structure is explicitly available to us in the PDF format. The logical structure must be rediscovered by the process of *document understanding*, which attempts to reverse the document-authoring process by examining the *layout conventions* that have been used.

The *Lixto Visual Wrapper* [2], a product of research at our institute, currently makes use of the tree structure inherent in HTML documents to locate relevant data. Extraction patterns are modelled in the *Elog* web extraction language which locates an element's position in the HTML parse tree and allows additional constraints to be specified. As the user is fully guided through this process by a visual interface, the complexity of the wrapper program is completely hidden.

2. OVERVIEW OF THE PROJECT

The first step in the document understanding process is *layout analysis* or *segmentation*; to break down the document into blocks that can be said to be *atomic*, i.e. to constitute the smallest logical entity in the document's structure. Typically these consist of paragraphs, headlines, titles and captions. We have experimented with some algorithms for this process, and they are described in section 3.

Once this process is complete, we plan to investigate the following methods for extracting the data:

- **ontology-based wrapping:** developing and using a document-generic ontology to represent the relationships between the various objects in the document. This makes it possible to express several types of relationships between the objects, rather than just a simple hierarchy. Extraction will take

* This work is funded in part by the Austrian Federal Ministry for Transport, Innovation and Technology under the FIT-IT programme line as a part of the NEXTWRAP project.

ISBN: 972-8924-02-X © 2005 IADIS

place using established tools for ontological reasoning. This approach also can naturally be extended to ontologies that are specific to a particular document class.

- **conversion to a structured format:** the use of rules, expressed in a logical or procedural language, to find the logical structure of the document. This will allow us to represent the content in a hierarchical structure such as XML. Thus the current techniques within the Lixto suite for wrapping from HTML could be adapted to work with this format.
- **spatial reasoning:** the process of extracting document objects according to how they occur (or relate to other objects) on the physical page, without any higher-level document understanding taking place. Sometimes it may be simpler for a wrapper designer to sidestep the document understanding process and extract objects simply based on their physical location.

Figure 1 illustrates this process.

Abbildung 3.1: Ein Ausschnitt einer zu analysierenden PDF-Datei [28].

```
ABSTRACT
A wrapper  is a program that automatically navigates a data structure such as a web site, selecting and extracting the
relevant content and delivering it in the form of structured da ta (such as XML) into database s and other applications.  The
Lixto Visual Wrapper [2] allows a user to visually create wrappers to extract data from similarly structured web pages, or
from web pages whose content changes ove r time.  The wrapper is based on us er-defined patterns exploiting the
hierarchical structure of the HTML source.
Many commercial applications also require the extractio n of data from PDF documents.  There appear to be no
general-purpose approaches to fulfil this  need.  We are currently investigating PDF data extraction in the NEXTWRAP
project.  This paper presents  our work in progress, with particular re ference to low-level segmentation algorithms.
KEYWORDS
data extraction,  document understanding,  PDF,  XML,  logical structure,  geometric structure,  unstructured
documents
1. INTRODUCTION
Much of the information on t o d a y s  web is published in the form of PDF documents.  Documents can be analysed on two levels;
the geometric  level and the logical  level; each with their corresponding structure.
Only the geometric structure is explicitly available to us in the PDF format.  The logical structure must be
rediscovered by the process of document understanding , which attempts to reverse the document-authoring
process by examining the layout conventions  that have been used.
The Lixto Visual Wrapper  [2], a product of research at our in stitute, currently makes use of the tree
structure inherent in HTML documents to locate relevant data.  Extraction patterns are modelled in the Elog
web extraction language which locates an e l e m e n t s  pos ition in the HTML parse tree and allows additional
constraints to be specified.  As the user is fully gu ided through this process by a visual interface, the
complexity of the wrapper program is completely hidden.
2. OVERVIEW OF THE PROJECT
The first step in the document understanding process is layout analysis  or segmentation ; to break down the
document into blocks that can be said to be atomic , i.e. to constitute the smallest logical entity in the
d o c u m e n t s  structure.  Typically these consist of paragraphs, headlines, titles and captions.  We have
experimented with some algorithms for this process, and they are described in section 3.
Once this process is complete, we plan to investigate the following methods for extracting the data:
    ontology-based wrapping: developing and using a document-generic ontology to represent the
relationships between the various objects in the docum ent.  This makes it possible to express several
types of relationships between the objects, rather th an just a simple hierarchy.  Extraction will take

* This work is funded in part by the Austrian Federal Minist ry for Transport, Innovation and Technology under the FIT-IT
program me
line as a part of the NEXTWRAP project.
IADIS International Conference on WWW/Internet 2005
371
place using established tools for ontol ogical reasoning.  This approach  also can naturally be extended
to ontologies that are specific to  a particular document class.
    conversion to a st ructured format: the use of rules, expressed in a logical or procedural language,
to find the logical structure of the document.  This will allow us to represent the content in a
hierarchical structure such as XM L.  Thus the current techniques within the Lixto suite for wrapping
from HTML could be adapted to work with this format.
    spatial reasoning: the process of extracting document objects according to how they occur (or relate
to other objects) on the physical  page, without any higher-level doc ument understanding taking place.
Sometimes it may be simpler for a wrapper desi gner to sidestep the document understanding process
and extract objects simply base d on their physical location.
Figure 1 illustrates this process.
```

Listing 3.2: Extrahierter Text von einer PDF Datei durch Pypdf2

Leider hat die Verwendung des PyPDF2-Pakets negativen Aspekten beziehungsweise Nachteile davon, dass dieses Paket zwar extrahiert Text, behält aber nicht die Struktur des Textes bei, die in der ursprünglichen PDF-Datei vorhanden war. Zusätzlich enthält der extrahierte Text zusätzliche Leerzeichen und Zeilenumbrüche, die nicht erforderlich waren, und die Struktur einer Tabelle wird in keiner Weise beibehalten. Diese Fehler können auch bei der Verwendung fortgeschrittener PDF-Bibliotheken auftreten und sind dann möglicherweise schwer zu entdecken. Noch schwieriger wird es, wenn die PDF-Datei eine Kombination aus Text mit zugänglichen zugrunde liegenden Textinformationen und scanartigen Bereichen enthält, d. h. Stellen, an denen Text sichtbar ist, aber keine textlichen Informationen erfasst werden können [25, 26, 27].

3.2.2 PDFMiner

Wenn es um den Umgang mit PDF-Dateien geht, kann auch das Python-Modul PyPDF [29] genutzt werden. Es verfügt nicht nur über einen großen Funktionsumfang, sondern ist auch recht einfach zu bedienen. Wenn es jedoch um die Extraktion von Text geht, ist PDFMiner die genauere und vertrauenswürdigere Option. PDFMiner wurde speziell für das Erkennen und Extrahieren von Text aus PDF-Dokumenten entwickelt. Wenn es um den Umgang mit PDF-Dateien geht, gibt es viele Situationen, in denen eine Bibliothek einer anderen in vielerlei Hinsicht überlegen ist. Neben den bereits erwähnten PDFMiner ist eine Software, die Informationen aus einer PDF-Datei extrahieren und ihre Struktur analysieren kann. Anschließend wandelt es die analysierte Struktur in einfachen Text, XML oder HTML um und unterteilt sie in Absätze, Zeilen und Zeichen. Es unterscheidet sich von anderen Werkzeugen, die mit PDFs verbunden sind, dadurch, dass sein einziger Zweck darin besteht, Textdaten zu erhalten und zu analysieren. PDFMiner ermöglicht es, die genaue Platzierung von Text auf einer Seite zu bestimmen und Informationen über Schriftarten, Zeilen usw. zu erhalten. Das Tool verfügt über einen PDF-Konverter, mit dem die PDF-Dateien in andere Textformen (z. B. HTML oder XML) umgewandelt werden können. Die Textanalyse ist nur eine von vielen möglichen Anwendungen für diesen vielseitigen PDF-Parser.

```
!pip install pdfminer

from io import StringIO
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.pdfpage import PDFPage
from pdfminer.pdfparser import PDFParser

output_string = StringIO()
with open('/content/Intelligent_Text_Extraction_from_PDF_Documents.pdf',
'rb') as in_file:
    parser = PDFParser(in_file)
    doc = PDFDocument(parser)
    rsrcmgr = PDFResourceManager()
    device = TextConverter(rsrcmgr, output_string, laparams=LAParams())
    interpreter = PDFPageInterpreter(rsrcmgr, device)
    for page in PDFPage.create_pages(doc):
        interpreter.process_page(page)

print(output_string.getvalue())
```

Listing 3.3: Python-Beispiel für das Package PDFMiner

Das oben stehende Code-Stück 3.3 erklärt, wie PDFMiner aufgerufen wird und wie es den Text von einer Pdf-Datei (siehe Abbildung 3.1) nach der Untersuchung und Bearbeitung extrahiert und dann das unten stehende Ergebnis gibt. Dafür wurden die dafür erforderlichen Klassen und Objekten aufgerufen, die miteinander verbunden sind. **LAParams** ist das Objekt, das die Layout-Parameter mit einem bestimmten Standardwert enthält. **PDFDocument** speichert die Daten, die durch **PDFParser** aus der Datei geholt werden und **PDFResourceManager** wird verwendet, um gemeinsam genutzte Ressourcen wie Fonts oder Bilder zu speichern während **PDFPageInterpreter** verarbeitet den Seiteninhalt. Ein **String-I-Objekt** enthält die endgültigen Text-Darstellung der PDF-Datei. Die zur Bearbeitung stehende PDF-Datei wird in der Zeile 12 aufgerufen und in den Zeilen 18 und 19 werden den Inhalt jeder Seite der ursprünglichen PDF-Datei verarbeitet. Abschließend wird der gesamten Inhalt der Datei durch **outputstring.getvalue()** in der Zeile 21 geliefert [30].

```
1  ABSTRACT
2  A wrapper is a program that automatically navigates a data structure such as a web site, selecting and extracting the
3  relevant content and delivering it in the form of structured data (such as XML) into databases and other applications.  The
4  Lixto Visual Wrapper [2] allows a user to visually create wrappers to extract data from similarly structured web pages, or
5  from web pages whose content changes over time.   The wrapper is based on user-defined patterns exploiting the
6  hierarchical structure of the HTML source.
7
8  Many commercial applications also require the extraction of data from PDF documents.   There appear to be no
9  general-purpose approaches to fulfil this need.  We are currently investigating PDF data extraction in the NEXTWRAP
10 project.  This paper presents our work in progress, with particular reference to low-level segmentation algorithms.
11
12 KEYWORDS
13 data extraction,  document understanding,  PDF,  XML,  logical structure,  geometric structure,  unstructured
14 documents
15 1.  INTRODUCTION
16 Much of the information on todays web is published in the form of PDF documents.  Documents can be
17 analysed on two levels; the geometric level and the logical level; each with their corresponding structure.
18 Only the geometric structure is explicitly available to us in the PDF format.  The logical structure must be
19 rediscovered by the process of document understanding, which attempts to reverse the document-authoring
20 process by examining the layout conventions that have been used.
21
22 The  Lixto  Visual  Wrapper  [2],  a  product  of  research  at  our  institute,  currently  makes  use  of  the  tree
23 structure inherent in HTML documents to locate relevant data.  Extraction patterns are modelled in the Elog
24 web extraction language which locates an elements position in the HTML parse tree and allows additional
25 constraints  to  be  specified.   As  the  user  is  fully  guided  through  this  process  by  a  visual  interface,  the
26 complexity of the wrapper program is completely hidden.
27 2.  OVERVIEW OF THE PROJECT
28 The first step in the document understanding process is layout analysis or segmentation; to break down the
29 document  into  blocks  that  can  be  said  to  be  atomic,  i.e.  to  constitute  the  smallest  logical  entity  in  the
30 documents  structure.   Typically  these  consist  of  paragraphs,  headlines,  titles  and  captions.   We  have
31 experimented with some algorithms for this process, and they are described in section 3.
32
33 Once this process is complete, we plan to investigate the following methods for extracting the data:
34
35 ontology-based  wrapping:  developing  and  using  a  document-generic  ontology  to  represent  the
36 relationships between the various objects in the document.  This makes it possible to express several
37 types of relationships between the objects, rather than just a simple hierarchy.  Extraction will take
38 ⋆ This work is funded in part by the Austrian Federal Ministry for Transport, Innovation and Technology under the FIT-IT
   programme
39 line as a part of the NEXTWRAP project.
40 IADIS International Conference on WWW/Internet 2005371
41
42 place using established tools for ontological reasoning.  This approach also can naturally be extended
43 to ontologies that are specific to a particular document class.
44 conversion to a structured format: the use of rules, expressed in a logical or procedural language,
45 to  find  the  logical  structure  of  the  document.   This  will  allow  us  to  represent  the  content  in  a
46 hierarchical structure such as XML.  Thus the current techniques within the Lixto suite for wrapping
47 from HTML could be adapted to work with this format.
48 spatial reasoning: the process of extracting document objects according to how they occur (or relate
49 to other objects) on the physical page, without any higher-level document understanding taking place.
50 Sometimes it may be simpler for a wrapper designer to sidestep the document understanding process
51 and extract objects simply based on their physical location.
52
53 Figure 1 illustrates this process.
```

Listing 3.4: Extrahierter Text von einer PDF-Datei durch PDFMiner

3.2.3 PyMuPDF

Die Bibliotheken PDFMiner und PyPDF2 sind beide vollständig in Python geschrieben. PyMuPDF hingegen basiert auf MuPDF, das ein kleiner, aber funktionsreicher PDF-Reader ist. Große Vorteile von PyMuPDF ergeben sich beim Umgang mit komplexen PDFs. Darüber hinaus weist PyMuPDF in einer Reihe von Benchmarks einen großen Geschwindigkeitsvorteil gegenüber PDFMiner und PyPDF2 auf. Artifex ist das Unternehmen, das das Produkt PyMuPDF entwickelt hat. Die Software kann kostenlos verwendet werden und wird unter einer Open-Source-Freeware-Lizenz vertrieben. PyMuPDF ist eine Programmierbibliothek, die die Programmiersprache Python verwendet und ihre Homepage ist auf GitHub [1] zu finden. Die Textextraktion ist nur eine vieler Möglichkeiten, die PyMuPDF bietet, und es unterstützt eine große Anzahl der Funktionen, die im Programm MuPDF verfügbar sind. Die Textextraktion ist, wie alle anderen Funktionen im PyMuPDF auch, für ihre hohe Leistung und hervorragende Rendering-Qualität bekannt. Dieses Paket kümmert sich automatisch um die Textbereinigung der Vorverarbeitung, da es den Text bereinigt, indem es alle unerwünschten Leerzeichen aus dem Text entfernt. Die Struktur des Dokuments in seiner ursprünglichen Form bleibt bei diesem Vorgang erhalten. Das Problem der erfolgreichen Extraktion von Tabellen in ihrem ursprünglichen Format besteht jedoch nach wie vor, was auch bei anderen Paketen der Fall ist. Wenn die Informationen in den Tabellen beibehalten werden sollen, muss es auf ein anderes Paket umstiegen werden [31, 27].

Das folgende Listing 3.5 zeigt ein Beispiel für einen einfachen Code, der verwendet werden kann, um den Klartext zu erhalten:

```
!pip install  pymupdf
import fitz
from fitz import TextPage
#Beispiel 1
filepath = "/content/Intelligent_Text_Extraction_from_PDF_Documents.pdf"
doc= fitz.open(filepath )
for page in doc:
    text = page.get_text("text")
    print(text)
#Beispiel 2
FILE_PATH = "/content/Intelligent_Text_Extraction_from_PDF_Documents.pdf"
pdf = fitz.open(FILE_PATH)
pdf.page_count
page_content = pdf.load_page(1).get_text().replace("\t", " ")
print(pdf.page_count,"pages","\n",page_content)
```

Listing 3.5: Python Beispiel für das Package PyMuPDF

[1] https://github.com/pymupdf/PyMuPDF, zuletzt aufgerufen am 24.11.2022

Im obenstehenden Beispiel wurde **PyMuPDF** installiert und das Package **fitz** (Zeile 2) aufgerufen, durch das die Eingabe-Datei aufgerufen und geöffnet werden kann. Von fitz wird **TextPage** importiert (Zeile 3), um deren Funktionen für Lesen und Extrahieren den Seiteninhalt zu benutzen. Im ersten Beispiel wurde der ganze Inhalt (siehe Listing 3.6) durch die **for**-Schleife entnommen (Zeile 7) und im zweiten Beispiel wurde nur der Inhalt von einer bestimmten Seite rausgeholt (siehe Listing 3.7) und zusätzlich die Anzahl der Datei-Seiten (Zeile 13 und 14) ausgegeben.

```
1  See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/4242676
2  Intelligent Text Extraction from PDF Documents
3  Conference Paper    December 2005
4  DOI: 10.1109/CIMCA.2005.1631436   Source   : IEEE Xplore
5  CITATIONS
6  19
7  READS
8  2,059
9  2 authors:
10 Some of the authors of this publication are also working on these related projects:
11 SEMAMO View project
12 Tamir Hassan
13 University Konstanz
14 17 PUBLICATIONS  235 CITATIONS
15 SEE PROFILE
16 Robert Baumgartner
17 TU Wien
18 62 PUBLICATIONS   1,939 CITATIONS
19 SEE PROFILE
20 All content following this page was uploaded by Robert Baumgartner on 16 March 2014.
21 The user has requested enhancement of the downloaded file.
22 INTELLIGENT TEXT EXTRACTION FROM PDF*
23 Tamir Hassan, Robert Baumgartner
24 Database & Artificial Intelligence Group, Vienna University of Technology, Austria
25  ABSTRACT
26 A wrapper is a program that automatically navigates a data structure such as a web site, selecting and extracting the
27 relevant content and delivering it in the form of structured data (such as XML) into databases and other applications.  The
28 Lixto Visual Wrapper [2] allows a user to visually create wrappers to extract data from similarly structured web pages, or
29 from web pages whose content changes over time.  The wrapper is based on user-defined patterns exploiting the
30 hierarchical structure of the HTML source.
31 Many commercial applications also require the extraction of data from PDF documents.  There appear to be no
32 general-purpose approaches to fulfil this need.  We are currently investigating PDF data extraction in the NEXTWRAP
33 project.  This paper presents our work in progress, with particular reference to low-level segmentation algorithms.
34 KEYWORDS
35 data extraction,  document understanding,  PDF,  XML,  logical structure,  geometric structure,  unstructured
36 documents
37 1. INTRODUCTION
38 Much of the information on t o d a y s web is published in the form of PDF documents.  Documents can be
39 analysed on two levels; the geometric level and the logical level; each with their corresponding structure.
40 Only the geometric structure is explicitly available to us in the PDF format.  The logical structure must be
41 rediscovered by the process of document understanding, which attempts to reverse the document-authoring
42 process by examining the layout conventions that have been used.
43 The Lixto Visual Wrapper [2], a product of research at our institute, currently makes use of the tree
```

Listing 3.6: extrahierter Text durch PyMuPDF-Code-Beispiel 1

```
4 pages
 INTELLIGENT TEXT EXTRACTION FROM PDF*
Tamir Hassan, Robert Baumgartner
Database & Artificial Intelligence Group, Vienna University of Technology, Austria

ABSTRACT
A wrapper is a program that automatically navigates a data structure such as a web site, selecting and extracting the
relevant content and delivering it in the form of structured data (such as XML) into databases and other applications.  The
Lixto Visual Wrapper [2] allows a user to visually create wrappers to extract data from similarly structured web pages, or
from web pages whose content changes over time.  The wrapper is based on user-defined patterns exploiting the
hierarchical structure of the HTML source.
Many commercial applications also require the extraction of data from PDF documents.  There appear to be no
general-purpose approaches to fulfil this need.  We are currently investigating PDF data extraction in the NEXTWRAP
project.  This paper presents our work in progress, with particular reference to low-level segmentation algorithms.
KEYWORDS
data extraction,  document understanding,  PDF,  XML,  logical structure,  geometric structure,  unstructured
documents
1. INTRODUCTION
Much of the information on t o d a y s web is published in the form of PDF documents.  Documents can be
analysed on two levels; the geometric level and the logical level; each with their corresponding structure.
Only the geometric structure is explicitly available to us in the PDF format.  The logical structure must be
rediscovered by the process of document understanding, which attempts to reverse the document-authoring
process by examining the layout conventions that have been used.
The Lixto Visual Wrapper [2], a product of research at our institute, currently makes use of the tree
structure inherent in HTML documents to locate relevant data.  Extraction patterns are modelled in the Elog
web extraction language which locates an e l e m e n t s position in the HTML parse tree and allows additional
constraints to be specified.  As the user is fully guided through this process by a visual interface, the
complexity of the wrapper program is completely hidden.
2. OVERVIEW OF THE PROJECT
The first step in the document understanding process is layout analysis or segmentation; to break down the
document into blocks that can be said to be atomic, i.e. to constitute the smallest logical entity in the
 d o c u m e n t s structure.  Typically these consist of paragraphs, headlines, titles and captions.  We have
experimented with some algorithms for this process, and they are described in section 3.
Once this process is complete, we plan to investigate the following methods for extracting the data:
```

Listing 3.7: Extrahierter Text durch PyMuPDF-Code-Beispiel 2

Die vorangehenden Listings 3.6 und 3.7 zeigen, dass im ersten Szenario alle Textinformationen, die nicht in Textabschnitten enthalten sind, von der ersten Seite der Datei extrahiert wurden. Danach wurde der Prozess zur zweiten Seite der Datei fortgesetzt, wo der Text auf dieser Seite extrahiert wurde, und so weiter, bis zur letzten Seite der Datei. Im zweiten Szenario wurde die Anzahl der in der Datei enthaltenen Seiten angegeben, und der Text wurde einer bestimmten Seite entnommen. Daher kann behauptet werden, dass es einen gewissen Spielraum gibt, wie diese Bibliothek entsprechend den Präferenzen des Benutzers genutzt werden kann. Wenn man sich den Text ansieht, der aus jedem Dokument abgerufen werden kann, scheint dies auf eine ähnliche Weise wie bei PDFMiner zu geschehen.

3.2.4 Apache Tika

Apache Tika ist ein Werkzeug zur Inhaltsanalyse, das Dokumenttypen identifizieren und relevante Informationen aus einer Vielzahl von Dateitypen extrahieren kann. Excel-Tabellen, Word-Dokumente, Bilddateien, PDFs– sie alle fallen in den Bereich des Werkzeugkastens, mit dem sich Metadaten und Text extrahieren lassen. Tika ist für eine Vielzahl von Anwendungen hilfreich, darunter die Indexierung in Suchmaschinen, die Inhaltsanalyse, die Übersetzung und vieles mehr, da es all diese verschiedenen Dateiformate über eine einzige Schnittstelle lesen und verarbeiten kann. Es ist erwähnenswert, dass Tika-Python eine Python-Anbindung an die Apache Tika REST-Dienste ist, die es ermöglicht, Tika nativ in der Sprache Python aufzurufen. Tika ist ein Projekt der Apache Software Foundation und war früher ein Unterprojekt von Apache Lucene und ist ein Java-basiertes Paket [26, 32, 33].

```
1  !pip install tika
2
3  from tika import parser as p
4
5  def get_data_from_given_path(file_path):
6      results = p.from_file(file_path)
7      return results
8
9  pdf_file_path="/content/Intelligent_Text_Extraction_from_PDF_Documents.pdf"
10 results = get_data_from_given_path(pdf_file_path)
11 print(results["content"].strip())
```

Listing 3.8: Beispiel für das Toolkit Apache Tika

Um den Inhalt aus der aufgerufenen PDF-Datei (Zeile 9) zu extrahieren, muss die Methode **from_file()** (Zeile 6) (siehe Zeile 6 in 3.8) vom **Parser _Objekt** (Zeile 3) verwendet werden. Durch Anwendung der Funktion **get _data _from _given _path** (Zeile 5) extrahiert und durch die Anweisung **print** gezeigt.

```
ABSTRACT

A wrapper is a program that automatically navigates a data structure such as a web site, selecting and extracting the
relevant content and delivering it in the form of structured data (such as XML) into databases and other applications.  The
Lixto Visual Wrapper [2] allows a user to visually create wrappers to extract data from similarly structured web pages, or
from web pages whose content changes over time.  The wrapper is based on user-defined patterns exploiting the
hierarchical structure of the HTML source.

Many commercial applications also require the extraction of data from PDF documents.  There appear to be no
general-purpose approaches to fulfil this need.  We are currently investigating PDF data extraction in the NEXTWRAP
project.  This paper presents our work in progress, with particular reference to low-level segmentation algorithms.

KEYWORDS

data extraction,  document understanding,  PDF,  XML,  logical structure,  geometric structure,  unstructured
documents

1. INTRODUCTION

Much of the information on t o d a y s web is published in the form of PDF documents.  Documents can be
analysed on two levels; the geometric level and the logical level; each with their corresponding structure.
Only the geometric structure is explicitly available to us in the PDF format.  The logical structure must be
rediscovered by the process of document understanding, which attempts to reverse the document-authoring
process by examining the layout conventions that have been used.

The Lixto Visual Wrapper [2], a product of research at our institute, currently makes use of the tree
structure inherent in HTML documents to locate relevant data.  Extraction patterns are modelled in the Elog
web extraction language which locates an e l e m e n t s position in the HTML parse tree and allows additional
constraints to be specified.  As the user is fully guided through this process by a visual interface, the
complexity of the wrapper program is completely hidden.

2. OVERVIEW OF THE PROJECT

The first step in the document understanding process is layout analysis or segmentation; to break down the
document into blocks that can be said to be atomic, i.e. to constitute the smallest logical entity in the
 d o c u m e n t s structure.  Typically these consist of paragraphs, headlines, titles and captions.  We have
experimented with some algorithms for this process, and they are described in section 3.

Once this process is complete, we plan to investigate the following methods for extracting the data:

ontology-based wrapping: developing and using a document-generic ontology to represent the
relationships between the various objects in the document.  This makes it possible to express several
types of relationships between the objects, rather than just a simple hierarchy.  Extraction will take


This work is funded in part by the Austrian Federal Ministry for Transport, Technology under the FIT-IT programme
line as a part of the NEXTWRAP project.

IADIS International Conference on WWW/Internet 2005

371




place using established tools for ontological reasoning.  This approach also can naturally be extended
to ontologies that are specific to a particular document class.

    conversion to a structured format: the use of rules, expressed in a logical or procedural language,
to find the logical structure of the document.  This will allow us to represent the content in a
hierarchical structure such as XML.  Thus the current techniques within the Lixto suite for wrapping
from HTML could be adapted to work with this format.

    spatial reasoning: the process of extracting document objects according to how they occur (or relate
to other objects) on the physical page, without any higher-level document understanding taking place.
Sometimes it may be simpler for a wrapper designer to sidestep the document understanding process
and extract objects simply based on their physical location.
```

Listing 3.9: Extrahierter Content durch das Toolkit Apache Tika

Nach Verwendung einer einfachen Vorlage aus dem Tika-Programm zeigt das vorherige Ergebnis einen kleinen textlichen Teil von einem großen Text, der aus einer PDF-Datei extrahiert wurde. Daraus können wir ersehen, dass die Struktur dieses Textteils in einer Weise organisiert ist, die der Struktur der Primärdatei ähnelt. Da diese Bibliothek in der Lage ist, eine gründlichere Analyse der Datei durchzuführen als ihre Vorgänger, sind die daraus extrahierten Informationen auch präziser. Diese Fähigkeit verleiht dieser Bibliothek einen Vorteil gegenüber ihren Konkurrenten.

3.2.5 tabula-py

tabula-py ist ein Paket, mit dem PDFs nicht nur ausgelesen, sondern auch direkt in CSV-Dateien umgewandelt werden können. Dies ist eine äußerst nützliche Kombination von Funktionen. Es ist ein in Python geschriebener Wrapper für tabula-java [34], der Tabellen aus PDF-Dateien lesen kann. Tabellen können aus PDF-Dokumenten gelesen werden, und die aus diesen Tabellen extrahierten Informationen werden in einem Python-DataFrame gespeichert. Dieser DataFrame kann dann später in ein CSV-, Excel- oder JSON-Dateiformat konvertiert werden. tabula-py ist hilfreich für die Durchführung von Analysen in einer eher informellen Umgebung unter Verwendung von Jupyter-Notebook oder Google Collaboratory. Beim Versuch, den gesamten Text aus PDFs zu extrahieren, kann es von Vorteil sein, Tabula _py in Verbindung mit den anderen im vorherigen Satz beschriebenen Tools zu verwenden [34, 35].

Im Folgenden zeigen die Listings 3.10 und 3.11 Beispiele für die Verwendung von tabula-py für die Extraktion der Tabellen in einer zweiseitigen PDF-Datei (siehe Abbildung 3.2) und wie die Ergebnisse (siehe Abbildungen 3.3 und 3.4) aussehen können.

```
!pip install tabula-py

import tabula

pdf_path = "/content/ast_sci_data_tables_sample-1.pdf"
#dfs: Liste von Tabellen in der zweiten Seite der Pdf datei
dfs = tabula.read_pdf(pdf_path, pages='2')

print(len(dfs)) #Lange der Liste

print(dfs[0])  #erste Tabelle in der Liste
#erste Tabelle in der zweiten Seite in csv Format
dfs[0].to_csv("first_table.csv")

#print(dfs)
for i in range(len(dfs)):   #extrahieren aller Tabellen in der zweiten Seite
  print(dfs[i])

#alle Tabellen in der Datei in csv Format
tabula.convert_into(pdf_path, "all_table.csv", output_format="csv", pages='all')
```

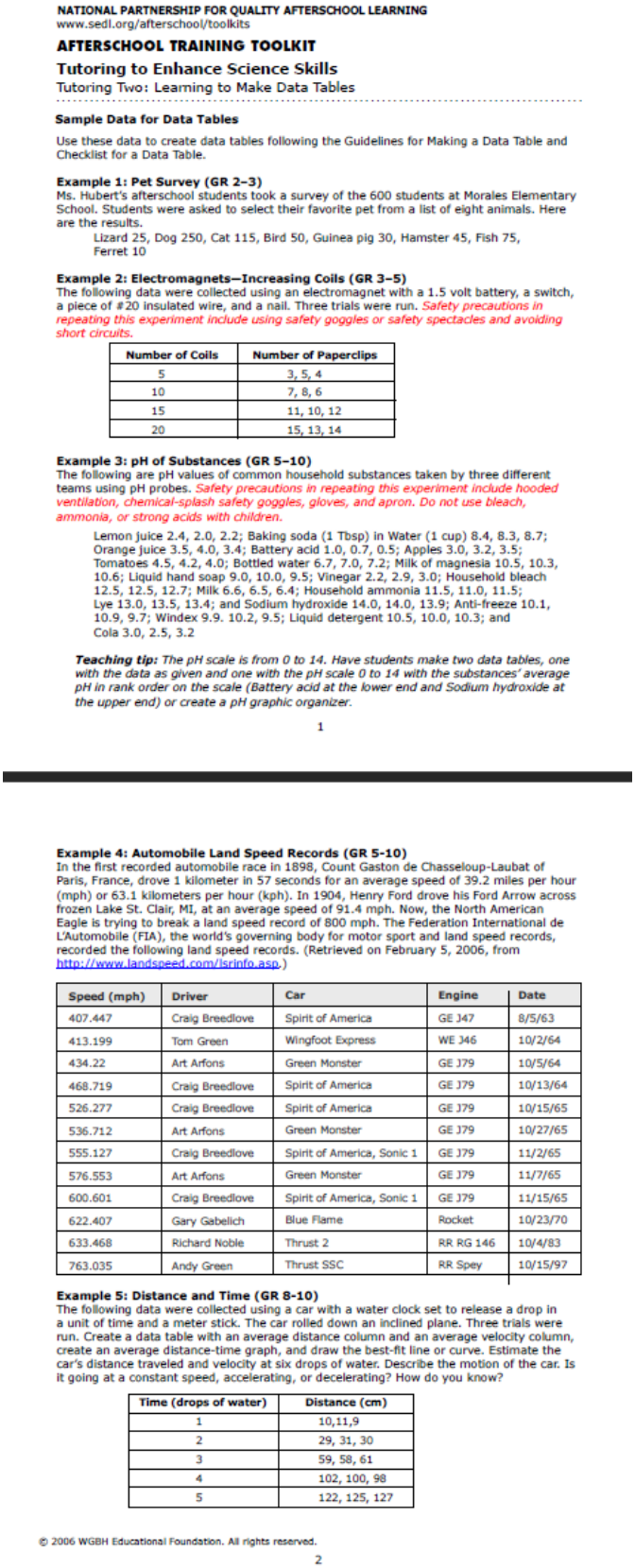
Listing 3.10: Beispiel für tabula-py

```
dfs = tabula.read_pdf(pdf_path, pages='all')

print(len(dfs))
print(dfs)

#Jede Tabelle in der Datei in einer getrennten CSV Datei
for i in range(len(dfs)):
    dfs[i].to_csv(f"table_all_{i}.csv")
```

Listing 3.11: Beispiel für tabula-py



NATIONAL PARTNERSHIP FOR QUALITY AFTERSCHOOL LEARNING
www.sedl.org/afterschool/toolkits

AFTERSCHOOL TRAINING TOOLKIT

Tutoring to Enhance Science Skills
Tutoring Two: Learning to Make Data Tables

Sample Data for Data Tables

Use these data to create data tables following the Guidelines for Making a Data Table and Checklist for a Data Table.

Example 1: Pet Survey (GR 2–3)
Ms. Hubert's afterschool students took a survey of the 600 students at Morales Elementary School. Students were asked to select their favorite pet from a list of eight animals. Here are the results.

Lizard 25, Dog 250, Cat 115, Bird 50, Guinea pig 30, Hamster 45, Fish 75, Ferret 10

Example 2: Electromagnets—Increasing Coils (GR 3–5)
The following data were collected using an electromagnet with a 1.5 volt battery, a switch, a piece of #20 insulated wire, and a nail. Three trials were run. *Safety precautions in repeating this experiment include using safety goggles or safety spectacles and avoiding short circuits.*

Number of Coils	Number of Paperclips
5	3, 5, 4
10	7, 8, 6
15	11, 10, 12
20	15, 13, 14

Example 3: pH of Substances (GR 5–10)
The following are pH values of common household substances taken by three different teams using pH probes. *Safety precautions in repeating this experiment include hooded ventilation, chemical-splash safety goggles, gloves, and apron. Do not use bleach, ammonia, or strong acids with children.*

Lemon juice 2.4, 2.0, 2.2; Baking soda (1 Tbsp) in Water (1 cup) 8.4, 8.3, 8.7; Orange juice 3.5, 4.0, 3.4; Battery acid 1.0, 0.7, 0.5; Apples 3.0, 3.2, 3.5; Tomatoes 4.5, 4.2, 4.0; Bottled water 6.7, 7.0, 7.2; Milk of magnesia 10.5, 10.3, 10.6; Liquid hand soap 9.0, 10.0, 9.5; Vinegar 2.2, 2.9, 3.0; Household bleach 12.5, 12.5, 12.7; Milk 6.6, 6.5, 6.4; Household ammonia 11.5, 11.0, 11.5; Lye 13.0, 13.5, 13.4; and Sodium hydroxide 14.0, 14.0, 13.9; Anti-freeze 10.1, 10.9, 9.7; Windex 9.9. 10.2, 9.5; Liquid detergent 10.5, 10.0, 10.3; and Cola 3.0, 2.5, 3.2

Teaching tip: *The pH scale is from 0 to 14. Have students make two data tables, one with the data as given and one with the pH scale 0 to 14 with the substances' average pH in rank order on the scale (Battery acid at the lower end and Sodium hydroxide at the upper end) or create a pH graphic organizer.*

1

Example 4: Automobile Land Speed Records (GR 5-10)
In the first recorded automobile race in 1898, Count Gaston de Chasseloup-Laubat of Paris, France, drove 1 kilometer in 57 seconds for an average speed of 39.2 miles per hour (mph) or 63.1 kilometers per hour (kph). In 1904, Henry Ford drove his Ford Arrow across frozen Lake St. Clair, MI, at an average speed of 91.4 mph. Now, the North American Eagle is trying to break a land speed record of 800 mph. The Federation International de L'Automobile (FIA), the world's governing body for motor sport and land speed records, recorded the following land speed records. (Retrieved on February 5, 2006, from http://www.landspeed.com/lsrinfo.asp.)

Speed (mph)	Driver	Car	Engine	Date
407.447	Craig Breedlove	Spirit of America	GE J47	8/5/63
413.199	Tom Green	Wingfoot Express	WE J46	10/2/64
434.22	Art Arfons	Green Monster	GE J79	10/5/64
468.719	Craig Breedlove	Spirit of America	GE J79	10/13/64
526.277	Craig Breedlove	Spirit of America	GE J79	10/15/65
536.712	Art Arfons	Green Monster	GE J79	10/27/65
555.127	Craig Breedlove	Spirit of America, Sonic 1	GE J79	11/2/65
576.553	Art Arfons	Green Monster	GE J79	11/7/65
600.601	Craig Breedlove	Spirit of America, Sonic 1	GE J79	11/15/65
622.407	Gary Gabelich	Blue Flame	Rocket	10/23/70
633.468	Richard Noble	Thrust 2	RR RG 146	10/4/83
763.035	Andy Green	Thrust SSC	RR Spey	10/15/97

Example 5: Distance and Time (GR 8-10)
The following data were collected using a car with a water clock set to release a drop in a unit of time and a meter stick. The car rolled down an inclined plane. Three trials were run. Create a data table with an average distance column and an average velocity column, create an average distance-time graph, and draw the best-fit line or curve. Estimate the car's distance traveled and velocity at six drops of water. Describe the motion of the car. Is it going at a constant speed, accelerating, or decelerating? How do you know?

Time (drops of water)	Distance (cm)
1	10,11,9
2	29, 31, 30
3	59, 58, 61
4	102, 100, 98
5	122, 125, 127

2

Abbildung 3.2: Eine zu analysierenden PDF-Datei durch tabula-py [36].

	A	B	C	D	E
1	Number of Coils	Number of Paperclips			
2	5	3, 5, 4			
3	10	7, 8, 6			
4	15	11, 10, 12			
5	20	15, 13, 14			
6	Speed (mph)	Driver	Car	Engine	Date
7	407.447	Craig Breedlove	Spirit of America	GE J47	8/5/1963
8	413.199	Tom Green	Wingfoot Express	WE J46	10/2/1964
9	434.22	Art Arfons	Green Monster	GE J79	10/5/1964
10	468.719	Craig Breedlove	Spirit of America	GE J79	10/13/1964
11	526.277	Craig Breedlove	Spirit of America	GE J79	10/15/1965
12	536.712	Art Arfons	Green Monster	GE J79	10/27/1965
13	555.127	Craig Breedlove	Spirit of America, Sonic 1	GE J79	11/2/1965
14	576.553	Art Arfons	Green Monster	GE J79	11/7/1965
15	600.601	Craig Breedlove	Spirit of America, Sonic 1	GE J79	11/15/1965
16	622.407	Gary Gabelich	Blue Flame	Rocket	10/23/1970
17	633.468	Richard Noble	Thrust 2	RR RG 146	10/4/1983
18	763.035	Andy Green	Thrust SSC	RR Spey	10/15/1997
19	Time (drops of water)	Distance (cm)			
20	1	10,11,9			
21	2	29, 31, 30			
22	3	59, 58, 61			
23	4	102, 100, 98			
24	5	122, 125, 127			

Abbildung 3.3: Die 3 extrahierten Tabellen von einer PDF-Datei durch tabula-py

first_table.csv ×

1 to 12 of 12 entries Filter

	Speed (mph)	Driver	Car	Engine	Date
0	407.447	Craig Breedlove	Spirit of America	GE J47	8/5/63
1	413.199	Tom Green	Wingfoot Express	WE J46	10/2/64
2	434.22	Art Arfons	Green Monster	GE J79	10/5/64
3	468.719	Craig Breedlove	Spirit of America	GE J79	10/13/64
4	526.277	Craig Breedlove	Spirit of America	GE J79	10/15/65
5	536.712	Art Arfons	Green Monster	GE J79	10/27/65
6	555.127	Craig Breedlove	Spirit of America, Sonic 1	GE J79	11/2/65
7	576.553	Art Arfons	Green Monster	GE J79	11/7/65
8	600.601	Craig Breedlove	Spirit of America, Sonic 1	GE J79	11/15/65
9	622.407	Gary Gabelich	Blue Flame	Rocket	10/23/70
10	633.468	Richard Noble	Thrust 2	RR RG 146	10/4/83
11	763.035	Andy Green	Thrust SSC	RR Spey	10/15/97

Show 25 per page

Abbildung 3.4: Die extrahierte Tabelle von einer PDF-Datei durch Tabula-py

Die zuvor besprochenen Softwarebibliotheken gehören zu den bedeutendsten und bekanntesten der Welt, wenn es um die Extraktion von Textinformationen und Inhalten aus Dateien im Allgemeinen und aus PDF-Dateien im Besonderen geht. In den vorangegangenen Abschnitten wurden diese Bibliotheken und ihre Anwendung auf einfache Dateibeispiele und die sich daraus ergebenden Ausgaben behandelt. Es gibt jedoch eine große Anzahl anderer Bibliotheken, die praktisch den gleichen Zweck erfüllen, die kurz beschrieben werden sollen.

3.2.6 PdfBox

PdfBox[2] ist eine PDF-Bibliothek, die weit verbreitet ist und von Apache vertrieben wird. Sie ist in der Lage, jede beliebige PDF-Datei in einfachen Text zu konvertieren. Sie ist jedoch nicht in der Lage, Ligaturen und Zeichen mit diakritischen Zeichen (z.B. (´), (`) und (ˆ) oder die deutschen Umlaute ä, ö, ü) zu verarbeiten. Es ist jedoch nicht in der Lage, die Grenzen von Absätzen oder die semantischen Funktionen, die sie erfüllen, zu erkennen [1].

3.2.7 pdftotext

pdftotext[3] ist das PDF-Extraktionsprogramm, mit dem jede PDF-Datei in eine einfache Textdatei umgewandelt wird, ohne jedoch die Grenzen zwischen den Absätzen, die semantischen Rollen des Dokuments oder sogar den Textkörper selbst zu bestimmen[1].

3.2.8 Camelot

Camelot[4] ist ein Python-Paket, mit dem sich einfach Datentabellen aus PDF-Dokumenten extrahieren lassen. Allerdings ist diese Bibliothek nur mit PDFs kompatibel, die Text enthalten, und nicht mit gescannten PDFs. Zur Verbesserung der Extraktion von Daten aus Tabellen stehen in Camelot zahlreiche Optionen zur Verfügung, die je nach Bedarf angepasst werden können. Im Vergleich zu den anderen derzeit verfügbaren Bibliotheken bietet dies ein höheres Maß an Kontrolle über den Extraktionsprozess [37].

[2] `https://pypi.org/project/python-pdfbox/`, zuletzt aufgerufen am 4.12.2022
[3] `https://pypi.org/project/pdftotext/`, zuletzt aufgerufen am 4.12.2022
[4] `https://pypi.org/project/camelot-py/`, zuletzt aufgerufen am 4.12.2022

3.2.9 PyPDF4

PyPDF4 ist eine PDF-Bibliothek, die vollständig in Python geschrieben ist und die Seiten von PDF-Dateien ändern sowie diese teilen, zusammenführen, beschneiden und kombinieren kann. PDF-Dateien können mit diesem Tool auch mit einem Passwortschutz geschützt, mit Anzeigeoptionen und beliebigen Daten angereichert werden. Es ist in der Lage, Text und Informationen aus PDFs zu extrahieren und viele Dateien zu einer einzigen zusammenzufassen [38].

3.3 Vergleich zwischen Python-Bibliotheken für die PDF-Inhalte-Extraktion

Das Extrahieren von Informationen aus PDF-Dateien ist selbst mithilfe von Spezialbibliotheken nicht immer einfach und manchmal auch fehleranfällig. Das liegt daran, dass PDF-Dateien eine Reihe von Hindernissen enthalten. Um bessere Ergebnisse bei der Extraktion von Informationen zu erzielen, müssen diese Bibliotheken innerhalb bestimmter Algorithmen und Prozesse verwendet werden. In den vorangegangenen Abschnitten wurden verschiedene Python-Bibliotheken und ihre Funktionen sowie die Art und Weise, in der diese Bibliotheken zur Extraktion von Informationen aus PDF-Dateien verwendet werden können, erörtert. Die folgende Tabelle 3.1 vergleicht diese Bibliotheken. Wie aus der Tabelle hervorgeht, ist es nicht einfach, eine einzige Bibliothek zu haben, die alle Aufgaben zum Extrahieren von Inhalten aus Dateien erfüllt. Daher ist es notwendig, mehrere Bibliotheken in eine spezifische Software zu integrieren und sie zusammen zu verwenden, um eine korrekte und klare Extraktion der textuellen und nicht-textuellen Inhalte bei der Arbeit mit Dateien des Formats zu erreichen.

	Extraktion der Texte	Extraktion der Bilder	Extraktion der Tabellen	Extraktion der Kopf- und Fußzeilen
PyPDF2	✓	✓	✗	✓
PDFMiner	✓	✓	✗	✓
PyMuPDF	✓	✓	✗	✓
Python wrapper für Apache Tika	✓	✓	✗	✓
tabula-py	✗	✗	✓	✗
Camelot	✗	✗	✓	✗

Tabelle 3.1: Python-Bibliotheken für die Extraktion von PDF-Inhalten.

Abschließend stellt das Diagramm[5] in Abbildung 3.5 einen umfassenden Plan für die Arbeit mit PDF-Dateien dar, wobei es zeigt, dass Python je nach Zielsetzung eine Vielzahl von Optionen bietet. Diese Optionen können zum Lesen und Verarbeiten von Daten sowie zum Extrahieren von Texten und Tabellen aus Dateien verwendet werden, oder sie können zum Erstellen neuer Dateien im PDF-Dateiformat verwendet werden [39].

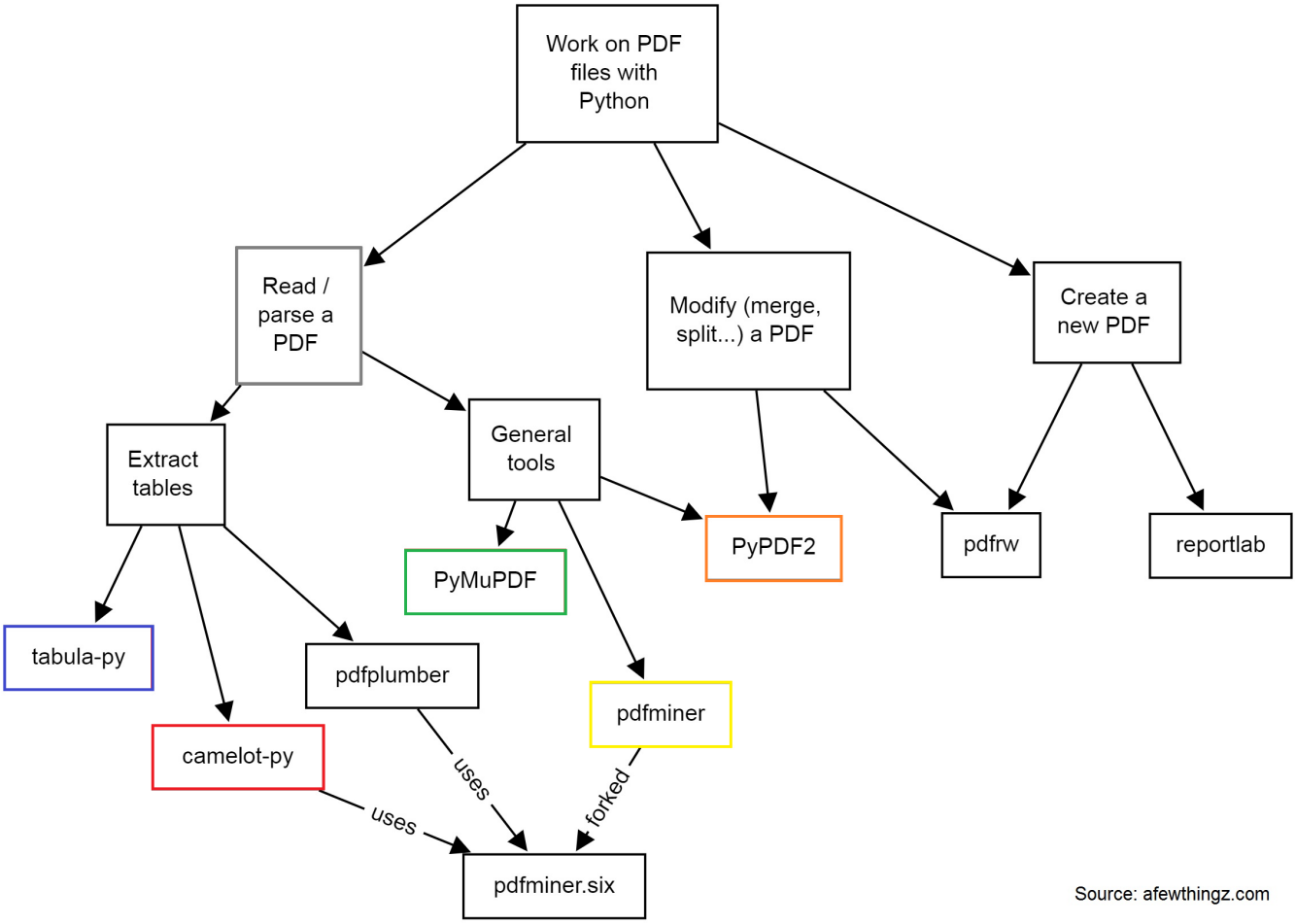


Abbildung 3.5: Bearbeitung von PDF-Dateien mit Python-Bibliotheken [39]

[5] https://afewthingz.com/, zuletzt aufgerufen am 31.12.2022

Kapitel 4

Konzeption des Verfahrens

In diesem Kapitel wird beschrieben, wie das Hauptziel dieser Arbeit erreicht wird, und es bildet den Schwerpunkt dieses Kapitels. Nachdem in den vorangegangenen Abschnitten PDF-Dateien und die Softwarebibliotheken erläutert wurden, die zur Extraktion von Textinformationen aus diesen Dateien verwendet werden, wird in diesem Abschnitt das Konzept erläutert, das zur Analyse der verschiedenen PDF-Dateien und zur Extraktion der darin enthaltenen Texte aus Absätzen, Informationsfeldern, Bildkommentaren und Tabellen verwendet wird.

In der Abbildung 4.1 ist der grobe Ablauf des Tools P2X **(PDF to XML)** als Ablaufdiagramm dargestellt. Mit dem P2X werden die PDF-Dateien verarbeitet und deren textliche Inhalte extrahiert und dann in XML-Dateien umgewandelt und gespeichert. Der Benutzer bereitet die Eingaben für das Tool als PDF-Dateien vor, die er analysieren und aus denen er den Textinhalt extrahieren möchte. Diese Eingaben werden analysiert und erforderliche Elemente werden identifiziert und dann extrahiert, wobei unerwünschte Elemente zurückbleiben. Diese extrahierten Elemente werden gefiltert und separat gespeichert, die später an bestimmten Stellen in der Ausgabedatei hinzugefügt werden. Wenn das Tool die Analyse aller Seiten der Eingabedatei beendet hat, wird eine XML-Ausgabedatei mit strukturiertem Inhalt der Textinformationen in der Eingabedatei erhalten.

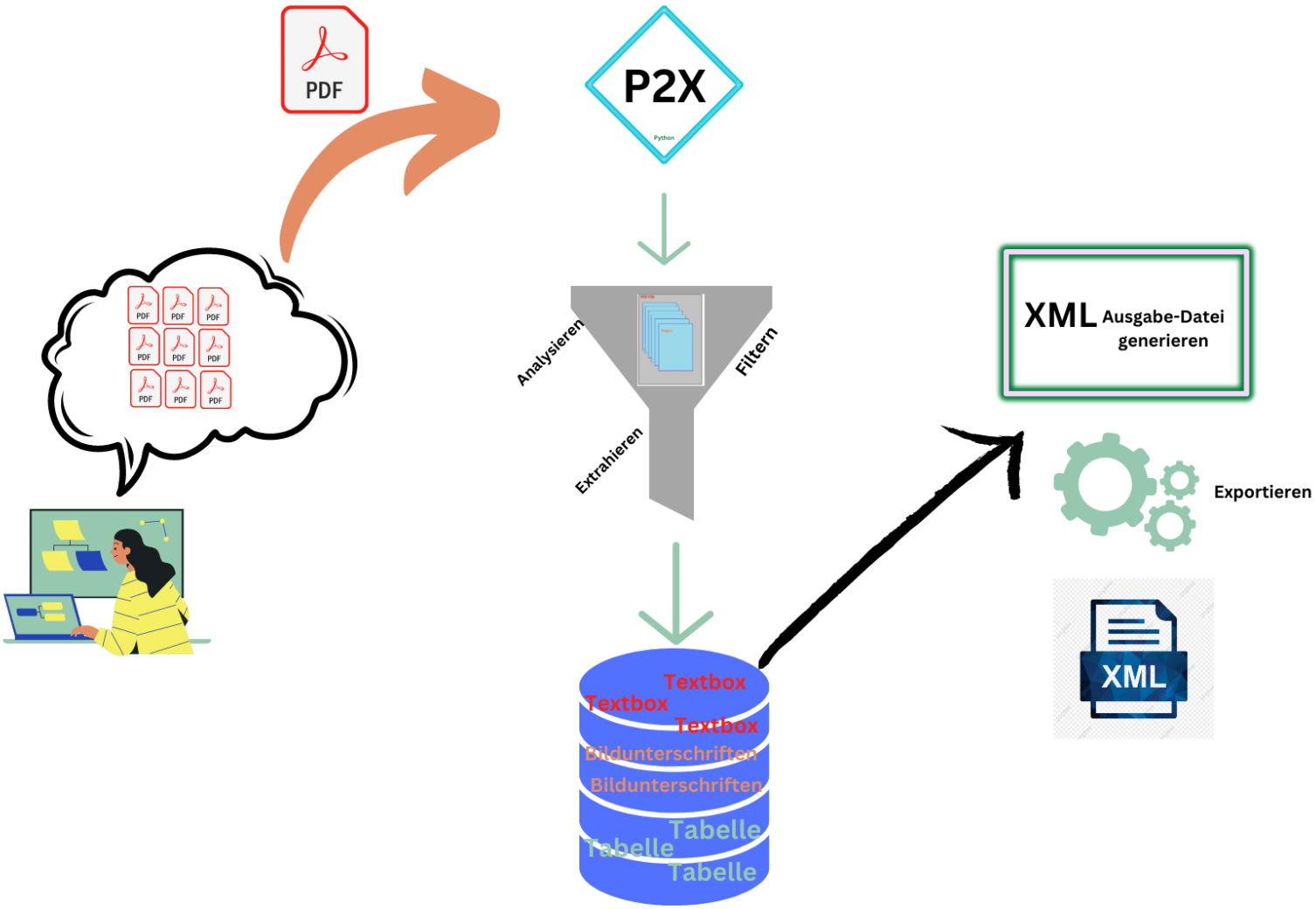


Abbildung 4.1: Ablaufdiagramm des Tools.

Konzeptschritte:

1. Alle Seiten der Datei analysieren
2. Identifizierung der Elemente auf jeder Seite und Angabe ihres Typs
3. Extrahieren ausgewählter Elemente in ihre eigenen Tags in der Ausgabedatei

All dies wird auf den nächsten Seiten erklärt.

4.1 Schritt 1: Lesen der Datei und Analyse der Seiten

Damit das Hauptziel dieses Projekts erreicht werden kann, muss das P2X-Tool effektiv und angemessen funktionieren. Folglich müssen die Dateien, die in diesem Verfahren verwendet werden, mit dem Tool kompatibel sein. Die PDF-Dateien, die sich als Eingabe für das Inhaltsextraktionstool eignen, sind diejenigen, die das grundlegende Format der PDF-Dateien aufweisen, das in Abschnitt 2.3.4 näher erläutert wurde. Dies ist notwendig, damit der Algorithmus des Tools auf die betreffenden PDF-Dateien angewendet werden kann und die gewünschten Ergebnisse liefert. Andererseits wird die Verwendung von gescannten PDF-Dateien nicht das erreichen, was man mit dem Werkzeug erreichen möchte, da das Werkzeug nicht in der Lage ist, PDF-Dateien zu verarbeiten, deren Inhalt nicht aus textlichen Komponenten, sondern aus gescannten Inhalten besteht.

Der erste Schritt besteht darin, dass der Benutzer die richtigen PDF-Dateien für das Programm auswählt und vorbereitet. In diesem Fall ist der Benutzer für den grundlegenden Schritt verantwortlich. Nach der Auswahl der Datei wird diese dann in das Tool übertragen, damit sie verarbeitet werden kann. Da das Tool nicht mehr als eine Datei gleichzeitig verarbeiten kann, führt es zunächst eine vollständige Analyse der ersten benötigten Datei durch und liefert die Ergebnisse, die die Anwendung bei dieser Datei erzielt hat, bevor es zur nächsten Datei übergeht.

Aufgrund ihrer einfachen Syntax und der großen Vielfalt an Bibliotheken ist Python die ideale prototypische Programmiersprache für den Zweck dieser Arbeit. Das in der Programmiersprache Python geschriebene Tool beginnt damit, einen PDF-Datei zu öffnen, die gesamte Struktur der Datei zu analysieren und alle Seiten der Datei zu lesen bzw. zu interpretieren (siehe Abbildung 4.2), beginnend mit der ersten Seite und bis zur letzten Seite. Dieses Lesungsverfahren ist abhängig von der Analyse des Layouts für jede Seite in der Datei und der Dekodierung der mit diesem Layout verbundenen Elemente. Auf diese Weise werden alle Elemente auf der Oberfläche des Layouts initial erkannt. Einige Beispiele für diese Elemente sind Textblöcke, die hauptsächlich durch Absätze dargestellt werden, Bilder, Bildunterschriften und Diagramme.

Dieser Prozess wird für jede Seite der Datei durchgeführt, und danach wird in der nächsten Phase die Kategorie jeder Komponente des Layouts der Seite ermittelt. Aufgrund der Art des Elements wird es unabhängig behandelt und zusammen mit anderen Komponenten, die mit ihm vergleichbar sind, aus der Seite herausgezogen, um eine für es geeignete Ausgabekette zu erstellen. Als Ergebnis erhalten wir verschiedene Sammlungen für die verschiedenen Arten von Komponenten, und diese Sammlungen werden dann vorbereitet, um an die Ausgabedatei angehängt zu werden. Auf den nächsten Seiten werden die einzelnen Schritte detaillierter beschrieben.

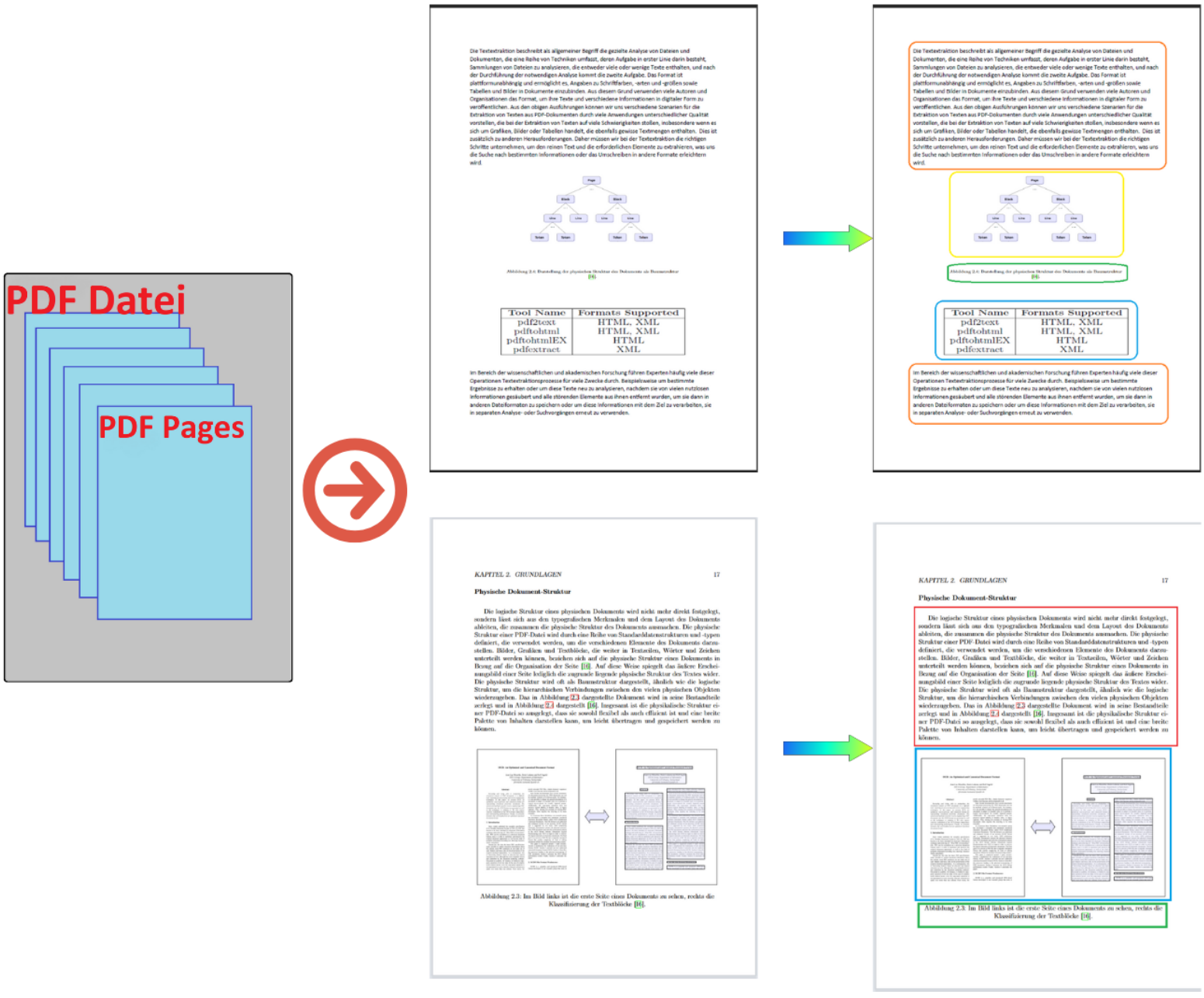


Abbildung 4.2: Interpretieren der Seiten der Datei und Elemente jeder Seite festlegen.

4.2 Schritt 2: Identifizierung der Elemente

Eine erfolgreiche Analyse der Seiten der ausgewählten Datei und die genaue Identifizierung der Elemente auf jeder Seite ist die Voraussetzung für den nächsten Schritt, nämlich die Klassifizierung der ausgewählten Elemente, die Angabe des Elementtyps und die Sortierung der Elemente in einzelne Baugruppen. Dieser Schritt muss abgeschlossen sein, bevor mit dem nächsten Schritt begonnen werden kann. Wie in Abbildung 4.1 zu sehen ist, besteht die Hauptaufgabe des Filters darin, eine Folge von Elementen auszuwählen, die der Benutzer nach Abschluss des gesamten Extraktionsprozesses benötigt. Dies kann als die Hauptverantwortung des Filters angesehen werden.

Nachdem die Untersuchung der Seite abgeschlossen ist, beginnt P2X mit der Sortierung der bereitgestellten Elemente. Dabei untersucht es jedes Element und bestimmt seine Art, um es der am besten geeigneten Kategorie zuzuordnen (siehe Abbildung 4.3). Das System beginnt mit der Suche nach jedem Textfeld auf der Seite. Dazu untersucht es jedes Element, ob es sich um Text, ein Bild oder eine Tabelle handelt, und stellt dann alle Ergebnisse zusammen. Die Funktion des Tools wird in keiner Weise von der Positionierung der Textfelder oder Absätze auf der Seite beeinflusst. Es gibt keine Verbindung zwischen den beiden. Befindet sich beispielsweise ein Teil des Textabsatzes am Anfang der Seite und der verbleibende Teil am Ende und ist durch ein Bild getrennt, so kombiniert das Werkzeug die beiden Teile, nachdem sie von der Seite gezogen wurden.

Alle Textfelder auf der Seite, unabhängig davon, wie groß oder klein sie sind oder wie viele Wörter sie enthalten, werden miteinander verbunden, um eine Textformation zu erzeugen, die die Textinformationen darstellt, die zuvor auf der zu analysierenden Seite geschrieben wurden. Nach Abschluss dieses Vorgangs fährt das Tool mit seiner Aufgabe fort und beginnt mit der Suche nach den Bildern, die sich auf das Layer dieser Seite befinden. Das Tool setzt seine Arbeit fort, indem es eine Analyse der verbleibenden Elemente durchführt, die alle Bilder aufzeigt, die, wenn sie auf der Seite vorhanden sind, das Tool zerlegt und ihre Bildunterschriften separat extrahiert, ohne sie mit der Textausgabe der Absätze auf der Seite zu verschmelzen. Wenn die Bilder nicht auf der Seite gefunden werden, fährt das Tool mit dem nächsten Schritt fort. Nachdem das Programm die Identifizierung der Textabsätze und Bilder auf der Seite abgeschlossen hat, sucht es als Nächstes nach Tabellen auf der Seite, die Textinformationen enthalten. Wenn eine Tabelle gefunden wird, wird der Textinhalt dieser Tabelle extrahiert. Auf diese Weise schließt das Tool den Prozess der Analyse aller Elemente der Seite ab und ordnet sie in spezifische Untergruppen ein, die verschiedene Arten von Informationen enthalten. Diese spezifischen Zusammenstellungen bilden den Teil der Ausgabedatei, der sich auf den Inhalt der analysierten Seite in der Eingabedatei bezieht.

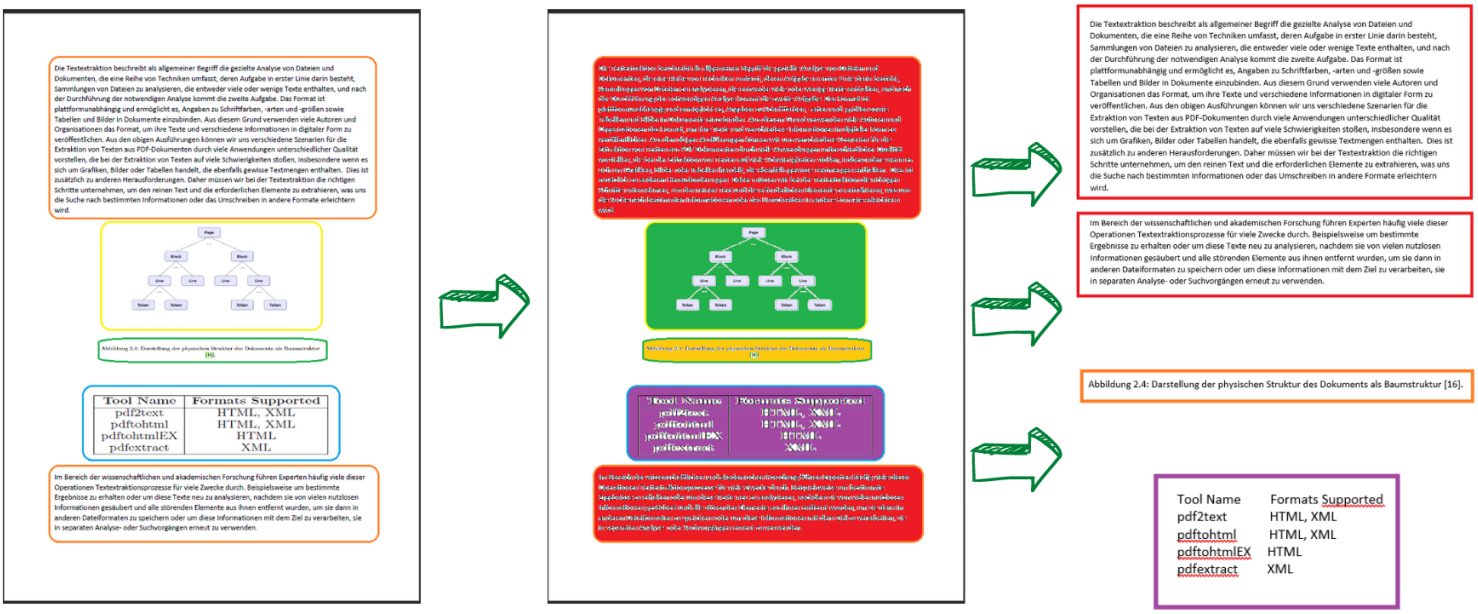


Abbildung 4.3: Identifizierung der Elemente der zu analysierenden Seite.

4.3 Schritt 3: Extrahieren und Exportieren

Wie bereits in den vorangegangenen Abschnitten dieses Kapitels erwähnt, ist es nach der Verarbeitung der ausgewählten Datei, der Analyse der Dateiseiten und der Kategorisierung und Extraktion der Komponenten erforderlich, ein Speicherformat für die extrahierten Daten zu finden. Denn das Speicherformat wird zur Speicherung der extrahierten Daten verwendet. Nach einem Bereinigungs- und Filterungsprozess der PDF-Datei können nun strukturierte Daten abgerufen werden, die effizient genutzt werden können. Dies wird durch das durchgeführte Verfahren besser ermöglicht als wenn die Analyse direkt auf der Eingabedatei angewandt wird. Wie in Abbildung 4.1 angedeutet, besteht der letzte Schritt des Programms darin, eine Extraktionsdatei zu erstellen und die Daten von jeder Seite der Eingabedatei darin zu speichern. Diesem Schritt folgt das Speichern der Datei. Mit diesem letzten Schritt ist der Arbeit des Tools fertiggestellt.

Wenn es um die Auswahl des Speicherformats geht, ist XML die beste Option. XML ist eine Auszeichnungssprache, die in großem Umfang für die Speicherung und Änderung von Dokumenten verwendet wird. Dabei wird der Inhalt der Dokumente beibehalten und gleichzeitig auf eine organisierte und effiziente Weise kodiert. Der Benutzer kann XML-Dateien genauso einfach lesen und auswerten wie jede andere Art von Datei. XML-Dokumente können eine Vielzahl von Strukturen haben, und diese Strukturen ermöglichen es, die Daten auf unterschiedliche Weise darzustellen. Der Hauptzweck dieser Struktur besteht darin, die Regeln zu definieren, die für diese Dokumente gelten. Diese Regeln können z. B. festlegen, welche Elemente und Attribute in diesen Dokumenten enthalten sind, sowie deren Anzahl und Anordnung, und sie können auch den Typ dieser Elemente und Attribute festlegen.

Das Tool P2X erzeugt eine XML-Datei, um die Informationen, die durch das Parsen und Verarbeiten der Seiten der PDF-Eingabedatei gewonnen wurden, in einer bestimmten Struktur zu speichern. Um die Daten jeder Seite in einer Komponente mit der gleichen Seitennummer zu speichern, erzeugt P2X Elemente mit der gleichen Anzahl der Seiten in der Eingabedatei, sobald es erfolgreich auf die PDF-Datei zugegriffen hat. Dadurch wird sichergestellt, dass die Daten nicht verloren gehen. Dieser Schritt verhindert Fehler beim Speichern von Daten und verhindert, dass sich die Inhalte der Seiten miteinander überschneiden. Jedes Element, auch Tag bezeichnet, setzt sich aus drei sekundären Tags zusammen. Die Hauptfunktion des ersten Tags besteht darin, den Text zu speichern, der aus den Textabschnitten der Seite geparst wurde. Im zweiten Tag werden die Bildunterschriften gespeichert, die sich auf den Bildern auf der Seite beziehen. Die Daten für die Tabelle sind im dritten Tag der Seite enthalten. Das bedeutet, dass nach der Analyse der Seite und der Extraktion ihrer Elemente nach der Kategorisierung diese Elemente den Tags hinzugefügt werden, die ihnen je nach Typ zugewiesen wurden, und dann wird dieses Tag an der dafür vorgesehenen Stelle in der Struktur der endgültigen XML-Extraktionsdatei hinzugefügt.

Wie in Abbildungen 4.4 und 4.5 zu sehen ist, hat die Ausgabedatei die Form einer Reihe von Tags, die aufeinander folgen. Jedes Tag bezieht sich auf eine Seite in der Eingabedatei, und das Tag selbst speichert den Inhalt dieser Seite in einer Weise, die nach bestimmten Kriterien organisiert und angeordnet ist. Wie der Arbeitsprozess im praktischen Sinne all dieser zuvor geklärten Schritte erreicht werden kann, wird im nächsten Abschnitt erörtert.

Abbildung 4.4: Speichern der extrahierten Elemente in einem XML-Tag.

```
<?xml version="1.0" ?>
<PDF_DATEI>
  <Pages>
    <Page nr=1>
    <Absaetze>
    <Absatz nr=1>
    </Absatz nr=1>
    .
    .
    .
    <Absatz nr=n>[redacted]</Absatz nr=n>
    </Absaetze>

    <Bildunterschriften>

    </Bildunterschriften>

    <Tabelle>
    <Tabelle nr=1>
    <tr nr=1>
    <td>Zelle 1</td>...<td>Zelle n</td>
    </tr nr=1>
    .
    .
    .
    <tr nr=n>
    <td>Zelle x</td>...<td>Zelle z</td>
    </tr nr=n>
    </Tabelle nr=1>
    .
    .
    <Tabelle nr=n>[redacted]</Tabelle nr=n>
    </Tabelle>
    </Page nr=1>

    <Page nr=2>[redacted]</Page nr=2>
    <Page nr=3>[redacted]</Page nr=3>
    <Page nr=4>[redacted]</Page nr=4>
    <Page nr=5>[redacted]</Page nr=5>
    <Page nr=6>[redacted]</Page nr=6>
    <Page nr=7>[redacted]</Page nr=7>
    <Page nr=8>[redacted]</Page nr=8>
    <Page nr=9>[redacted]</Page nr=9>
    <Page nr=10>[redacted]</Page nr=10>
    .
    .
    .
    <Page nr=n>[redacted]</Page nr=n>
    .
    .
    .
    <Page_y>[redacted]</Page_y>

  </Pages>
</PDF_DATEI>
```

Abbildung 4.5: Die finale Form der XML-Ausgabedatei.

Kapitel 5

Demonstratorische Implementierung

In diesem Kapitel wird gezeigt, wie die in dem letzten Kapitel 4 vorgeschlagene Konzeption zur Textextraktion in der Praxis unter Verwendung der Programmiersprache Python angewendet werden kann. In diesem Kapitel wird Schritt für Schritt gezeigt, wie diese Konzeption auf PDF-Dateien implementiert und Text aus diesen Dateien extrahiert wird. Die Implementierung umfasst alle notwendigen Konfigurationen und Codefragmente, die für die Durchführung der Extraktion erforderlich sind. Darüber hinaus enthält dieses Kapitel Beispiele für das Ergebnis, das mit dieser Konzeption erzielt werden kann, um ihre Nützlichkeit und Präzision zu unter Beweis zu stellen. Ziel dieses Kapitels ist es, dem Leser ein besseres Verständnis dafür zu vermitteln, wie die vorgeschlagene Vorgehensweise in realen Situationen umgesetzt werden kann und welche potenziellen Vorteile sie bei einer solchen Anwendung bietet.

5.1 Erforderlichen Anforderungen

Wie vorhin erwähnt, dass das erwünschte System durch Verwendung der Programmiersprache Python erreicht werden kann, mit der PDFMiner 3.2.2 geschrieben worden ist. PDFMiner ist ein unverzichtbares Werkzeug und ihre Unterstützung für die Extraktion von Text aus PDF-Dateien ist umfangreich, da sie mit mehreren PDF-Formaten umgehen kann. PDFMiner ist ein open-source-code Werkzeug, auf das jeder zugreifen und es nutzen kann. PDFMiner ist eine zuverlässiges und vielseitiges Bibliothek, die eine Vielzahl von Textextraktionsaufgaben bewältigen kann, einschließlich der Erhaltung des Layouts bei der Textextraktion, der Extraktion von Text aus nur bestimmten Seiten und der Extraktion von Text aus bestimmten Bereichen einer Seite. Da PDFMiner aus Python stammt, ist es mit einer Vielzahl von Python-basierten Anwendungen und Dienstprogrammen kompatibel. Dies bedeutet, dass Programmierer bessere Textextraktionswerkzeuge entwickeln können, indem sie die Vorteile der umfangreichen Python-Bibliothek nutzen. Insgesamt kann PDFMiner eine unschätzbare Quelle für Entwickler und Wissenschaftler sein, die mit PDF-Dokumenten arbeiten. In diesem Abschnitt werden die PDFMiner-Pakete und Methoden erläutert, die erforderlich für die Analyse und Extraktion sind.

In dem folgenden Codeausschnitt wird das Aufrufen der benötigten PDFMiner-Werkzeuge gezeigt.

```
from pdfminer.pdfparser import PDFParser
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfpage import PDFPage
from pdfminer.pdfpage import PDFTextExtractionNotAllowed
from pdfminer.pdfinterp import PDFResourceManager
from pdfminer.pdfinterp import PDFPageInterpreter
from pdfminer.pdfdevice import PDFDevice
from pdfminer.layout import LTLine, LAParams, LTFigure,
LTTextBox,LTImage,LTTextLine ,LTTextBoxHorizontal
from pdfminer.converter import PDFPageAggregator
```

Listing 5.1: Aufrufen der benötigten Klassen und Modulen

Die ersten Zeilen bringen Klassen aus den Komponenten **pdfminer.pdfparser, pdfminer.pdfdocument** und **pdfminer.pdfpage** ein, die **PDFParser, PDFDocument, PDFPage** und **PDFTextExtractionNotAllowed** heißen. Mit diesen Modulen ist es möglich, PDF-Dokumente zu lesen und zu verstehen und sogar bestimmte Textpassagen herauszunehmen. In den nachfolgenden Zeilen werden die Klassen **PDFResourceManager, PDFPageInterpreter** und **PDFDevice** des Moduls **pdfminer.pdfinterp** aufgerufen. Diese Klassen erleichtern die Verwaltung von Daten, die zum Parsen und Analysieren von PDF-Dateien nach textuellen Inhalten benötigt werden. Die Klassen **LTLine, LAParams, LTFigure, LTTextBox, LTImage, LTTextLine** und **LTTextBoxHorizontal** werden vom pdfminer.layout-Paket bereitgestellt. Diese Klassen repräsentieren die verschiedenen PDF-Layoutkomponenten und ermöglichen die Textextraktion aus diesen Elementen.

Das folgende Codestück gibt drei Funktionen an, die verwendet werden können, um Text aus Tabellen in einer PDF-Datei abzurufen.

```
def rectIntersectVertical(rect1 , rect2 ) ->  bool :
    outerRect = []
    innerRect = []

    if rect1[1] < rect2[1]:
        outerRect = rect1
        innerRect = rect2
    else:
        outerRect = rect2
        innerRect = rect1

    return   innerRect[1] >= outerRect [1] and innerRect[1] < outerRect[3]
-----------------------------------------------
```

```
14 def rectIntersectHorizontal(rect1 , rect2 ) ->  bool :
15     outerRect = []
16     innerRect = []
17 
18     if rect1[0] < rect2[0]:
19         outerRect = rect1
20         innerRect = rect2
21     else:
22         outerRect = rect2
23         innerRect = rect1
24 
25     return   innerRect[0] >= outerRect [0] and innerRect[0] < outerRect[2]
26 ------------------------------------------------
27 def addTextInTable(text , page) :
28     index = -1
29     for rect in page["rect"]:
30         index =index +1
31         x = rectIntersectVertical(rect , text )
32         y = rectIntersectHorizontal(rect , text )
33         if x and y :
34             page["table"][index].append(text)
35             return True
36     return False
```

Listing 5.2: Tabellen suchen und deren Inhalt bestimmen durch Hilfsfunktionen

Die erste Funktion mit der Bezeichnung **rectIntersectVertical** prüft zwei Rechtecke, die jeweils als eine Sammlung von vier Ganzzahlen (x1, y1, x2 und y2) dargestellt werden, und stellt fest, ob sich die Rechtecke vertikal schneiden oder nicht. Die zweite Funktion heißt **rectIntersectHorizontal** und untersucht zwei Rechtecke, um herauszufinden, ob es eine horizontale Überschneidung zwischen ihnen gibt. Die dritte Methode mit der Bezeichnung **addTextInTable** erfordert ein Textfeld und ein Seitenobjekt, um ordnungsgemäß zu funktionieren. Jedes Rechteck in der Rect-Liste des Page-Objekts wird durchlaufen, um zu sehen, ob das Textrechteck in vertikaler und horizontaler Richtung mit ihm übereinstimmt. Ist dies der Fall, wird der Inhalt in die entsprechende Spalte der im Seitenobjekt enthaltenen Tabellenliste eingefügt. Die Methode liefert False, wenn das Textfeld keines der Rechtecke aus der angegebenen Liste enthält. Im Falle einer erfolgreichen Hinzufügung von Zellen wird True zurückgegeben. Zusammen können diese Operationen bestimmen, welche Textfelder mit welchen Tabellenzellen in einem PDF-Dokument korrelieren, und dann den extrahierten Text in der entsprechenden Zelle der Tabellenliste speichern. Diese Methode setzt voraus, dass die PDF-Tabelle eine einheitliche rechteckige Form hat, mit separaten Rechtecken für jede Spalte.

Die nächste zwei Funktionen dienen für die Extraktion der Bildunterschriften, die in der Seite befinden. Die Aufgabe der ersten Funktion (**findBelowElement**) ist Festlegung des Textboxes, der direkt unter dem in der Seite bestehende Bild steht. Die zweite Funktion (**checkImage**) prüft es, ob der unter dem Bild befindliche Text den Begriff (Abbildung) oder (Figur) enthält, was beides darauf hindeutet, dass es sich um eine Abbildung handelt. Ist dies der Fall, erstellt die Funktion eine Kopie der Datenstruktur (absatz), die den Ergebnistext enthält und fügt sie an die Liste (Figure) in der Datenstruktur an. Damit der Inhalt von (absatz) in Zukunft nicht erneut hinzugefügt werden kann, wird das Original gelöscht.

```
def findBelowElement (element , elements) :
    index = -1
    i =-1
    for e in elements: #Schleife fuer alle Elemente (Textfelder)
        i = i+1
        #Pruefen, ob e unter Element
        if e[1] < element[1] and e[3] < element[3]:
            if index == -1 : #das erste Ergebnis
                index = i
            elif e[1] < elements[index] [1]:
                index =i
    return index
-------------------------------------------------------------------------
def checkImage(page):
    for image in page["images"]:
        belowElementIndex = findBelowElement(image, page["absatz"])
        if belowElementIndex == -1:  #keinen Text unter dem Bild gefunden
            continue
        #Ergebnistext der Funktion findBelowElement nehmen
        absatz = page["absatz"][belowElementIndex]

        text = absatz[4]
        #Text mit bestimmten Worten beginnen, dann ist es Text unter Bild
        isThisFigure = text.find("abbildung") == 0 or
                       text.find("Abbildung") == 0 or
                       text.find("figure") == 0 or
                       text.find("Figure") == 0

        if not isThisFigure:
            continue

        # das Ergebnis unter dem Bildtext hinzufuegen
        res = absatz[:]  #den Absatz in die Abbildungsdaten kopieren
        #Daten zu den Abbildungslisten hinzufuegen
        page["figure"] .append(res)

        #Bildtext entfernen,damit er nicht in (Abstatz) eingetragen wird
        absatz[4] =  ""
```

Listing 5.3: Bildtext suchen und ziehen durch Hilfsfunktionen

5.2 Zugriff auf Datei

In diesem Codeausschnitt wird die PDF-Datei aufgerufen und überprüft, ob es zur Analyse gilt oder nicht. Es werden auch ein paar wichtigen Parametern vorbereitet, die in den nächsten Quellcodeausschnitten erforderlich sind.

```
# Open a PDF file.
fp = open('mypdfmahmoud.pdf', 'rb')
#fp = open('mypdfTa.pdf', 'rb')
#fp = open('mypdfSa.pdf', 'rb')
#fp = open('mypdfTarreq.pdf', 'rb')
# Create a PDF parser object associated with the file object.
parser = PDFParser(fp)
# Create a PDF document object that stores the document structure.
# Supply the password for initialization.
document = PDFDocument(parser)
# Check if the document allows text extraction. If not, abort.
if not document.is_extractable:
    raise PDFTextExtractionNotAllowed
# Create a PDF resource manager object that stores shared resources.
rsrcmgr = PDFResourceManager()
# Create a PDF device object.
device = PDFDevice(rsrcmgr)
# Create a PDF interpreter object.
interpreter = PDFPageInterpreter(rsrcmgr, device)

# Set parameters for analysis.
laparams = LAParams()

# Create a PDF page aggregator object.
device = PDFPageAggregator(rsrcmgr, laparams=laparams)
interpreter = PDFPageInterpreter(rsrcmgr, device)
```

Listing 5.4: Aufrufen und Vorbereitung der Datei

5.3 Vollständiger Prozess der Seite

Seite für Seite analysiert dieser Codeausschnitt 5.5 ein PDF-Dokument, nimmt Layoutkomponenten wie horizontale und vertikale Linien, Textbereiche und Bilder und speichert sie in einer Wörterbuchsammlung namens **raw**. Pro Seite wird eine Datenbank mit den folgenden Schlüsseln erzeugt: **Seitennummer**, **hLines**, **vLines**, **rawText**, **Bilder, absatz, Tabelle und Abbildung**. Die Positionen und Inhalte der Textfelder werden unter der Variablen **rawText** gespeichert. Die Positionen der Bilder werden unter dem Schlüssel **images** gespeichert. Die Einträge **absatz**, **table** und **figure** können verwendet werden, um alles zu speichern, von strukturiertem Text über Tabellendaten bis hin zu Bildunterschriften.

```
# Process each page contained in the document.
raw = []  # data of the page
# read all text AND  lines
for page in PDFPage.create_pages(document):
    resPage = {}
    # prepare the raw data
    resPage["hLines"] = []
    resPage["vLines"] = []
    resPage["rawText"] = []
    resPage["images"] = []
   # prepare the result lists
    resPage["absatz"] = []
    resPage["table"] = {}
    resPage["figure"] = []

    interpreter.process_page(page)
    layout = device.get_result()
    for el in layout:
        if isinstance( el , LTLine): # line elemnt
            hh = el.get_pts()
            points = hh.split(",")
            x1 = float(points[0])
            y1 = float(points[1])
            x2 = float(points[2])
            y2 = float(points[3])
            if y1 == y2 :  # horzontal line
                if x2 > x1 : resPage["hLines"].append([x1,y1,x2,y2])
                else:  resPage["hLines"].append([x2,y2,x1,y1])

            else: # vertical line
                if x2 > x1 :  resPage["vLines"].append([x1,y1,x2,y2])
                else :  resPage["vLines"].append([x2,y2,x1,y1])

        if isinstance( el , LTTextBoxHorizontal):   # text box  element
            xzxzxzxz =  el.get_text()
            resPage["rawText"].append([el.bbox[0],el.bbox[1],el.bbox[2],
el.bbox[3] , el.get_text()  ])

        if  isinstance(el, LTTextLine):    # no data found here ignore
            dff = el.getText()
            d = 0
        if isinstance(el, LTTextBox)  :# no data found here ignore
            dff = el.get_text()
            x = 0
        if isinstance(el, LTFigure):  # image element
            resPage["images"].append( [el.x0 , el.y0 , el.x1 , el.y1])

    raw.append(resPage)
```

Listing 5.5: Vollständige Bearbeitung der Seite

Dieses Codefragment verarbeitet jede Seite in der Liste **raw** und extrahiert Textdaten aus der Liste **rawText**. Die Methode **addTextInTable** fügt den Text dem Tabellen-Wörterbuch im Wörterbuch **resPage** hinzu, wenn festgestellt wird, dass der Text in eine Tabelle gehört. Wenn der Text nicht Teil einer Tabelle ist, wird er der Liste **absatz** hinzugefügt, die im Wörterbuch **resPage** für diese bestimmte Seite enthalten ist. Die Liste **Absatz** enthält regulären Text, der nicht in einer Tabelle oder einem Bild enthalten ist, und ist in einem separaten Bereich zu finden.

```
# if text in table add to table otherwise added to absatz
for page in raw :
    for text in page["rawText"]:
        t = addTextInTable(text , page ) #add text to table cells
        if t == False :
        #add text to absatz ( normal text )
            page["absatz"] .append( text)
```

Listing 5.6: Absätze- und Tabellen-Inhalte Einordnung

Durch diese **for**-Schleife wird die Funktion **checkImage(page)** für jede Seite aufgerufen und entfernt den Text aus Absatz, wenn der Text unter einem Bild steht und mit dem Wort Figure oder Abbildung beginnt.

```
#Text aus Absatz entfernen, wenn der Text unter einem Bild steht
#und mit Figure oder abbildung beginnt
for page in raw :
    checkImage(page)
```

Listing 5.7: Bildtext suchen und extrahieren

5.4 XML-Ausgabedatei generieren

Das Ziel dieses Systems ist es, eine XML-Datei zu erstellen, die den textlichen Inhalt der PDF-Datei enthält. In diesem Teil wird erläutert, wie eine XML-Datei erzeugt wird und wie die aus der PDF-Datei extrahierten Informationen nach der Verarbeitung der einzelnen Seiten eingefügt werden.

5.4.1 Absätze exportieren

Dieser Codeausschnitt erzeugt ein XML-Dokument, indem er über die Seiten eines PDF-Dokuments iteriert, das durch eine Datenstruktur namens **raw** dargestellt wird. Für jede Seite extrahiert der Code den Text der Absätze und erstellt ein XML-Element <page> mit einem Attribut n, das die Seitenzahl darstellt. Innerhalb jedes Seitenelements erstellt der Code ein XML-Element <Absaetze>, um die Absätze zu gruppieren, und für jeden Absatz wird ein XML-Element <absatz> mit einem Attribut n erstellt, das die

Absatznummer und den Text des Absatzes als Inhalt des Elements darstellt. Der Code wendet auch einige Filter an, um Absätze auszuschließen, die für die Ausgabe nicht relevant sind. Insbesondere werden Absätze übersprungen, die bestimmte Schlüsselwörter wie Inhaltsverzeichnis, Literaturverzeichnis, Abbildungsverzeichnis, Tabellenverzeichnis, Anhang, Listings und Abkürzungsverzeichnis enthalten. Außerdem werden Absätze übersprungen, die mit Abbildung oder Figure beginnen, sowie Absätze mit weniger als 10 Wörtern oder einer Schriftgröße kleiner als 10. Schließlich werden auch Absätze ausgeschlossen, die URLs enthalten.

```
# generate xml result -------------
xml= ''
xml = xml +'<?xml version="1.0" encoding="UTF-8"?>'
xml = xml +'<PDF_DATEI>'
xml = xml +'<pages>'

p =-1
for page in raw :
    p = p +1
    xml = xml +'<page n="' + str(p) + '">'

    # add all text --------------
    a = -1
    xml = xml +'<Absaetze>'
    #get the text of the first paragraph
    first_absatz = page["absatz"][0][4]
    if not first_absatz.lower().startswith(('inhaltsverzeichnis',
                                            'literatur',
                                            'literaturverzeichnis',
                                            'abbildungsverzeichnis',
                                            'tabellenverzeichnis',
                                            'anhang', 'listings')):
        for absatz in page["absatz"]:
            text = absatz[4]
            if 'http' not in text:
                if not text.lower().startswith(('abbildung', 'figure')):
                    #check if font size is greater than 9
                    if absatz[0] > 10:
                        words = text.split() #split the text into words
                        #check if there are more than 10 words
                        if len(words) > 8:
                            a += 1
                            lines = text.split("\n")
                            text = ' '.join(lines)
                            xml+='<absatz n="'+str(a)+'">'+text+'</absatz>'

    xml = xml +'</Absaetze>'
```

Listing 5.8: Texte der Absätze bearbeiten und extrahieren

5.4.2 Bildunterschriften exportieren

Dieser Code erzeugt einen XML-Block mit Bildunterschriften. Er prüft zunächst, ob Bilder auf der Seite vorhanden sind und fügt deren Bildunterschriften dem XML-Block hinzu. Dann wird der Text jedes Absatzes auf der Seite überprüft und alle Bildunterschriften, die mit **Abbildung** oder **Figure** beginnen, werden dem XML-Block hinzugefügt. Der resultierende XML-Block enthält alle Bildunterschriften auf der Seite.

```
#add all Bildunterschriften----------------------------------------
xml = xml +'<Bildunterschriften>'

b = -1
for Figure in page ["figure"]:
    b = b +1
    xml = xml + '<Bildunterschrift n="' + str(b) +  '">' + Figure [4]+
'</Bildunterschrift>'

if not first_absatz.lower().startswith(('inhaltsverzeichnis',
                                        'literatur',
                                        'abbildungsverzeichnis',
                                        'tabellenverzeichnis',
                                        'anhang', 'listings')):
    for absatz in page["absatz"]:
        text = absatz[4]
        if text.lower().startswith(("abbildung", "figure")):
            words = text.split()  # split the text into words
            if len(words) > 1:  # check if there are more than 10 words
                b += 1
                xml = xml + '<Bildunterschrift n="' + str(b) + '">' +
                                    text + '</Bildunterschrift>'

xml = xml + '</Bildunterschriften>'
```

Listing 5.9: Bildtext suchen und extrahieren

5.4.3 Tabellen exportieren

Der folgende Code erzeugt einen XML-Block, der Tabellen und deren Zellen enthält. Er durchläuft alle Tabellen auf der Seite in einer Schleife und fügt ihre Zellen dem XML-Block hinzu. Der resultierende XML-Block enthält alle Tabellen und ihre Zellen auf der Seite, mit einem eindeutigen Bezeichner (n) für jede Tabelle und Zelle.

```
# add all the tables ----------------------------------------
t = -1
xml = xml +'<Tabellen>'
for tableID in page["table"] :
    table = page["table"][tableID]
    if len(table) == 0 :
        continue
    t = t +1
    xml = xml + '<Tabelle n="' + str(t) +  '">'

    c = -1
    for cell in table :
        c = c +1
        xml = xml + '<Zelle n="' + str(c) +  '">' + cell[4] + '</Zelle>'

    xml = xml+ '</Tabelle>'
xml = xml +'</Tabellen>'
```

Listing 5.10: Tabellen extrahieren

Als letzte Schritt wird die Bearbeitung der aktuellen Seite mit </**page**> beendet, um mit der nächsten Seite zu beginnen. Nach der Anwendung des Systems auf allen Seiten der Datei wird die ganze Exportierung der Informationen von der Datei beendet und nachher wird die XML-Datei generiert und darin alle erwünschten Ergebnisse.

```
    xml = xml +'</page>'

xml = xml +'</pages>'
xml = xml +'</PDF_DATEI>'



#write text result to file
f = open("result.xml", "w", encoding='utf-8')
f.write(xml)
f.close()
```

Listing 5.11: End der Bearbeitung der Datei und XML-Ausgabedatei generieren

Da die vom System gelieferten Ausgaben sehr umfangreich sind, werden einige Beispiele für Eingaben 5.1 und 5.3 und Ausgaben 5.2 und 5.6 angeführt, da die Eingabe- und Ausgabedateien extern an die Datei dieser Arbeit angehängt werden.

2.3.3 PDF-Struktur

Die Interpretation eines Dokuments hängt von der eigenen Interpretation des Lesers ab, wobei ein Dokument strukturiert, halbstrukturiert oder unstrukturiert sein kann. In den meisten Fällen hat ein Dokument, das von Menschen gelesen werden kann, sowohl eine physische Form als auch eine logische Struktur. Ein Dokument hat Teile. Ein Abschnitt kann einen Titel, einen Abschnittsteil oder eine darin verschachtelte Struktur haben. Ein Abschnittswechsel kann in Form von zusätzlichem Abstand, einer oder mehreren Leerzeilen oder einer Abschnittsüberschrift für die Komponente, die nach dem Abschnitt kommt, erfolgen. Abschnitte sind die Komponenten, die visuell voneinander getrennt sind [15]. Die Abbildung 2.1 zeigt ein PDF-Dokument-Modell[1] als eine Sammlung von Teilen, Unterabschnitten usw.

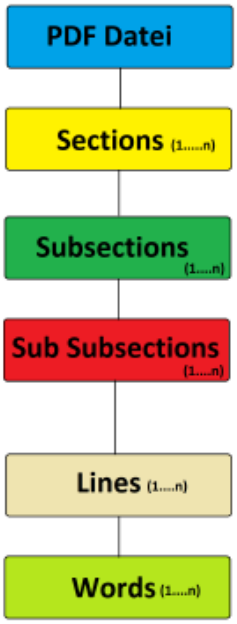


Abbildung 2.1: Ein PDF-Dokument-Modell [15]

Verschiedene Anwendungen zur Informationsextraktion und -abfrage, zur Kategorisierung und Zusammenfassung von Dokumenten und zur Textanalyse können alle von der Identifizierung der logischen Abschnitte eines Dokuments und der Organisation dieser Abschnitte in einer Standardstruktur profitieren [15].

[1] https://ebiquity.umbc.edu/_file_directory_/papers/857.pdf

Abbildung 5.1: Eine Seite der PDF-Datei als Teil der Eingabe-Datei.

```
</page>
<page n="15">
    <Absaetze>
        <absatz n="0">Die Interpretation eines Dokuments hängt von der eigenen
            Interpretation des Lesers ab, wobei ein Dokument strukturiert, halbstrukturiert
            oder unstrukturiert sein kann. In den meisten Fällen hat ein Dokument, das von
            Menschen gelesen werden kann, sowohl eine physische Form als auch eine logische
            Struktur. Ein Dokument hat Teile. Ein Abschnitt kann einen Titel, einen
            Abschnittsteil oder eine darin verschachtelte Struktur haben. Ein
            Abschnittswechsel kann in Form von zusätzlichem Abstand, einer oder mehreren
            Leerzeilen oder einer Abschnittsüberschrift für die Komponente, die nach dem
            Abschnitt kommt, erfolgen. Abschnitte sind die Komponenten, die visuell
            voneinander getrennt sind [15]. Die Abbildung 2.1 zeigt ein PDF-Dokument-Modell1
            als eine Sammlung von Teilen, Unterabschnitten usw. </absatz>
        <absatz n="1">Verschiedene Anwendungen zur Informationsextraktion und -abfrage, zur
            Kategori- sierung und Zusammenfassung von Dokumenten und zur Textanalyse können
            alle von der Identifizierung der logischen Abschnitte eines Dokuments und der
            Organisation dieser Abschnitte in einer Standardstruktur profitieren [15]. </absatz>
    </Absaetze>
    <Bildunterschriften>
        <Bildunterschrift n="0">Abbildung 2.1: Ein PDF-Dokument-Modell [15]
        </Bildunterschrift>
    </Bildunterschriften>
    <Tabellen></Tabellen>
</page>
```

Abbildung 5.2: Die extrahierten Ergebnisse von der Seite als XML-Tag in der Ausgabe-Datei.

4 Konzept der WossiDiA-Python-API

- Clique Expansion: Die ist wohl die populärste Technik zur Umwandlung eines Hypergraphen in einen Normal Graphen. Der Cliquen-Expansions Algorithmus konstruiert einen Graphen $G^b(V, E^b)$ aus dem ursprünglichen Hypergraphen $G(V, E)$. Bei der Cliquenexpansion wird jedes Hyperkante zu einer Clique umgewandelt. In dem jede Hyperkante durch eine normale Kante für jedes Paar von Knoten in der Hyperkanten ersetzt wird [ABB06].
- Star Expansion: Hier wird ein Graph $G^b(V^b, E^b)$ aus dem Hypergraph $G(V, E)$ konstruiert, indem für jede Hyperkante $e \in E$ ein neuer Knoten erzeugt wird. Außerdem verbindet sich der neue Knoten mit jedem Knoten in der Hyperkante[ABB06].
- Lawler-Expansion: Hier werden zwei neue Knoten $e1$ und $e2$ aus der Hyperkante e erzeugt. Für jeden $v \in e$ wird einen gerichtete Kante von v zu $e1$ und eine gerichtete Kante von $e2$ zu v eingefügt. Schließlich wird noch eine gerichtete Kante von $e1$ nach $e2$ eingefügt[BKV21].

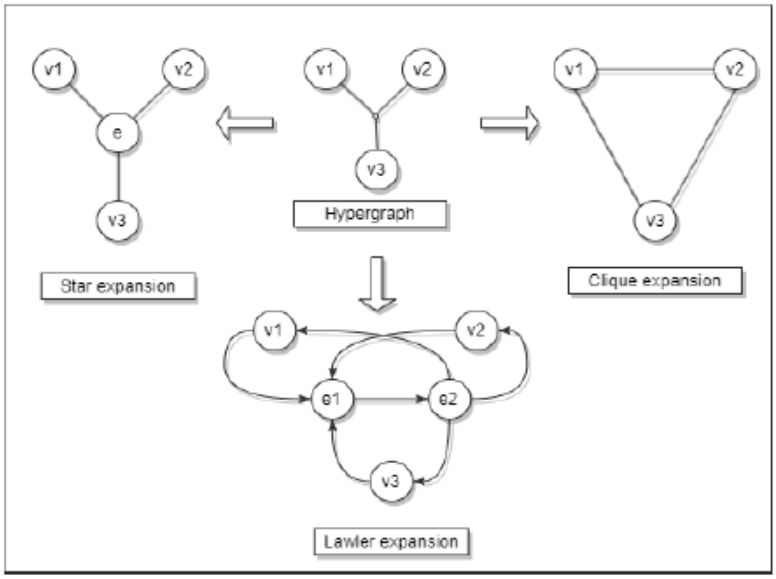


Abbildung 4.3. Hypergraph-Expansion

35

Abbildung 5.3: Eine Seite der PDF-Datei als Teil der Eingabe-Datei [40].

```
<page n="37">
    <Absaetze>
        <absatz n="0">• Clique Expansion: Die ist wohl die populärste Technik zur Umwandlung
            eines Hy- pergraphen in einen Normal Graphen. Der Cliquen-Expansions Algorithmus
            kon- struiert einen Graphen Gb(V, Eb) aus dem ursprünglichen Hypergraphen G(V,
            E). Bei der Cliquenexpansion wird jedes Hyperkante zu einer Clique umgewandelt.
            In dem jede Hyperkante durch eine normale Kante für jedes Paar von Knoten in der
            Hyperkanten ersetzt wird [ABB06]. </absatz>
        <absatz n="1">• Star Expansion: Hier wird ein Graph Gb(V b, Eb) aus dem Hypergraph
            G(V, E) kon- struiert, indem für jede Hyperkante e ∈ E ein neuer Knoten erzeugt
            wird. Außerdem verbindet sich der neue Knoten mit jedem Knoten in der
            Hyperkante[ABB06]. </absatz>
        <absatz n="2">• Lawler-Expansion: Hier werden zwei neue Knoten e1 und e2 aus der
            Hyperkante e erzeugt. Für jeden v ∈ e wird einen gerichtete Kante von v zu e1
            und eine gerichtete Kante von e2 zu v eingefügt. Schließlich wird noch eine
            gerichtete Kante von e1 nach e2 eingefügt[BKV21]. </absatz>
    </Absaetze>
    <Bildunterschriften>
        <Bildunterschrift n="0">Abbildung 4.3. Hypergraph-Expansion
        </Bildunterschrift>
    </Bildunterschriften>
    <Tabellen></Tabellen>
</page>
```

Abbildung 5.4: Die extrahierten Ergebnisse von der Seite als XML-Tag in der Ausgabe-Datei.

5.5 Mögliche Einschränkungen

Obwohl die Software in der Lage ist, Absätze, Bildunterschriften und Tabellen aus PDF-Dateien zu extrahieren, gibt es einige wichtige Einschränkungen, die beachtet werden müssen.

1. Eine mögliche Einschränkung ist die Genauigkeit der Textextraktion. Obwohl PDF-Miner ein effektives Tool zur Extraktion von Text aus PDF-Dateien ist, kann die Qualität der extrahierten Daten beeinträchtigt werden, wenn die PDF-Datei unklar strukturiert ist oder wenn sie Bilder oder Grafiken enthält, die den Text verdecken.

2. Ein weiteres Problem ist die Kompatibilität mit anderen Sprachen. Obwohl das Tool auf Deutsch funktioniert, kann es möglicherweise nicht mit anderen Sprachen wie Chinesisch, Japanisch oder Arabisch umgehen, da diese Sprachen besondere Schriftzeichen oder Schreibrichtungen haben.

3. Eine weitere Herausforderung ist ein zweispaltiges Layout in einer PDF-Datei. Ein zweispaltiges Layout in einer PDF-Datei besteht aus zwei Spalten, die nebeneinander angeordnet sind. Dies ist eine übliche Formatierungsmethode in wissenschaftlichen Zeitschriften und anderen Publikationen, insbesondere wenn viele Abbildungen, Tabellen oder Formeln enthalten sind. Obwohl es sich um eine gängige Formatierungsmethode handelt, kann ein zweispaltiges Layout zu Problemen bei der Textextraktion führen. Wenn die Software nicht in der Lage ist, mit einem zweispaltigen Layout umzugehen, kann dies zu Fehlern bei der Extraktion des Textes führen. Einige der häufigsten Probleme sind:

   (a) Falsche Textreihenfolge: Wenn der Text in zwei Spalten aufgeteilt ist, kann der Software Schwierigkeiten haben, den Text in der richtigen Reihenfolge zu extrahieren. Wenn der Text in der einen Spalte zu Ende geht und in der anderen Spalte fortgesetzt wird, kann dies dazu führen, dass der Software den Text falsch extrahiert und die Reihenfolge durcheinanderbringt.

   (b) Verlust von Informationen: Wenn der Text in einer Spalte so eng beieinander liegt, dass er sich überlappt oder Teile des Textes abgeschnitten werden, kann dies dazu führen, dass der Software Teile des Textes verliert oder unvollständig extrahiert.

   (c) Fehlerhafte Strukturierung: Wenn der Text in den beiden Spalten unterschiedliche Absatzlängen aufweist oder in verschiedenen Textgrößen oder Schriftarten formatiert ist, kann dies dazu führen, dass der Software die Struktur des Textes nicht richtig erkennt. Dadurch kann es schwieriger werden, Absätze oder andere strukturierte Elemente des Textes zu identifizieren und zu extrahieren.

   Um diese Probleme zu vermeiden, gibt es mehrere Ansätze, die verfolgt werden können, um der Software zu verbessern. Eine Möglichkeit besteht darin, eine Software zu implementieren, die in der Lage ist, das zweispaltige Layout automatisch zu

erkennen und den Text in jeder Spalte getrennt zu extrahieren. Eine andere Möglichkeit besteht darin, eine Vorverarbeitung des Dokuments durchzuführen, um das zweispaltige Layout in eine einzelne Spalte umzuwandeln. Es gibt eventuell auch andere spezielle Bibliotheken und Tools, die auf die Extraktion von Text aus PDF-Dateien mit einem zweispaltigen Layout spezialisiert sind. Wenn der Software also Schwierigkeiten hat, mit dieser Art von PDF-Dateien umzugehen, könnten diese Optionen eine Alternative sein. Insgesamt ist es wichtig, zu wissen, dass das Extrahieren von Text aus PDF-Dateien mit einem zweispaltigen Layout eine Herausforderung darstellt und dass die Implementierung von Lösungen, die diese Herausforderung bewältigen können, Zeit und Ressourcen erfordern kann.

4. Da der Software die Absätze, die in zwei Abschnitte getrennt sind, nicht automatisch zusammenfügen kann, kann das zu mehreren Fehlern und Problemen kommen, die die Qualität der extrahierten Texte beeinträchtigen können.

   (a) Ein möglicher Fehler ist, dass der Text des ersten Abschnitts unvollständig ist, da der zweite Abschnitt nicht extrahiert wird. Wenn beispielsweise der letzte Satz des ersten Abschnitts eine wichtige Information enthält, die für das Verständnis des Textes unerlässlich ist, kann es schwierig sein, den Text vollständig zu erfassen, wenn dieser Satz im zweiten Abschnitt verborgen ist.

   (b) Ein weiterer möglicher Fehler ist, dass die beiden Abschnitte in der falschen Reihenfolge extrahiert werden. Wenn beispielsweise der zweite Abschnitt vor dem ersten Abschnitt extrahiert wird, kann der Text unverständlich werden, da die chronologische Reihenfolge der Informationen gestört wird.

   (c) Ein weiterer möglicher Fehler ist, dass die beiden Abschnitte nicht richtig zusammengeführt werden, was zu einem unleserlichen oder fehlerhaften Text führt. Wenn beispielsweise ein Abschnitt mit einem Satz endet, der mit einem Wort beginnt, das im nächsten Abschnitt wiederholt wird, kann der Software diesen Satz möglicherweise nicht richtig zusammenführen, was zu einem unverständlichen Text führt.

   (d) Darüber hinaus kann es schwierig sein, die exakte Position zu bestimmen, an der die beiden Abschnitte zusammengeführt werden sollen. Wenn beispielsweise der zweite Abschnitt nur aus einem Satz besteht, der mit einer neuen Zeile beginnt, kann es schwierig sein, zu entscheiden, ob der Satz zum ersten Abschnitt gehört oder ein eigener Abschnitt ist.

   Insgesamt kann die Unfähigkeit der Software, getrennte Absätze automatisch zu erkennen und zusammenzuführen, zu mehreren Fehlern führen, die die Qualität der extrahierten Texte beeinträchtigen können. Daher ist es wichtig, diese Einschränkung im Hinterkopf zu behalten und gegebenenfalls manuell die beiden Absätze miteinander zu verbinden, um sicherzustellen, dass der Text korrekt und verständlich ist.

5. Eine weitere Herausforderung ist die Probleme, die bei der Verarbeitung von PDF-Zeitungsartikel mit dem Software auftreten können. Die PDF-Zeitungsartikel sind oft sehr unterschiedlich gestaltet und können eine Vielzahl von Formatierungen aufweisen. Zum Beispiel können sie mehrere Spalten, Überschriften, Bilder und Textboxen enthalten, die alle auf derselben Seite angeordnet sind. Solche Dateien können für Ihren Software schwierig sein, da die Formatierung unvorhersehbar sein kann und der Text nicht immer in klare Absätze oder Tabellen organisiert ist.

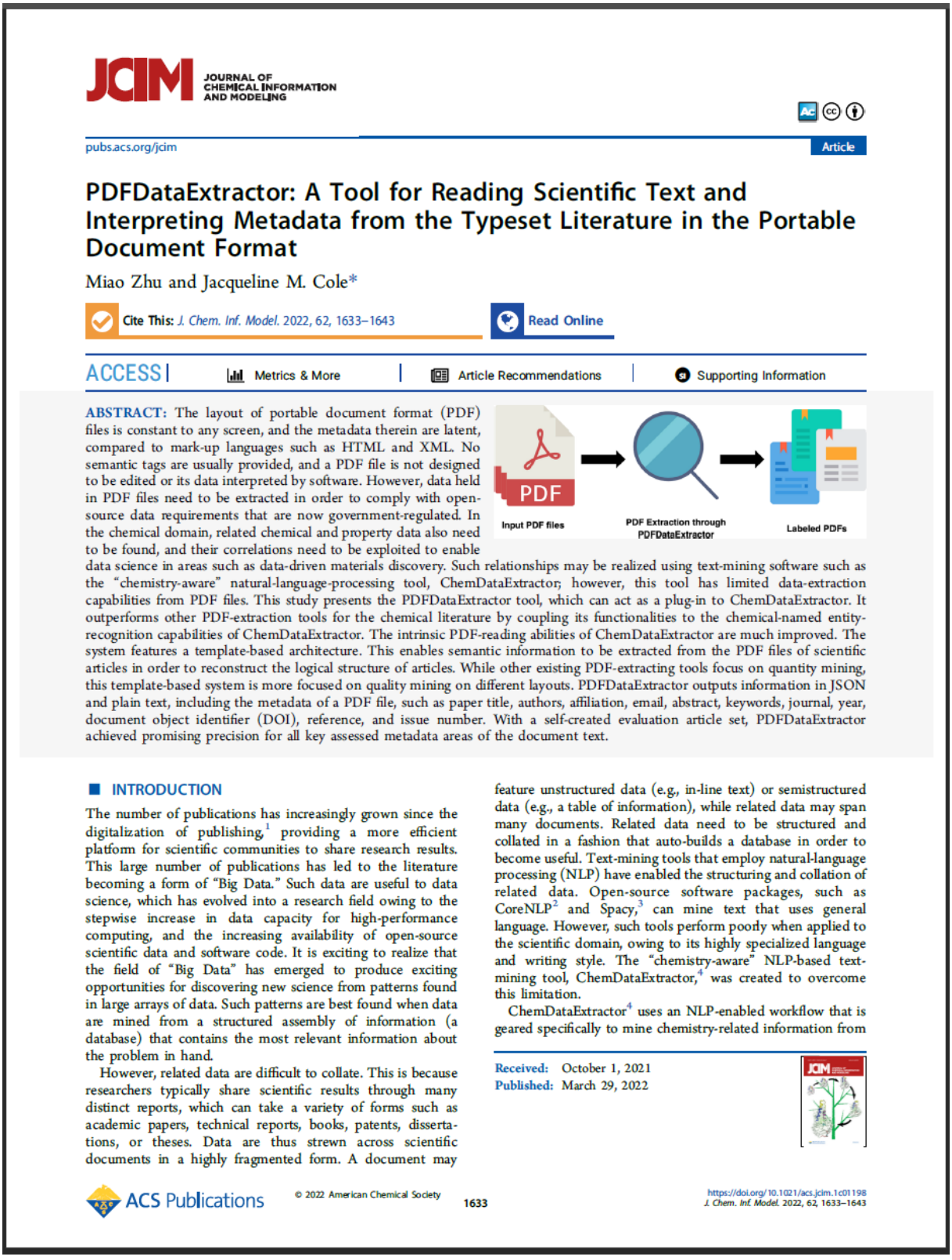
   (a) Eines der Probleme, die bei der Verarbeitung der PDF-Zeitungsartikel auftreten können, ist das Fehlen eindeutiger Absatz- oder Tabellenstrukturen. In solchen Dateien ist es oft schwierig, die einzelnen Absätze oder Tabellen voneinander zu trennen und zu extrahieren. Wenn der Text nicht in eindeutigen Absätzen oder Tabellen organisiert ist, kann der Software Schwierigkeiten haben, den Text korrekt zu extrahieren.

   (b) Ein weiteres Problem besteht darin, dass die Zeitungsartikel oft Überschriften und Unterüberschriften enthalten, die in einer anderen Schriftart, -größe oder -farbe als der restliche Text geschrieben sind. Diese Überschriften und Unterüberschriften können für den Software schwierig zu identifizieren sein und möglicherweise nicht korrekt extrahiert werden.

   Es können bei der Verarbeitung von PDF-Zeitungsartikel mit der Software eine Vielzahl von Problemen auftreten, einschließlich der Schwierigkeit, eindeutige Absatz- und Tabellenstrukturen zu erkennen, das Extrahieren von Überschriften und Unterüberschriften. Es ist wichtig, diese Einschränkungen zu berücksichtigen und alternative Methoden zu prüfen, wenn es mit PDF-Zeitungsartikel gearbeitet werden soll.

Schließlich ist auch die Genauigkeit der Ergebnisse eine mögliche Einschränkung. Obwohl das Tool in der Lage ist, Absätze, Bildunterschriften und Tabellen zu extrahieren, können die Ergebnisse möglicherweise nicht perfekt sein und müssen manuell überprüft werden. In einigen Fällen kann es auch erforderlich sein, manuelle Anpassungen vorzunehmen, um die Genauigkeit der extrahierten Daten zu verbessern.

Zusammenfassend gibt es mehrere Fehler, die bei der Verarbeitung von PDF-Dateien mit zweispaltigem Layout oder als Zeitungsartikel sowie bei getrennten Absätzen auftreten können. Diese Fehler können dazu führen, dass der Text unvollständig, inkorrekt oder verwirrend extrahiert wird, was zu einer ungenauen Darstellung des Dokuments führen kann. Um diese Probleme zu lösen, können spezielle Tools und Technologien verwendet werden, um den Text in PDF-Dateien genau und vollständig zu extrahieren.



JCIM JOURNAL OF CHEMICAL INFORMATION AND MODELING

pubs.acs.org/jcim Article

PDFDataExtractor: A Tool for Reading Scientific Text and Interpreting Metadata from the Typeset Literature in the Portable Document Format

Miao Zhu and Jacqueline M. Cole*

Cite This: J. Chem. Inf. Model. 2022, 62, 1633−1643 Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: The layout of portable document format (PDF) files is constant to any screen, and the metadata therein are latent, compared to mark-up languages such as HTML and XML. No semantic tags are usually provided, and a PDF file is not designed to be edited or its data interpreted by software. However, data held in PDF files need to be extracted in order to comply with open-source data requirements that are now government-regulated. In the chemical domain, related chemical and property data also need to be found, and their correlations need to be exploited to enable data science in areas such as data-driven materials discovery. Such relationships may be realized using text-mining software such as the "chemistry-aware" natural-language-processing tool, ChemDataExtractor; however, this tool has limited data-extraction capabilities from PDF files. This study presents the PDFDataExtractor tool, which can act as a plug-in to ChemDataExtractor. It outperforms other PDF-extraction tools for the chemical literature by coupling its functionalities to the chemical-named entity-recognition capabilities of ChemDataExtractor. The intrinsic PDF-reading abilities of ChemDataExtractor are much improved. The system features a template-based architecture. This enables semantic information to be extracted from the PDF files of scientific articles in order to reconstruct the logical structure of articles. While other existing PDF-extracting tools focus on quantity mining, this template-based system is more focused on quality mining on different layouts. PDFDataExtractor outputs information in JSON and plain text, including the metadata of a PDF file, such as paper title, authors, affiliation, email, abstract, keywords, journal, year, document object identifier (DOI), reference, and issue number. With a self-created evaluation article set, PDFDataExtractor achieved promising precision for all key assessed metadata areas of the document text.

INTRODUCTION

The number of publications has increasingly grown since the digitalization of publishing,[1] providing a more efficient platform for scientific communities to share research results. This large number of publications has led to the literature becoming a form of "Big Data." Such data are useful to data science, which has evolved into a research field owing to the stepwise increase in data capacity for high-performance computing, and the increasing availability of open-source scientific data and software code. It is exciting to realize that the field of "Big Data" has emerged to produce exciting opportunities for discovering new science from patterns found in large arrays of data. Such patterns are best found when data are mined from a structured assembly of information (a database) that contains the most relevant information about the problem in hand.

However, related data are difficult to collate. This is because researchers typically share scientific results through many distinct reports, which can take a variety of forms such as academic papers, technical reports, books, patents, dissertations, or theses. Data are thus strewn across scientific documents in a highly fragmented form. A document may feature unstructured data (e.g., in-line text) or semistructured data (e.g., a table of information), while related data may span many documents. Related data need to be structured and collated in a fashion that auto-builds a database in order to become useful. Text-mining tools that employ natural-language processing (NLP) have enabled the structuring and collation of related data. Open-source software packages, such as CoreNLP[2] and Spacy,[3] can mine text that uses general language. However, such tools perform poorly when applied to the scientific domain, owing to its highly specialized language and writing style. The "chemistry-aware" NLP-based text-mining tool, ChemDataExtractor,[4] was created to overcome this limitation.

ChemDataExtractor[4] uses an NLP-enabled workflow that is geared specifically to mine chemistry-related information from

Received: October 1, 2021
Published: March 29, 2022

https://doi.org/10.1021/acs.jcim.1c01198
J. Chem. Inf. Model. 2022, 62, 1633−1643

Abbildung 5.5: Beispiel für ein zweispaltiges Layout einer PDF-Seite [41].

Freiburger *Wochen*bericht
13.08.2014

Schlaflos in Freiburg

Zwölf Stunden als Anwohner am Augustinerplatz – Protokoll einer Wochenend-Feiernacht

Seit 1998 setzen sich die Innenstadtbewohner dafür ein, in Ruhe schlafen zu können, seit 2000 gibt es einen Runden Tisch zum Thema. Im Oktober 2013 beschloss der Gemeinderat die Einrichtung eines „Kommunalen Ordnungsdienstes" (KOD) mit knapper Mehrheit. Aufgrund neuer Mehrheitsverhältnisse könnte der Beschluss nun wieder gekippt werden. Anlass, die nächtliche Situation vor Ort zu erleben.

Hochbetrieb um 23.30 Uhr: der Augustinerplatz in einer Ferien-Freitagnacht FOTOS: HOFMAIER

Freitag, 8. August, 19.30 Uhr, 28 Grad: Ich beziehe mein Quartier am Augustinerplatz. Bei geöffnetem Fenster ist Stimmengemurmel und verhaltenes Gelächter in angenehmer Lautstärke zu vernehmen.

20.30 Uhr, 26 Grad: Die Treppe am „August" füllt sich, Freiburger und Touristen genießen in Sommerkleidung den warmen Abend, Kinder hopsen, Flaschen kreisen.

21 Uhr, 25 Grad: Es wird langsam dunkel. Die Treppe ist voll, die Besucher setzen sich einzeln und in Grüppchen aufs Pflaster. Bierverkauf aus mitgebrachten Kästen. Erste Flaschensammler machen sich auf den Weg.

21.15 Uhr: Live-Musik in der Besetzung Kontrabass, Trompete, Gesang. Südamerikanische Rhythmen.

21.30 Uhr: Beifall, Animation „Kuku" und Sprechgesang. Ca. 200 Menschen trinken sich warm. Die „Säule der Toleranz" glüht in blau-pink-orange-gelb.

21.45 Uhr: Immer mehr Menschen strömen auf den Platz. Die akustische Kulisse schwillt an. Frauen kreischen, Männer grölen.

22 Uhr: Kleines Gerangel unter Freunden. Grüppchen überall auf dem Platz, nun auch vor den Geschäften. Die Säule strahlt grün, das Rot wächst.

22.15 Uhr: Solo-Gitarrenkünstler gibt mit gezupftem „La Bamba" den Startschuss zum Hit dieser (nur dieser?) Nacht.

22.30 Uhr, 22 Grad: „La Bamba" wird von einigen Anwesenden mit frenetischem Gegröle aufgenommen. Fehlt nur noch das Lagerfeuer. Die Säule der Toleranz ist nun zur Hälfte errötet.

23 Uhr, 21 Grad: Deutliche Steigerung der Lautstärke, ca. 400 Personen auf dem Platz. Flaschen klirren, erste Verbrüderungen zwischen den Grüppchen. Die Säule ist durchgehend rot. Keiner beachtet sie.

23.30 Uhr, 20 Grad: Rundgang über den „August". Alkoholdunst in der Luft, friedlich-kollektives Besäufnis am Boden. Die Flaschensammler haben gut zu tun.

Ruhe nach Mitternacht? Von wegen!

Samstag, 9. August, 0.15 Uhr: Flamenco per Gitarre, begleitet von rhythmischem Klatschen.

1 Uhr: „Eijeijeijeijei" gellt es über den Platz. „Guantanamera" macht „La Bamba" Konkurrenz.

2 Uhr: Die Gitarre wird neu gestimmt, um zum fünften „La Bamba" mit „Eijeijeijeijei" anzuheben. Einige Anwesende begleiten das Spektakel mit Flaschenrollen vom höchsten Punkt des Platzes aus.

2.30 Uhr, 17 Grad: Spontane Chorgründung zum Absingen spanischer Gassenhauer. Flaschenwerfen macht offensichtlich großen Spaß.

2.45 Uhr, Platzregen, 14 Grad: Im Schatten der Mauer trotzen 15 Unentwegte dem Nass von oben mit ohrenbetäubender Bekräftigung der deutsch-spanischen Freundschaft und dem unerfüllten Wunsch „It's raining men". Diverse internationale Stammesgesänge münden in die deutsche Nationalhymne.

3 Uhr: Wir lernen einen neuen Klatsch-Rhythmus, um damit nahtlos in die italienische Ballade „Volare" zu wechseln. Die Säule glüht rot.

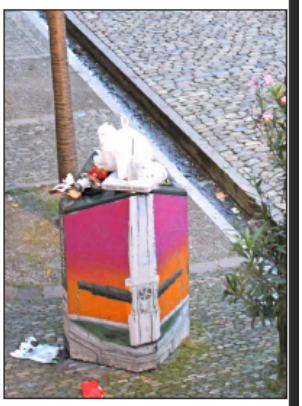
4 Uhr: In die zögerlich einkehrende Ruhe nach Flaschensammeln und Scherbenkicken mischt sich lautes Geklirr. Am nächsten Morgen ist das Geräusch zuordenbar: Die Scheibe vom Spielzeugladen „Holzpferd" hat ein riesiges Loch.

6 Uhr: Die Stadtreinigung fährt dreimal über den komplett vermüllten Platz. Kosten pro Jahr: 60.000 bis 100.000 Euro. Die Säule wird blass. Die Luft ist rein, der Platz ist leer, der neue Tag beginnt.

7 Uhr: Die Aussteller des ab 10 Uhr stattfindenden „Regionalmarktes" messen ab und bauen ihre Stände auf.

Persönliches Fazit: Eine weitestgehend schlaflose Nacht mit Erkenntniswert und Nachahmungs-Empfehlung. Wer wirklich wissen will, wie sich die - teilweise seit Generationen - hier lebenden Anwohner fühlen, für den müsste „Schlaflos in Freiburg" Pflichtprogramm sein. Eine einzige Nacht heilt von der nachgeplapperten Idee, es handele sich um „mediterranes Flair". Mitnichten. Es handelt sich um grobe Ruhestörung mit Werten über 70 Dezibel (zulässig sind 40 dB) und Spitzen bis 110 Dezibel. Lärm und Schlafmangel machen krank. Nicht mehr und nicht weniger. Alles andere ist Ideologie.

Meine Gastgeber fanden übrigens, es sei eine verhältnismäßig ruhige Nacht gewesen ... **Sigrid Hofmaier**



Kaum der Rede wert: nur ein bisschen Müll im Eimer...

Abbildung 5.6: Beispiel für ein mehrspaltiges Layout einer PDF-Zeitungsartikel [42].

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

In dieser Bachelorarbeit wird untersucht, wie man Text aus PDF-Dateien extrahieren kann, wobei der Schwerpunkt auf Absätzen, Bilduntertiteln und Tabellen liegt. Es wird eine Reihe von Python-Bibliotheken untersucht, aber am Ende wird PDFMiner als die effektivste Anwendung für diese Aufgabe ausgewählt. Das Ziel der Arbeit war es, eine effektive Methode zur Extraktion von Texten aus PDF-Dokumenten zu finden. Die Arbeit umfasst eine detaillierte Beschreibung des Arbeitsablaufs, einschließlich der Implementierung und Ausführung der PDFMiner-Codebibliothek. Die Arbeit beschreibt die Verfahren, die befolgt werden müssen, um Text aus einer PDF-Datei zu extrahieren, sowie die Schwierigkeiten und Beschränkungen, die dabei auftreten können.

6.2 Ausblick

Obwohl die Ergebnisse vielversprechend sind, gibt es noch Raum für Verbesserungen und Weiterentwicklungen in diesem Bereich. In Zukunft könnte es möglich sein, automatische Texterkennungstechnologien in PDF-Extraktionsprozesse zu integrieren, um eine höhere Genauigkeit und Zuverlässigkeit zu gewährleisten. Eine andere mögliche Erweiterung wäre die Integration von maschinellem Lernen, um eine bessere Identifizierung von Texten in PDF-Dateien zu ermöglichen. Außerdem könnte die Integration von Textanalysesoftware in die Extraktionsprozesse die Analyse von extrahierten Texten automatisieren und vereinfachen. Auch die Erkennung von Nicht-Text-Komponenten wie Bildern, Infografiken und anderen Arten von Abbildungen muss unbedingt verbessert werden. Ein weiteres Hindernis, das es zu überwinden gilt, ist die Optimierung und Anpassung der Textextraktion aus PDF-Dateien, die in anderen Sprachen als Englisch verfasst sind. Darüber hinaus können auch Methoden zur Extraktion von Texten aus anderen Dokumentformaten, wie z.B. Microsoft Word, Excel oder PowerPoint, untersucht werden. Die Kombination von Text-

extraktion aus verschiedenen Dokumentformaten kann zur Entwicklung von umfassenden Tools zur Automatisierung von Arbeitsprozessen beitragen.

Die Extraktion von Text aus PDF-Dateien wird in Zukunft ein wesentlicher Bestandteil der Datenanalyse sein und in einer Vielzahl von Anwendungsbereichen hilfreich sein. Die Verarbeitung von PDF-Dateien wird in Anbetracht der zunehmenden Verbreitung digitaler Technologien zu einer immer wichtigeren Fähigkeit. Es ist zu erwarten, dass in nicht allzu ferner Zukunft immer mehr Unternehmen und Organisationen auf PDF-Extraktionsverfahren angewiesen sein werden, um wichtige Informationen aus PDF-Dateien zu extrahieren. Infolgedessen wird dieser Sektor weiter an Bedeutung gewinnen und eine Vielzahl neuer Fortschritte und Durchbrüche mit sich bringen.

Literatur

[1] Hannah Bast und Claudius Korzen. *A benchmark and evaluation for text extraction from PDF*. 19. Juni 2017. DOI: 10.5555/3200334.3200346. URL: http://ieeexplore.ieee.org/document/7991564/.

[2] Adobe Marketing Cloud. *PDF ist 20 Jahre alt geworden und bekannter als der neue Papst*. 17. Juni 2013. URL: https://blog.adobe.com/de/publish/2013/06/17/pdf-ist-20-jahre-alt-geworden-und-bekannter-als-der-neue-papst (besucht am 06. 11. 2022).

[3] Gesellschaft Für Informatik. *Informationsextraktion*. 16. Mai 2014. URL: https://gi.de/informatiklexikon/informationsextraktion (besucht am 06. 11. 2022).

[4] Yorick Wilks. "Information extraction as a core language technology". In: *Information Extraction A Multidisciplinary Approach to an Emerging Information Technology* (1997), S. 1–9. DOI: 10.1007/3-540-63438-x_1. URL: http://dx.doi.org/10.1007/3-540-63438-x_1.

[5] Jing Jiang. "Information Extraction from Text". In: *Mining Text Data* (2012), S. 11–41. DOI: 10.1007/978-1-4614-3223-4_2. URL: http://dx.doi.org/10.1007/978-1-4614-3223-4_2.

[6] Andreas Henrich. *Information Retrieval 1*. 2008, S. 11–22.

[7] Norbert Fuhr. *Einführung in Information Retrieval Skriptum zur Vorlesung im WS 11/12*. 2011), S. 3–5.

[8] Librarianship Studies. *Information Retrieval*. 21. Feb. 2020. URL: https://www.librarianshipstudies.com/2020/02/information-retrieval.html (besucht am 06. 11. 2022).

[9] *IE IR - Information Extraction and Information Retrieval | TALP Language and Speech Technologies and Applications*. URL: https://www.talp.upc.edu/EXPERTISE-AREA-DETAIL/430 (besucht am 06. 11. 2022).

[10] John Ockerbloom. "Archiving and Preserving PDF Files". In: *RLG DigiNews* 5.1 (15. Feb. 2001).

[11] Adobe. *PDF Reference fifth edition*. 2004, S. 1–9.

[12] Mads R. Dahl, Eivind O. Simonsen und Christian B. Hoyer. "*What you see is not what you get in the PDF document format*". In: *Health Informatics Journal* 17.1 (März 2011), S. 24–32. DOI: 10.1177/1460458210397851. URL: http://dx.doi.org/10.1177/1460458210397851 (besucht am 11. 11. 2022).

[13] *Fünf PDF-Formate und ihre Einsatzmöglichkeiten*. 14. Aug. 2020. URL: https://www.foxit.com/blog/funf-pdf-formate-und-ihre-einsatzmoglichkeiten/ (besucht am 11. 11. 2022).

[14] Bill Willoughby. *7 Different PDF File Formats You Should Know About*. URL: https://www.gflesch.com/blog/7-different-pdf-file-formats-you-should-know-about (besucht am 11. 11. 2022).

[15] *Deep Understanding of a Document's Structure*. URL: https://www.researchgate.net/publication/321460638_Deep_Understanding_of_a_Document's_Structure (besucht am 31. 12. 2023).

[16] *Physical and Logical Structure Recognition of PDF Documents*. URL: http://www.bloechle.ch/jean-luc/pub/Bloechle_Thesis.pdf (besucht am 31. 12. 2023).

[17] *PDF Grundlagen*. URL: https://www.pdf-tools.com/public/downloads/whitepapers/Whitepaper-PDF-Grundlagen-DE.pdf (besucht am 31. 12. 2023).

[18] Ashish Choudhary. *Data Extraction from Unstructured PDFs*. 21. Juni 2021. URL: https://www.analyticsvidhya.com/blog/2021/06/data-extraction-from-unstructured-pdfs/ (besucht am 19. 11. 2022).

[19] *Wie extrahiert man Daten aus PDF, ohne zu Programmieren?* URL: https://www.octoparse.de/blog/wie-extrahiert-man-daten-aus-pdf-ohne-zu-programmieren (besucht am 19. 11. 2022).

[20] Mitchell Sloan. *4 Tipps um Daten aus PDFs zu extrahieren*. 1. Okt. 2021. URL: https://www.acodis.io/de/blog/4-tipps-um-daten-aus-pdfs-zu-extrahieren (besucht am 19. 11. 2022).

[21] Christopher Stahl u. a. *DeepPDF: A Deep Learning Approach to Extracting Text from PDFs*. 1. Mai 2018. URL: https://www.osti.gov/biblio/1460210 (besucht am 19. 11. 2022).

[22] David Spisla. *Erkennung von Fließtext in PDF-Dokumenten*. 1. Aug. 2016. URL: https://ad-publications.cs.uni-freiburg.de/theses/Bachelor_David_Spisla_2016.pdf (besucht am 19. 11. 2022).

[23] Fabian Schillinger. *Strukturierte Extraktion von Text aus PDF*. 12. Mai 2015. URL: https://ad-publications.informatik.uni-freiburg.de/theses/Master_Fabian_Schillinger_2015.pdf (besucht am 19. 11. 2022).

[24] *Tiobe Index*. 3. Juni 2022. URL: https://www.tiobe.com/tiobe-index/ (besucht am 26. 11. 2022).

[25] *PyPDF2*. 20. Nov. 2022. URL: https://pypi.org/project/PyPDF2/ (besucht am 20. 11. 2022).

[26] Rucha Sawarkar. *Python Packages for PDF Data Extraction - Analytics Vidhya*. 6. Jan. 2022. URL: https://medium.com/analytics-vidhya/python-packages-for-pdf-data-extraction-d14ec30f0ad0 (besucht am 27. 11. 2022).

[27] dida Datenschmiede GmbH. *How to extract text from PDF files*. URL: https://dida.do/blog/how-to-extract-text-from-pdf (besucht am 26. 12. 2022).

[28] Tamir Hassan und Robert Baumgartner. "Intelligent Text Extraction from PDF Documents". In: *Computational Intelligence for Modelling, Control and Automation* (28. Nov. 2005). DOI: 10.1109/cimca.2005.1631436.

[29] *PDFMiner — pdfminer-docs 0.0.1 documentation*. URL: https://pdfminer-docs.readthedocs.io/pdfminer_index.html (besucht am 27. 11. 2022).

[30] *Programming with PDFMiner — pdfminer-docs 0.0.1 documentation*. URL: https://pdfminer-docs.readthedocs.io/programming.html (besucht am 27. 11. 2022).

[31] *Introduction — PyMuPDF 1.21.0 documentation*. Deutsch. URL: https://pymupdf.readthedocs.io/en/latest/ (besucht am 27. 11. 2022).

[32] *Apache Tika – Apache Tika*. URL: https://tika.apache.org/ (besucht am 27. 11. 2022).

[33] *Tutorialspoint, TIKA - Overview*. URL: https://www.tutorialspoint.com/tika/tika_overview.htm (besucht am 27. 11. 2022).

[34] *tabula-py: Read tables in a PDF into DataFrame — tabula-py documentation*. URL: https://tabula-py.readthedocs.io/en/latest/ (besucht am 03. 12. 2022).

[35] Andrew Treadway. *3 ways to scrape tables from PDFs with Python*. 1. Mai 2020. URL: http://theautomatic.net/2019/05/24/3-ways-to-scrape-tables-from-pdfs-with-python/ (besucht am 03. 12. 2022).

[36] Misha Sv. *How to Extract Tables from PDF using Python*. 17. Okt. 2021. URL: https://www.youtube.com/watch?v=tEFAFQXaOWw (besucht am 03. 12. 2022).

[37] *Educative, What is Camelot?* URL: https://www.educative.io/answers/what-is-camelot (besucht am 03. 12. 2022).

[38] *PyPDF4 Documentation, Cameron Laird, GitHub*. 2019. URL: https://github.com/claird/PyPDF4 (besucht am 03. 12. 2022).

[39] *A few thingz, Working on PDF files with Python*. URL: https://afewthingz.com/ (besucht am 31. 12. 2022).

[40] Tarreq Dahrouj. "Entwicklung einer WossiDiA-Python-Schnittstelle". Bachelorarbeit. Universität Rostock, Sep. 2022.

[41] Miao Zhu und Jacqueline M. Cole. *PDFDataExtractor: A Tool for Reading Scientific Text and Interpreting Metadata from the Typeset Literature in the Portable Document Format*. 29. März 2022. DOI: 10.1021/acs.jcim.1c01198. (Besucht am 13. 03. 2023).

[42] Sigrid Hofmaier. "Schlaflos in Freiburg". In: *Freiburger Wochenbericht* (13. Aug. 2014). URL: https://www.freiburger-wochenbericht.de/home/ (besucht am 13. 03. 2023).

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Anhang A

```
<!ELEMENT PDF-Datei (Pages)>
<!ELEMENT Pages (Page+)>
<!ELEMENT Page (Absaetze*, Bildunterschriften*, Tabellen*)>
<!ELEMENT Absaetze (Absatz+)>
<!ELEMENT Bildunterschriften (Bildunterschrift+)>
<!ELEMENT Bildunterschrift (#PCDATA)>
<!ELEMENT Tabellen (Tabelle+)>
<!ELEMENT Tabelle (td+)>
<!ELEMENT td (#PCDATA)>

<!ATTLIST PDF-Datei version CDATA #REQUIRED>
<!ATTLIST Page number CDATA #REQUIRED>
<!ATTLIST Bildunterschrift align (left|center|right) "left">
<!ATTLIST Tabelle width CDATA #REQUIRED>
<!ATTLIST td rowspan CDATA #IMPLIED>
<!ATTLIST td colspan CDATA #IMPLIED>
```

Listing A.1: Darstellung vom Aufbau der XML-Datei in DTD-Format

```
<?xml version="1.0" ?>
<PDF_DATEI>
  <Pages>
    <Page nr=1>
  <Absaetze>
  <Absatz nr=1> </Absatz nr=1>
  .
  .
  .
  <Absatz nr=n> </Absatz nr=n>
  </Absaetze>

  <Bildunterschriften>   </Bildunterschriften>

  <Tabelle>
  <Tabelle nr=1>
  <tr nr=1>
    <td>Zelle 1</td>...<td>Zelle n</td>
    </tr nr=1>
  .
  .
    <tr nr=n>
    <td>Zelle x</td>...<td>Zelle z</td>
    </tr nr=n>
    </Tabelle nr=1>
  .
  .
  <Tabelle nr=n>  </Tabelle nr=n>
  </Tabelle>
    </Page nr=1>

  <Page nr=2>...</Page nr=2>
  <Page nr=3>...</Page nr=3>
  <Page nr=4>... </Page nr=4>
  <Page nr=5>...</Page nr=5>
  <Page nr=6>...</Page nr=6>
  <Page nr=7>...</Page nr=7>
  <Page nr=8>...</Page nr=8>
  <Page nr=9>...</Page nr=9>
  <Page nr=10>...</Page nr=10>
    .
    .
    <Page nr=n>
  <Absaetze>
  <Absatz nr=1>Die Textextraktion beschreibt als allgemeiner Begriff die gezielte Analyse von
Dateien und Dokumenten, die eine Reihe von Techniken umfasst, deren Aufgabe in erster Linie
darin besteht, Sammlungen von Dateien zu analysieren, die entweder viele oder wenige Texte
enthalten, und nach der Durchf hrung der notwendigen Analyse kommt die zweite Aufgabe. Das
Format ist plattformunabh ngig und erm glicht es, ".............".
  </Absatz nr=1>
  <Absatz nr=2>Im Bereich der wissenschaftlichen und akademischen Forschung fuehren Experten
h ufig viele dieser Operationen Textextraktionsprozesse f r viele Zwecke durch.
Beispielsweise um bestimmte Ergebnisse zu erhalten oder um diese Texte neu zu analysieren,
nachdem sie von vielen nutzlosen Informationen ges ubert und alle st renden Elemente aus
ihnen entfernt wurden, um sie dann in anderen Dateiformaten zu speichern oder um diese
Informationen mit dem Ziel zu verarbeiten, sie in separaten Analyse- oder Suchvorg ngen
erneut zu verwenden.
  </Absatz nr=2>
  </Absaetze>
  <Bildunterschriften>
  Darstellung der physischen Struktur des Dokuments als Baumstruktur [16].
  </Bildunterschriften>
  <Tabelle>
```

```
54   <Tabelle nr=1>
55   <tr nr=1>
56     <td>Tool Name</td>...<td>Formats Supported</td>
57     </tr nr=1>
58     <tr nr=2>
59     <td>pdf2text</td>...<td>HTML, XML</td>
60     </tr nr=2>
61     <tr nr=3>
62     <td>pdftohtml</td>...<td>HTML, XML</td>
63     </tr nr=3>
64     <tr nr=4>
65     <td>pdftohtmlEX</td>...<td>HTML</td>
66     </tr nr=4>
67     <tr nr=5>
68     <td>pdfextract</td>...<td>XML</td>
69     </tr nr=5>
70     </Tabelle nr=1>
71   </Tabelle>
72     </Page nr=n>
73     .
74     .
75     .
76   <Page_y>...</Page_y>
77   </Pages>
78 </PDF_DATEI>
```

Listing A.2: Beispiel für die komplette Darstellung der XML-Ausgabedatei

```
from pdfminer.pdfparser import PDFParser
from pdfminer.pdfdocument import PDFDocument
from pdfminer.pdfpage import PDFPage
from pdfminer.pdfpage import PDFTextExtractionNotAllowed
from pdfminer.pdfinterp import PDFResourceManager
from pdfminer.pdfinterp import PDFPageInterpreter
from pdfminer.pdfdevice import PDFDevice
from pdfminer.layout import LTLine, LAParams, LTFigure, LTTextBox,LTImage,LTTextLine ,LTTextBoxHorizontal
from pdfminer.converter import PDFPageAggregator

#-------------------------------------------------------------------------
#--11111111111------------------------------------------------------------

def rectIntersectVertical(rect1 , rect2 ) ->  bool :
    outerRect = []
    innerRect = []

    if rect1[1] < rect2[1]:
        outerRect = rect1
        innerRect = rect2
    else:
        outerRect = rect2
        innerRect = rect1

    return   innerRect[1] >= outerRect [1] and innerRect[1] < outerRect[3]

#-------------------------------------------------------------------
#---22222222222222---------------------------------------------

def rectIntersectHorizontal(rect1 , rect2 ) ->  bool :
    outerRect = []
    innerRect = []

    if rect1[0] < rect2[0]:
        outerRect = rect1
        innerRect = rect2
    else:
        outerRect = rect2
        innerRect = rect1

    return   innerRect[0] >= outerRect [0] and innerRect[0] < outerRect[2]

#-----------------------------------------------------------------------
#----3333333333333-------------------------------------------------

def addTextInTable(text , page) :
    index = -1
    for rect in page["rect"]:
        index =index +1
        x = rectIntersectVertical(rect , text )
        y = rectIntersectHorizontal(rect , text )
        if x and y :
            page["table"][index].append(text)
            return True # hier bestatigung dass der text in der table und das code endet hier beim ersten return
    return False     # wenn if nicht funktioniert dann erster return nicht funktioniert kommt zu diesem return

#-------------------------------------------------------------------------
#----4444444444444-----------------------------------------------------

# return -1 if not found , otherwise return index in elements of nearst element below the element
def findBelowElement (element , elements) :
    index = -1
    i =-1
    for e in elements: # loop throw element
        i = i+1
        if e[1] < element[1] and e[3] < element[3] : # check e under element
            if index == -1 : # this is the first result
                index = i
            elif e[1] < elements[index] [1]:# replace the prevoise elment as e is under elements[index]
                index =i
    return index

#-----------------------------------------------------------------------------------------------
#-----55555555555555-----------------------------------------------------------------------

def checkImage(page):
    for image in page["images"]:
        belowElementIndex = findBelowElement(image, page["absatz"])
        if belowElementIndex == -1:  # no text below image found
            continue
        # take the result text  from findBelowElement function
        absatz = page["absatz"][belowElementIndex]

        text = absatz[4]
        # if text start with specific words then it's text under image example Abbildung 304
```

```
86          isThisFigure = text.find("abbildung") == 0 or text.find("Abbildung") == 0 or text.find("figure") == 0 or
    text.find("figure") == 0
87
88          if not isThisFigure:
89              continue
90
91          # add the result to under image text
92          res = absatz[:]  # clone the absatz to the figure data
93          page["figure"] .append(res)  # add the data to the figure lists
94
95          # clear the abstatz so it will not be added in the abstatz later
96          absatz[4] = ""
97  #-------------------------------------------------------------------------
98  #-------------------------------------------------------------------------
99
100 # Open a PDF file.
101 fp = open('mypdfmahmoud.pdf', 'rb')
102 #fp = open('mypdfTa.pdf', 'rb')
103 #fp = open('mypdfSa.pdf', 'rb')
104 #fp = open('mypdfTarreq.pdf', 'rb')
105 # Create a PDF parser object associated with the file object.
106 parser = PDFParser(fp)
107 # Create a PDF document object that stores the document structure.
108 # Supply the password for initialization.
109 document = PDFDocument(parser)
110 # Check if the document allows text extraction. If not, abort.
111 if not document.is_extractable:
112     raise PDFTextExtractionNotAllowed
113 # Create a PDF resource manager object that stores shared resources.
114 rsrcmgr = PDFResourceManager()
115 # Create a PDF device object.
116 device = PDFDevice(rsrcmgr)
117 # Create a PDF interpreter object.
118 interpreter = PDFPageInterpreter(rsrcmgr, device)
119
120 # Set parameters for analysis.
121 laparams = LAParams()
122
123  # Create a PDF page aggregator object.
124 device = PDFPageAggregator(rsrcmgr, laparams=laparams)
125 interpreter = PDFPageInterpreter(rsrcmgr, device)
126
127 #-------------------------------------------------------------------------
128 #-------------------------------------------------------------------------
129
130
131 # Process each page contained in the document.
132
133 raw = []  # data of the page
134 # read all text AND  lines
135 for page in PDFPage.create_pages(document):
136     resPage = {}
137     # prepare the raw data
138     resPage["hLines"] = []
139     resPage["vLines"] = []
140     resPage["rawText"] = []
141     resPage["images"] = []
142
143    # prepare the result lists
144     resPage["absatz"] = []
145     resPage["table"] = {}
146     resPage["figure"] = []
147
148     interpreter.process_page(page)
149     layout = device.get_result()
150     for el in layout:
151
152         if isinstance( el , LTLine): # line elemnt
153             hh = el.get_pts()
154             points = hh.split(",")
155             x1 = float(points[0])
156             y1 = float(points[1])
157             x2 = float(points[2])
158             y2 = float(points[3])
159             if y1 == y2 :  # horzontal line
160                 if x2 > x1 :
161                     resPage["hLines"].append([x1,y1,x2,y2])
162                 else:
163                     resPage["hLines"].append([x2,y2,x1,y1])
164             else: # vertical line
165                 if x2 > x1 :
166                     resPage["vLines"].append([x1,y1,x2,y2])
167                 else :
168                     resPage["vLines"].append([x2,y2,x1,y1])
169
170         if isinstance( el , LTTextBoxHorizontal):   # text box  element
```

```
            xzxzxzxz =  el.get_text()
            resPage["rawText"].append([el.bbox[0],el.bbox[1],el.bbox[2], el.bbox[3] , el.get_text()  ])

        if  isinstance(el, LTTextLine):    # no data found here ignore
            dff = el.getText()
            d = 0
        if isinstance(el, LTTextBox)  :# no data found here ignore
            dff = el.get_text()
            x = 0
        if isinstance(el, LTFigure):  # image element
            resPage["images"].append( [el.x0 , el.y0 , el.x1 , el.y1])

    raw.append(resPage)

#--------------------------------------------------------------------------
#-------------------------------------------------------------------------

# dieses codestueck nicht beschreiben in der wissenschaftlichen arbeit
# calc all the rectangles according to the lines intersection on the page
for page in raw :
    page["rect"] = []

    for vLine in page["vLines"]:
        for hLine in page["hLines"]:
            # check if 2 lines intersect
            if vLine[0] >= hLine[0] and vLine[0] <= hLine[3] and hLine[1] >= vLine[1] and hLine[1] <= vLine[3] :
                # vline and hline intersect , make rect from both of them ( create out of 4 corner )
                page["rect"] .append(
                    [
                    min(hLine[0] , hLine[2] , vLine[0] , vLine[2] ) ,
                    min(hLine[1] , hLine[3] , vLine[1] , vLine[3] ) ,
                    max(hLine[0] , hLine[2] , vLine[0] , vLine[2] ) ,
                    max(hLine[1] , hLine[3] , vLine[1] , vLine[3] )
                     ])
                page["table"] [len(page['rect'])-1 ]= [] ## add empty dictionary in order to add text per table


#---------------------------------------------------------
#----------------------------------------------------------------


# if text in table add to table otherwise added to absatz
for page in raw :
    for text in page["rawText"]:
        t = addTextInTable(text , page ) # add text to table cells
        if t == False :
            page["absatz"] .append( text)  # add text to absatz ( normal text ) noimage text , no table

# hier kann man das code verandern, wobei dass wenn das zeile mit underscore endet, wird mit dem nachsten zeile
zusammengefuegt.                                                  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<


# remove text from absatz if the text is under an image and start with Figure , abbildung
for page in raw :
    checkImage(page)


#--------------------------------------------------------------------------------------
#----------------------------------------------------------------------------------------------
# generate xml result -------------------------------------------------

xml= ''
xml = xml +'<?xml version="1.0" encoding="UTF-8"?>'
xml = xml +'<PDF_DATEI>'
xml = xml +'<pages>'

p =-1
for page in raw :
    p = p +1
    xml = xml +'<page n="' + str( p) + '">'

    # add all text ----------------------------------------------------------
    a = -1
    xml = xml +'<Absaetze>'

    first_absatz = page["absatz"][0][4]  # get the text of the first paragraph         meeeeeeeeeeeeeeeeeeeeee
    if not first_absatz.lower().startswith(('inhaltsverzeichnis', 'literatur', 'literaturverzeichnis',
'abbildungsverzeichnis', 'tabellenverzeichnis', 'anhang', 'listings', 'abk rzungsverzeichnis')):
#meeeeeeeeeeeeeeeeeeeeeeeeeeee
        for absatz in page["absatz"]:  # hier muessen die verbesserungen erledigt werden fuer die absetze
    <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
            text = absatz[4]
            if 'http' not in text:
                if not text.lower().startswith(('abbildung', 'figure')):
                    if absatz[0] > 10:  # check if font size is greater than 9
                        words = text.split()  # split the text into words
```

```
                        if len(words) > 8:  # check if there are more than 10 words
                            a += 1
                            lines = text.split("\n")
                            text = ' '.join(lines)
                            xml += '<absatz n="' + str(a) + '">' + text + '</absatz>'
            # a = a +1
            # xml = xml + '<absatz n="' + str(a) +  '">' + absatz [4]+ '</absatz>'


    xml = xml +'</Absaetze>'


    # add all Bildunterschriften-------------------------------------------------------
    xml = xml +'<Bildunterschriften>'

    b = -1
    for Figure in page ["figure"]:
        b = b +1
        xml = xml + '<Bildunterschrift n="' + str(b) +  '">' + Figure [4]+ '</Bildunterschrift>'


    if not first_absatz.lower().startswith(('inhaltsverzeichnis', 'literatur', 'abbildungsverzeichnis',
'tabellenverzeichnis', 'anhang', 'listings', 'abkrzungsverzeichnis')):
        for absatz in page["absatz"]:
            text = absatz[4]
            if text.lower().startswith(("abbildung", "figure")):
                words = text.split()  # split the text into words
                if len(words) > 1:  # check if there are more than 10 words
                    b += 1
                    xml = xml + '<Bildunterschrift n="' + str(b) + '">' + text + '</Bildunterschrift>'


    xml = xml + '</Bildunterschriften>'


    # add all the tables ----------------------------------------------------------------
    t = -1

    xml = xml +'<Tabellen>'
    for tableID in page["table"] :
        table = page["table"][tableID]
        if len(table) == 0 :
            continue
        t = t +1
        xml = xml + '<Tabelle n="' + str(t) +  '">'

        c = -1
        for cell in table :
            c = c +1
            xml = xml + '<Zelle n="' + str(c) +  '">' + cell[4] + '</Zelle>'

        xml = xml+ '</Tabelle>'
    xml = xml +'</Tabellen>'

    xml = xml +'</page>'

xml = xml +'</pages>'
xml = xml +'</PDF_DATEI>'


#write text result to file
f = open("result.xml", "w", encoding='utf-8')
f.write(xml)
f.close()
```

Listing A.3: Der komplette Code

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.

Rostock, den 15. März 2023 Mahmoud Ahmad Alkhamis

mahmoud/alkhamis