Bachelorarbeit

über das Thema

Erkennung von Fließtext in PDF-Dokumenten

David Spisla

01.08.2016

Albert-Ludwigs-Universität Freiburg
Technische Fakultät
Institut für Informatik

Autor: David Spisla
david.spisla@dms80.de

Gutachterin: Prof. Dr. Hannah Bast

Betreuer: Claudius Korzen

Abgabedatum: 01.08.2016

I Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Datum: .....

(Unterschrift)

II Danksagung

An dieser Stelle möchte ich mich bei Frau Prof. Dr. Hannah Bast für die Bereitstellung dieses interessanten Themas bedanken. Außerdem möchte ich mich bei Claudius Korzen für die Betreuung während der Bearbeitung und für das beständige Bemühen um eine Verbesserung der empirischen Analyse bedanken.

Ein besonderer Dank geht an meine Eltern, die mir durch Ihre fortlaufende Unterstützung die Vollendung dieses Studiums ermöglicht haben. Ebenso danke ich meiner restlichen Familie und meinen Freunden für die zahlreichen Ermutigungen im Laufe dieses Studiums.

III Abstract

Diese Arbeit beschreibt ein Verfahren zur Erkennung und Extraktion von Fließtext in PDF-Dokumenten. Eine Extraktion von Fließtext erleichtert die inhaltliche Suche nach Informationen in einem PDF. Das Verfahren nutzt Techniken des Maschinellen Lernens, um den Fließtext in einem ersten Schritt von den übrigens Bestandteilen (Tabellen, Bilder, Überschriften, Referenzen, usw.) auszusondern. Dies erleichtert den zweiten Schritt, in welchem zusammengehörende Abschnitte, die z.B. durch Bilder oder ein Seitenende getrennt werden, erkannt und zusammengeführt werden. In beiden Schritten wird die Erkennung mittels sorgfältig ausgewählter Featurevektoren ermöglicht. Dies ermöglicht zum Schluss eine Zusammenführung aller Fließtextelemente, die dann für eine inhaltliche Suche genutzt werden können. Trainings- und Testphasen werden mithilfe der Python-Bibliothek *scikit-learn* durchgeführt, welche eine Reihe von bekannten Lernalgorithmen bereitstellt.

Die empirische Analyse hat für alle Lernmodelle nahezu identische Ergebnisse ergeben. 2 Varianten (einmal je 100 Trainings- und je 100 Testdaten und einmal 200 Trainingsdaten, die auch als Testdaten verwendet werden) werden ausgetestet. Bei Variante 1 werden für die Erkennung von Fließtext folgende durchschnittliche Analysewerte ermittelt: *precision*= 98 %, *recall*= 98 % und *f1-score*= 98 %. Bei Variante 2 liegen diese Werte alle bei 97 %. Für die Erkennung der zusammengehörenden Abschnitte werden in allen Analysewerten 100 % erreicht. Aufgrund der günstigen Laufzeit kommt das Lernmodell *Bernoulli Naive Bayes* im eigentlichen Extraktions-Programm zur Anwendung.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Anzahl wissenschaftlicher Artikel, die elektronisch veröffentlicht werden, ist in den letzten Jahren rasant gestiegen. Dominierend ist hierbei das PDF-Dokumentenformat, welches erstmalig 1993 durch das Unternehmen Adobe Systems veröffentlicht wurde. Das Format ist plattformunabhängig und bietet u.a. den Vorteil, dass Informationen über die Schriftart, -größe und -farbe genauso in das Dokument integriert sind wie Tabellen und Bilder [ISO08]. Daher wird das Format von vielen Organisationen und Autoren verwendet, um Texte und Informationen aller Art digital zu publizieren.

Informationen in PDF-Dokumenten zu erkennen, kann für viele Anwendungsfälle wichtig sein. Zu denken ist hierbei an eine Aufbereitung des Inhaltes für Menschen mit Sehbehinderungen, ein Transfer der Inhalte in andere Dateiformate oder eine generelle Verbesserung der Lesbarkeit durch Veränderung der Schriftgröße und ein Entfernen störender Elemente wie Referenzen, Tabellen, Bildern, u.a.. Eine weitere interessante Anwendungsmöglichkeit bietet sich im Bereich der semantischen Suche, in welcher versucht wird, den Inhalt der Suchanfrage zu verstehen und auf dieser Basis die entsprechenden Ergebnisse zu generieren. Anwendung findet diese Art der Suche z.B. in der Suchmaschine *Broccoli*, die am Lehrstuhl für Algorithmen und Datenstrukturen an der Universität Freiburg entwickelt wird [BBBH12]. Damit *Broccoli* auf die Inhalte eines wissenschaftlichen Artikels zugreifen kann, müssen diese Inhalte im Vorfeld aus dem Dokument extrahiert werden d.h. die Extraktion von inhaltlich zusammenhängenden Sätzen, in denen die relevanten Informationen ablesbar sind. Eine solche Suche kann auch für das System *icecite* verwendet werden. *Icecite* ist ein Dokumentenverwaltungsprogramm für wissenschaftliche Artikel im PDF-Format, welches am gleichen Lehrstuhl entwickelt wird [BK13]. Es bietet u.a die Möglichkeit, automatisiert Metadaten und Referenzen zu extrahieren, per Mausklick die entsprechenden Referenzartikel aus dem Internet herunterzuladen und Annotationen zu erstellen. Das System arbeitet webbasiert, ein Herunterladen von Software ist nicht notwendig. Dokumente können aber auch gelesen oder annotiert werden, wenn der Rechner offline ist.

Das hier vorgestellte Verfahren basiert auf Systeme, die von *icecite* verwendet werden und verfolgt das Ziel, den eigentlichen Fließtext aus allen übrigen Elementen eines PDF zu extrahieren. Fließtext bezeichnet einen in einem beliebigen Schriftdokument durchgängigen Text, der unabhängig von Überschriften, Tabellen, Abbildungen, Fußnoten, Referenzangaben und Ähnlichem eine sprachlich-syntaktische und logisch-semantische Bedeutungseinheit darstellt und daher unabhängig von diesen Elementen gesetzt werden

kann. Der Einfachheit halber wird im folgenden Fließtext als *relevant* und kein Fließtext als *irrelevant* bezeichnet. Ist die Extraktion der relevanten Elemente erreicht, so erleichtert dies eine Suche nach bestimmten Informationen.

1.2 Schwierigkeiten

Das Extrahieren bestimmter Informationen aus einem solchen Dokument gestaltet sich jedoch nicht einfach, da PDF ein rein layout-basiertes Format ist, d.h. das PDF enthält z.B. nur die Informationen, welcher Buchstabe in welcher Schriftart an welcher Stelle im Dokument steht. Der strukturelle Zusammenhang zwischen allen Buchstaben ist jedoch meistens nicht ablesbar. Doch genau dies ist für eine inhaltliche Suche im Fließtext wichtig. Es gibt bereits viele Programme, die mit sehr einfachen Mitteln versuchen, relevanten Text aus PDF zu extrahieren. Dies geschieht jedoch unzureichend, da einzelne Sätze durch Abbildungen oder Tabellen getrennt werden und viele irrelevante Elemente nicht entfernt werden. Es existieren jedoch bereits regelbasierte Verfahren, die das Ziel einer strukturierten Extraktion verfolgen. Allerdings arbeiten diese Verfahren in der Praxis oftmals nicht zufriedenstellend, da sie aufgrund ihres statischen Regelwerkes wenig flexibel auf ungewöhnliche Layout-Elemente bzw. Unregelmäßigkeiten und Fehler im Dokumentenaufbau reagieren. Durch Maschinelles Lernen sollen diese Nachteile überwunden werden. Die Möglichkeiten Maschinellen Lernens werden bisher überwiegend für die Extraktion von Metadaten (Titel, Autor, Datum der Veröffentlichung, u.a.) genutzt, siehe Kapitel 2.

Diese Technik bietet die Möglichkeit, Wissen empirisch zu gewinnen. Hierbei werden einem System sog. Trainingsdaten zur Verfügung gestellt, anhand derer es wiederkehrende Muster und Gesetzmäßigkeiten erkennt. Im Anschluss daran kann das System unbekannte Daten bzw. Testdaten analysieren und Wissen extrahieren. Diese Technik wird in vielen Bereichen der Informatik eingesetzt, wie z.B. in der Bildverarbeitung und -analyse und im Bereich des *Information Retrieval* zur Klassifizierung von Informationen. Es existieren bereits zahlreiche Bibliotheken in den verschiedensten Programmiersprachen, die allesamt solche Lernsysteme anbieten. Das hier vorgestellte Verfahren nutzt die Python-Bibliothek *scikit-learn* [SKLEARN].

Die Qualität eines Systems, das Maschinelles Lernen zur Klassifikation einsetzt, ist in erster Linie durch die bereitgestellten Trainings- und Testdaten als auch durch die Auswahl geeigneter Featurevektoren bedingt. Featurevektoren enthalten parametrisierbare Eigenschaften von Daten in Form eines Vektors. Die verschiedenen Dimensionen eines Vektors repräsentieren somit unterschiedliche Eigenschaften eines Datenmusters. Ein Feature entspricht somit der Eigenschaft eines Datenmusters. In Kapitel 3 findet der Leser detaillierte Erläuterungen zum Maschinellen Lernen sowie den hier verwendeten Feature-

vektoren und Lernverfahren.
Um jedoch Trainings- und Testdaten aus einem PDF zu erstellen, müssen im Vorfeld geeignete Informationen aus einem PDF extrahiert und entsprechend aufbereitet werden. Dies soll gerade durch die oben erwähnten Systeme aus *icecite* erreicht werden. Das folgende Unterkapitel diskutiert die Möglichkeiten und Schwierigkeiten, die mit der Aufbereitung der Trainings- und Testdaten in diesem Verfahren verbunden sind.

1.3 Lösungsansatz

Das Erstellen geeigneter Trainings- und Testdaten kann u.U. zu den Nachteilen des überwachten Maschinellen Lernens zählen. Bevor überhaupt unbekannte Daten analysiert werden können, braucht es im Vorfeld eine Trainingsphase, die je nach Implementierung Rechenzeit beansprucht und damit die Performance eines Systems beeinträchtigen kann. Ebenso sollten die Trainingsdaten als solche, also hier z.B. konkrete PDF-Dokumente, sorgfältig ausgewählt werden, damit das System anhand einer möglichst hohen Vielfalt an unterschiedlichen Layoutelementen lernen kann. Wählt man viele Dokumente aus, die sehr homogen in ihrem Aufbau sind, so wird das System in einer späteren Testphase ein PDF mit abweichender Gestaltung eventuell ungenügend klassifizieren können. Um für das hier vorgestellte Verfahren geeignete Daten bereit zu stellen, wird auf das bereits o.g. System *icecite* zurückgegriffen. Darin befindet sich ein Verfahren, das es ermöglicht, aus einem PDF die einzelnen Elemente (z.B. Wörter, Textzeilen und Paragraphen) herauszulesen und systematisch im tsv-Format darzustellen (*tsv = tab-separated values*). Dieses Format ist eine spezifisch angeordnete Textdatei zur Speicherung von einfach strukturierten Daten. In diesem Fall werden die einzelnen Informationen in einer Zeile durch ein TAB-Zeichen getrennt. Die erste Zeile benennt die Art der Information und die folgenden Zeilen enthalten die einzelnen Textelemente eines PDF-Dokumentes entsprechend aufgelistet.

Um die Position, Schriftart, Schriftgröße und Farbe der einzelnen Buchstaben zu erkennen, wird innerhalb dieses Verfahrens die *JAVA* Bibliothek *Apache PDFBox* verwendet. Mit *PDFBox* lassen sich sowohl einzelne Buchstaben als auch ganze Wörter extrahieren. Insgesamt werden mithilfe dieses Verfahrens 17 Information (z.B. das Textelement als solches, die Position im Dokument, häufigste Schriftart -größe und -farbe, u.a.) über ein extrahiertes Textelement ermittelt. Wie in Abbildung 1 zu sehen ist, enthält z.B. das Feature „Text“ das extrahierte Textelement als solches. Diese Informationen sind für die Erstellung der Featurevektoren von Bedeutung.

Innerhalb von *icecite* befindet sich ein weiteres Verfahren, dass aus einer TeX-Datei (im folgenden TEX_FILE genannt) die einzelnen Textelemente extrahiert und in eine txt-Datei

Eigenschaft	Text	Seite	minX	minY	maxX	maxY	häufigste Schriftart	häufigste Schriftgröße	häufigste Schriftfarbe	(...)	Rolle
paragraph	Reducing quasi-ergodicity (...)	1	126.0	626.0	484.2	663.4	font-33	17.2	color-0		title
line	Reducing quasi-ergodicity (...)	1	126.0	647.9	484.2	663.4	font-33	17.2	color-0		title
line	by Tsallis Monte Carlo (...)	1	188.9	626.0	424.0	641.5	font-33	17.2	color-0		title
paragraph	Masao Iwamatsu†∗and Yutaka Okabe† ∗Department of (...)	1	134.6	526.9	475.6	609.1	font-32	12.0	color-0		unknown
line	Masao Iwamatsu†∗and Yutaka Okabe†	1	208.1	598.9	401.6	609.1	font-32	12.0	color-0		unknown

Abbildung 1: Schematische Darstellung des Ausschnittes einer tsv-Datei (im Folgenden TSV_FILE genannt) mit den entsprechenden Elementen eines PDF-Dokumentes. Die Markierung „paragraph“ wird einem ganzen Paragraphen zugewiesen. Unter jedem Paragraphen sind die einzelnen Zeilen dieses Paragraphen angegeben, daher hier die Bezeichnung „line“. *minX, minY, maxX, maxY* geben die jeweiligen Koordinaten des umschließenden Rechtecks des Paragraphen an. *Rolle* meint die Funktion innerhalb des Dokumentes. Die übrigen Werte sind selbsterklärend, wobei sich die Bezeichnung „häufigste“ auf Basis der Buchstaben errechnen lässt.

(im folgenden TXT_FILE genannt) schreibt. Die einzelnen Elemente sind ebenfalls mit zusätzlichen Informationen versehen, wie die „Rolle“ eines Paragraphen (z.B. abstract, section, text, usw.), die Start- und Endzeile, die Koordinaten im TEX_FILE und der Text des Paragraphen als solches. Das TXT_FILE hilft dabei, den entsprechenden Einträgen des Featurevektors die korrekte Klassifizierung zu geben. In diesem Sinne handelt es hier um ein sog. *Supervised Learning* d.h. für alle Eingaben sind die entsprechend korrekten Ausgaben vorhanden. Das hier vorgestellte Klassifizierungsverfahren arbeitet also mit den Daten, welches die beiden Verfahren aus *icecite* liefern. Etwaige Fehlertoleranzen werden nicht eingebaut.

Rolle	Startzeile	Endzeile	Koordinaten im TEX_FILE	Text
title	5	6	(1;125,99;529,35;484,25;663,38]), (1;125,79;180,31;386, 83; 187,98])	Reducing quasi-ergodicity (...)
authors	10	10	(1;[125,79;460,26;457,18;505,53])	Masao Iwamatsu[formula]
abstract	24	31	(1;[153,07;375,11;457,18;465,72])	A new Monte Carlo scheme based on the system of (...)
heading	43	43	(2;[125,79;695,17;241,09;705,13])	Introduction
text	45	59	(2;[125,79;521,77;484,45;678,39])	The ergodic hypothesis is fundamental to statistical mechanics. This (...)

Abbildung 2: Schematische Darstellung des Ausschnittes eines TXT_FILE mit den entsprechenden Elementen eines TEX_FILE. Die „Rolle“ meint die Funktion eines Paragraphen innerhalb des Dokumentes (z.B. abstract, authors, text, usw.). Zusätzlich erhält man die Start- und Endzeile des Elementes im TEX_FILE, die Koordinaten des Textelementes im TEX_FILE und den Text als solchen. Die Koordinaten beinhalten als ersten Wert die Angabe der Seitenzahl. In den eckigen Klammern sind der Reihe nach *minX, minY, maxX, maxY* angegeben, welche die jeweiligen Koordinaten des umschließenden Rechteckes des Paragraphen beinhalten. Ist ein Paragraph im PDF aufgeteilt (z.B. durch einen Seitenumbruch), so erhalten wir zusätzlich die Angaben für den Beginn der Fortsetzung (ebenfalls mit Seitenzahl und Koordinaten).

1.3.1 Fließtext erkennen

In der Klasse *CreateCSV* innerhalb des eigens erstellen Python-Programmes *create_csv.py* wird die Extraktion der Daten realisiert (siehe Kapitel 7.2). Zunächst werden mit den Funktionen *read_from_tsv_file* bzw. *read_from_txt_file* das TSV_FILE als auch das TXT_FILE eingelesen. Die Funktion *extract_feature_values_isText* erlaubt nun eine Extraktion derjenigen Daten, die für die Erstellung der Featurevektoren zur Fließtexterkennung notwendig sind. Eine Erläuterung zu den Featurewerten ist in Kapitel 3.3 zu finden.

Die Ermittlung der Featurevektorenwerte erfolgt nun in der Funktion *create_csvIsText*. Speziell für die Erstellung der Trainings- und Testdaten ist hier die Unterfunktion *identify_text* von Bedeutung. Die Schwierigkeit in der Klassenzuordnung zur Fließtexterkennung ergibt sich aus dem Umstand, dass es eben kein zuverlässiges Verfahren gibt, das eine solche Zuordnung erlaubt. Um eine äußerst aufwendige, manuelle Zuordnung zu vermeiden, ist es im Laufe des Entwicklungsprozesses gelungen, durch eine Anpassung der Parser (zur Erstellung eines TSV_FILE bzw. TXT_FILE) aus dem System *icecite*, eine brauchbare Klassifizierung zu erstellen. Hierzu werden die Koordinaten eines jeden tsv-Paragraphen (im folgenden TSV_PARA genannt), den es zu klassifizieren gilt, mit den Koordinaten derjenigen txt-Paragraphen (im folgenden TXT_PARA genannt) verglichen, die auf der gleichen Seite vorkommen. Der Parser zur Erstellung des TXT_FILE markiert Fließtext ziemlich zuverlässig, kann dies jedoch nicht aus einem PDF selbst, sondern nur aus dem zugehörigen TEX_FILE erstellen. Daher werden auch nur diejenigen TXT_PARA betrachtet, die als Fließtext markiert sind. Das Aussortieren aller übrigen Textelemente ist bereits beim Einlesen des TEX_FILE vollzogen worden. Der Vergleich wird durch die Hilfsfunktion *check_for_overlap* ermöglicht. Da die Werte der Koordinaten aus dem TSV_FILE und TXT_FILE nicht immer identisch sind, wird eine Überlappung der Rechtecke geprüft, die sich aus den Koordinaten errechnen lässt. Kommt es zu einer Überlappung, so wird dieser Paragraph als Fließtext markiert.

1.3.2 Zusammengehörende Absätze erkennen

Auch hier wird die Erstellung der csv-Daten (im folgenden CSV_DATA genannt) in der Klasse *CreateCSV* realisiert. Während für die Erstellung der Featurevektorenwerte zur Fließtexterkennung auf die Daten der entsprechenden Werte in den allen Paragraphen des TSV_FILE zugegriffen wird, erfolgt die Erstellung der Daten zur Erkennung zusammengehörender Textelemente auf Grundlage der einzelnen Zeilen eines TSV_PARA (siehe Kapitel 3.4). Diese werden ebenfalls in der Funktion *read_from_tsv_file* extrahiert und dann in der Funktion *extract_feature_values_isSection* entsprechend verarbeitet. Die

Daten des TXT_FILE werden hier nicht benötigt. Die eigentliche Erstellung der csv-Datei (im folgenden CSV_FILE genannt) erfolgt dann in der Funktion *create_csvIsSection*.

1.3.3 Visualisierung der markierten Trainings- und Testdaten

Sind die CSV_DATA erstellt, so lassen sich mit der Funktion *markText* die für die Trainings- und Testphase erstellten Klassenzuordnungen visualisieren. Hierbei werden in das betreffende PDF die als Fließtext markierten Textelemente farblich umrandet. Auch die Klassenzuordnungen des zweiten Featurevektors sind darin integriert. Beginnt ein neuer Fließtextabschnitt, so erhält dieser eine grüne Umrandung und das Label „New“. Ein Folge-Paragraph wird orange umrandet und mit dem Label „Old“ versehen (siehe Abbildung 3).

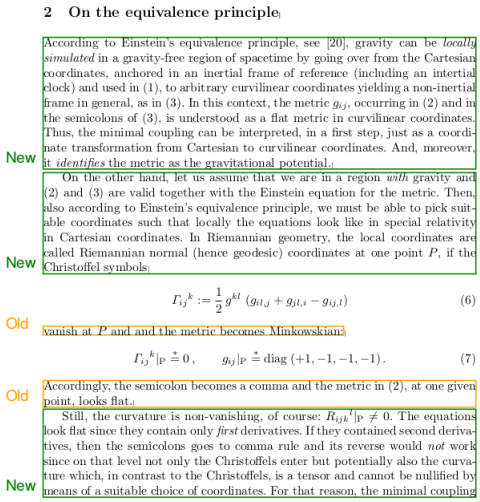

Abbildung 3: Ausschnitt eines PDFs, dessen Fließtextelemente anhand der erstellen Trainings- und Testdaten markiert wurden. An diesem Beispiel ist zu sehen, dass Folge-Paragraphen korrekt orange umrandet und mit dem Label „Old“ versehen werden, weil sie zu dem vorherigen Fließtextelement gehört. Beginnt ein neuer Fließtextabschnitt, so erhält dieser eine grüne Umrandung und das Label „New“.

1.3.4 Auswahl geeigneter Verfahren

Als eigentliche Lernmodelle werden für dieses Verfahren 5 bekannte Modelle eingesetzt: Logistic Regression, Naive Bayes, Decision Trees, k-Nearest-Neighbors und die Support Vector Machine. Eine ausführliche theoretische Analyse erfolgt in Kapitel 3.5, während die praktische Anwendung in Kapitel 4 diskutiert wird.

2 Stand der Forschung

2.1 Regelbasierte Verfahren und PDF

Im Bereich der Texterkennung von PDF-Dokumenten lassen sich auf Anhieb viele Programme, Anleitungen, Bibliotheken und wissenschaftliche Publikationen finden. Die meisten dieser Systeme extrahieren den relevanten Text unzureichend. Interessanter hingegen sind die Systeme, die auf Grundlage eines regelbasierten Verfahrens arbeiten. Zu nennen wäre z.B. das Framework *PTX*, welches mit einem festen Regelwerk die hierarchische Struktur eines wissenschaftlichen Artikels extrahiert. Dieses System erstellt jedoch nicht explizit eine Klassifizierung in relevanten und irrelevanten Text, sondern extrahiert Informationen wie Titel, Autor, Abstract, Textkörper oder Literaturverzeichnis und speichert diese in eine XML-Datei [HLT05]. Ein weiteres System ist *Layout-Aware PDF Text Extraction* (LA-PDFText), welches für die *BioNLP*-Community entwickelt wurde [RPHB12]. In 3 Arbeitschritten erreicht das System eine ähnliche Klassifizierung, wie sie hier angestrebt wird, wobei das Regelwerk jedoch in der Literatur als wenig flexibel angesehen wird [SCHILL15]. Beide Verfahren und weitere werden hier ausführlicher diskutiert: [SCHILL15, S. 10ff]. Etwas flexibler gestaltet sich ein System, das 2015 im Rahmen einer Masterarbeit am Lehrstuhl für Algorithmen und Datenstrukturen an der Universität Freiburg entwickelt wurde [SCHILL15]. Es extrahiert in mehreren Schritten irrelevante Elemente aus dem PDF heraus, übrig bleibt zum Schluss ausschließlich der relevante Teil. Angestrebt wird somit das gleiche Ziel wie das hier verfolgte Klassifizierungsverfahren, welches das Ziel jedoch mit Maschinellem Lernen erreicht und sich dadurch in der Vorgehensweise deutlich unterscheidet. Die Nachteile regelbasierter Verfahren sollen so umgangen werden.

Regelbasierte Verfahren haben zwar den Vorteil, dass die formulierten Regeln zumeist kleine Wissenselemente darstellen, die einfach zu verstehen, bequem zu erweitern und damit leicht anzuwenden sind. Durch diesen Aufbau sind die gezogenen Schlüsse leicht nachzuvollziehen und ein Problemlösen mittels eines Regelsystems ist nicht nur effizient sondern stimmt oftmals mit dem menschlichen Problemlösen überein. Nachteile ergeben sich jedoch dadurch, dass bei einer Vielzahl an Regeln die Übersicht verloren geht, was es schwierig macht, den Problemlöseprozess nachzuvollziehen [HAAS06, S.44f.]. Durch den statischen Aufbau kann es leicht passieren, dass bei ungewöhnlichen Layout-Elementen eines PDF-Dokumentes die entsprechenden Regeln nicht greifen. Für das System ist es dann nur schwer möglich, flexibel zu reagieren. Eine Ausnahmebehandlung ist notwendig, die aber das Regelwerk erneut verkompliziert. Maschinelles Lernen kann diese Nachteile umgehen, da es flexibler auf abweichende Muster reagieren kann (siehe Kapitel 3.2).

2.2 Maschinelle Verfahren zur Erkennung von Metadaten

Es existieren einige Verfahren, die mittels Maschinellem Lernen versuchen, Metadaten aus PDF-Dokumenten zu extrahieren. Aus dem Jahr 2003 existiert ein Beispiel, welches das Ziel verfolgt, diese Metadaten digitalen Bibliotheken zur Verfügung zu stellen [HGMZZF03]. Der Fokus liegt hier auf wissenschaftliche Dokumente und das Extrahieren und Klassifizieren der Daten erfolgt mittels einer Support Vector Machine. Das gleiche Ziel verfolgt ein anderer Ansatz mit Conditional Random Fields (CFR) [PM06]. In dem Jahr 2005 veröffentlicht ist hingegen ein Verfahren, aus allgemeinen Dokumenten lediglich den Titel zu extrahieren [HHCMZ05]. Speziell aus wissenschaftlichen Artikeln will *SciPlore Xtract* den Titel extrahieren [BGSF10], aber auch der *Docear's PDF Inspector* [BLGN13]. Erwähnenswert ist auch *HUMB*, welches versucht, Schlüsselwörter in technischen und wissenschaftlichen Dokumenten zu erkennen [LR10]. Es lassen viele weitere Beispiele aufzeigen. Für das hier vorgestellte Verfahren sind diese Systeme jedoch wenig wegweisend, da keines von ihnen eine explizite Klassifizierung in relevanten und irrelevanten Text vornimmt.

2.3 Maschinelle Verfahren zur Erkennung von Fließtext

Näher zu betrachten sind also die Systeme, die neben den Metadaten auch den relevanten Text extrahieren wollen.

SectLabel

SectLabel z.B. ist ein solches System [KLN10], das mit CRFs arbeitet und binäre Featurefunktionen verwendet. Es versucht, die logische Struktur eines Dokumentes zu ermitteln und verwendet dabei, falls zugänglich, Informationen, die mittels *OCR* (Abkürzung für *optical character recognition*, damit lassen sich in Bildern Texte erkennen) ermittelt werden. Dazu gehören Eigenschaften der Schrift und die Positionen des Textes im Dokument. Ausgelegt ist das System für wissenschaftliche Artikel und es besteht aus zwei Komponenten. Die primäre Komponente, *Logical Structure* (LS), erhält den ganzen Text als Eingabe und ermöglicht eine Klassifizierung in 23 Kategorien wie Titel, Überschrift, Autor, Referenzen, Fließtext, u.a.. Die untergeordnete zweite Komponente, *Generic Section* (GS), analysiert ausschließlich die Überschriften des Dokumentes und bietet 13 Kategorien.
Die Hauptkomponente LS arbeitet in der Trainingsphase mit 4 Feature-Klassen: *Location*, *Number*, *Punctuation* und *Length*. In der Featureklasse *Location* wird untersucht, an welcher Position im Dokument sich das Textelement befindet. So lernt das System, dass der Titel sich im oberen Drittel befindet, während Referenzen eher im unteren Teil des Doku-

mentes zu finden sind. *Number* untersucht das Dokument nach bestimmten Pattern, wie etwa „http“, was auf eine Webseite hinweist, oder auch Hinweise auf eine Überschrift, wie „1.1“. Durch *Punctuation* lassen sich z.B. Formeln erkennen, indem nach einer bestimmten Anzahl Klammern gesucht wird. Schließlich bietet *Length* die Möglichkeit, mittels der Zeilenlänge eine Klassifizierung durchzuführen. Relevanter Text wird dabei tendenziell eine höhere Zeilenlänge aufweisen als irrelevanter Text.

Die zweite Komponente GS arbeitet mit 3 Feature-Klassen: *Position*, *First and Second Words* und *Whole Header*. Durch *Position* wird die Überschrift durch die Position innerhalb des Dokumentes klassifiziert. *First and Second Words* untersucht die ersten beiden Wörter aus der Überschrift und durchsucht es nach Schlüsselwörtern. Dadurch ist die semantische Einteilung eines Bereiches möglich. Ein Abschnitt „related work“ kann z.B. die Schlüsselwörter „Related Work“ oder „Literature Review“ in der Überschrift enthalten. *Whole Header* konkateniert alle Überschriften zu einem Erinnerungsmodell, welches im gesamten Trainingsverlauf hilfreich sein kann. Nach Ablauf der Trainingsphase erhält man zum einen das „logical structure model“ als auch das „generic section model“. Diese Modelle enthalten alle relevanten Informationen, mit denen in der Testphase die Elemente eines wissenschaftlichen Artikels klassifiziert werden können.

Obwohl *SectLabel* keine explizite Klassifizierung in relevanten Teil und irrelevanten Teil vornimmt, sind die Feature-Klassen eine interessante Anregung für das hier vorgestellte Verfahren. Laut den Autoren werden für die Analyse der Textelemente in der LS-Komponente akzeptable Performancewerte erzielt (ca. 76 Macro F1) [KLN10, S.16]. Diese Werte lassen sich durch weitere Modifikationen verbessern.

3 Theoretische Analyse

3.1 Maschinelles Lernen im Allgemeinen

Maschinelles Lernen ist ein Forschungsgebiet, welches sich mit der computergestützten Modellierung und Realisierung von Lernphänomenen beschäftigt [GOERZ14, S. 406]. In diesem Zusammenhang ist es hilfreich, den Begriff des Lernens näher zu beschreiben. Die Diskussion um den Begriff des Lernens ist letztlich sehr weitreichend und die folgende Definition kann nur eine erste Orientierung bieten:

> *Lernen ist jede Veränderung eines Systems, die es ihm erlaubt, eine Aufgabe bei der Wiederholung derselben Aufgabe oder einer Aufgabe derselben Art besser zu lösen.* [SIMON83]

Gemeint ist hier die Fähigkeit eines Systems, Wissen aus Erfahrung zu generieren. Das System erhält in einer Trainingsphase viele Beispiele eines bestimmten Datentypus und lernt, wiederholende Muster und Regelmäßigkeiten zu erkennen und sich diese zu merken. In der nun folgenden Testphase kann dieses erlernte Wissen angewendet werden, um bisher unbekannte Daten zu analysieren.

Wir können dies veranschaulichen, indem wir die inzwischen zahlreichen Anwendungsmöglichkeiten solcher Lernsysteme betrachten [ALPAYDIN14, S. 4ff]. Im Bankwesen lässt sich z.B. das Risiko eines Kredites aus Sicht der Bank berechnen, indem die Daten des Antragstellers sowie der Kreditbetrag mit einer Datenbank abgeglichen werden, in der die Daten früherer Kreditvergaben gesammelt sind. Zunächst wird das Lernsystem in der Trainingsphase mit der Datenbank trainiert, um anschließend in der Testphase anhand der aktuellen Daten zu entscheiden, ob der Antragsteller kreditwürdig ist oder nicht. Wir haben hier also den klassischen Fall einer binären Klassifizierung. Ein weiteres großes Anwendungsfeld ist die Mustererkennung, wie z.B. Bildanalyse, Gesichtserkennnung, Handschriftenerkennung, automatische Spracherkennung und verschiedene Formen der Personenerkennung anhand biometrischer Daten. Auch in der Medizin kommen rechnergestützte Lernsysteme zum Einsatz, um die Diagnose von Krankheiten zu verbessern.

Das hier vorgestellte Verfahren lässt sich dem Forschungsbereich des *Information Retrieval* zuordnen. In den letzten Jahren werden hier verstärkt Techniken des Maschinellen Lernens angewendet, sei es im Bereich der Suchmaschinen zur Optimierung von Suchergebnissen oder eben die Klassifikation von Elementen eines PDF-Dokumentes.

3.2 Vor- und Nachteile Maschinellen Lernens

Vorteile

Durch Maschinelles Lernen ist es möglich, die Nachteile regelbasierter Verfahren zu umgehen. Maschinelles Lernen ist in der Lage, flexibel zu reagieren, da kein statisches Regelwerk zum Einsatz kommt. Ausgewählte Trainingsdaten können jederzeit angepasst werden, um die Qualität der Mustererkennung zu verbessern und die zunehmende Rechenleistung moderner Computeranlagen ermöglicht es, viele Daten (z.B. viele PDF-Dokumente mit unterschiedlichen Elementen und Layouts) zu verarbeiten. Im Rahmen des hier vorgestellten Verfahrens sollen dadurch ungewöhnliche Layout-Elemente bzw. Unregelmäßigkeiten und Fehler im Dokumentenaufbau einer korrekten Klassifizierung nicht im Wege stehen. Die rasante Verbreitung computergestützter Lernsysteme zeigt, dass diese Verfahren gute Resultate in den entsprechenden Anwendungsfeldern liefern.

Nachteile

Wie bereits oben erwähnt, kann die Trainingsphase zu einem Nachteil werden. Dies ist dann der Fall, wenn die Datenmenge zu groß ist und damit die vorhandenen Rechenkapazitäten überlastet werden. Andererseits kann es zu einem sog. *overfitting* kommen d.h. eine zu hohe Anzahl an Variablen in einem Featurevektor oder eine zu hohe Anzahl an Trainingsschritten, welche die Erkennungsleistung nicht mehr erhöhen, sondern verringern. Auf weitere mögliche Stolpersteine in der Anwendung Maschinellen Lernens wird hier eingegangen: [DOMINGOS12].

3.3 Featurevektoren zur Fließtexterkennung

Bereits in Kapitel 2.3 wird auf die Bedeutung von Featurevektoren eingegangen. Die Auswahl der geeigneten Features erfolgt über die Informationen aus dem TSV_FILE, die mittels des entsprechenden Verfahrens aus dem System *icecite* extrahiert werden. Jedoch können nicht alle diese 17 Informationen genutzt werden, da einige von Ihnen keine sinnvollen Informationen liefern, die zu einer effektiven Unterscheidung zwischen relevanten und irrelevanten PDF-Elementen führen. Auch gilt es ein *overfitting* zu vermeiden. Daher folgt nun eine Diskussion über die Relevanz derjenigen Informationen, die für das hier verfolgte Vorhaben von Interesse sind:

häufigsteSchriftart, häufigsteSchriftgröße: Hier erhalten wir Informationen über die im Textelement am häufigsten vorkommende Schriftart und Schriftgröße. Relevante Textelemente haben häufig eine Schriftart, die im Dokument am häufigsten vorkommt, da der

größte Teil eines PDFs in der Regel aus Fließtext besteht. Allerdings kann es vorkommen, dass bei PDFs mit vielen Bildern und Referenzen dies nicht der Fall ist. Demnach wird die Schriftart eines jeden Textelementes hier mit der häufigsten Schriftart derjenigen Textelemente verglichen, die im Dokument als „body-text“ markiert sind (eine Erklärung zu „body-text“ findet sich weiter unten). Aus diesen Textelementen wählen wir die häufigste Schriftart aus. Analog ist die andere Information zu sehen. Da der Wert der Schriftgröße im TSV_FILE als Dezimalzahl mit einer Nachkommastelle ausgegeben wird, wird dieser Wert zunächst gerundet und dann zu einer Ganzzahl umgewandelt. Dies soll etwaige Unstimmigkeiten abfangen, die innerhalb eines Dokumentes auftreten können. Diese Informationen sind somit wichtige Features für die Klassifizierung. Die binäre Featurefunktion ist definiert durch:

$$häufigsteSchriftart = \begin{cases} 1 & \text{wenn häufigste Schriftart im Textelement der} \\ & \text{häufigsten Schriftart im Dokument entspricht} \\ 0 & \text{sonst} \end{cases}$$

Analog die Funktion für „häufigsteSchriftgröße“.

minX, minY, maxX, maxY: Die entsprechenden XY-Koordinaten, welche die Start- bzw. Endposition des umschließenden Rechteckes eines Textelementes im PDF angeben. Aus den Werten von *minX* und *maxX* lässt sich die Breite des Textelementes bestimmen. Relevante Textelemente haben häufig eine Breite, die im Dokument am häufigsten vorkommt, da der größte Teil eines PDFs in der Regel aus Fließtext besteht. Allerdings kann es vorkommen, dass bei PDFs mit vielen Bildern und Referenzen dies nicht der Fall ist. Demnach wird die Breite eines jeden Textelementes hier mit der häufigsten Breite derjenigen Textelemente verglichen, die im Dokument als „body-text“ markiert sind (eine Erklärung zu „body-text“ findet sich weiter unten). Aus diesen Textelementen wählen wir die häufigste Breite aus. Ebenfalls wie bei der Schriftgröße erfolgt eine Rundung und Umwandlung der Breite (*maxX* - *minX*) zu einer Ganzzahl. So erhalten wir ein weiteres Feature für die Klassifizierung. Die Höhe eines Textelementes (*maxY* - *minY*) ist hingegen weniger von Bedeutung, da relevante Elemente auch nur aus einer Zeile bestehen können und daher in ihrer Höhe z.B. mit einer Überschrift gleich sein können. Die binäre Featurefunktion ist definiert durch:

$$häufigsteTextbreite = \begin{cases} 1 & \text{wenn Textbreite des Textelementes der} \\ & \text{häufigsten Textbreite im Dokument entspricht} \\ 0 & \text{sonst} \end{cases}$$

Text: Dies ist das Textelement als solches und dieses gilt es zu klassifizieren. Irrelevante Elemente, wie etwa Bildbeschreibungen, enthalten oft zu Beginn bestimmte Schlüsselwörter, die eher selten zu Beginn eines relevanten Elementes auftreten. Dazu gehören z.B. „Figure“, „Table“, „FIG“, u.a.. Indem wir das Textelement untersuchen, ob es mit einem Schlüsselwort beginnt oder nicht, können wir bereits einen Hinweis auf ein relevantes Element finden. Für die Klassifizierung ist dieses Feature hilfreich. Die binäre Featurefunktion ist definiert durch:

$$enthältSchlüsselwort = \begin{cases} 1 & \text{wenn Textelement ein Schlüsselwort enthält} \\ 0 & \text{sonst} \end{cases}$$

Rolle: Diese Information beinhaltet eine Klassifizierung des betreffenden Textelementes basierend auf Regeln. Relevanter Text (zu dem der Text des abstracts nicht dazugehört) erhält die Bezeichnung „body-text“, während etwa Referenzen schlicht als „reference“ bezeichnet werden. Weitere Klassifizierungen sind z.B. „figure-caption“, „table-caption“, „title“, „section-header“, „figure“, u.a.. Dieses Klassifizierungsverfahren liefert jedoch keine zuverlässige Unterscheidung zwischen relevanten und irrelevanten Textelementen, da es vereinzelnd vorkommen kann, dass Formeln oder Teile der Überschrift als „body-text“ markiert werden. Als alleiniges Feature wäre es zu ungenau, jedoch im Verbund mit den anderen Features ein guter Hinweis auf ein relevantes Textelement. Daher wird es in die Liste der Features aufgenommen. Die binäre Featurefunktion ist definiert durch:

$$istBodyText = \begin{cases} 1 & \text{wenn Textelement als body-text markiert ist} \\ 0 & \text{sonst} \end{cases}$$

Aus *Text* und *Rolle* erhalten wir ein weiteres Feature. Zur Unterscheidung zwischen Referenzen und Fließtext, die beide häufig dieselbe Schriftart, -größe und Textbreite besitzen können, führen wir das Feature *istReferenz* ein. Demnach wird ein Textparagraph als 1 markiert, wenn es mit der Rolle *reference* markiert ist. Ist es als *unknown* markiert, so wird mittels String-Patterns geprüft, ob dieses Textelement mit *[.*]* (eckige Klammern mit beliebigem Inhalt) oder *d+.** (1-2 Ziffern mit beliebigem Zeichen danach) beginnt.

Ist dies der Fall, so wird ein Textparagraph ebenfalls mit 1 markiert. Die binäre Featurefunktion ist definiert durch:

$$istReferenz = \begin{cases} 1 & \text{wenn Textelement Referenz ist} \\ 0 & \text{sonst} \end{cases}$$

Es liegt nun ein Featurevektor der Dimension 6 vor. Mittels ihm soll ein unbekanntes Textelement korrekt als relevant oder irrelevant klassifiziert werden d.h. wir haben in diesem Verfahren ein binäres Klassifizierungsproblem vorliegen mit den Ergebnisklassen $C_1 = 1$ (Fließtext) und $C_2 = 0$ (kein Fließtext). Wenn nun eine unbekannte Klassifizierung von Objekten der Menge X (alle Textelemente eines PDFs) erstellt werden soll mit

$$f : X \to \{1, 0\}$$

so wird durch die Featurefunktion

$$\phi : X \to \mathbb{R}^6$$

jedem Textelement $x \in X$ der Featurevektor $\phi(x) = (\phi_1(x), ..., \phi_6(x))$ zugeordnet, wobei $\phi(X) = \{\phi(x) \mid x \in X\}$ als Feature-Raum bezeichnet wird [SCHNITGER11, S.97]. Jedes ϕ_i mit $i \in \{1, ..., 6\}$ ist wie folgt den entsprechenden Features zugeordnet:

$$i = \begin{cases} 1 & \text{häufigsteSchriftart} \\ 2 & \text{häufigsteSchriftgröße} \\ 3 & \text{häufigsteTextbreite} \\ 4 & \text{enthältSchlüsselwort} \\ 5 & \text{istBodyText} \\ 6 & \text{istReferenz} \end{cases}$$

Laufzeitabschätzung

Die einzelnen Funktionen zum Einlesen der Daten aus dem TSV_FILE und TXT_FILE sowie die Extraktion der Daten zur Featurewerterstellung laufen in linearer Zeit durch und gehören damit der Laufzeitklasse $O(n)$ an. Auch die eigentliche Erstellung der csv-Datei in der Funktion *create_csvIsText* erfolgt in linearer Zeit. Jeder TSV_PARA muss zur Klassenzuordnung nur mit denjenigen TXT_PARA verglichen werden, die auf der gleichen Seite des PDFs liegen. Dadurch wird verhindert, dass im Worst Case jeder TSV_PARA mit allen TXT_PARA verglichen wird. Theoretisch ist es denkbar, dass ein PDF mit einer

häufigsteSchriftart	häufigsteSchriftgröße	häufigsteTextbreite	enthältSchlüsselwort	istBodyText	istReferenz	ErgebnisKlasse
0	0	1	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
1	0	0	0	0	0	0
1	0	0	0	0	0	0
0	0	0	0	0	0	0
1	1	0	0	0	0	0
0	0	0	0	0	0	0
1	1	1	0	1	0	1

Abbildung 4: Schematische Darstellung des Ausschnittes eines CSV_FILE zur Fließtexterkennung mit den entsprechenden Werten der Featurevektoren, die je einem Textelement zugeordnet sind. Man erkennt an diesem Beispiel, dass eine Zustimmung in den Features 1-3 sowie 5 in der Ergebnisklasse zu einer positiven Markierung als relevanten Text führt.

Seite aus lauter Fließtextelementen besteht. In diesem Fall wäre die Laufzeit quadratisch, wobei die Anzahl der Paragraphen hier niedrig ist und daher nicht ins Gewicht fällt. Die Laufzeit von *create_csvIsText* liegt also in der Laufzeitklasse $O(n)$, wobei n die Menge aller TSV_PARA ist.

3.4 Featurevektoren zur Erkennung zusammengehörender Fließtextelemente

Aus den bereits in Kapitel 3 teilweise näher vorgestellten 17 Informationen werden nun ebenfalls einige zur Erstellung der Featurevektorenwerte für zusammengehörende Paragraphen verwendet. Wie bereits in Kapitel 1.3.2 angedeutet, werden hier die einzelnen Zeilen eines TSV_PARA berücksichtigt. Jede Zeile wird ebenso wie ein ganzer Paragraph hinsichtlich dieser 17 Informationen aufbereitet. Auch hier folgt nun eine Diskussion über die Relevanz derjenigen Informationen, die für das hier verfolgte Vorhaben von Interesse sind:

minX: In wissenschaftlichen PDFs ist es üblich, dass zu Beginn eines neuen Paragraphen die erste Zeile etwas nach rechts eingerückt ist. Vergleicht man die minX-Koordinaten der ersten und zweiten Zeile eines Paragraphen, so lässt sich leicht feststellen, ob die erste Zeile zu Beginn etwas nach rechts verschoben ist. Ist dies der Fall, so erhalten wir ein sicheres Indiz für eine Einrückung und damit einen Hinweis auf den Beginn eines neuen Paragraphen. Die binäre Featurefunktion ist definiert durch:

$$Einrückung = \begin{cases} 1 & \text{wenn erste Zeile im Paragraph eingerückt ist} \\ 0 & \text{sonst} \end{cases}$$

Rolle: In wissenschaftlichen PDFs ist es ebenfalls üblich, dass nach einer Überschrift

ein neuer Paragraph beginnt, hier aber die erste Zeile nicht eingerückt ist. Wir überprüfen also den vorherigen Paragraphen, inwiefern dieser mit der Rolle „section-heading“ versehen ist. Ist dies der Fall, so lässt sich mit großer Wahrscheinlichkeit sagen, dass es sich hier um einen neuen Paragraphen handelt. Die binäre Featurefunktion ist definiert durch:

$$nachÜberschrift = \begin{cases} 1 & \text{wenn erste Zeile im Paragraph nach einer Überschrift ist} \\ 0 & \text{sonst} \end{cases}$$

Text: In wissenschaftlichen PDFs kann es auch vorkommen, dass viele Fließtextparagraphen aufeinander folgen und neue Paragraphen nicht durch eine Einrückung in der ersten Zeile markiert sind, sondern durch eine Leerzeile. Für diesen Fall wird ein drittes Feature erstellt. Es wird überprüft, ob der erste Buchstabe in der ersten Zeile ein Großbuchstabe ist und ob die letzte Zeile des vorherigen Paragraphen mit einem Punkt endet. Ist dies der Fall, so erhalten wir ein Indiz für einen neuen Paragraphen. Die binäre Featurefunktion ist definiert durch:

$$GroßbuchstabeUndPunkt = \begin{cases} 1 & \text{wenn Großbuchstabe zu Beginn eines Paragraphen} \\ & \text{und letzter Paragraph mit Punkt endet} \\ 0 & \text{sonst} \end{cases}$$

Es liegt nun ein Featurevektor der Dimension 3 vor. Mittels ihm soll ein bereits erkanntes Fließtextelement korrekt als Folge-Paragraph oder als neuer Paragraph klassifiziert werden d.h. wir haben auch hier ein binäres Klassifizierungsproblem vorliegen mit den Ergebnis-Klassen $C_1 = 1$ (Folge-Paragraph) und $C_2 = 0$ (neuer Paragraph). Wenn nun eine unbekannte Klassifizierung von Objekten der Menge X (alle bereits erkannten Fließtextelemente eines PDFs) erstellt werden soll mit

$$f : X \to \{1, 0\}$$

so wird durch die Featurefunktion

$$\phi : X \to \mathbb{R}^3$$

jedem Fließextelement $x \in X$ der Featurevektor $\phi(x) = (\phi_1(x), \phi_2(x), \phi_3(x))$ zugeordnet. Jedes ϕ_i mit $i \in \{1, 2, 3\}$ ist wie folgt den entsprechenden Features zugeordnet:

$$i = \begin{cases} 1 & \text{Einrückung} \\ 2 & \text{nachÜberschrift} \\ 3 & \text{GroßbuchstabeUndPunkt} \end{cases}$$

tsv-Paragraph	Einrückung	nachÜberschrift	GroßbuchstabeUndPunkt	ErgebnisKlasse
8	0	1	1	0
9	1	0	1	0
10	1	0	1	0
12	0	1	0	0
14	0	0	0	1
16	0	0	0	1
18	0	0	0	1
21	0	0	0	1
24	0	0	0	1

Abbildung 5: Schematische Darstellung des Ausschnittes eines CSV_FILE zur Erkennung zusammengehörender Paragraphen mit den entsprechenden Werten der Featurevektoren, die je einem Fließtextelement zugeordnet sind. Die erste Spalte enthält den Index für den entsprechenden Fließtextparagraphen. Mit diesem Index kann das Verfahren intern den entsprechenden Fließtextparagraphen aus dem CSV_FILE identifizieren. Man erkennt an diesem Beispiel, dass mindestens eine Zustimmung in den Features 1-3 zu einer Markierung als neuen Paragraphen führt. Erfolgt keine einzige Zustimmung, so handelt es sich um einen Folge-Paragraphen. Der Featurevektor entspricht somit dem logischen Modelle eines NOR-Gatters mit 3 Eingängen.

Laufzeitabschätzung

Ebenfalls wie bei der Erstellung des CSV_FILE für die Fließtexterkennung liegt die Laufzeit für das Einlesen der Informationen und die Extraktion der Featuredaten in der Klasse $O(n)$. Die eigentliche Erstellung des CSV_FILE in der Funktion *create_csvIsSection* läuft ebenfalls in $O(n)$, da hier nicht zusätzlich auf die Textelemente des TXT_FILE zugegriffen werden muss, sondern die TSV_PARA nur einmal durchlaufen werden. In der optionalen Funktion *markText* werden die TSV_PARA m mal durchlaufen, wobei m die Anzahl der Seiten des PDFs ist. Bei n Paragraphen haben wir hier eine Laufzeit von $O(mn)$, was ebenso zur Laufzeitklasse *O(n)* gehört.

3.5 Analyse unterschiedlicher Lernmodelle

Im Bereich des Maschinellen Lernens gibt es eine Vielzahl an Lernmodellen. Es werden im Folgenden ausschließlich die Verfahren diskutiert, die sich für eine binäre Klassifikation am besten eignen.

3.5.1 Logistic Regression

Allgemeine Regressionsverfahren werden zur Bestimmung eines unbekannten numerischen Attributwertes verwendet und beschreiben damit quantitative Zusammenhänge zwi-

schen Variablen. Sie dienen allgemein der Vorhersage oder Prognose, um z.B. Verkaufszahlen eines Produktes oder das Verhalten von Kunden vorherzusagen [ALPAYDIN14, S. 34ff]. Da in dem hier beschriebenen Verfahren eine dichotomische Zielgröße (hier: 1 = relevant, 0 = irrelevant) vorliegt, sind solche Verfahren nicht geeignet, da außerhalb von [1, 0] liegende Schätzwerte nicht interpretiert werden können. Die *Logistische Regression* ermöglicht hingegen die Bestimmung einer dichotomischen Zielgröße. Sie verfolgt das Ziel, die Auftretenswahrscheinlichkeit eines Ereignisses in Abhängigkeit verschiedener Größen zu ermitteln.

Der Ablauf dieses Verfahrens beginnt zunächst mit einer Modellformulierung (Bestimmung der Hypothese, Einflußgrößen, Zielgrößen). Daran schließt sich die Formulierung einer logistischen Regressionsgleichung an:

$$p_i = \frac{1}{1+e^{-(a+b*x_i)}}$$

wobei p_i die Wahrscheinlichkeit dafür ist, dass ein Untersuchungsobjekt i die Ausprägung 1 erhält in Abhängigkeit von der Einflußgröße x_i. Die Regressionskoeffizienten a und b werden mit dem Maximum-Likelihood-Verfahren geschätzt. Dabei werden sie so bestimmt, dass die Wahrscheinlichkeit des Auftretens der vorhandenen Stichprobendaten maximal ist. Auf Grundlage dieser Gleichung kann im Anschluss eine Auswertung der Daten vorgenommen werden und das Modell in seiner Gesamtheit als auch die Merkmalsvariablen (= Featurevariablen) beurteilt werden [DK03, S. 4ff].

Bewertung

Da die *Logistische Regression* auf die Bestimmung dichotomischer Zielgrößen zugeschnitten ist, kann sie für die hier behandelte Fragestellung als Lernmodell angewendet werden. Dadurch, dass man bei der Logistischen Regression ein Optimierungsproblem löst, ist die Laufzeit abhängig von dem gewählten Optimierungsverfahren. Die Bibliothek *sklearn* implementiert in der Klasse *LogisticRegression* den linearen Solver *LIBLINEAR* [SKLEARNLR]. Daher wird hier ein Klassifizierer mit einer linearen Laufzeit $O(n)$ in Abhängigkeit von der Trainingsmenge n angewendet.

3.5.2 Naive Bayes

Naive Bayes ist ebenfalls ein bekanntes probalistisches Verfahren, welches für $C = 2$ zur binären Klassifikation verwendet werden kann. Es ist ein einfaches Verfahren zur Klassifikation eines Objektes und es wird dabei von der Annahme ausgegangen, dass, vorausgesetzt eine bestimmte Klasse ist gegeben, jedes Feature unabhängig von allen an-

deren Features ist. Soll z.B. ein Textparagraph klassifiziert werden, so betrachtet *Naive Bayes* jedes Feature dieses Objektes einzeln, ohne gleichzeitig bestimmte Korrelationen mit anderen Features zu berücksichtigen. Dabei kommen die Bayeschen Grundannahmen zur Geltung: Seien A und B Ereignisse im Wahrscheinlichkeitsraum Ω und die Wahrscheinlichkeit $A \cap B$ im Raum B als $Pr(A|B)$ geschrieben, dann gilt:

(1) $Pr(A|B) = Pr(A \cap B)/Pr(B)$
(2) $Pr(A|B) * Pr(B) = Pr(B|A) * Pr(A)$

Aus diesen Annahmen lässt sich folgende Berechnungsvorschrift zur Klassifikation ableiten. In der Trainingsphase wird zunächst für jedes Feature errechnet, mit welcher Wahrscheinlichkeit es in einer Klasse einen bestimmten Wert annimmt. Für diesen Zweck bietet *sklearn* verschiedene Möglichkeiten an: Die Gaussche Normalverteilung, die in der Klasse *GaussianNB* realisiert wird. Dann die Bernoulli-Verteilung (Klasse *BernoulliNB*, die besonders für binäre Featurefunktionen geeignet) ist und schließlich eine Multinomial-Verteilung (Klasse *MultinomialNB*) [SKLEARNNB]. Anschließend kann die Wahrscheinlichkeit einer Klasse C, gegeben Paragraph P mit den Featurevariablen $x_1, ..., x_n$, errechnet werden:

$$Pr(C|P) = Pr(C) * \prod_{i=1,...,n} Pr(x_i|C)/Pr(P)$$

Für den hier vorliegenden Fall bedeutet dies, dass, je öfter ein Feature den Wert 1 (bzw. je nach Feature auch 0) in einer bestimmten Ergebnisklasse C annimmt, desto höher auch die Wahrscheinlichkeit ist, dass der dazugehörige Paragraph zu dieser Klasse gehört.

Bewertung

Trotz ihrer Einfachheit erzielen *Naive Bayes*-Klassifikatoren in der Anwendung häufig gute Ergebnisse, weshalb sie im Rahmen des hier vorliegenden Klassifizierungsproblems angewendet werden. Auch dieser Klassifizierer arbeitet in linearer Laufzeit $O(n)$ in Abhängigkeit von der Trainingsmenge n [SKLEARNNB].

3.5.3 Decision Trees

Entscheidungsbäume repräsentieren hierarchisch geordnete Entscheidungsregeln. Für *C=2* handelt es ich um einen geordneten und gerichteten Binärbaum. Jeder Entscheidungsbaum hat mindestens einen Wurzelknoten und beliebig viele innere Knoten, die von

der Anzahl der verwendeten Features abhängen, und mindestens 2 Blätter. Jeder Knoten steht für eine bestimmte Entscheidungsregel (hier: binäre Featurefunktion) und im Falle eines Binärbaumes kann diese Regel nur 2 Werte annehmen. In den dazugehörigen Blättern wird dann die Antwort repräsentiert. Abbildung 6 veranschaulicht die Funktionsweise eines Binärbaumes.

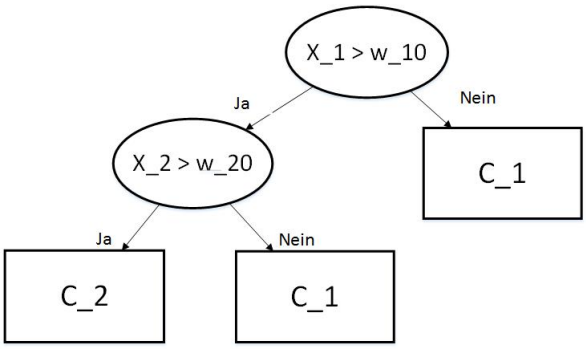

Abbildung 6: Einfaches Beispiel eines binären Entscheidungsbaumes [ALPAYDIN14, S. 214]. Ellipsen (Knoten) symbolisieren Entscheidungsregeln und Rechtecke (Blätter) die entsprechenden Antworten. Im Wurzelknoten wird die Entscheidung getroffen, ob Feature x_1 größer als ein bestimmter Featurewert w_{10} ist. Wenn *Nein*, dann erfolgt die Klassifizierung zur Klasse C_1, wenn *Ja*, dann wird eine weitere Entscheidung getroffen. Ist Feature x_2 kleiner als der Featurewert w_{20}, so ist das Objekt ebenfalls als C_1 markiert, ist es größer, so als C_2.

Bewertung

Durch den einfachen logischen Aufbau sind solche Entscheidungsbäume leicht zu verstehen und die hierarchische Strukturierung erlaubt eine schnelle Lokalisierung des zu klassifizierenden Objektes. Bei einem Binärbaum kann im besten Falle jede Entscheidung die Hälfte aller Entscheidungsfälle (Knoten) eliminieren. Die Laufzeit eines binären Entscheidungsbaumes hängt von der Anzahl an unterschiedlichen Fällen n aus der Trainingsphase ab und liegt im günstigsten Falle bei $O(n * log_2 n)$ [ALPAYDIN14, S. 213ff]. *Decision Trees* finden eine weite Verbreitung, da sie zur Lösung von Entscheidungsfindungen aller Art herangezogen werden können. Mögliche Anwendungsfelder sind die Biologie, Klimaforschung, Gerichtsmedizin und innerhalb des *Information Retrieval*. Sie eignen sich daher für die hier vorfolgte Fragestellung. In *sklearn* sind Entscheidungsbäume als Klasse *DecisionTreeClassifier* realisiert [SKLEARNDT].

3.5.4 k-Nearest-Neighbors

Der k-Nearest-Neighbors-Algorithmus ist ein Klassifikationsverfahren, bei dem die Klassenzuweisung eines Objektes x (hier: der Featurevektor eines Textparagraphen) anhand seiner k nächsten Nachbarn durchgeführt wird. Der Algorithmus funktioniert derart, dass für jedes Objekt zunächst anhand eines Abstandsmaßes (oft ist es der Euklidische Ab-

stand) seine z.B. $k = 5$ Nachbarn ermittelt werden. Ist ein Nachbar der Klasse $C = 1$ zugehörig und 4 Nachbarn der Klasse $C = 0$, so weist der Algorithmus dieses Objekt der Klasse $C = 0$ zu. Im hier vorliegenden Fall wäre ein Textparagraph nicht als Fließtext markiert. k sollte ungerade gewählt werden, um ein Unentschieden zu verhindern. Bei dieser Art der Klassifikation handelt es sich um eine parameterfreie Methode, die davon ausgeht, dass ähnliche Eingaben ähnliche Ausgaben besitzen. Die Klassenzuweisung erfolgt nicht durch eine Funktion mit fest definierten Parametern (wie z.B. bei *Logistic Regression*). Für die Ermittlung der Abstandswerte müssen alle Objekte abgespeichert werden, dies wird auch als *lazy learning* bezeichnet [ALPAYDIN14, S. 185f. und S. 190f.]. Der Algorithmus ist nicht nur für binäre Klassifikation geeignet, sondern auch für Klassifikation mit $C > 2$. Abbildung 7 veranschaulicht die Funktionsweise dieses Verfahrens.

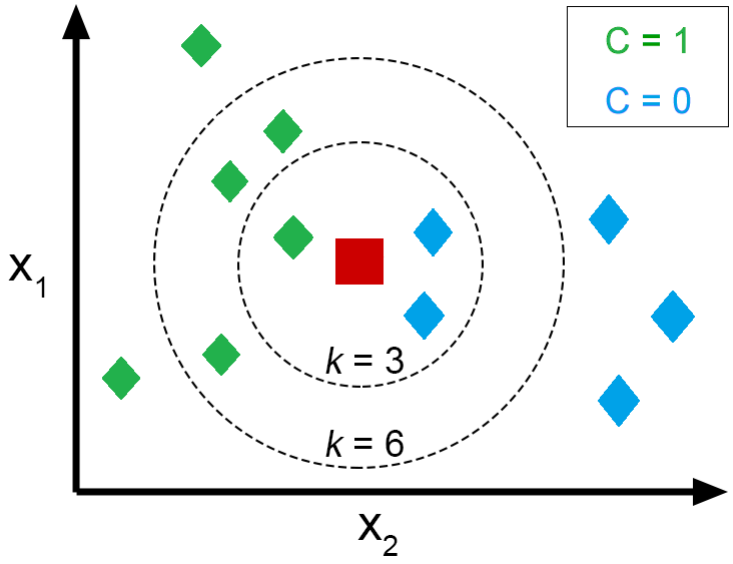

Abbildung 7: Veranschaulichung des k-Nearest-Neighbors-Algorithmus. Der Einfachheit wegen hier lediglich im zweidimensionalen Raum mit 2 Featurevariablen X_1 und X_2. Für ein zu klassifizierendes Objekt (rotes Quadrat) werden entsprechend dem Wert für k seine Nachbarn (Karos) ermittelt. Für $k = 3$ erfolgt eine Zuweisung zur Klasse $C = 0$, für $k = 6$ eine Zuweisung zu $C = 1$.

Bewertung

In diesem einfachen Verfahren muss keine Trainingsphase durchlaufen werden, die Klassifizierungsphase ist jedoch bei größeren Datenmengen teuer und ist abhängig vom angewendeten Suchverfahren. Mit einem linearen Suchverfahren ist die Laufzeit proportional zur Menge der Datenobjekte. Angenommen, es gibt n Objekte mit Dimension d. Dann brauchen wir $O(d)$ um die Distanz zu einem anderen Objekt auszurechnen. Um einen

nächsten Nachbarn auszurechnen, brauchen wir dann $O(nd)$ und schließlich $O(knd)$, um die k nächsten Nachbarn zu ermitteln. Es gibt jedoch Bemühungen, die Laufzeit mithilfe des LSH-Algorithmus (LSH = *locality-sensitive hashing*) zu reduzieren [AI08].
Dennoch ist dieses Verfahren in vielen Klassifizierungs- und Regressionproblemen erfolgreich eingesetzt worden. Gerade auch durch die Eigenschaft, dass es keine parametrische Methode ist, kann es bei Entscheidungsproblemen mit unregelmäßigen Entscheidungsgrenzen erfolgreich sein. Die Bibliothek *sklearn* implementiert in der Klasse *KNeighborsClassifier* einen Basis-Algorithmus, der mit Gewichtungen arbeitet [SKLEARNKNN].

3.5.5 Support Vector Machine

Dieses Verfahren eignet sich sowohl für Regression als auch für Klassifikation. Es gehört zur Klasse der *Large-Margin-Classifier*. Grundlage für dieses Verfahren ist eine Trainingsmenge mit bereits bekannten Klassenzugehörigkeiten. Jedes dieser Objekte wird durch einen Vektor dargestellt, dessen Dimension von der Anzahl der Features abhängig ist. Dies ist in der hier vorliegenden Problemstellung der Fall. Anhand dieser Menge wird in den Vektorraum eine Hyperebene eingebunden, so dass diese als eine Art Trennlinie die vorhandenen Trainingsobjekte in 2 Klassen aufteilt. Dabei wird die Hyperebene so ausgewählt, dass zwischen den am nächsten liegenden Vektoren ein möglichst großer Abstand entsteht. Diese Vektoren sind für die exakte Beschreibung der Hyperebene wichtig und werden deshalb Stützvektoren genannt. Daher auch der Name *Support Vector Machine* für das ganze Verfahren. In der Regel werden zur Erstellung einer solchen Hyperebene die Trainingsdaten mehrmals durchlaufen und die Ebene nach und nach optimiert. Ist eine einfach lineare Trennung zwischen den Datenobjekten möglich, so genügt eine solch einfache, nicht-verbiegbare Ebene aus. In der Realität ist dies jedoch häufig nicht der Fall und daher verfügt die SVM über nichtlineare Kernelfunktionen, die den gesamten Vektorraum mitsamt Trainingsobjekten in einen höherdimensionalen Raum umrechnen. Dort wird eine neue Hyperebene erstellt, mit der sich nun die Trainingsmenge klassifizieren lässt. Bei der Rückführung in den niedrig-dimensionalen Raum erhalten wir eine nicht-lineare Hyperebene [ALPAYDIN14, S.349ff]. Abbildung 8 veranschaulicht die Funktionsweise dieses Verfahrens.

Bewertung

In der Trainingsphase muss sowohl für linear trennbare als auch nicht-linear trennbare Objekte ein quadratisches Optimierungsproblem gelöst werden. Dadurch ist in dieser Phase eine Laufzeit von $O(n^3)$ gegeben [ALPAYDIN14, S. 353]. Trotz des hohen Rechenaufwandes zur Berechnung der optimalen Hyperebene werden SVMs in der Praxis häufig eingesetzt (z.B. in der Bioinformatik, zur Gesichtserkennung oder als Spamfilter)

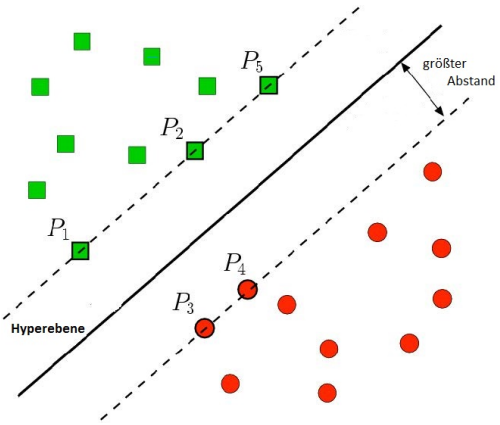

Abbildung 8: Veranschaulichung der Funktionsweise einer SVM mit einfachen, linear trennbaren Datenobjekten zur binären Klassifizierung. Die grünen Quadrate und roten Kreise sind alle Trainingsobjekte in Vektorform $(\vec{x_1}, y_1)...(\vec{x_n}, y_n)$ wobei $y_i \in \{-1, 1\}$. Die SVM errechnet nun eine Hyperebene mit $H = H(\vec{w}, b) := \{\vec{x} \in FS | \langle \vec{x}, \vec{w} \rangle + b = 0\}$. Die Vektoren P_1 bis P_5 sind die Stützvektoren und der Abstand zur Hyperebene beträgt $dist(\vec{x}, H) = |\frac{1}{\|\vec{x}\|}(\langle \vec{x}, \vec{w} \rangle + b = 0)|$ [ALPAYDIN14, S.351ff].

und können dort eine gute Performance erreichen. SVMs ermöglichen nach erfolgreicher Trainingsphase eine schnelle und präzise Klassifizierung, da auch nicht-lineare vorliegende Datenobjekte klassifiziert werden können. Die SVM ist daher auch für Datenobjekte mit vielen Merkmalen gut geeignet. *sklearn* implementiert in der Klasse *SVC* eine SVM. Sämtliche Berechnungen werden mithilfe der Bibliotheken *LIBSVM* und *LIBLINEAR* realisiert [SKLEARNSVM].

4 Empirische Analyse

4.1 Einleitung

Bereits die Erstellung der Featurewerte für die beiden Vektoren hat deutlich gemacht, dass dieses Verfahren auf wissenschaftliche PDFs ausgerichtet ist. Daher werden auch für die Erstellung der Trainings- und Testdaten 200 verschiedene PDFs aus dem Bereich der sog. MINT-Fächer (*MINT* = zusammenfassende Bezeichnung für Mathematik, Informatik, Naturwissenschaft und Technik) ausgewählt, die allesamt auf englisch verfasst sind und vom Lehrstuhl bereitgestellt werden. Speziell für die Trainingsphase werden PDFs ausgewählt, die sich möglichst fehlerfrei zu einem TXT_FILE und TSV_FILE parsen lassen. Eine schlechte Auswahl an Trainingsdaten erschwert die Bildung von guten Modellen für die Testphase. Für beide Phasen wird darauf geachtet, PDFs mit verschiedenen und auch von der Norm abweichenden Layouteigenschaften auszuwählen.

Im Folgenden werden nun die einzelnen Analyseergebnisse (*precision*, *recall*, *f1-score* und die Wahrheitsmatrix) für jedes Lernmodell präsentiert. *precision* gibt an, wieviele der als positiv markierten Testdaten auch wirklich positiv sind. *recall* gibt hingegen an, wieviele der positiven Testdaten vom System als positiv erkannt werden. Der *f1-score* ist der gewichtete Durchschnitt aus *precision* und *recall* und kann daher als eine Art Kompromiss betrachtet werden. *support* gibt an, wie oft die jeweilige Ergebnisklasse im Ergebnisvektor vorkommt. Die Wahrheitsmatrix (engl. = confusion matrix) ist eine Matrix A, in der jedes $A_{i,j}$ die Anzahl der Klassenzuordnungen angibt, von denen man weiß, dass sie der Klasse i angehören, die aber der Klasse j zugeordnet werden. *Sklearn* bietet hier die Klasse *Metrics* mit entsprechender Dokumentation an [SKLEARNMETRICS]. Da *sklearn* ebenso erlaubt, für jedes Lernmodell verschiedene Parameter manuell auszuwählen, wird ebenfalls angeführt, welche Parametereinstellungen zu den besten Ergebnisse geführt haben. Details können der jeweiligen Dokumentation entnommen werden, die für jedes Lernmodell verfügbar sind [SKLEARN]. Die Auswahl des Lernmodells und das Einlesen der Daten erfolgen im Python-Programm *modelling.py* (siehe auch Kapitel 7.3).

Evaluiert werden 2 Varianten. In Variante 1 werden 100 PDFs als Trainingsdaten und die übrigen 100 als Testdaten verwendet. Variante 2 verwendet alle 200 PDFs sowohl zum Trainieren als auch zum Testen.

4.2 Logistic Regression

Für beide Varianten werden, abweichend von den Ergebnissen mit den default-Einstellungen, eine Verbesserung mit der Veränderung des Parameters *class_weight* auf *class_weight={0:0.25, 1:0.75}* erreicht. Dieser Parameter erlaubt es, einer Ergebnisklasse eine stärkere Gewichtung zu geben. Eine weitere Gewichtung zugunsten von Klasse 1 verschlechtert die Ergebnisse. Die Verwendung anderer Solver verändert die Ergebnisse nicht. Für die Erkennung der zusammengehörenden Absätze werden die default-Einstellungen verwendet.

Fließtexterkennung				
class	precision	recall	f1-score	support
0.0	0.99 / 0.99	0.97 / 0.96	0.98 / 0.98	7996 / 15853
1.0	0.95 / 0.94	0.98 / 0.98	0.97 / 0.96	4717 / 9675
avg / total	0.98 / 0.97	0.98 / 0.97	0.98 / 0.97	12713 / 25528

Confusion Matrix		
	0 zugewiesen	1 zugewiesen
0 tatsächlich	7776 / 15291	220 / 562
1 tatsächlich	84 / 204	4633 / 9471

Absatzerkennung				
class	precision	recall	f1-score	support
0.0	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	2851 / 5642
1.0	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	1866 / 4029
avg / total	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	4717 / 9671

Confusion Matrix		
	0 zugewiesen	1 zugewiesen
0 tatsächlich	2851 / 5642	0 / 0
1 tatsächlich	0 / 0	1866 / 4029

Tabelle 1: Analysewerte und Wahrheitsmatrizen für die Fließtext- und Absatzerkennung mit Logistic Regression. Der linke Wert ist für Variante 1 (je 100 Trainings- und Testdaten) und der rechte Wert für Variante 2 (200 PDFs als Trainings- und Testdaten). *class* meint hier die möglichen Ergebnisklassen ($0 = keinFließtext, 1 = Fließtext$ bzw $0 = neuerAbsatz, 1 = Folge{-}Absatz$), *precision* gibt an, wieviele der als positiv markierten Testdaten auch wirklich positiv sind. *recall* gibt hingegen an, wieviele der positiven Testdaten vom System als positiv erkannt werden. Der *f1-score* ist der gewichtete Durchschnitt aus *precision* und *recall* und kann daher als eine Art Kompromiss betrachtet werden. *support* gibt an, wie oft die jeweilige Ergebnisklasse im Ergebnisvektor vorkommt. Die Wahrheitsmatrix (engl. = confusion matrix) ist eine Matrix A, in der jedes $A_{i,j}$ die Anzahl der Klassenzuordnungen angibt, von denen man weiß, dass sie der Klasse i angehören, die aber der Klasse j zugeordnet werden.

4.3 Naive Bayes

Das Bernoulli Naive Bayes Modell erzielt hier die besten Ergebnisse. Für beide Varianten werden, abweichend von den Ergebnissen mit den default-Einstellungen, eine Verbesserung mit der Veränderung des Parameters *class_prior* auf *class_prior=[1, 2]* erreicht. Dieser Parameter erlaubt es, der Wahrscheinlichkeit einer Ergebnisklasse einen Vorrang zu gewähren. Gewährt man der Klasse 1 einen noch stärkeren Vorrang, so verändern sich die Ergebnisse nicht mehr. Für die Erkennung der zusammengehörenden Absätze werden die default-Einstellungen verwendet.

Fließtexterkennung				
class	precision	recall	f1-score	support
0.0	0.99 / 0.99	0.97 / 0.97	0.98 / 0.98	7996 / 15853
1.0	0.95 / 0.95	0.98 / 0.98	0.97 / 0.97	4717 / 9675
avg / total	0.98 / 0.97	0.98 / 0.97	0.98 / 0.97	12713 / 25528

Confusion Matrix		
	0 zugewiesen	1 zugewiesen
0 tatsächlich	7776 / 15381	220 / 472
1 tatsächlich	84 / 214	4633 / 9461

Absatzerkennung				
class	precision	recall	f1-score	support
0.0	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	2851 / 5642
1.0	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	1866 / 4029
avg / total	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	4717 / 9671

Confusion Matrix		
	0 zugewiesen	1 zugewiesen
0 tatsächlich	2851 / 5642	0 / 0
1 tatsächlich	0 / 0	1866 / 4029

Tabelle 2: Analysewerte und Wahrheitsmatrizen für die Fließtext- und Absatzerkennung mit Bernoulli Naive Bayes. Der linke Wert ist für Variante 1 (je 100 Trainings- und Testdaten) und der rechte Wert für Variante 2 (200 PDFs als Trainings- und Testdaten). *class* meint hier die möglichen Ergebnisklassen ($0 = keinFließtext, 1 = Fließtext$ bzw $0 = neuerAbsatz, 1 = Folge{-}Absatz$), *precision* gibt an, wieviele der als positiv markierten Testdaten auch wirklich positiv sind. *recall* gibt hingegen an, wieviele der positiven Testdaten vom System als positiv erkannt werden. Der *f1-score* ist der gewichtete Durchschnitt aus *precision* und *recall* und kann daher als eine Art Kompromiss betrachtet werden. *support* gibt an, wie oft die jeweilige Ergebnisklasse im Ergebnisvektor vorkommt. Die Wahrheitsmatrix (engl. = confusion matrix) ist eine Matrix A, in der jedes $A_{i,j}$ die Anzahl der Klassenzuordnungen angibt, von denen man weiß, dass sie der Klasse i angehören, die aber der Klasse j zugeordnet werden.

4.4 Decision Trees

Für beide Varianten werden auch hier keine Verbesserung mit der Veränderung eines Parameters erzielt. Eher kommt es dadurch zu einer Verschlechterung. Für die Erkennung der zusammengehörenden Absätze werden ebenso die default-Einstellungen verwendet.

Fließtexterkennung				
class	precision	recall	f1-score	support
0.0	0.99 / 0.99	0.97 / 0.97	0.98 / 0.98	7996 / 15853
1.0	0.95 / 0.95	0.98 / 0.98	0.97 / 0.97	4717 / 9675
avg / total	0.98 / 0.97	0.98 / 0.97	0.98 / 0.97	12713 / 25528

Confusion Matrix		
	0 zugewiesen	1 zugewiesen
0 tatsächlich	7776 / 15381	220 / 472
1 tatsächlich	84 / 214	4633 / 9461

Absatzerkennung				
class	precision	recall	f1-score	support
0.0	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	2851 / 5642
1.0	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	1866 / 4029
avg / total	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	4717 / 9671

Confusion Matrix		
	0 zugewiesen	1 zugewiesen
0 tatsächlich	2851 / 5642	0 / 0
1 tatsächlich	0 / 0	1866 / 4029

Tabelle 3: Analysewerte und Wahrheitsmatrizen für die Fließtext- und Absatzerkennung mit Decision Trees. Der linke Wert ist für Variante 1 (je 100 Trainings- und Testdaten) und der rechte Wert für Variante 2 (200 PDFs als Trainings- und Testdaten). *class* meint hier die möglichen Ergebnisklassen ($0 = keinFließtext, 1 = Fließtext$ bzw $0 = neuerAbsatz, 1 = Folge{-}Absatz$), *precision* gibt an, wieviele der als positiv markierten Testdaten auch wirklich positiv sind. *recall* gibt hingegen an, wieviele der positiven Testdaten vom System als positiv erkannt werden. Der *f1-score* ist der gewichtete Durchschnitt aus *precision* und *recall* und kann daher als eine Art Kompromiss betrachtet werden. *support* gibt an, wie oft die jeweilige Ergebnisklasse im Ergebnisvektor vorkommt. Die Wahrheitsmatrix (engl. = confusion matrix) ist eine Matrix A, in der jedes $A_{i,j}$ die Anzahl der Klassenzuordnungen angibt, von denen man weiß, dass sie der Klasse i angehören, die aber der Klasse j zugeordnet werden.

4.5 k-Nearest-Neighbors

Für beide Varianten werden auch hier keine Verbesserung mit der Veränderung eines Parameters erzielt. Eher kommt es dadurch zu einer Verschlechterung. Für die Erkennung der zusammengehörenden Absätze werden ebenso die default-Einstellungen verwendet.

Fließtexterkennung				
class	precision	recall	f1-score	support
0.0	0.99 / 0.99	0.97 / 0.97	0.98 / 0.98	7996 / 15853
1.0	0.95 / 0.95	0.98 / 0.98	0.97 / 0.97	4717 / 9675
avg / total	0.98 / 0.97	0.98 / 0.97	0.98 / 0.97	12713 / 25528

Confusion Matrix		
	0 zugewiesen	1 zugewiesen
0 tatsächlich	7776 / 15381	220 / 472
1 tatsächlich	84 / 214	4633 / 9461

Absatzerkennung				
class	precision	recall	f1-score	support
0.0	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	2851 / 5642
1.0	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	1866 / 4029
avg / total	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	4717 / 9671

Confusion Matrix		
	0 zugewiesen	1 zugewiesen
0 tatsächlich	2851 / 5642	0 / 0
1 tatsächlich	0 / 0	1866 / 4029

Tabelle 4: Analysewerte und Wahrheitsmatrizen für die Fließtext- und Absatzerkennung mit k-Nearest-Neighbors. Der linke Wert ist für Variante 1 (je 100 Trainings- und Testdaten) und der rechte Wert für Variante 2 (200 PDFs als Trainings- und Testdaten). *class* meint hier die möglichen Ergebnisklassen ($0 = keinFließtext, 1 = Fließtext$ bzw $0 = neuerAbsatz, 1 = Folge{-}Absatz$), *precision* gibt an, wieviele der als positiv markierten Testdaten auch wirklich positiv sind. *recall* gibt hingegen an, wieviele der positiven Testdaten vom System als positiv erkannt werden. Der *f1-score* ist der gewichtete Durchschnitt aus *precision* und *recall* und kann daher als eine Art Kompromiss betrachtet werden. *support* gibt an, wie oft die jeweilige Ergebnisklasse im Ergebnisvektor vorkommt. Die Wahrheitsmatrix (engl. = confusion matrix) ist eine Matrix A, in der jedes $A_{i,j}$ die Anzahl der Klassenzuordnungen angibt, von denen man weiß, dass sie der Klasse i angehören, die aber der Klasse j zugeordnet werden.

4.6 Support Vector Machine

Bereits die default-Einstellungen der Klasse *SVC()* ergeben für Variante 1 die besten Ergebnisse. Für Variante 2 konnte die Verwendung des Kernels *kernel='poly'* bei der Erkennung von Klasse 0 geringfügig bessere Ergebnisse erzielen. Der Kernel einer SVM stellt die zentrale mathematische Berechnungsvorschrift einer SVM dar. Die alternativen Varianten der SVM, *NuSVC* und *LinearSVC* aus der Bibliothek *sklearn*, erzielen für beide Varianten geringfügig schlechtere Ergebnisse. Für die Erkennung der zusammengehörenden Absätze werden auch die default-Einstellungen verwendet.

Fließtexterkennung				
class	precision	recall	f1-score	support
0.0	0.99 / 0.99	0.97 / 0.97	0.98 / 0.98	7996 / 15853
1.0	0.95 / 0.95	0.98 / 0.98	0.97 / 0.97	4717 / 9675
avg / total	0.98 / 0.97	0.98 / 0.97	0.98 / 0.97	12713 / 25528

Confusion Matrix		
	0 zugewiesen	1 zugewiesen
0 tatsächlich	7776 / 15381	220 / 472
1 tatsächlich	84 / 214	4633 / 9461

Absatzerkennung				
class	precision	recall	f1-score	support
0.0	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	2851 / 5642
1.0	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	1866 / 4029
avg / total	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	4717 / 9671

Confusion Matrix		
	0 zugewiesen	1 zugewiesen
0 tatsächlich	2851 / 5642	0 / 0
1 tatsächlich	0 / 0	1866 / 4029

Tabelle 5: Analysewerte und Wahrheitsmatrizen für die Fließtext- und Absatzerkennung mit einer SVM. Der linke Wert ist für Variante 1 (je 100 Trainings- und Testdaten) und der rechte Wert für Variante 2 (200 PDFs als Trainings- und Testdaten). *class* meint hier die möglichen Ergebnisklassen ($0 = keinFließtext, 1 = Fließtext$ bzw $0 = neuerAbsatz, 1 = Folge - Absatz$), *precision* gibt an, wieviele der als positiv markierten Testdaten auch wirklich positiv sind. *recall* gibt hingegen an, wieviele der positiven Testdaten vom System als positiv erkannt werden. Der *f1-score* ist der gewichtete Durchschnitt aus *precision* und *recall* und kann daher als eine Art Kompromiss betrachtet werden. *support* gibt an, wie oft die jeweilige Ergebnisklasse im Ergebnisvektor vorkommt. Die Wahrheitsmatrix (engl. = confusion matrix) ist eine Matrix A, in der jedes $A_{i,j}$ die Anzahl der Klassenzuordnungen angibt, von denen man weiß, dass sie der Klasse i angehören, die aber der Klasse j zugeordnet werden.

5 Diskussion

5.1 Qualität der Ergebnisse

Zunächst fällt auf, dass für Variante 1 hinsichtlich der Fließtexterkennung die Ergebnisse bei allen Lernmodellen identisch sind, sowohl hinsichtlich der Wahrheitsmatrix als auch für *precision*, *recall* und *f1-score*. Allerdings sind diese Werte bei der Anwendung der Modelle *Logistic Regression* und *Bernoulli Naive Bayes* durch eine Anpassung der Parameter-Werte zustandegekommen. Auch für Variante 2 sind die Ergebnisse bei allen Lernmodellen identisch, bis auf *Logistic Regression*, bei der einige Werte geringfügig schlechter sind. Die Ähnlichkeit der Ergebnisse ist von Seiten des Autors nicht erklärbar. Insgesamt sind bei Variante 2 die Durchschnittswerte um 1 % schlechter als bei Variante 1. Eine Erhöhung der Trainingsmenge scheint keine Verbesserung zu bringen, was eigentlich zu erwarten wäre. Möglicherweise liegt hier ein *overfitting* vor d.h. die Trainingsmenge hat einen Umfang angenommen, welche die Erkennungsleistung verringert. Um die optimale Menge an Trainingsdaten auszutesten, können weitere empirische Analysen durchgeführt werden.

Hinsichtlich der Erkennung zusammengehörender Abschnitte werden für beide Varianten und bei allen Lernmodellen 100 % erzielt. Dies lässt sich dadurch erklären, dass wir hier bezüglich der Featurewertebestimmung das logische Modell eines NOR-Gatters mit 3 Eingängen vorliegen haben (siehe auch Kapitel 3.4).

Insgesamt lassen sich die Ergebnisse als zufriedenstellend bezeichnen. Verbesserungsvorschläge werden im folgenden Unterkapitel diskutiert.

5.2 Verbesserungsvorschläge

Um die Erkennung relevanter Textelemente weiter zu verbessern und sich damit den Höchstwerten anzunähern, kann zum einen eine weitere Verbesserung des tex- und pdf-Parsers angestrebt werden. Einige TEX_FILES aus der vom Lehrstuhl bereitgestellten Dokumentenmenge werden ungenügend geparsed, was sich z.B. darin zeigt, dass innerhalb des TXT_FILE die Rollenzuweisungen nicht 100 % zuverlässig sind (Referenzen oder Autorenangaben zu Beginn des Dokumentes werden z.B. als „text“ markiert, u.a.). Entsprechendes gilt für den tsv-Parser, der ebenso in den Rollenzuweisungen nicht 100 % sicher ist und auch gelegentlich Formeln zusammen mit Fließtext als einen Paragraphen extrahiert. Hier ist eine weitere systematische Sichtung erforderlich, in der die Art und Anzahl der Fehler entsprechend dokumentiert werden. Auf dieser Grundlage können dann nicht nur die Parser, sondern auch der Algorithmus zur Erstellung der Featurevektorenwer-

te verbessert werden. Der Implementierungsprozess hat gezeigt, dass eine kontinuierliche Verbesserung dieser Parser (und damit bessere Trainings- und Testdaten) die Erkennung von Fließtext kontinuierlich steigern konnte. Weiterhin kann eine Gewichtung bestimmter Featurewerte angewendet und evaluiert werden. Als fortgeschrittene Techniken können auch eine semantische Erkennung bzw. Fehlertoleranzen eingebaut werden.

Obwohl die Ergebnisse für die Erkennung zusammengehörender Abschnitte 100 % erreichen, zeigt eine manuelle Besichtigung der markierten Trainings- und Test-PDFs, dass es gelegentlich zu falschen Zuordnungen kommt. So werden z.B. nach Überschriften manche Abschnitte nicht als neue Abschnitte markiert, sondern dem vorherigen als Folge-Paragraph zugeordnet oder auch neue Abschnitte trotz Einrückung in der ersten Zeile als solche nicht erkannt. Zwar sind diese Zuordnungen hinsichtlich des Featurewerte-Modells korrekt, jedoch möglicherweise unkorrekt im semantischen Sinne. Auch hier ist eine weitere systematische Sichtung erforderlich, in der die Art und Anzahl der Fehler entsprechend dokumentiert werden. Auf dieser Grundlage können dann nicht nur die Parser, sondern auch der Algorithmus zur Erstellung der Featurevektorenwerte verbessert werden. Auch muss berücksichtigt werden, dass die Ergebnisse für die Erkennung zusammengehörender Abschnitte von den Klassenzuordnungen aus der Fließtexterkennung abhängig sind, da nur diejenigen Textelemente betrachtet werden, die als Fließtext vorher markiert werden. Weiterhin kann hier eine Gewichtung bestimmter Featurewerte angewendet und evaluiert werden. Das Feature *GroßbuchstabeUndPunkt* kann dahingehend verbessert werden, dass hier die Zeilenbreite in der letzten Zeile eines Paragraphen berücksichtigt wird. Endet ein relevanter Paragraph mit einem Punkt und ist die Zeilenbreite gegenüber den anderen Zeilen geringer, so ist in der Regel der nächste relevante Paragraph ein neuer Abschnitt. Auszutesten ist auch ein neues Feature, dass die erste Zeile eines relevanten Paragraphen auf einen Kleinbuchstaben untersucht, da ein Folge-Paragraph häufig damit beginnt. Ebenfalls zu überprüfen ist die Anwendung semantischer Techniken bzw. Fehlertoleranzen.

Das eigentliche Programm zur Extraktion eines Fließtextes *extract_text.py* (siehe Kapitel 7.4) kann dahingehend verbessert werden, dass der Benutzer als Ausgabe nicht nur eine Datei mit dem reinen Fließtext erhält, sondern noch zusätzlich eine Datei, in der sämtliche Paragraphen des PDF zusammen mit zusätzlichen Informationen (z.B. Fließtext / kein Fließtext, Seite im PDF, u.a.) aufgelistet sind. Eine solche Datei würde in ihrem Aufbau dem TSV_FILE ähnlich sein.

5.3 Auswahl des Lernmodells

Dasjenige Lernmodell, welches nach Ansicht des Autors im eigentlichen Programm zur Extraktion eines Fließtextes (siehe Kapitel 7.4) zur Anwendung kommen soll, ist *Bernoulli Naive Bayes*. Zum einen spricht für dieses Modell, dass mit ihm in beiden Varianten gleich gute Ergebnisse wie etwa im Vergleich zur SVM erzielt werden. Zum anderen spricht dafür, dass dieses Modell mit einer Laufzeit innerhalb *O(n)* schneller arbeitet als etwa die SVM und auch für Anwendung mit binären Featurefunktionen gut geeignet ist.

6 Zusammenfassung

In dieser Bachelorarbeit wurde ein System vorgestellt, dass es erlaubt, zusammengehörende Fließtextparagraphen aus einem PDF zu erkennen und zu extrahieren. Dies soll durch die Anwendung von Lernmodellen aus dem Bereich des Maschinelllen Lernens realisiert werden. Große Teile der Implementierungsphase wurden dafür aufgewendet, zuverlässige Trainings- und Testdaten zu erstellen. Grundlage für diese Erstellung bildeten Daten, die mithilfe des System *icecite* gewonnen werden konnten. Im Laufe des Implementierungsprozesses konnte die Qualität dieser Daten kontinuierlich verbessert werden. Schließlich konnten Featurevektoren sowohl für die reine Erkennung von Fließttext als auch für die Erkennung von zusammengehörenden Fließtextparagraphen erstellt werden. Jeder dieser Vektoren enthält eine Reihe von Features, deren Werte durch eine binäre Featurefunktion errechnet wurden. Da es sich hier um ein *Supervised Learning* handelt, beinhalten alle Trainings- und Testdaten einen Ergebnisvektor, dessen Werte ebenso durch eine binäre Funktion errechnet wurden ($0 = keinFließtext, 1 = Fließtext$ bzw. $0 = neuerAbsatz, 1 = Folge - Absatz$). Speziell die Erstellung des Ergebnisvektors für die reine Erkennung von Fließtext hat sehr viel Zeit in Anspruch genommen.

In einer empirischen Analyse konnten diese Trainings- und Testdaten (erstellt aus 200 wissenschaftlichen PDF-Dokumenten) ausgetestet werden. Es hat sich gezeigt, dass im Rahmen der verwendeten Dokumentenmenge brauchbare Ergebnisse mittels Maschinellem Lernen erzielt wurden. Zur Anwendung kamen folgende Lernmodelle: Logistic Regression, Naive Bayes, Decision Trees, k-Nearest-Neighbors und die Support Vector Machine. Evaluiert wurden 2 Varianten. In Variante 1 wurden 100 PDFs als Trainingsdaten und die übrigen 100 als Testdaten verwendet. Variante 2 verwendete alle 200 PDFs sowohl zum Trainieren als auch zum Testen. Für Variante 1 konnten hinsichtlich der Fließtexterkennung durschnittliche Werte von 98 % erreicht werden (*precions*, *recall* und *f1-score*). Die Ergebnisse bei Variante 2 waren um 1 % schlechter. Beobachtet wurde insgesamt, dass bei allen Lernmodellen und in den beiden Varianten nahezu identische Ergebnisse erzielt wurden.

Ausblick

Um die Qualität der Erkennung weiter zu steigern, werden folgende Weiterentwicklungen vorgeschlagen:

- weitere empirische Analysen, um die optimale Menge an Trainingsdaten auszutesten.
- eine systematische Sichtung der Trainings- und Testdaten, in der die Art und Anzahl der Fehler entsprechend dokumentiert werden (zur Verbesserung der Parser aus dem

System *icecite* und des Algorithmus zur Erstellung der Featurevektorenwerte)

- Verbesserung des Features *GroßbuchstabeUndPunkt* zur Erkennung von zusammengehörenden Paragraphen
- Entwicklung eines weiteren Features zur Erkennung von zusammengehörenden Paragraphen
- für beide Featurevektoren kann eine Gewichtung einzelner Features ausgetestet werden
- Überlegungen, inwiefern semantische Techniken bzw. Fehlertolerenzen die Ergebnisse steigern können
- Verbesserung des eigentliches Programmes zur Extraktion eines Fließtextes *extract_text.py* (siehe Kapitel 7.4)

7 Benutzerhinweise

7.1 icecite

Das System *icecite* ist Voraussetzung zur Benutzung der Programme *create_csv.py* und *extract_text.py*. Über diese Webadresse lässt sich das Programm herunterladen:
https://github.com/ckorzen/icecite
Entsprechend der Anleitung auf dieser Seite lässt sich das Programm mittels *git* herunterladen. Installiert wird der PDF-Parser wie folgt:

```
cd icecite/pdf-parent
mvn install
```

Um ein PDF entsprechend den Bedürfnissen von *create_csv.py* zu parsen, müssen folgende Befehle ausgeführt werden:

```
cd icecite/pdf-cli
java -jar pdf-cli/target/pdf-cli-*-jar-with-dependencies.jar
--feature paragraphs_with_lines --format tsv $PDF
```

$PDF ist ein Platzhalter für den Pfad zur PDF-Datei.

Der TEX-Parser wird wie folgt installiert:

```
cd /icecite/commons
mvn -DskipTests install
cd /icecite/arxiv-benchmark/arxiv-benchmark/
mvn -DskipTests install
cd /icecite/arxiv-benchmark/tex-paragraph-parser/
```

und folgendermaßen benutzt:

```
cd /icecite/arxiv-benchmark/tex-paragraph-parser
java -jar target/tex-paragraph-parser-0.0.1-SNAPSHOT-jar-with-dependencies.jar
-i <TEX-FILE> -o <OUTPUT-FILE> -v <VISUALIZATION-PATH> -bb
```

$< TEX - FILE >$ gibt den Pfad zum TEX_FILE an, $< OUTPUT - FILE >$ gibt den Pfad des zu erstellenden TXT_FILE an (die Datei muss vorher nicht existieren) und $< VISUALIZATION - PATH >$ den Pfad für das markierte PDF, in dem die Paragraphen rot umrandet und mit der entsprechenden Rolle beschriftet werden. Der Parameter *-bb* ist wichtig, da hiermit die Koordinaten der Paragraphen ausgegeben werden.

7.2 create_csv.py

Dieses Programm benötigt zur Erstellung eines CSV_FILE die durch die *icecite*-Parser erzeugten TXT_FILES und TSV_FILES. Es liest diese Dateien ein und berechnet anhand der Informationen die Werte für beide Featurevektoren. Es ist über diese Webadresse verfügbar:
https://ad-websvn.informatik.uni-freiburg.de/student-theses/david-spisla
Das Programm wird ausgeführt mit dem Befehl:

```
python3 create_csv.py PDF-NUMBER yes/no
```

Bei *PDF-NUMBER* soll der Name des PDF ohne das Suffix *.pdf* eingegeben werden. Heißt eine Datei z.B. 7.pdf, so muss hier eine 7 als erster zusätzlicher Parameter eingegeben werden. Für eine Stapelverarbeitung empfiehlt es sich, alle PDFs und die dazugehörigen Dateien hinsichtlich der Benennung durchzunummerieren (auch der Ordner, in dem alle relevanten Dateien eines PDFs liegen, sollte 7 heißen. Daher empfiehlt es sich, einen Ordner pro PDF zu benutzen). Eine solche Verarbeitung kann über das einfache Hilfsprogramm *create_all_csv.py* erreicht werden, welches über die gleiche Webadresse verfügbar ist. Wünscht der Benutzer eine Ausgabe der markierten PDFs zur Überprüfung der erstellen Klassenzuordnungen, so kann durch den zweiten zusätzlichen Parameter *yes* bzw. *no* eine solche Ausgabe erzeugt werden oder nicht.

Zu beachten ist auch die richtige Setzung der relativen Pfadangaben, in welcher die PDF-Dokumente liegen und in welcher die erstellen Trainings- und Testdaten hineingelegt werden. In der Datei *create_csv.py* kann dies am oberen Ende des Codes getan werden. Weitere Einzelheiten lassen sich der Dokumentation entnehmen.

7.3 modelling.py

Dieses Programm benötigt zur Anwendung eines Lernmodells die durch *create_csv.py* erzeugten CSV_FILES. Es liest diese Dateien ein, wendet darauf das durch den Benutzer gewählte Lernmodell an und erstellt zum Schluss die Analysewerte. Es ist ebenfalls über diese Webadresse verfügbar:
https://ad-websvn.informatik.uni-freiburg.de/student-theses/david-spisla
Das Programm wird ausgeführt mit dem Befehl:

```
python3 create_csv.py MODEL
```

Bei *MODEL* muss das entsprechende Lernmodell mit einem Kürzel angegeben werden. Es stehen folgende Kürzel zur Verfügung:

```
lr nb dt knn svm
```

Auch hier ist die Setzung der relativen Pfadangaben zu beachten, die benötigt werden, um die Trainings- und Testdaten zu laden. Dies kann am oberen Ende des Codes getan werden. Das Programm lädt alle im jeweiligen Ordner befindlichen CSV_FILES. Es ist darauf zu achten, dass nur korrekt erstellte Trainings- und Testdateien verwendet werden. Weitere Einzelheiten lassen sich der Dokumentation entnehmen.

7.4 extract_text.py

Dieses Programm benötigt zur Anwendung eines Lernmodells die durch *create_csv.py* erzeugten CSV_FILES sowie den installierten PDF-Parser, der von *icecite* bereitgestellt wird. Es erstellt von einem PDF mithilfe des Parsers ein TSV_FILE, aus der dann gleich die Werte für beide Featurevektoren (aber ohne Klassenzuordnungen) erstellt werden. Anschließend werden die CSV_FILES eingelesen und das Lernmodell trainiert. Auf das trainierte Modell werden dann in der Testphase die Featurevektoren des eingelesenen PDFs angewendet. So erhält man die Klassenzuordnungen und damit die zusammengehörenden Fließtextparagraphen. Die erstellten Featurevektorenwerte des PDFs werden nur intern verarbeitet, eine Datei wird nicht erstellt. Es ist ebenfalls über diese Webadresse verfügbar:
https://ad-websvn.informatik.uni-freiburg.de/student-theses/david-spisla
Das Programm wird ausgeführt mit dem Befehl:

```
python3 extract_text.py <PATH-ICECITE> <PDF-FILE> yes/no
```

Bei $< PATH - ICECITE >$ wird der absolute Pfad zum System *icecite* eingegeben. $< PDF - FILE >$ gibt den relativen Pfad zum PDF an, aus dem der Fließtext extrahiert werden soll. Mit *yes* bzw. *no* kann der Benutzer bestimmen, ob ein markiertes PDF erstellt werden soll, in dem alle Fließtextelemente, die durch das Programm ermittelt werden, farblich umrandet sind. Ist ein relevanter Text grün umrandet, so handelt es sich um einen neuen Fließtext-Paragraphen, eine gelbe Markierung kennzeichnet einen Folge-Paragraphen.

Auch hier ist die Setzung der relativen Pfadangaben zu beachten, die benötigt werden, um die Trainingsdaten zu laden. Dies kann am oberen Ende des Codes getan werden. Das Programm lädt alle im jeweiligen Ordner befindlichen CSV_FILES als Trainingsdaten. Es ist darauf zu achten, dass nur korrekt erstellte Trainingsdateien verwendet werden. Weitere Einzelheiten lassen sich der Dokumentation entnehmen.

8 Literatur

[ALPAYDIN14, 2014] Alpaydin, Ethem: Introduction To Machine Learning. The MIT Press, Cambridge, Massachusetts, London, England, (2014)

[AI08, 2008] Andoni, Alexandr; Indyuk, Piotr: Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In: Communication of the ACM, January 2008/ Vol. 51,No.1. S.117-122. The MIT Press, Cambridge, Massachusetts, London, England, (2014)

[BBBH12, 2012] Bast, Hannah; Bäurle, Florian; Buchhold, Björn; Haussmann, Elmar: Broccoli: Semantic Full-Text Search at your Fingertips. In: CoRRabs/1207.2615 (2012)

[BGSF10, 2010] Beel, J., Gipp, B., Shaker, A. and Friedrich, N. 2010. SciPlore Xtract: Extracting Titles from Scientific PDF Documents by Analyzing Style Information (Font Size). Research and Advanced Technology for Digital Libraries, Proceedings of the 14th European Conference on Digital Libraries (ECDL'10) (Glasgow (UK), Sep. 2010), S.413–416

[BK13, 2013] Bast, Hannah; Korzen, Claudius: The Icecite Research Paper Management System. In: Lin, Xuemin (Hrsg.); Manolopoulos, Yannis(Hrsg.); Srivastava, Divesh (Hrsg.); Huang,Guangyan (Hrsg.): Web Information Systems Engineering - WISE 2013 Bd. 8181. Springer Berlin Heidelberg, 2013

[BLGN13, 2013] Beel, Joeran; Langer, Stefan; Genzmehr, Marcel; Müller, Christoph: Docears PDF Inspector: Title Extraction from PDF Files. In: Proceeding of the 13th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'13), S.443-444. ACM, 2013

[DOMINGOS12, 2012] Domingos, Pedros: A Few Useful Things to Know about Machine Learning. In: Magazine Communications of the ACM, Volume 55 Issue 10 (2012), S.78-87

[DK03, 2003] Diaz-Bone, R.; Kuenemund, H.: Einfuehrung in die binaere logistische Regression. Mitteilungen aus dem Schwerpunktbereich Methodenlehre Heft Nr. 56 (2003). S. 4ff. Entnommen aus: http://www.rainer-diaz-bone.de/Logreq.pdf
(letzter Zugriff: 31.07.2016)

[GOERZ14, 2014] Görz, Günther [Hrsg]: Handbuch der Künstlichen Intelligenz – München, Oldenburg: 2014

[HAAS06, 2006] Haas, Matthias : Methoden der künstlichen Intelligenz in betriebswirtschaftlichen Anwendungen. (Diplomarbeit an der Fachhochschule Wismar, Februar 2006)

[HLT05, 2005] Hollingsworth, Bill; Lewin, Ian; Tidhar, Dan: Retrieving hierarchical text structure from typeset scientific articles–a prerequisite for e-science text mining. In: Proc. of the 4th UK E-Science All Hands Meeting, 2005, S.267–273

[HGMZZF03, 2003] Han, H., Giles, C.L., Manavoglu, E., Zha, H., Zhang, Z. and Fox, E.A.: Automatic document metadata extraction using support vector machines. Proceedings of the 3rd ACM/IEEE-CS Joint Conference on Digital libraries (2003), S.37–48

[HHCMZ05, 2005] Hu, Y., Hang, L., Cao, Y., Meyerzon, D., Zheng, Q.: Automatic Extraction of Titles from General Documents using Machine Learning. Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital libraries (2005), S.145–154

[ISO08, 2008] ISO: Document management—Portable document format—Part 1: PDF 1.7 / International Organization for Standardization. Geneva, Switzerland, 2008 (32000-1:2008)

[KLN10, 2010] Kan, Min-Yen; Luong, Minh-Thang; Nguyen, Thuy D.: Logical Structure Recovery in Scholarly Articles with Rich Document Features. In: Int. J. Digit. Library Syst. 1 (2010), Oktober, Nr.4, S.1–23. – ISSN 1947–9077

[LR10, 2010] Lopez, Patrice; Romary, Laurent: HUMB: Automatic Key Term Extraction from Scientific Articels in GROBID. In: Proceedings of the 5th International Workshop on Semantic Evaluation, ACL (Uppsala, Sweden, 2010), S.248-251

[PM06, 2006] Peng, F., McCallum, A.: Information extraction from research papers using conditional random fields. In: Information Processing and Management: an International Journal Volume 42 Issue 4, (2006), S.963-979

[RPHB12, 2012] Ramakrishnan, Cartic ; Patnia, Abhishek ; Hovy, Eduard ; Burns,Gully: Layout-Aware Text Extraction from Full-text PDF of Scientific Articles. In: Source Code for Biology and Medicine 7 (2012), Nr. 1, S.7. – ISSN 1751–0473

[SCHILL15, 2015] Schillinger, Fabian : Strukturierte Extraktion von Text aus PDF. (Masterarbeit an der Universität Freiburg, Mai 2015)

[SCHNITGER11, 2011] Schnitger, Georg : Computational Learning Theory. (Skript zur Vorlesung an der Goethe-Universität Frankfurt a.M., 2011)

[SIMON83, 1983] Simon, H.: Why Should Machines Learn? In: Michalski, R., Carbonell, J. und Mitchell, T. [Hrsg]: Machine Learning: An Artificial Intelligence Approach, S.25-38. Tioga, Palo Alto, CA

[SKLEARN] Scikit-learn. Machine Learning in Python. Internet: http://scikit-learn.org/stable
(letzter Zugriff: 31.07.2016)

[SKLEARNKNN] Scikit-learn. Nearest Neighbors. Internet: http://scikit-learn.org/stable/modules/neighbors.html
(letzter Zugriff: 31.07.2016)

[SKLEARNLR] Scikit-learn. Generalized Linear Models. Internet: http://scikit-learn.org/stable/modules/linear_model.html
(letzter Zugriff: 31.07.2016)

[SKLEARNNB] Scikit-learn. Naive Bayes. Internet: http://scikit-learn.org/stable/modules/naive_bayes.html
(letzter Zugriff: 31.07.2016)

[SKLEARNDT] Scikit-learn. Decision Trees. Internet: http://scikit-learn.org/stable/modules/tree.html
(letzter Zugriff: 31.07.2016)

[SKLEARNSVM] Scikit-learn. Support Vector Machines. Internet: http://scikit-learn.org/stable/modules/svm.html
(letzter Zugriff: 31.07.2016)

[SKLEARNMETRICS] Scikit-learn. API Reference. Internet: http://scikit-learn.org/stable/modules/classes.html
(letzter Zugriff: 31.07.2016)