Bachelor-Arbeit

Erkennung von Textparagraphen in wissenschaftlichen Publikationen mit Techniken des maschinellen Lernens

Maximilian Dippel

23. August 2016

Albert-Ludwigs-Universität Freiburg im Breisgau
Technische Fakultät
Institut für Informatik

Gutachter/in
Prof. Dr. Hannah Bast

Betreuer/in
Claudius Korzen

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.

Ort, Datum	Unterschrift

Abstract

Im Rahmen dieser Arbeit werden Methoden zur automatisierten Erkennung und Visualisierung von menschlich-intuitiven Paragraphen in wissenschaftlichen Publikationen behandelt. Diese extrahierten Paragraphen sollen für weitere Verwendungszwecke wie etwa eine Klassifizierung der Paragraphen in Überschriften, Bildunterschriften, inhaltlichem Fließ-Text etc. bereitgestellt werden. Ferner sollen Teile der so klassifizierten Paragraphen perspektivisch als Basis einer semantischen Suche dienen.

Es werden verschiedene Techniken des maschinellen Lernens eingesetzt und ihre Ergebnisse evaluiert. Hierzu werden geeignete Featurevektoren und entsprechende Trainingsdaten erstellt. Wir werden sehen, dass Support Vector Machines mit einer Precision von 0.98 und einem Recall von 0.98 die besten Ergebnisse liefern.

Inhaltsverzeichnis

1 Einleitung

1.1 Problemdefinition

Ausgehend von einem gegebenen PDF-Dokument werden wir mithilfe eines externen Algorithmus [1] ein TSV-Dokument erstellen, das für jedes Wort und jede Zeile des PDFs Informationen über Text, Position, Ausmaße, Font und Farbe liefert. Auf Basis dieses TSV-Dokuments berechnen wir für jede Zeile, ob sie den Anfang eines Paragraphen beschreibt. Die Ergebnisse dieser Berechnung werden visualisiert und in einem TSV-Dokument ausgegeben.

1.2 Motivation

Das PDF-Format hat sich längst als gängiges, plattformübergreifendes Format für Dokumente aller Art durchgesetzt. Ob wissenschaftliche Arbeiten, Bücher oder Prospekte – bei Veröffentlichung und Verbreitung bedienen sich die meisten Autoren des PDFs. Im Vergleich zu anderen Formaten ermöglicht es einen zuverlässigen Austausch zwischen Lesern, ohne dass dabei Formatierungen, Schriftarten, -größen oder -formen durch unterschiedliche Dekodierungen, Versionen oder Programme verändert werden.

Während das gängige PDF-Format also offenkundige Vorteile im Vergleich zu anderen Formaten – wie etwa die Plattformunabhängigkeit – bietet, weist es dabei allerdings auch vermeintliche Nachteile auf, wie beispielsweise die mangelnde Editierbarkeit der Dokumente. Des Weiteren ist PDF ein rein Layout-basiertes Format und liefert nur Eigenschaften (Schrift, Schriftgröße, Position) über einzelne Buchstaben, nicht jedoch über strukturelle Zusammenhänge zwischen den Buchstaben.

Auch im Zuge der Literaturrecherche beim Schreiben von wissenschaftlichen Arbeiten ist die Verarbeitung und Sichtung anderer Text-Quellen, häufig im PDF-Format, unerlässlich. Je nach Umfang der Arbeit werden diese Referenz-Dokumente in großer Vielfalt auftreten und mitunter schwer zu verwalten sein, sofern sie nur als PDFs vorliegen. Hier könnte die Handhabung erleichtert werden, indem man beispielsweise bei einer Wort-Suche die Möglichkeit bereitstellt, bestimmte Paragraphen zu filtern (z.B. Bildunterschriften) oder bei Treffern gleich den ganzen Paragraphen zu liefern.

An diesen Beispielen sieht man, dass bestimmte Informationen fehlen, die die effiziente Suche und vor allem die Verwaltung der Suchergebnisse erleichtern würden. Wie bereits angedeutet, kann uns dabei die Einteilung in Paragraphen helfen, beispielsweise wenn wir zu Treffern immer auch den gesamten Paragraphen extrahieren wollen. Sind wir im Besitz der separierten Paragraphen, können wir perspektivisch auch deren Kategorisierung in Überschriften, Bildunterschriften, inhaltlichen Fließtext etc. vornehmen. Somit könnte man nur in Paragraphen suchen, die eine bestimmte Eigenschaft erfüllen, was zur Güte der Ergebnisse beitragen würde. Die Erkennung von Paragraphen liefert also verschiedene Vorteile bei der Verarbeitung von Dokumenten. Dennoch ist die Extraktion von Paragraphen ein Feature, über das keiner der gängigen PDF-Reader verfügt – maximal ist eine Einteilung von vorab aufbereiteten PDFs möglich.

Im Rahmen dieser Arbeit wollen wir uns also mit der Bereistellung und Analyse einer algorithmischen Lösung für die Unterteilung eines PDFs in Para-

graphen beschäftigen. Unser Hauptproblem besteht darin, dass das PDF selbst keine Struktur-Informationen liefert, wir also bei der Erstellung von Trainingsdaten als Vergleichsbasis den Umweg über andere Formate wie LaTeX gehen müssen. Das Generieren der Trainingsdaten wird einen großen Teil der Arbeit ausmachen. Ausgehend von einer Menge an Trainingsdaten werden wir unter Verwendung verschiedener Methoden des maschinellen Lernens gute Features suchen, implementieren und analysieren.

Die Erstellung möglichst korrekter Trainingsdaten sowie deren Verwendung im Rahmen von maschinellem Lernen stellen somit den Kern dieser Arbeit dar.

1.3 Das 'richtige' Ergebnis

Will man die im Rahmen dieser Arbeit erhaltenen Ergebnisse bewerten, so muss man sich zunächst verdeutlichen, dass die Einteilung eines Textes in Paragraphen keinesfalls eindeutig und häufig sehr subjektiv ist. Zwar mag in manchen Situationen jeder die gleichen Abschnitte aufteilen, in anderen können die Ergebnisse allerdings weit voneinander abweichen (vgl. Abbildung 1). Ist beispielsweise eine Aufzählung von Stichpunkten ein einziger Paragraph? Oder sind die einzelnen Stichpunkte selbst Paragraphen? Diese Entscheidung hängt nicht zuletzt vom persönlichen Kriterien ab.

Vor diesem Hintergrund ist es unter manchen Aspekten schwierig, Ergebnisse als 'falsch' oder 'richtig' zu bewerten, da die Algorithmen lediglich versuchen, das zu reproduzieren, was sie anhand von Trainingsdaten gelernt haben – entweder durch gesetzte Parameter oder Trainingsdaten.

Da die Trainingsdaten in dieser Arbeit maßgeblich anhand von LaTeX-Dokumenten generiert werden (s. Kapitel 5), entspricht die hier verwendete Unterteilung von Paragraphen der Unterteilung in die folgenden semantischen Boxen und Elemente innerhalb von LaTeX-Dateien:

- Titel, z.B. \title{...}
- Autoren, z.B. \author{...}
- Affiliationen, \affil{...}
- Überschriften, z.B. \section{...}, \subsection{...},\subsubsection{...}, \paragraph{...}
- Abstract, z.B. \begin{abstract} ... \end{abstract}
- Aufzählungen, z.B. jedes \item in \begin{itemize} ... \end{itemize}
- Referenzen, z.B. jedes \bibitem in \begin{thebibliography} ... \end{thebibliography}
- Acknowledgements, z.B. \begin{acknowledgements} ... \end{acknowledgements}
- Bildunterschriften, z.B. \caption in \begin{figure} ... \end{figure}
- Tabellenunterschriften, z.B. \caption in \begin{table} ... \end{table}
- Abgesetzte Formeln, z.B. $$ ... $$ oder \begin{align} ... \end{align} oder \begin{eqnarray} ... \end{eqnarray}
- Fußnote, \footnote{...}

- Textblöcke, die durch eine Leerzeile getrennt sind.
- Explizit definierte Paragraphen, z.B. mit \par{...}

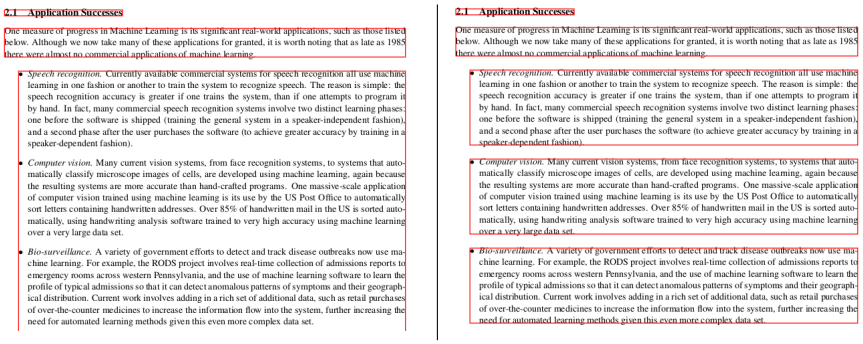


Abbildung 1: Beispiel unterschiedlicher Einteilungen eines Ausschnitts einer wissenschaftlichen Arbeit in rot umrandete Paragraphen. Auf der linken Seite sieht man eine eher grobe Einteilung, auf der rechten Seite eine eher feine Einteilung.

2 Vergleichbare Arbeiten

Bei der Arbeit mit PDF-Extraktion und -Aufbereitung stellt man fest, dass Software mit einer Funktionalität, die unserer Problemstellung entsprechen würde, kaum zu finden ist. Zwar können viele Programme Text-Abschnitte erkennen, doch ist es hier in erster Linie zusammenhängender Fließtext (samt Überschriften, Fußnoten etc.), der erkannt wird, ungeachtet der Einteilung in inhaltliche Paragraphen oder Abschnitte.

Die meisten Ansätze werden dadurch erschwert, dass PDFs nur Positionen von Characters speichern und darüber hinaus kaum Informationen liefern. Software wie PDFBox [2] bildet darauf aufbauend Wörter und Zeilen auf Basis von Schätzungen, geht aber nicht so weit, dass eine Paragraphen-Erkennung möglich ist.

Auch wenn wir keine direkte Lösung unseres Problems in vergleichbaren Arbeiten finden, so wollen wir doch im Folgenden exemplarisch auf vergleichbare Ansätze oder Lösungen von Teilproblemen in anderen Ausarbeitungen eingehen. Vor- und Nachteile werden dabei stets auch auf unser Problem bezogen, nicht (nur) auf die Güte des Programms als solches.

2.1 Tagged-PDF

Im Zuge der PDF-Extraktion wird man sehr wahrscheinlich auch auf den Begriff der Tagged-PDFs stoßen. Dabei handelt es sich um aufbereitete PDFs, die zusätzlich zu Positionen noch Informationen bezüglich Struktur oder Logik der Textbausteine enthalten [3]. Von einem Tagged-PDF spricht man insbesondere, wenn folgende Eigenschaften vorliegen:

- Dem Inhalt liegt eine logische Lese-Reihenfolge bei
- Bilder besitzen korrekte alternative Beschreibungen
- Tabellen sind mit Struktur-Informationen versehen
- Text-Paragraphen sind ausgeprägt und enthalten erweiterte Informationen
- Kodierungs-Informationen liegen für alle Bereiche vor

Vorteile
Offenkundig liefern Tagged-PDFs eine Vielzahl an Informationen, die uns nicht zuletzt bei der Einteilung in Paragraphen helfen könnten.

Nachteile
Leider bilden Tagged-PDFs nicht zwangsläufig die Lösung des Problems, da die automatische Erstellung von Tags zu einem PDF bei komplizierteren Dokumenten oftmals an ihre Grenzen stößt und wenig zufriedenstellende Ergebnisse liefert. Dies wird dadurch verstärkt, dass selbst kleine Ungenauigkeiten in der Interpretation die komplette vermeintliche Struktur des PDFs verändern können, was sich drastisch in der Wiedergabe des entsprechenden Dokuments niederschlagen kann.

Fazit
Da also Programme zur Erstellung von Tagged-PDFs in der Masse keine zufriedenstellenden Ergebnisse liefern, entspricht die Erstellung von Tags im Kern dem Problem dieser Arbeit, der Strukturierung von PDFs. Wir haben also die gleichen Schwierigkeiten und keinen echten Vorteil.

2.2 parsCit

Das Programm parsCit [4] ermöglicht zum einen das Parsen von Referenzen (bspw. im Literaturverzeichnis) und den zugehörigen Zitaten im Text, zum anderen die Erkennung logischer Strukturen in wissenschaftlichen Dokumenten. Letzteres wird in einer Hierarchie abgebildet, die folgende Komponenten erkennen soll:

- Titel
- Autor
- Gliederungen
- Abstracts
- Text-Abschnitte und Unterabschnitte
- Referenzen
- Bilder
- Tabellen

Diese logische Unterteilung erfolgt in einem Unterprogramm names SectLabel.

Die Basis der Algorithmen bietet ein Konzept aus dem Bereich des maschinellen Lernens, die sog. Conditional Random Fields [5].

Anwendung
Das Programm steht als Download oder in verschiedenen Ausführungen im Internet als Web-Demo zur Verfügung. Dabei können Dateien in verschiedenen Formaten hochgeladen werden.

Vorteile
Hat man ein Dokument im TXT- oder XML-Format vorliegen, können oben beschriebene Informationen gut extrahiert und sogar visualisiert werden.

Nachteile
In verschiedenen Beispielen zeigt sich, dass die Erkennung der Informationen für ein PDF noch nicht ganz überzeugt. So werden Autor und Titel bspw. richtig erkannt, doch als 'Abstract' erkannte das Programm mitunter den gesamten Text des PDFs. Eine Ausgabe der einzelnen Text-Abschnitte muss man dem erstellten XML-Dokument entnehmen, das die Text-Blöcke allerdings sehr grob und unpräzise zusammenfügt. Zusätzlich kann die Laufzeit sehr hoch sein, insbesondere da die effizienteren Varianten des Programms nicht für PDFs, sondern

lediglich für TXT-Files implementiert sind. In einem Beispiel-Versuch betrug die Dauer der Verarbeitung von einem PDF mit zehn Seiten Länge eine Minute.

Fazit
Auch wenn das Programm Lösungs-Ansätze in Richtung unseres Problems unternimmt, so ist es doch im jetzigen Stadium eher unbefriedigend in der Anwendung auf PDF-Dateien. Zusätzlich ist es primär für XML- und maximal noch für TXT-Dateien ausgelegt, was eine – zugegeben nicht schwierige – Konvertierung des PDFs erfordert.

2.3 pdfx

Das Modul pdfx v1.9 [6] ist ein automatisierter PDF-to-XML-Converter für wissenschaftliche Dokumente. Dabei wird eine Hierarchie logischer Elemente erstellt und in einem XML-Dokument gespeichert. Extrahiert werden können folgende Elemente (Bezeichnung der Elemente der Vergleichbarkeit wegen in Englisch):

- Front Matter
  - Titel
  - Abstract
  - Autor
  - Fußnoten
- Body Matter
  - Text-Körper
  - Überschriften
  - Bilder
  - Tabelle
  - Bild- und Tabellen-Überschriften
  - Bild- und Tabellen-Referenzen
  - Literaturangaben
  - Literaturverweise
- Extras
  - Kopfzeile
  - Fußzeile
  - Seitenzahlen
  - Anmerkungen
  - E-Mail-Adressen
  - URL-Adressen
  - Literaturangaben
  - Literaturverweise

Anwendung
Die Anwendung des Moduls erfolgt über ein Web-Interface, das den Upload von PDFs ermöglicht und deren Verarbeitung im Rahmen von einzelnen Jobs mit *job_ids* ermöglicht. Dabei werden das zugehörige XML-Dokument sowie zusätzlich das HTML-Dokument und ein TAR-GZ-Archiv erstellt. Diese stehen zum sofortigen Download sowie 24h zur nachträglichen Einsicht unter der *job_id* zur Verfügung.

Alternativ kann das Programm über einen HTTP-client wie *curl* über die command-line bedient werden [6].

Vorteile
Es erfolgt eine Übersetzung des PDFs in verschiedene Formate, wobei nach logischen Elementen getrennt wird. Dabei kann jedes PDF verarbeitet werden und die Zeitkosten sind akzeptabel. So benötigt die Übersetzung eines PDFs mit ca. 10 Seiten etwa 15 - 20 Sekunden.

Nachteile
Während die Beschreibung des Programms wie die Lösung des Gesamtproblems dieser Arbeit anmuten mag, so stößt es doch nach einigen Testversuchen auf recht offenkundige Grenzen. Zwar werden zum Teil gute Unterscheidungen von verschiedenen Elementen vorgenommen – also bspw. von Überschriften und Fließtext –, doch entspricht die Einteilung innerhalb der Elemente nicht unseren Anforderungen. So sind längere Fließtexte als ganzes zusammengefasst und nicht als einzelne Paragraphen gemäß Einrückungen oder Zeilenabständen.

Fazit
Da die Anforderungen der Entwickler an ihr Programm nicht ganz den unseren entsprechen, lässt sich dieses Programm nicht zur Lösung unseres Problems anwenden. Die inhaltlichen Text-Paragraphen werden logischen Text-Elementen untergeordnet und gehen somit als Information verloren.

2.4 Automatic Paragraph Identification

Die gleichnahmige Studie [7] befasst sich mit der automatisierten Erkennung von Paragraphen in Texten verschiedener Sprachen (Englisch, Deutsch, Griechisch). Anhand von formellen, syntaktischen und inhaltlichen Features wird eine Erkennung von Paragraphen mithilfe von maschinellem Lernen verfolgt. Ausgehend von vorab erkannten Sätzen werden die folgenden Features betrachtet:

Nicht-syntaktische Features:
Diese Features beziehen sich in erster Linie auf visuell/formell messbare Werte.

- Distance
  Entfernung des aktuellen Satzes zum letzten Paragraphen-Ende, gemessen in dazwischen liegenden Sätzen und Wörtern.
- Sentence Length
  Länge des aktuellen Satzes gemessen in der Anzahl der Wörter.

- Relative Position
  Position im Verhältnis zum Gesamttext.
- Quotes
  Gibt an, ob der vorherige oder der aktuelle Satz ein Zitat beinhaltet.
- Final Punctation
  Gibt an, mit welchem Zeichen der letzte Satz (bspw. '?') geendet ist.
- Words
  Beschreibt die ersten drei Wörter sowie die Gesamtheit aller Wörter des Satzes.

Syntaktische Features:
Auf Basis einer Darstellung von Sätzen als Syntax-Baum werden verschiedene Werte des Baumes gemessen, um Informationen über die Komplexität des aktuellen Satzes zu erhalten.

- Parsed
  Gibt an, ob der Satz überhaupt als Baum dargestellt werden konnte.
- Number of phrases
  Misst Komplexität in der Anzahl der Teilbäume, respektive Teilsätze.
- Signature
  Stellt Informationen über textbezogene, vorab generierte Tags für Teilsequenzen von Wörtern bereit.
- Children of Top-Level Nodes
  Misst die Komplexität des Syntax-Baumes auf dem höchsten Level (top-level complexity). Es wird gelistet, aus welchen Teil-Typen ein Satz besteht.
- Branching Factor
  Misst die durchschnittliche Anzahl an Kindern eines Satzteils [8].
- Tree Depth
  Misst die Tiefe des Syntax-Baumes.
- Cue Words
  Hebt vordefinierte Schlüsselwörter am Angang, in der Mitte und am Ende des Satzes hervor.

Inhaltliche Features:
Auf Basis von gemessenen Sprachmodellen und Häufigkeiten von Wörtern und Sätzen werden zwei Features generiert:

- Häufigkeit eines Satzes in der betreffenden Sprache
- Entropie-Rate jedes einzelnen Wortes (Häufigkeit des Satzes geteilt durch die Anzahl der Wörter des Satzes)

Fazit
Von den Autoren angestellte Tests ergaben eine Genauigkeit der Erkennung zwischen 71.83% und 83.92% [7]. Informationen über die statistische Zusammensetzung dieser Werte werden nicht geliefert. Insbesondere muss hierbei erwähnt werden, dass die Auswertung ausschließlich auf Texten aus dem Bereich der Nachrichten erfolgte. Sämtliche Modelle und vorab generierte Erhebungen sind unter dieser Fokussierung erstellt worden. Somit kann die Methode nur auf einer vergleichsweise geringen Anzahl an spezifischen Texten ausgeführt werden, was ein großes Hinderniss bei der allgemeinen Anwendung darstellen kann. Des Weiteren führen inhaltliche Features zu Hürden bei den zulässigen Sprachen des Textes. Momentan arbeitet das Verfahren nur auf Englisch, Deutsch und Griechisch.

Nicht zuletzt sei erwähnt, dass die vorgestellte Studie auch lediglich als solche begriffen wurde, es also kein zugängliches Programm oder eine Materialisierung der beschriebenen Konzepte für den Endnutzer gibt.

Den Einschränkungen sowie dem theoretischen Charakter der Studie ist geschuldet, dass diese lediglich als interessante Idee und nicht als Lösung des Problems betrachtet werden kann.

2.5 Andere Arbeiten

An dieser Stelle wollen wir weitere vergleichbare Arbeiten nennen, diese jedoch nur in Kürze betrachten.

2.5.1 GROBID

Das Modul GROBID [9] bedient sich maschinellen Lernens, um Inhalt aus Dokumenten wie PDFs zu extrahieren und zu strukturieren. Das Resultat ist dabei ein TEI-kodiertes [10] Dokument. Entwickelt wird das Projekt seit 2008, und seit 2011 ist es als open source verfügbar. GROBID verspricht folgende Funktionalitäten:

- Extraktion von Gesamt-Text sowie dessen Segmentierung und Strukturierung von Text-Körpern
- Extraktion von Headern
- Extraktion von Referenzen
- Parsen von Namen, bspw. des Autors im Header oder in Referenzen
- Parsen von Adress-Blöcken oder Anschriften
- Parsen von zeitlichen Daten
- Parsen von Meta-Daten

Anwendung
Nach vorangegangener Installation lässt sich das Programm auf einem eigenen

Server starten und im Browser aufrufen. Von dort aus kann man über die Benutzeroberfläche PDFs hochladen und im Rahmen verschiedener Services analysieren. Da die Ausgabe jedoch TEI-kodiert ist, müsste unter Umständen eine weitere Konvertierung in gewünschte Formate erfolgen.

2.5.2 pdf-extract

Das Modul pdf-extract [11] dient der Extraktion verschiedener Text-Bereiche anhand ihrer Klassifikation. Dabei wird eine strukturelle Analyse durchgeführt, um verschiedene Elemente innerhalb eines Dokuments zu erkennen. In diesem Rahmen sollen u.a. erkannt werden:

- Textspalten
- Kopfzeilen
- Fußzeilen
- Text-Abschnitte
- Überschriften
- Referenzen

Anwendung
Das Moduls steht als freier Download zur Verfügung. Die Bedienung erfolgt über das Terminal, wobei man als Optionen die Elemente angeben kann, die extrahiert werden sollen, bspw. headers.

2.6 Fazit

Die Tatsache, dass zum eingangs beschriebenen Problem keine wirklich überzeugende Lösung zu existieren scheint, birgt eine besondere Motivation, sich im Rahmen dieser Arbeit damit zu beschäftigen und einen verwendbaren Lösungsansatz zu präsentieren.

3 Ansatz

3.1 Ausgangspunkt

Grundlage des vorgestellten Verfahrens bilden verschiedene, bereits existierende Algorithmen und Konzepte. Genannt sei hier in erster Linie ein Algorithmus zur Erkennung von Wörtern und Zeilen in einem gegebenen PDF [1], der ebenfalls am Lehrstuhl für Algorithmen und Datenstrukturen entwickelt wurde. Konkret liefert der Algorithmus Informationen über Typ, Text, Position, Schriftart, Schriftgröße, Farbe (vgl. Abbildung 2).

Es folgt eine Übersicht aller bereitgestellten Werte und ihre Beschreibung. Können den Einträgen einer Spalte verschiedene bzw. weitere Bedeutungen zukommen, werden diese in Klammern genannt.

- **feature**
  Gibt den Typ des Eintrags an, also ob eine Textzeile (*line*), ein Wort (*word*) oder eine Klasse (z.B. *color* oder *font*) vorliegt. Klassen-Einträge dienen hierbei anderen *line*- und *word*-Zeilen als Referenzen, die als solche in anderen Spalten als Werte dienen können. Beispielsweise kann es sich um eine *font*-Klasse handeln, auf die in einer *line*-Zeile in der Spalte mostCommonFont verwiesen wird.

- **text (reference name)**
  Gibt bei *line*-Einträgen den String-Inhalt des Elements an. Dementsprechend entweder das zugehörige Wort oder die zugehörige Zeile im PDF. Bei Klassen steht hier der eindeutige Referenzname.

- **page (font: name, color: value red)**
  Gibt an, zu welcher Seite im Ausgangs-PDF der Eintrag gehört. Bei *font*-Klassen steht hier der Name der Schrift. Bei *color*-Klassen steht hier der *Rot*-Wert der Farbe.

- **minX (font: bold, color: value green)**
  Minimale x-Koordinate des Elements in PDF-Einheiten. Es wird vom linken unteren Seitenrand gemessen. Bei *font*-Klassen steht hier ein binärer Wert, der angibt, ob die Schrift *bold* ist (1) oder nicht (0). Bei *color*-Klassen steht hier der *Grün*-Wert der Farbe.

- **minY (font: italic, color: value blue)**
  Minimale y-Koordinate des Elements in PDF-Einheiten. Es wird vom linken unteren Seitenrand gemessen. Bei *font*-Klassen steht hier ein binärer Wert, der angibt, ob die Schrift *italic* ist (1) oder nicht (0). Bei *color*-Klassen steht hier der *Blau*-Wert der Farbe.

- **maxX**
  Maximale x-Koordinate des Elements in PDF-Einheiten. Es wird vom linken unteren Seitenrand gemessen. Bei *font*-Klassen steht hier ein binärer Wert, der angibt, ob die Schrift *type 3* ist (1) oder nicht (0).

- **maxY**
  Maximale y-Koordinate des Elements in PDF-Einheiten. Es wird vom linken unteren Seitenrand gemessen.

- **mostCommonFont**
  Gibt die *font*-Klasse des in dieser Zeile am häufigsten verwendeten Fonts an.
- **startFont**
  Gibt die *font*-Klasse des in dieser Zeile am Anfang verwendeten Fonts an.
- **endFont**
  Gibt die *font*-Klasse des in dieser Zeile am Ende verwendeten Fonts an.
- **startFontSize**
  Gibt die Größe in *pt*. des in dieser Zeile am Anfang verwendeten Fonts an.
- **endFontSize**
  Gibt die Größe in *pt*. des in dieser Zeile am Ende verwendeten Fonts an.
- **mostCommonColor**
  Gibt die *color*-Klasse der in dieser Zeile am häufigsten verwendeten Farbe an.
- **startColor**
  Gibt die *color*-Klasse der in dieser Zeile am Anfang verwendeten Farbe (*color*) an.
- **endColor**
  Gibt die *color*-Klasse der in dieser Zeile am Ende verwendeten Farbe (*color*) an.
- **role**
  Gibt die Einteilung in verschiedene Rollen an. Beispiele seien hier *title* und *body-text*.

	A	B	C	D	E	F	G	H	I
1	**feature**	**text**	**page**	**minX**	**minY**	**maxX**	**maxY**	**mostCommonFont**	...
2	word	Hello	1	142	695	167	702	font-31	...
3	word	World!	1	170	695	203	702	font-31	...
4	word	Bye	1	142	679	160	689	font-31	...
5	word	World!	1	164	681	193	689	font-31	...
6	line	Hello World!	1	142	695	203	702	font-31	...
7	line	Bye World!	1	142	679	193	689	font-31	...
8	font	Font-31	cmr1	1	0	0			
9	color	Color-0	0.1	0.2	0.0				

Abbildung 2: Teil einer Beispiel-Ausgabe des Algorithmus zur Erkennung von Wörtern und Zeilen eines PDFs. Die Ausgabe erfolgt als TSV-Dokument.

Auf dieser Basis werden nachfolgende Überlegungen und Algorithmen arbeiten.

Ebenso sei bemerkt, dass Implementierungen im Rahmen dieser Arbeit in Python geschrieben werden, da Python zum einen gute Module hinsichtlich des maschinellen Lernens bietet und zum anderen als Script-Sprache das schnelle Testen von Ansätzen und Verfahren ermöglicht.

3.2 Lineare Vorgehensweise

Im folgenden Abschnitt wollen wir versuchen, das beschriebene Problem durch eine lineare Vorgehensweise zu lösen, d.h. wir arbeiten unsere Input-Tabelle (Abbildung 2) Zeile für Zeile ab und vergleichen dabei jeweils zwei *line*-Einträge miteinander. Dies geschieht unter der Einschränkung, dass wir Paragraphen nur innerhalb einer Seite und innerhalb einer Textspalte erkennen wollen. D.h. Paragraphen, die über eine Seite oder Spalte hinausreichen, werden geteilt.

Gegeben eine Folge von Text-Zeilen eines PDFs in geordneter, inhaltlich-richtiger Reihenfolge, verarbeiten wir diese linear und treffen eine Paragraphen-Einteilung basierend auf vorgegebenen Kriterien. Man entscheidet also für jede verarbeitete Zeile, ob sie den Anfang eines neuen Paragraphen markiert oder noch zum vorherigen gehört. Dies geschieht durch einen direkten Vergleich mit der vorangegangenen Zeile.

An dieser Stelle muss überlegt werden, auf welche Art und Weise der Algorithmus eine solche Entscheidung vornehmen könnte. Beispielsweise hätte er die Möglichkeit, auf den Abstand der Zeilen (Formel 1), den Schrifttyp oder die Positionen der Zeilen zu referenzieren.

$$\begin{aligned} a &= vorherigeZeile \\ b &= aktuelleZeile \\ d(a,b) &= minY_a - maxY_b \end{aligned} \tag{1}$$

Auf den ersten Blick ermöglichen diese Parameter für das menschliche Auge eine sinnvolle Erkennung und Unterscheidung von Paragraphen. Doch ignorieren wir dabei, dass wir selbst nicht etwa gesetzte Schwellwerte haben, sondern situationsbedingte Grenzen ziehen. Da für einen Algorithmus alle PDFs und ferner alle Stellen des PDFs lediglich Folgen von Buchstaben sind, müsste er mit gesetzten und immer gleichen Schwellwerten in allen Situationen arbeiten. Man kann sich leicht Fälle überlegen, wo beispielsweise ein bestimmter Zeilenabstand den Anfang eines neuen Paragraphen bedeutet und in anderen Fällen das genaue Gegenteil. Ein Beispiel für eine solche Problematik bezüglich des Zeilenabstands sehen wir in Abbildung 3. Lassen wir zunächst die von LaTeX statuierte, einzig richtige Lösung außer Acht und beschränken uns unabhängig davon auf den Kern des beschriebenen Problems, so scheint im oberen Beispiel der Abbildung die linke Einteilung sinnvoller als die rechte, da die zweite Zeile lediglich eine kurze Beschreibung der genannten Person darstellt. Die Zeilen gehören also dem Empfinden nach zusammen, auch wenn sie durch einen größeren Zeilenabstand getrennt werden. Im unteren Beispiel der Abbildung wirkt die rechte Einteilung sinnvoller, da es sich klar um zwei abgeschnittene Paragraphen handelt. Hier soll der Zeilenabstand also sehr wohl zu einer Teilung der Paragraphen führen. Ähnliche Überlegungen kann man auch für alle anderen möglichen Parameter anstellen.

Somit muss man bei einer linearen Vorgehensweise mindestens eine Vielzahl von Parametern betrachten, um ein verwendbares Ergebnis zu erhalten. Doch an dieser Stelle stoßen wir auf das Problem der Gewichtung. Angenommen, der Algorithmus hat für einen *line*-Eintrag berechnet, dass der Zeilenabstand sowie die Schriftfarbe auf einen neuen Paragraphen hindeuten, die Schriftgröße und Einrückung (minX) aber vermuten lassen, dass die Zeile zum vorherigen Paragraphen gehört. Es ergibt sich die Frage, ob nun nach Mehrheit der Kriterien

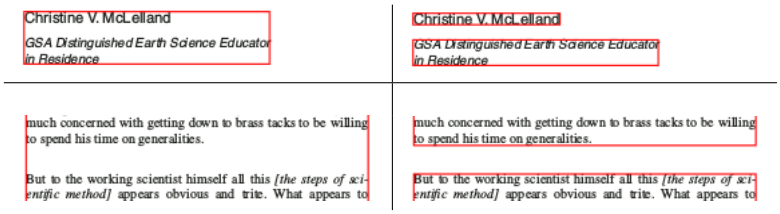


Abbildung 3: Bedeutung von Zeilenabständen in unterschiedlichen Umgebungen

entschieden wird. Dies könnte zu Problemen führen, falls der Zeilenabstand von großer Bedeutung ist, insbesondere wichtiger als Font-Größe und Einrückung. Diese Relation würde bei einer simplem Mehrheits-Gewichtung verloren gehen und mit ihr ein für die Einteilung wichtiges Kriterium.

Wir sehen also, dass wir eine Vielzahl von Entscheidungs-Parametern (im Folgenden Features genannt) führen, deren Gewichtigung von entscheidender Bedeutung ist. Gleichzeitig ist es schwierig, Prioritäten vorab festzulegen, da diese z.T. nur sehr subjektiv durch einen Menschen zu erheben sind.

Auch wenn die lineare Vorgehensweise auf Probleme zu treffen scheint, so liefert sie doch grundlegende Überlegungen, die wir im Folgenden aufgreifen wollen. Nicht zuletzt führen uns diese Überlegungen zu einer Anwendung des maschinellen Lernens.

3.3 Nutzung von maschinellem Lernen

Aus den Vorstellungen eines linearen Lösungsverfahrens lässt sich klar die Idee verschiedener Features zur Bewertung und Erkennung einzelner Paragraphen ableiten. Solche Features seien beispielsweise der Zeilenabstand, die Zeilenlänge oder der Font. Wie oben beschrieben, muss man eine Gewichtung der einzelnen Kriterien in Relation zueinander vornehmen, da auch Überschreitungen von Grenzwerten in einer Kategorie in der richtigen Verbindung mit anderen Feature-Ergebnissen durchaus positiv bewertet werden können. Da diese Gewichtung allerdings nur schwer von Hand vorgenommen werden kann, drängt sich die Anwendung des Prinzips des maschinellen Lernens auf.

4 Maschinelles Lernen als Lösungsverfahren

Wie eingangs festgestellt, liegen uns zahlreiche Informationen bzw. Kategorien (= Features) vor. Was uns fehlt, sind Erfahrungen, um diese Kategorien und ihre Ergebnisse in Relation zueinander bewerten zu können. Im Rahmen des maschinellen Lernens versuchen wir, vorab einen Wissensschatz anhand von bereits bewerteten Beispielen zu erstellen, um Muster in der Art und Weise der Bewertung der Daten zu erkennen. Diese Muster können dann als statistisches Modell auf noch nicht ausgewertete Daten angewendet werden, um Ergebnisse zu liefern, die den Erfahrungen entsprechen.

4.1 Grundlagen des maschinellen Lernens

Während die Grundidee des maschinellen Lernens stets dieselbe bleibt, so gibt es doch verschiedene Bereiche und Unterscheidungen. In erster Linie sei hier die Unterscheidung zwischen symbolischen und subsymbolischen Systemen genannt.

4.1.1 Symbolische und subsymbolische Systeme

Dem *Symbol* kommen in verschiedenen Kontexten mitunter unterschiedliche Bedeutungen bei (vgl. Abbildung 4). So kann man es als Buchstaben-Einheit in einem Alphabet, als ein Wort oder gar als Handlung bzw. Vorgang verstehen.

In der Informatik wird das *Symbol* häufig als atomare (d.h. kleinste, nicht weiter auflösbare) Gegenstands-Einheit verwendet, die als Repräsentant eines nicht direkt wahrnehmbaren geistigen Sachverhalts dient (vgl. [12], Seite 23) oder dienen kann.

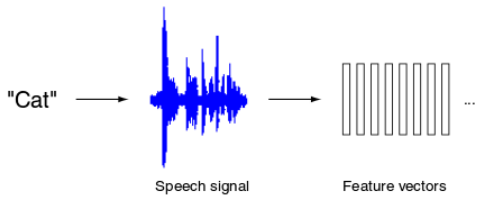


Abbildung 4: Verschiedene Auffassung eines Symbols [12]. Wir sehen von links nach rechts ein Wort in geschriebener Sprache, ein Wort als Audio-Signal in der Akustik und als Dateninformation (vielleicht binär) in der Informatik

Symbolische Systeme verarbeiten Trainings-Wissen und Bewertungs-Regeln auf eine Art, die einen nachträglichen Einblick in die Ergebnis-Findung ermöglicht. Diese Verfahren lassen sich somit von Menschen besser nachvollziehen, da sich eine Art menschliche Logik auf die Ergebnisse projizieren lässt.

Dem gegenüber stehen subsymbolische Systeme, deren Wissen eher implizit repräsentiert wird, d.h. Bewertungen und deren Herleitung sind in erster Linie maschinell verwertbar und nur schwer von einem Menschen zu lesen. In diesem Zusammenhang kann es vorkommen, dass sich bestimmte Ergebnisse weniger leicht nachvollziehen lassen als bei symbolischen Systemen.

4.1.2 Überwachtes und unüberwachtes Lernen

Eine weitere wichtige Unterscheidung im Bereich des maschinellen Lernens stellt die Art und Weise des Lernens dar. Hier stehen sich in erster Linie überwachtes und unüberwachtes Lernen gegenüber.

Spricht man von einem überwachten Lernen, so gibt es einen allwissenden Lehrer, der einem lernenden Agenten auf die Frage nach richtig oder falsch (bspw. nach einer ausgeführten Aktion) eine korrekte Antwort gibt. So entsteht ein direktes und unmittelbares Feedback [13].
Konkreter kann man sich hier beispielsweise die Eingabe eines Trainingssets vorstellen, also einer Menge gelöster Bewertungs-Probleme, die dem Agenten später bei eigenständigen Bewertungen als Muster helfen soll.

Demgegenüber steht das unüberwachte Lernen. Hierbei erhält ein lernender Agent auf die Frage, ob eine Aktion richtig oder falsch war, keine Antwort. Demnach muss er selbstständig eine Bewertung der Güte seiner Handlungen vornehmen, beispielsweise indem er Auswirkungen und Effekte misst [13].
In vielen Fällen wird bei diesem System ein Netz aus Informationen und Verbindungen gebildet, das auf eigenen Klassifikationen und Zusammenhängen beruht. Mit diesem Modell erklärt sich gewissermaßen die Maschine die aufgefundenen Sachverhalte selbst.

Eine dritte, eher kleinere, aber trotzdem erwähnenswerte Art des Lernens ist das sogenannte bestärkende Lernen, bei dem der Agent Belohnungen oder Bestrafungen abhängig von seiner gewählten Taktik bzw. Vorgehensweise erhält. Auf diese Weise will man eine eigenständige Entwicklung hin zur richtigen Auswahl von Aktionen entwickeln. Diese Lernform entspricht nicht zuletzt der des Menschen [13].

4.2 Auswahl

Im Bereich des maschinellen Lernens gibt es eine Vielzahl an Modellen, die man zur Lösung des hier behandelten Problems verwenden kann. Da wir zwangsläufig eine Auswahl treffen müssen, priorisieren wir im Rahmen dieser Arbeit die bekanntesten und gängigsten Modelle für flaches (symbolisches) überwachtes Lernen. Diese wären in unserem Fall:

- Logistische Regression
- Support Vector Machines
- Perceptrons
- Naive Bayes

Wir werden feststellen, dass diese Auswahl uns eine ausreichende Varianz in der Qualität und Natur der Ergebnisse liefert, die wir vergleichend gegenüberstellen können und werden.

4.3 Features

Ausgehend von einem Datenset wie in Abbildung 2 und Kapitel 3.1 beschrieben, fügen wir jedes Feature als neue Spalte hinzu. Die Werte in diesen neuen Spalten bestimmen wir durch lineares Abarbeiten der *line*-Einträge. Dabei vergleichen

wir stets zwei Zeilen und bestimmen aus den uns zur Verfügung stehenden Daten die Werte der Features.

Es folgt eine nähere Beschreibung der im Rahmen dieser Ausarbeitung verwendeten Features. Dabei sei in der Spalte 'Berechnung' die Veriable v stets der für dieses Feature berechnete und eingetragene Wert. Des Weiteren sei a die vorherige Zeile und b die aktuell betrachtete Zeile, für die wir den Wert berechnen. Für diese Zeilen verwenden wir die in Abbildung 2 und Kapitel 3.1 beschriebenen Werte.

Name	**Beschreibung**	**Berechnung**
FirstLineOfPage	Binäre Information, ob die Zeile die erste auf einer neuen Seite ist	$v = (page_a \neq page_b)$
Distance	Vertikaler Abstand zur vorherigen Zeile in PDF-Einheiten	$v = minY_a - maxY_b$
DifferenceInStart	Positionsunterschied der Startposition zur Startposition der vorherigen Zeile in PDF-Einheiten	$v = \|minX_a - minX_b\|$
AverageFont	Binäre Information, ob der am häufigsten verwendete Font der Zeile identisch zu dem der vorherigen Zeile ist	$v = (mostCommonFont_a == mostCommonFont_b)$
ConnectedFont	Binäre Information, ob der Font am Anfang der Zeile identisch zum Ende der vorherigen Zeile ist	$v = (endFont_a == startFont_b)$
SpecialFont	Binäre Information, ob der am häufigsten verwendete Font der Zeile identisch zu dem der vorherigen Zeile ist, **aber** nicht dem vorab berechneten häufigsten Font des Dokuments entspricht.	$v = (mostCommonFont_a == mostCommonFont_b)$ && $(mostCommonFont_a \neq totalMostCommonFont)$
AverageFontSize	Binäre Information, ob die Font-Größe der Zeile identisch zur vorherigen Zeile ist	$v = (mostCommonFontSize_a == mostCommonFontSize_b)$

Name	Beschreibung	Berechnung
ConnectedFontSize	Binäre Information, ob die Font-Größe am Anfang der Zeile identisch zum Ende der vorherigen Zeile ist	$v = (endFontSize_a == startFontSize_b)$
ConnectedFontColor	Binäre Information, ob die Font-Farbe am Anfang der Zeile identisch zum Ende der vorherigen Zeile ist	$v = (endColor_a == startColor_b)$
Role	Binäre Information, ob die Zeilen-Rolle, die dem Dokument entnommen werden konnte, der der vorherigen Zeile entspricht	$v = (role_a == role_b)$
IsPageNumber	Binäre Information, ob die Zeile eine Seitenzahl beschreibt	Berechnung anhand von Erkennungskriterien, bspw. der *Role* und der Struktur der Zeile
IsListing	Binäre Information, ob die Zeile den Beginn einer Auflistung beschreibt	Berechnung anhand von Erkennungskriterien, bspw. Aufzählungszeichen
AfterListing	Binäre Information, ob die Zeile auf den Beginn einer Auflistung folgt	Berechnung anhand von *IsListing* der vorherigen Zeile
IsTitle	Binäre Information, ob die Zeile den Beginn eines Titels beschreibt	Berechnung anhand von Erkennungskriterien, bspw. Aufzählungszeichen
AfterTitle	Binäre Information, ob die Zeile auf den Beginn eines Titels folgt	Berechnung anhand von *IsTitle* der vorherigen Zeile
FormulaSwap	Binäre Information, ob die vorherige Zeile eine Formel war und die aktuellen Zeile nicht **oder** ob die vorherige Zeile keine Formel war, aber in der aktuellen Zeile eine Formel beginnt	$v = (role_a == \text{'formula'} \;\&\&\; role_b \neq \text{'formula'}) \;\|\|\; (role_a \neq \text{'formula'} \;\&\&\; role_b == \text{'formula'})$
Overlap	Binäre Information, ob das Positions-Rechteck der aktuellen Zeile das Positions-Rechteck der vorherigen Zeile schneidet	$v = !(max(minX_a, minX_b) > min(maxX_a, maxX_b) \;\|\| \; min(maxY_a, maxY_b) < max(minY_a, minY_b))$

4.4 Ausgangspunkt

Im Folgenden gehen wir davon aus, dass uns Testdaten in Form einlesbarer Tabellen vorliegen. Diese Testdaten beinhalten die in Abbildung 2 und Kapitel 3.1 beschriebenen Informationen und werden algorithmisch um die vorangegangenen Features und ihre Werte ergänzt. Des Weiteren besitzen unsere Testdaten die Information 'is_start_of_paragraph', die für jede Zeile angibt, ob ein neuer Paragraph beginnt oder nicht. Diese Spalte bildet den Kern unserer Berechnungen, da sie im Fall der Trainingsdaten die zu lernende Lösung für den Agenten enthält und im Fall zu bewertender Daten das Ziel unserer Ergebnisse ist.

Auf die Erzeugung der Trainings-Daten wird in Kapitel 5 ausführlicher eingegangen.

Zusätzlich zu den Trainingsdaten liegt uns eine Menge zu bewertender Daten vor, die lediglich die Informationen aus Kapitel 3.1 enthalten. Diese werden algorithmisch um die vorangegangenen Features ergänzt.

4.5 Logistische Regression

4.5.1 Grundlagen

Ziel der logistischen Regression ist die Adjustierung der Einflüsse verschiedener Features, um eine möglichst geringe Verzerrung der Ergebnis-Schätzung zu erhalten [14].

Konkret beschreibt die logistische Regression die Modellierung einer diskreten Ergebnis-Variable Y bzw. deren Verteilung in Abhängigkeit der bereitgestellten Features X_i. Ein Beispiel wäre hier die Frage, ob ein Patient eine bestimmte Krankheit Y hat auf Basis des Blutdrucks X_1, des Blutzuckers X_2 und des Alters X_3.

Können in der Variable Y bei einem Modell mehr als zwei Ergebnis-Zustände erreicht werden, erfolgt eine Aufteilung in eine Menge von Hilfs-Variablen. In unserem konkreten Fall – wie auch in dem angeführten Beispiel – ist dies jedoch nicht vonnöten, da unsere Ergebnis-Variable 'is_start_of_paragraph' lediglich die Werte $True$ oder $False$, also 1 oder 0 annimmt. Hier sprechen wir davon, dass Y ein binäres Messniveau besitzt. Dies sei im Folgenden vorausgesetzt.

Im Kern der Methode geht es uns darum, die Wahrscheinlichkeit für ein positives Ergebnis $p = P(Y = 1)$ zu modellieren. Bei medizinischen Anwendungen wird p häufig das Risiko genannt. Ebenfalls von Bedeutung ist die sog. Chance, die beschrieben wird durch $p/(1-p)$. Das Risiko sowie die Chance können also – im Unterschied zu Y – beliebige reelle Werte zwischen 0 und 1 annehmen [14].

Gesetzt den Fall, uns liegt zur Bestimmung von Y nur ein einziges Feature X vor, bilden wir den Logarithmus der Chance als

$$logit(p) = log[\frac{p}{1-p}] = \alpha + \beta X \tag{2}$$

was äquivalent ist zu

$$p = \frac{exp(\alpha + \beta X)}{1 + exp(\alpha + \beta X)} \tag{3}$$

Hierbei handelt es sich um die sogenannte logistische Funktion, der die logistische Regression ihre Namensgebung verdankt [14]. Liegen $m > 1$ Features vor, so ersetzen wir βX durch eine Linearkombination der Features der Form $\beta_1 X_1 + \cdots + \beta_m X_m$.

4.5.2 Anwendung

In Python verwenden wir das Modul sklearn [15], genauer die Klasse linear_model. Diese ermöglicht uns die Erstellung eines logistischen Wahrscheinlichkeits-Modells und damit die spätere Bewertung von Daten. Liegen uns Trainingsdaten als Tabelle in einer Variable *data* vor, ist der Kern der Methode in Algorithmus 1 beschrieben.

Algorithm 1 Logistische Regression mit sklearn

```
1: X = data[features]   # features als Menge anzuwendender Features (4.3)
2: Y = data['is_start_of_paragraph']
3: self.logreg = linear_model.LogisticRegression()
4: self.logreg.fit(X, Y)
```

In der Tabelle X vereinen wir also alle Feature-Variablen, während wir in der Tabelle Y die Spalte mit den Ergebnissen finden. Der Parameter C ist eine Fließzahl, die das Inverse der Regulierungs-Stärke beschreibt. Dieser Parameter hilft uns, *overfitting* (d.h. wir spezialisieren unser Modell zu sehr auf unsere Testdaten) zu reduzieren, indem weit ausschlagende Feature-Werte mit *penalties* abgeschwächt werden.

Mittels *logreg.predict* können wir nun für andere Daten-Tabellen die Ergebnis-Spalte auf Basis unseres Modells vorhersagen.

4.6 Support Vector Machines

4.6.1 Grundlagen

Bei Support Vector Machines (kurz: SVMs) werden die zur Verfügung stehenden Testdaten in einen Hyperraum projiziert. Dort bilden sie einzelne Punkte in einem potenziell mehrdimensionalen Raum, wobei jeder Punkt einer Klasse zugeordnet wird. Ziel ist es nun, in diesem Raum eine Hyperebene zu erstellen, die die gegebenen Punkte nach Klassen trennt, wobei der Abstand zu allen Punkten nahe der Hyperebene maximiert werden soll (vgl. Abbildung 5). So werden später auch weniger eindeutige Daten der richtigen Klasse zugeordnet [16].

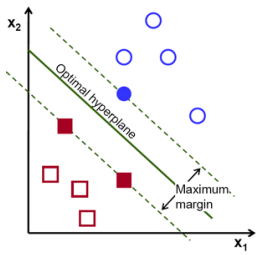


Abbildung 5: Ideale Hyperebene zur Trennung von Trainingsdaten [17]

Da sich die Hyperebene allerdings nicht verbiegen kann, kommt man schnell zu der Überlegung, dass nicht alle Trainingsdaten linear trennbar sind. In diesem Fall überführt man das Problem in die nächst höhere Dimension, bis eine Trennung möglich ist (vgl. Abbildung 6).

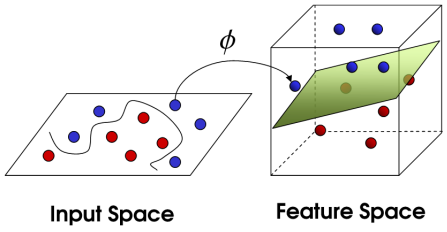


Abbildung 6: Dimensions-Erhöhung, falls die Daten in der aktuellen Dimension nicht linear trennbar sind[18]

Hat man in einem konkreten Fall Trainingsobjekte $x_1, \ldots, x_m \in \mathbb{R}^d$ mit zugehörigen Klassen $y_i = +1$ falls x_i in Klasse A ist und $y_i = -1$ falls x_i in Klasse B ist, so sei eine Hyperebene H bestimmt als

$$H = \{x \in \mathbb{R}^d \| w \bullet x\} \tag{4}$$

wobei $w \in \mathbb{R}^d$ der zur Hyperebene gehörende Normalenvektor und $b \in \mathbb{R}$ das Offset ist.

Die Distanz eines Trainingsobjektes zu H ergibt sich als

$$dist(x_i, H) = \frac{y_i \cdot (w \bullet x_i - b)}{|w|} \tag{5}$$

Beschreibt nun eine Variable r den minimalen Abstand eines Punktes zu unserer Hyperebene H, so wollen wir eine trennende Hyperebene H^* finden, sodass $2 \cdot r$ maximal ist, während $dist(x_i, H) \geq r$ für alle i.

Wir erhalten unsere trennende Hyperebene $H*$ also als Ergebnis eines Optimierungsproblems, auf dessen Lösung wir an dieser Stelle jedoch nicht weiter eingehen werden [16].

4.6.2 Anwendung

In unserem Fall bilden wir die einzelnen Zeilen im Hyperraum ab. Die Kategorisierung der Zeilen in der Spalte 'is_start_of_paragraph' mit 1 und 0 bildet unsere Klassifizierung, anhand derer wir unterscheiden wollen.

In Python verwenden wir auch hier das Modul sklearn [15], genauer die Klasse SVM. Diese ermöglicht uns die Erstellung eines Modells basierend auf SVM. Liegen uns Trainingsdaten als Tabelle in einer Variable *data* vor, ist der Kern der Methode in Algorithmus 2 beschrieben.

Algorithm 2 SVM mit sklearn

```
1: X = data[features]   # features als Menge anzuwendender Features (4.3)
2: Y = data['is_start_of_paragraph']
3: self.svm = svm.SVC()
4: self.svm.fit(X, Y)
```

4.7 Perceptrons

4.7.1 Grundlagen

Ähnlich wie Support Vector Machines sind auch Perceptrons ein linearer Klassifizierer. Hierbei werden iterativ der Normalenvektor w und der Offset-Vektor b einer Hyperebene H *greedy* berechnet, d.h. sie werden in jedem Schritt bestmöglich verbessert. Anhand dieser Ebene H erfolgt die Unterscheidung in die Klassen 1 und 0 eines Elements x wie folgt:

$$class(x) = \begin{cases} 1, w \bullet x - b > 0 \\ 0, w \bullet x - b \leq 0 \end{cases} \tag{6}$$

Wir berechnen also den Abstand eines Elements x zur Hyperebene H mit

$$dist(x, H) = w \bullet x - b \tag{7}$$

und klassifizieren es abhängig davon, auf welcher Seite von H sich das Element x befindet [19].

Vereinfacht werden die gesuchten Vektoren w und b wie in Algorithmus 3 berechnet. Hierbei referenzieren wir auf ein Trainingsset mit korrekt bewerteten Daten.

Algorithm 3 Bestimmung von w und b bei Perceptrons

```
1:  while changes in w do
2:      for all objects x in training set do
3:          if w • x − b gives right prediction then
4:              do nothing
5:          if w • x − b gives wrong prediction then
6:              if w • x − b ≤ 0 then
7:                  w = w + x
8:                  b = b − 1
9:              if w • x − b > 0 then
10:                 w = w − x
11:                 b = b + 1
```

Alternativ kann man als Abbruchbedingung eine gesetzte Anzahl an Iterationen oder ein Minimum an Veränderung verwenden. Ein Konvergenz-Beweis wird an dieser Stelle nicht angestrebt [19].

4.7.2 Anwendung

Identisch zur Anwendung von SVM bilden wir hier ebenfalls die einzelnen Zeilen im Hyperraum ab. Die Kategorisierung der Zeilen in der Spalte 'is_start_of_paragraph' mit 1 und 0 bildet unsere Klassifizierung, anhand derer wir unterscheiden wollen.

In Python verwenden wir auch hier das Modul sklearn [15], genauer die Klasse linear_model.perceptron. Liegen uns Trainingsdaten als Tabelle in einer Variable *data* vor, ist der Kern der Methode in Algorithmus 4 beschrieben.

Algorithm 4 Perceptrons mit sklearn

```
1: X = data[features]   # features als Menge anzuwendender Features (4.3)
2: Y = data['is_start_of_paragraph']
3: self.perceptron = linear_model.perceptron.Perceptron()
4: self.perceptron.fit(X, Y)
```

4.8 Naive Bayes

4.8.1 Grundlagen

Bei Naive Bayes arbeiten wir mit einer Menge an Klassen und einer Menge an Features bzw. Ausprägungen von unabhängig betrachteten Wahrscheinlichkeitsvariablen $F = \{f_1, \ldots, f_{|F|}\}$. Ein zu klassifizierendes Objekt $d = (v(f_1), \ldots, v(f_{|F|}))$ beschreibt eine Belegung v aller Features.

Des Weiteren haben wir aus Trainingsdaten die folgenden Informationen berechnet:

- $P(c_i)$ beschreibt die generelle Wahrscheinlichkeit, dass ein Objekt in Klasse c_i liegt. Dieser Wert kann durch simples Abzählen bestimmt werden.
- $P(v(f_j)|c_i)$ beschreibt die Wahrscheinlichkeit, dass ein Objekt mit einer bestimmten Ausprägung des Features j in Klasse c_i liegt

Haben wir ein zu bewertendes Objekt $d = (v(f_1), \ldots, v(f_{|F|}))$, berechnen wir für alle Klassen c_i:

$$P(c_i|d) = \frac{P(c_i) \cdot \prod_{f_j \in F} P(v(f_j)|c_i)}{P(d)} \tag{8}$$

$P(d)$ beschreibt hierbei die Wahrscheinlichkeit, dass wir als zu bewertendes Objekt d betrachten. In der Praxis lässt man den Nenner weg, da dieser für alle d identisch ist.

Haben wir also für alle Klassen c_i die Wahrscheinlichkeit $P(c_i, d)$ bestimmt, dass d in c_i liegt, wählen wir schlichtweg die Klasse mit der größten Wahrscheinlichkeit aus.

4.8.2 Anwendung

In unserem konkreten Fall betrachten wir als Klassen 1 und 0. Ein Objekt ist somit entweder Anfang eines neuens Paragraphen (also 1) oder nicht(also 0). Die Features f_i entsprechen den im Abschnitt 4.3 gelisteten Features. Ein Objekt entspricht also einer Zeile des TSV-Files, wobei die Werte der einzelnen Spalten als Ausprägungen der jeweiligen Features $v(f_i)$ dienen.

In Python verwenden wir auch hier das Modul sklearn [15], genauer die Klasse naive_bayes. Liegen uns Trainingsdaten als Tabelle in einer Variable *data* vor, ist der Kern der Methode in Algorithmus 5 beschrieben.

Algorithm 5 Naive Bayes mit sklearn

```
1: X = data[features]   # features als Menge anzuwendender Features (4.3)
2: Y = data['is_start_of_paragraph']
3: self.NB = naive_bayes.GaussianNB()
4: self.NB.fit(X, Y)
```

4.9 Post-Processing

Auch wenn im Folgenden mit Algorithmen des maschinellen Lernens gearbeitet wird, bleibt doch ein Restbestand an linearen Algorithmen. So werden Ergebnisse, die mittels eines Modells des maschinellen Lernens bestimmt wurden, anschließend durch ein sog. Post-Processing verfeinert, um Ausnahmen und Spezialfälle zu erkennen. Dabei werden die Zeilen des PDFs linear abgearbeitet und die berechneten Ergebnisse stellenweise korrigiert. Konkret sollen die folgenden Elemente besser erkannt werden:

- **Referenzen**
  Referenzen anhand von Features zu erkennen führt zu unbefriedigenden Ergebnissen. Eine lineare Erkennung erscheint hier deutlich einfacher und effektiver. Die Berechnung erfolgt anhand von Erkennungskriterien, bspw. eckigen Klammern oder – wenn vorhanden – den Informationen über die *role*.

- **Listen**
  Listen und Aufzählungen (auch Stichpunkte) werden anhand der verwendeten Features bereits gut erkannt. Um letzte Fehler zu beseitigen, erfolgt eine geringfügige lineare Überarbeitung, die vereinzelte Stichpunkte, die vorher nicht als solche erkannt wurden, nachträglich als Paragraphen kennzeichnet und in die Lösung einarbeitet.

- **Mehrzeilige Titel**
  Mehrzeilige Titel sind aufgrund ihrer Struktur mitunter schwierig anhand von Features zu erkennen. Sie weisen u.U. sehr drastische Positions-Unterschiede auf und erfordern daher eine lineare Überarbeitung.

- **Formeln**
  Da Formeln bereits initial Schwierigkeiten bei der Extraktion von Zeilen des PDFs in ein TSV-File bereiten können, kann darunter auch die Feature-basierte Erkennung leiden. Aus diesem Grund werden Formeln an manchen Stellen linear aufbereitet. Konkret werden hierbei Positions-Rechtecke vereint, die eine Überschneidung aufweisen. Das verhindert eine zu kleinteilige Erkennung.

5 Trainingsdaten

Ein wichtiger Faktor bei der Arbeit mit maschinellem Lernen stellen Trainingsdaten dar. Will ein Algorithmus bzw. unser statistisches Modell Vorhersagen über vorgelegte Textstellen liefern, muss es diese anhand von Erfahrungen treffen. Diese Erfahrungen spiegeln sich in unseren Testdaten wider.

Neben den untersuchten Features sind somit Trainingsdaten von essenzieller Bedeutung, da sie maßgeblich die Güte des Resultats bestimmen: hat man ein Trainingsset, in dem zufällig jeder Paragraph mit einem bestimmten Wort beginnt, wird der Algorithmus das als klares Muster für einen Paragraphen erkennen, auch wenn es sicher kein entscheidendes Kriterium in den meisten anderen Dokumenten darstellt. Ebenso ist es naheliegend, dass eine sehr geringe Menge an Trainingsdaten kaum zu guten Ergebnissen in allen Fällen führen wird, insbesondere nicht in Spezialfällen. Was das statistische Modell in seiner Trainingsphase nicht gesehen hat, kann es nicht oder nur schwer reproduzieren.

Auf der anderen Seite ist es auch naheliegend, dass eine Vielzahl von Daten in sehr unterschiedlicher Beschaffenheit zu einer starken Mittelung der Daten und ihrer Bewertung führt, was sich negativ auf die Ergebnisse auswirken könnte.

Im Folgenden wollen wir uns also näher mit der Erstellung der Trainingsdaten beschäftigen.

5.1 Anforderungen

Will man Trainingsdaten generieren, muss man sich zunächst überlegen, welche Beschaffenheit diese haben müssen oder konkreter, was wir eigentlich trainieren wollen. Typischerweise erzeugen wir hierfür ein neues Feature, das wir unserer Tabelle (vgl. Abbildung 2) als Spalte hinzufügen und mit Werten füllen. In unserem konkreten Fall sollen diese Werte klassifizierbar, also nicht etwa Fließzahlen sein (wie es bei der linearen Regression der Fall wäre).

Bezogen auf die Problemstellung erstreben wir ein Trainings-Feature, das uns für jede Textzeile angibt, ob diese Teil des vorangegangenen Paragraphen ist oder einen neuen markiert. Insofern erweitern wir die Tabelle um die eingangs beschriebene Spalte 'is_start_of_paragraph', die den Wert 1 annimmt, falls ein neuer Paragraph beginnt, und andernfalls 0 ist.

5.2 Erzeugung

Es stellt sich natürlich die Frage, wie man solche Trainingsdaten erhält, zumal sie in großen Mengen benötigt werden. Diese Problematik wollen wir im Folgenden behandeln.

Es muss kaum erwähnt werden, dass die manuelle Vorgehensweise in Summe mehr als unzulänglich ist. Angefangen bei enormem Zeitaufwand, kommt die Subjektivität und – schlimmer noch – die mangelnde Einheitlichkeit der Unterscheidung von Paragraphen hinzu.

Obwohl diese Methode zum schnellen Testen von Verfahren oder Trainings-Parametern verwendet werden kann, muss sie in größerem Rahmen zwangsläufig durch einen Algorithmus ersetzt werden. Will man jedoch Trainingsdaten maschinell generieren, muss man zunächst die aktuellen Rahmenbedingungen und die daraus resultierenden Möglichkeiten betrachten. Angenommen, man hat lediglich ein PDF vorliegen, kann man aus diesem kaum eine Einteilung von Para-

graphen als Trainings-Daten ableiten – andernfalls hätten wir die Lösung unseres Gesamtproblems bereits erreicht. Man muss also den Umweg über andere Hilfsmittel gehen, bspw. andere Format- oder Kompilierungs-Dateien. Hier bietet sich insbesondere das LaTeX-Format an, dessen Beschaffenheit bereits einigen Aufschluss über spätere Paragraphen liefern kann.

Haben wir für ein oder mehrere PDFs die Tabelle (vgl. Abbildung 2) mit den Grundwerten sowie das zugehörige LaTeX-File vorliegen, bedarf es also eines Algorithmus, der die einzelnen Zeilen der PDFs verarbeitet und in die Trainings-Spalte 'is_start_of_paragraph' eine 1 einfügt, wo nach unserer Einschätzung ein neuer Paragraph beginnt. Allen anderen Zeilen geben wir den Wert 0. In diesem Zusammenhang und im Zuge der in dieser Arbeit beschriebenen Ausarbeitungen wurden insbesondere die folgenden zwei Varianten ausprobiert. Beide setzen sich als Ziel die Erstellung guter Trainingsdaten anhand von Informationen, die wir aus einem LaTeX-Dokument entnehmen bzw. berechnen können. Die erste Variante ist dabei diejenige, die zuerst getestet wurde und intuitiv gute Ergebnisse liefern sollte. Wir werden jedoch feststellen, dass diese Variante nicht praktikabel ist und mit Variante 2 eine anwendbare Alternative präsentieren.

5.2.1 Variante 1 - Edit Distance

Bei dieser Variante bedienen wir uns des Konzeptes der Editier-Distanz, genauer der Levenshtein-Distanz [20]. Diese zählt, gegeben zwei Strings A und B, die kleinste Anzahl an Einfüge-, Vertausch- und Entfernen-Operationen von Buchstaben, um String A in String B zu überführen. Zur Berechnung einer Lösung, ferner der Ermittlung der Levenshtein-Distanz, gibt es einfache Algorithmen der dynamischen Programmierung, auf die hier allerdings nicht weiter eingegangen wird. Es sei jedoch erwähnt, dass sich eine entsprechende Implementierung in einer Grund-Bibliothek der meisten Programmiersprachen findet.

Basis des hier vorgestellten Ansatzes bildete ein Algorithmus des Instituts für Algorithmen und Datenstrukturen an der Universität Freiburg zur Extraktion von Paragraphen aus einem LaTeX-Dokument [1]. Konkret werden hierbei Textparagraphen durch doppelte Zeilenumbrüche sowie Befehle wie '\par' unterschieden und getrennt durch leere Zeilen in ein Textdokument geschrieben. Ferner werden Überschriften als Paragraphen separiert und Formel-Bereiche (bspw. durch '\equation') eingegrenzt und im ausgegebenen Textdokument durch Paragraphen der Form $[Formel]$ ersetzt.

Wir erhalten also ein Textdokument, dessen Inhalt eine Aufteilung des LaTeX-Dokument bzw. des resultierenden PDFs in einzelne Paragraphen beschreibt, die durch leere Zeilen getrennt werden. Stellt man dem nun die Zeilen unserer aus demselben PDF generierten Tabelle gegenüber (vgl. Abbildung 2), so muss explizit erwähnt werden, dass wir nun keinesfalls einen 1-zu-1-Vergleich der beiden Dokumente (der TXT-Datei mit den Paragraphen und der TSV-Datei mit den Zeilen-Daten) vornehmen können, um die Zeilen zu Paragraphen zu verbinden. Zunächst haben wir im TXT-Dokument eine gänzlich andere Zeileneinteilung als im TSV-Dokument, also der Datentabelle. Beispielsweise kann ein Paragraph in LaTeX durch eine einzelne lange Zeile beschrieben werden. Im PDF – und somit auch im TSV – wird sich diese LaTeX-Zeile auf mehrere PDF-Zeilen verteilen. Somit hätten wir im TXT-Dokument mit den erkannten Paragraphen nur eine einzelne Zeile, die keiner Zeile im TSV-Dokument entspricht, sondern einer Vielzahl. Zusätzlich muss die Reihenfolge der Paragraphen in einem LaTeX-File

nicht unbedingt mit der Reihenfolge im PDF übereinstimmen. Man denke hier beispielweise an das Element $figure$, das sich – wenn nicht weiter spezifiziert – nicht strikt in die Textreihenfolge einordnet.

Intuitiv würde man ausgehend von dieser Basis die Zeilen des TSV-Dokuments durchgehen und zeitgleich deren Konkatenation mit den Paragraphen im TXT-Dokument vergleichen. Liegt eine Übereinstimmung vor, hat man den aktuellen Paragraphen erkannt und arbeitet den nächsten ab.

Hier zeichnet sich allerdings sofort ein Problem bezüglich der eingangs beschriebenen Verarbeitung von Formel-Elementen aus LaTeX ab. Während unser Algorithmus zur Erkennung von Zeilen eines PDFs uns auch für Formel-Zeilen deren Inhalt in unserer Daten-Tabelle (vgl. Abbildung 2) wiedergibt, erhalten wir im TXT-Dokument mit den Paragraphen lediglich den Vermerk $[Formel]$. Dies führt dazu, dass inhaltlich gleiche Zeilen nicht mehr gleich in ihrer Form bzw. Repräsentation sind, was einen einfachen Vergleich unmöglich macht.

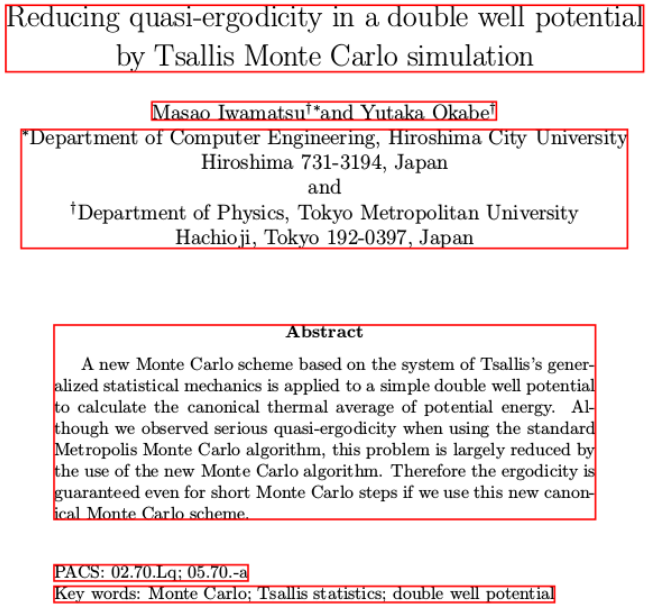
Hier kommt der zentrale Ansatz dieser Variante ins Spiel, die Editier-Distanz. Da inhaltlich gleiche Zeilen nun nicht mehr identisch sind, müssen wir einen näherungsweisen Vergleich anstellen. Wir betrachten also der Reihe nach unsere Paragraphen und fügen so lange Zeilen aus dem TSV-Dokument zusammen, bis die Editier-Distanz des Paragraphen zur Konkatenation aus bisher zusammengefügten Zeilen in Relation zur Anzahl der Buchstaben schlechter wird. Konkret gehen wir dabei wie in Algorithmus 6 vor.

Algorithm 6 Pseudo-Code für Variante 1

```
1:  procedure ExtractParagraphs
2:      lineCounter ← 1
3:      paragraphCounter ← 1
4:      prevLines ← emptyString()
5:      while lineCounter ≤ getTotalLines() do
6:          line ← getLineAt(lineCounter)
7:          paragraph ← getParagraphAt(paragraphCounter)
8:          ed1 ← editDistance(prevLines, paragraph)
9:          ed2 ← editDistance(prevLines + line, paragraph)
10:         if ed1/length(prevLines) < ed2/length(prevLines + line) then
11:             prevLines ← line
12:             yield result: line is start of paragraph
13:             paragraphCounter ← paragraphCounter + 1
14:         if ed1/length(prevLines) ≥ ed2/length(prevLines + line) then
15:             prevLines ← prevLines + line
16:         lineCounter ← lineCounter + 1
```

Ergebnis

Abbildung 7 zeigt die Visualisierung der erkannten Paragraphen mittels des vorgestellten Algorithmus. D.h. überall dort, wo wir eine Zeile mit 1 markiert haben, beginnt ein neuer Paragraph, also ein neues Rechteck. Wir sehen, dass die Erkennung hier – wie auch in vielen weiteren Fällen – sehr annehmbare Ergebnisse liefert, die wir als Testdaten verwenden könnten.



Reducing quasi-ergodicity in a double well potential by Tsallis Monte Carlo simulation

Masao Iwamatsu[†*] and Yutaka Okabe[†]

[*]Department of Computer Engineering, Hiroshima City University
Hiroshima 731-3194, Japan
and
[†]Department of Physics, Tokyo Metropolitan University
Hachioji, Tokyo 192-0397, Japan

Abstract

A new Monte Carlo scheme based on the system of Tsallis's generalized statistical mechanics is applied to a simple double well potential to calculate the canonical thermal average of potential energy. Although we observed serious quasi-ergodicity when using the standard Metropolis Monte Carlo algorithm, this problem is largely reduced by the use of the new Monte Carlo algorithm. Therefore the ergodicity is guaranteed even for short Monte Carlo steps if we use this new canonical Monte Carlo scheme.

PACS: 02.70.Lq; 05.70.-a
Key words: Monte Carlo; Tsallis statistics; double well potential

Abbildung 7: Beispiel-Ergebnis

Doch wie man anhand des Algorithmus schon erahnt, sind wir sehr abhängig von der Reihenfolge der Paragraphen, da wir auch die Zeilen linear abarbeiten. Werden also Paragraphen nicht in der visuell korrekten Reihenfolge erkannt (die Position eines *figure*-Elements in LaTeX entspricht nicht immer der im PDF), liefert der Algorithmus beliebige, inkorrekte Ergebnisse, schlimmer noch: beim ersten Auftreten einer Paragraphen-Vertauschung wird der Algorithmus somit fälschlicherweise einen Paragraphen mit den folgendenden Zeilen vergleichen, der keinerlei gewollte Übereinstimmungen zeigt. Es gibt also den Fall, dass die zusammengefügten Zeilen mehr zufällig dem Paragraphen entsprechen und eventuell als solche abgeschlossen werden. Es liegt jedoch nahe, dass der Fehler sich fortführen wird, da der Algorithmus nunmehr Zeilen mit den an dieser Stelle falschen Paragraphen vergleicht. Eventuell wird er dabei wieder auf einen Paragraphen treffen, der tatsächlich zu den nächsten Zeilen passt. Nun wird der Algorithmus wieder korrekt arbeiten, bis zum nächsten Vorkommen einer solchen Vertauschung.

Wir halten also fest, dass in den meisten Fällen gute Ergebnisse erzielt werden können, wir die Korrektheit allerdings nicht immer garantieren können. Da wir die Paragraphen aus den LaTeX-Dokumenten berechnen, werden wir bei Elementen wie '\figure' oder '\table' sehr wahrscheinlich beobachten, dass die Paragraphen-Reihenfolge diesbezüglich im LaTeX-Dokument nicht der des PDFs entspricht, da diese Elemente sehr häufig eine andere Position einnehmen – im Unterschied bspw. zu Fließtext, dessen Reihenfolge im LaTeX-Dokument der des PDFs entspricht.

Vor dem Hintergrund dieser Problematik lässt sich kaum mit dieser Variante arbeiten und wir benötigen einen alternativen Ansatz.

5.2.2 Variante 2 - Rectangle Matching

Variante 2 beschreibt ein Verfahren, das lediglich mit Positionen und Größen, bzw. den daraus abgeleiteten Rechtecken der Textbausteine arbeitet. So können wir etwa zu jeder erkannten Zeile aus einer TEX-Datei ihre entsprechende Position im PDF als Rechteck bestimmen und mit den Daten aus unserer Grund-Tabelle (vgl. Abbildung 2) abgleichen.

Auch bei diesem Verfahren bedienen wir uns eines bereits existierenden Algorithmus des Lehrstuhls für Algorithmen und Datenstrukturen der Universität Freiburg [1]. Dieser nimmt ebenfalls ein LaTeX-Dokument als Input und liefert dazu nicht nur den Text der Paragraphen (wie in Variante 1), sondern insbesondere auch die Positions-Rechtecke.

Basis hierfür ist das Konzept synctex [21], das eine Verbindung von LaTeX-Zeilen und zugehörigen PDF-Zeilen auf Basis von Positionen ermöglicht. Wichtig ist hierbei jedoch, dass zu einem zugehörigen LaTeX-Dokument ein Synctex-File erstellt wird, das die wesentlichen Informationen der Verbindung bereitstellt. Während diese Methodik in verschiedenen Programmen intern zum Einsatz kommt, lässt sich synctex auch über die Shell bedienen.

Abbildung 8: Synctex in der Shell. Für ein PDF test.pdf und ein TEX-File test.tex bestimmen wir die Position von Zeile 4 des TEX-File im Bezug auf test.pdf

Abbildung 8 zeigt examplarisch die Informationen eines synctex-Aufrufs. Hierbei arbeiten wir auf einem LaTeX-File *test.tex*, zu dem wir vorab mit *pdflatex -synctex=-1 test.tex* das synctex-File und das entsprechende PDF erstellt haben. Dieses PDF *test.pdf* wird ebenfalls als Eingabe verwertet, damit die Verbindung zwischen LaTeX-File und PDF-File hergestellt werden kann. Die '4:0' des Aufrufs beschreibt hierbei, dass wir uns für Zeile 4, Spalte 0 (gleichbedeutend mit *alleSpalten*) des LaTeX-Files interessieren.

Als Ergebnis des Aufrufs erhalten wir eine Liste von Elementen und ihren Positions-Daten. In der Ausgabe von Abbildung 8 haben wir genau ein Element vorliegen. Dabei sind insbesondere die folgendenden Informationen relevant:

- **Output** - auf dieses PDF referenzieren sämtliche Werte
- **Page** - die zugehörige Seitenzahl im PDF
- **h** - die x-Position gemessen in Pixeln von links

- **v** - die y-Position gemessen in Pixeln von oben
- **W** - die Breite gemessen in Pixeln von oben
- **H** - die Höhe gemessen in Pixeln von oben

Anhand dieser Informationen können wir zu jeder LaTeX-Zeile das zugehörige Positions-Rechteck im PDF bestimmen. Falls synctex zu einer Zeile mehrere Elemente erkennt, verbinden wir diese zu einem Gesamt-Rechteck.

Wir können also wie gehabt LaTeX-Zeilen betrachten, die im PDF einen Paragraphen darstellen werden, und nun ihre Positions-Rechtecke verbinden. Diese Information extrahieren wir als eine Liste von Rechtecken (vgl. Abbildung 9).

page	minX	minY	maxX	maxY
1	71,999985	570,334351	538,251526	769,890015
1	71,999985	216,897522	538,251526	504,918854
1	71,999985	169,081787	538,251526	202,686584
1	71,999985	135,211304	538,251526	144,908264
...	...	...	...	...

Abbildung 9: Ausgabe der Rechtecke

Somit besitzen wir Informationen über die Rechtecke der Paragraphen im PDF. Darauf aufbauend können wir nun die Text-Zeilen des PDFs aus dem TSV-File betrachten (vgl. Abbildung 2). Für alle *line*-Zeilen suchen wir das Paragraphen-Rechteck mit der größten Überschneidung und verbinden diese *line*-Zeilen so zu Text-Paragraphen. An dieser Stelle können wir problemlos anhand der Koordinaten bestimmen, welche *line* die erste des Paragraphen ist. Bei diesen *line*-Elementen tragen wir in der Spalte 'is_start_of_paragraph' eine 1 ein und erhalten unsere gewünschten Testdaten.

6 Analyse

6.1 Theoretisch

Es folgt eine Analyse der verwendeten Algorithmen. N beschreibt hierbei die Anzahl der gesamten zu bewertenden Zeilen, M die Anzahl der Features.

6.1.1 Extern: Algorithmus zum Parsen eines PDFs

Input: PDF-File
Output: TSV-File mit Informationen zu Wörtern und Zeilen des Elements (vgl. Abbildung 2)
Performanz: $O(A1)$ [1]

6.1.2 Extern: Algorithmus zum Parsen eines TEX-Files

Input: TEX-File
Output: TXT-File mit Informationen zu Paragraphen-Rechteck im späteren PDF, die dem TEX-File entnommen werden konnten (vgl. Abbildung 9)
Performanz: $O(A2)$ [1]

6.1.3 Algorithmus zum Generieren von Trainingsdaten

Input: TEX-File
Output: TSV-File, das Trainingsdaten beschreibt, d.h. um die Spalte 'is_start_of_paragraph' erweitert wurde.
Performanz: $O(N^2 + A1 + A2)$
N^2 für den Fall, dass wir alle Zeilen miteinander vergleichen müssen. Tritt auf, wenn alle Zeilen aus dem TEX-File als Paragraph klassifiziert wurden.
$A1$ zum Parsen des aus dem TEX-File generierten PDF.
$A2$ zum Parsen des TEX-Files selbst.

6.1.4 Algorithmus zur Evaluation eines Modells

Input: Verwendetes Modell
Output: Precision, Recall und F1-Score des Models auf einem Trainingsdokument.
Performanz: $O(N \cdot M)$
Für jede der N Zeilen müssen alle M Features berechnet werden.

6.1.5 Haupt-Algorithmus zur Paragraphen-Bestimmung

Input: Menge an TSV-Files als Trainingsdaten, PDF zur Paragraphen-Bestimmung
Output: TSV-File mit Informationen zu den erkannten Paragraphen, d.h. ausgefüllter Spalte 'is_start_of_paragraph' für jede Zeile. Zusätzlich wird ein PDF mit Visualisierungen der Paragraphen erstellt.
Performanz: $O(N \cdot M + A1)$
Das PDF wird in $A1$ geparst. Anschließend werden für jede der N Zeilen alle M Features berechnet.

6.2 Empirisch

Betrachtet man die Arbeit bzw. die zugrundeliegenden Algorithmen als Ganzes, so ist vor dem Hintergrund der Problemstellung besonders die Qualität der Ergebnisse von Relevanz. Erreicht das Resultat eine ausreichende Güte, lassen sich etwaige Laufzeit-Probleme kompensieren.

Im Folgenden werden die verschiedenen in dieser Arbeit verwendeten Modelle jeweils unter verschiedenen Qualitätsindikatoren verglichen. Dabei wurden alle in 4.3 beschriebenen Features verwendet.

6.2.1 Precision und Recall

Im Bereich Information Retrieval beschreibt Precision den Anteil der erhaltenen positiven Ergebnisse, die tatsächlich relevant sind. Somit betrachten wir in unserem Fall, wie viele der Zeilen, die wir als Anfang eines Paragraphen bestimmen, auch wirklich Paragraphen-Anfänge sind. Precision legt folglich Wert darauf, dass keine falsch-positiven Ergebnisse auftreten, also solche Zeilen, die in der tatsächlichen Lösung mit 0, von unserem Algorithmus aber mit 1 bewertet wurden.

Recall beschreibt die Anzahl an relevanten Ergebnissen, die anteilig zur Gesamtmenge der relevanten Ergebnisse erkannt wurden. Wir interessieren uns also dafür, wie viele Paragraphen-Anfänge auch tatsächlich als solche erkannt wurden. Wie auch die Werte für Precision bewegt sich der Wert für Recall zwischen 0 und 1, wobei 1 der Idealwert ist.

Resultierend aus den Werten für Recall und Precision kann man den F1-Score bestimmen, der ein Maß für die Genauigkeit der Tests beschreibt.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{9}$$

Abbildung 10 zeigt eine Auswertung des Algorithmus auf 1000 repräsentativen Beispiel-Dokumenten. Es wird unterschieden zwischen den vier zur Verfügung stehenden Modellen.

Es ist unschwer zu erkennen, dass Support Vector Machines von den reinen Werten her die anderen Verfahren um Längen übertreffen. Am schlechtesten schneiden demnach Perceptrons ab. Zwar haben Perceptrons einen sehr hohen Precision-Wert, doch resultiert dieser aus dem Umstand, dass so gut wie keine Zeilen mit 1 bewertet werden. Diese sind dann ein Bruchteil der zu erkennenden Zeilen, was zu einem sehr hohen Precision-Wert führt. Vor dem Hintergrund des Recall-Wertes wird dieser Precision-Wert allerdings in Relation gesetzt und in seinem positiven Eindruck stark gemindert. Logistic Regression sowie Naive Bayes liefern bessere, aber nicht wirklich überzeugende Ergebnisse.

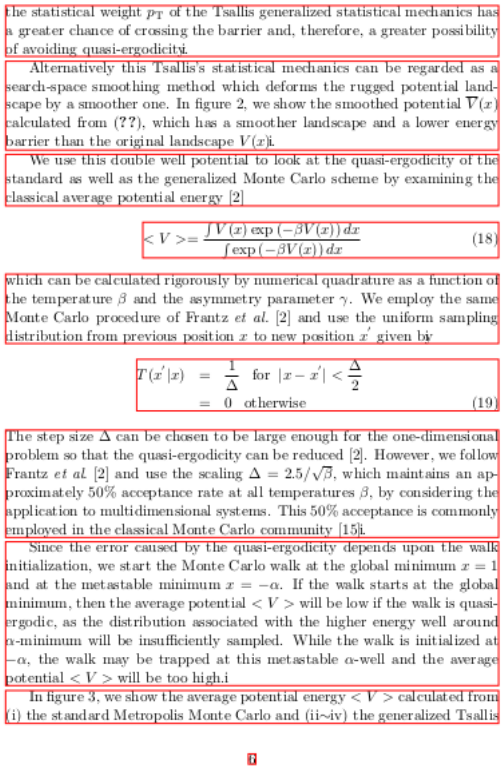
Model	**Precision**	**Recall**	**F1**
Logistic Regression	0.93	0.70	0.80
Support Vector Machines	0.98	0.98	0.98
Perceptrons	0.98	0.24	0.39
Naive Bayes	0.97	0.51	0.67

Abbildung 10: Precision, Recall und F1-Score auf 1000 Datensätzen

6.2.2 Intuitiv

Da unsere Zielsetzung die Erkennung von Paragraphen nach menschlichem Empfinden ist, muss die Qualität des Ergebnisses auch auf dieser Meta-Ebene analysiert werden. Zwar lassen sich menschliche Eindrücke im Hinblick auf das Ergebnisse nicht rational messen, doch liefern sie Tendenzen, die den maschinell berechneten Qualitäts-Werten ähneln.

So schneiden Support Vector Machines bei der Erkennung von Paragraphen eindeutig am besten ab. Ein Beispiel-Ergebnis zeigt Abbildung 11. Sowohl Formeln als auch normale Paragraphen werden hier gut erkannt.


the statistical weight p_T of the Tsallis generalized statistical mechanics has a greater chance of crossing the barrier and, therefore, a greater possibility of avoiding quasi-ergodicity.

Alternatively this Tsallis's statistical mechanics can be regarded as a search-space smoothing method which deforms the rugged potential landscape by a smoother one. In figure 2, we show the smoothed potential $\overline{V}(x)$ calculated from (??), which has a smoother landscape and a lower energy barrier than the original landscape $V(x)$.

We use this double well potential to look at the quasi-ergodicity of the standard as well as the generalized Monte Carlo scheme by examining the classical average potential energy [2]

$$< V >= \frac{\int V(x) \exp\left(-\beta V(x)\right) dx}{\int \exp\left(-\beta V(x)\right) dx} \tag{18}$$

which can be calculated rigorously by numerical quadrature as a function of the temperature β and the asymmetry parameter γ. We employ the same Monte Carlo procedure of Frantz *et al.* [2] and use the uniform sampling distribution from previous position x to new position $x^{'}$ given by

$$\begin{aligned} T(x^{'}|x) &= \frac{1}{\Delta} \quad \text{for } |x - x^{'}| < \frac{\Delta}{2} \\ &= 0 \quad \text{otherwise} \end{aligned} \tag{19}$$

The step size Δ can be chosen to be large enough for the one-dimensional problem so that the quasi-ergodicity can be reduced [2]. However, we follow Frantz *et al.* [2] and use the scaling $\Delta = 2.5/\sqrt{\beta}$, which maintains an approximately 50% acceptance rate at all temperatures β, by considering the application to multidimensional systems. This 50% acceptance is commonly employed in the classical Monte Carlo community [15].

Since the error caused by the quasi-ergodicity depends upon the walk initialization, we start the Monte Carlo walk at the global minimum $x = 1$ and at the metastable minimum $x = -\alpha$. If the walk starts at the global minimum, then the average potential $< V >$ will be low if the walk is quasi-ergodic, as the distribution associated with the higher energy well around α-minimum will be insufficiently sampled. While the walk is initialized at $-\alpha$, the walk may be trapped at this metastable α-well and the average potential $< V >$ will be too high.

In figure 3, we show the average potential energy $< V >$ calculated from (i) the standard Metropolis Monte Carlo and (ii~iv) the generalized Tsallis

Abbildung 11: Beispiel-Ergebnis mit SVM

7 Zusammenfassung

Ziel dieser Arbeit war die Erkennung von Paragraphen und vergleichbaren Strukturen in einem gegebenen PDF. Zu diesem Zweck haben wir aus LaTeX-Dateien Trainingsdaten automatisiert generiert, die bzgl. jeder Textzeile Informationen zu ausgewählten Features enthalten. Zusätzlich haben wir für jede dieser Textzeilen bestimmt, ob sie den Anfang eines Paragraphen beschreibt. Auf Basis dieser Daten wurde ein Modell des maschinellen Lernens erstellt, das für jede neue Textzeile anhand der Features entscheidet, ob es sich um den Anfang eines Paragraphen handelt. Liegt also ein zu bewertendes PDF vor, wird dieses geparst, um die Trainings-Features erweitert und anschließend von unserem Modell bewertet. Die Ergebnisse visualisieren und speichern wir.

7.1 Ausblick

Wie die Analyse verdeutlicht, liefert das vorgestellte Verfahren in vielen Fällen sehr gute Ergebnisse. Dennoch können die berechneten Werte täuschen, da manche Dokumente sehr gut und andere weniger gut bearbeitet werden. Verfügt man über eine ausreichende Menge an Trainingsdaten – deren Erstellung im Rahmen dieser Arbeit behandelt wurde – muss eine weitere Optimierung zwangsläufig im Hinblick auf die verwendeten Features oder die verwendeten Algorithmen erfolgen.

Features
Im jetzigen Stadium des Algorithmus werden in erster Linie Features mit Bezug auf Positions-Größen und Schriftarten verwendet. Da in diesem Bereich kein großes Verbesserungspotenzial besteht, wäre es ein naheliegender nächster Schritt, die Features auf inhaltliche Aspekte zu erweitern. So könnte man die Erkennung eines neuen Paragraphen an Features festmachen, die beispielsweise auf bestimmte Schlüsselwörter achten. *'Im Folgenden ...'* deutet sicher häufiger auf einen beginnenden Paragraphen hin als andere Zeilenanfänge.

Allerdings können inhaltliche bzw. semantische Features schnell an ihre Grenzen stoßen. Insbesondere kann es zu *overfitting* kommen, wenn man anhand einer geringen Anzahl von PDFs bestimmte Merkmale eines Paragraphen-Anfangs festmachen will, diese jedoch in anderem Zusammenhang andere Rückschlüsse liefern.

Des Weiteren kann die Realisierung der Features Probleme bereiten. Will man bspw. den inhaltlichen Zusammenhang von Textblöcken ermitteln, erfordert das eine sehr tiefe und mitunter sehr zeitintensive Berechnung.

Nichtsdestotrotz würde eine überzeugende Umsetzung semantischer Features zwangsläufig das Ergebnis verbessern.

Algorithmen
In der dargestellten Evaluation lieferten SVM eindeutig die besten Ergebnisse. Da diese Analyse aber lediglich in Relation zu einer geringen Anzahl an Modellen betrieben wurde, ist nicht auszuschließen, dass andere Algorithmen noch bessere Ergebnisse liefern.

Insbesondere ist hier an komplexere Ansätze zu denken wie neuronale Netzwerke, die allerdings über die Problemstellung der vorliegenden Untersuchung hinaus reichen.

7.2 Fazit

Obgleich wir also festgestellt haben, dass bei der Erkennung Verbesserungspotenzial besteht, liefert das vorgestellte Verfahren auch zum jetzigen Zeitpunkt bereits gute Ergebnisse. An dieser Stelle sei betont, dass die im Rahmen dieser Arbeit entstandene zuverlässige algorithmische Erstellung von Trainingsdaten eine weitere Optimierung nicht nur begünstigt, sondern explizit befördert. Somit kann man sich auf dieser Basis eine weitere Verbesserung der Methode vorstellen.

Insgesamt kann diese Arbeit somit nicht als vollständige Lösung des Problems betrachtet werden – diese anzustreben mag generell schwierig sein –, wohl aber als Analyse und Strukturierung der Problematik sowie als Rahmen zur weiteren Verbesserung und Optimierung.

Literaturverzeichnis

[1] Claudius Korzen. Icecite. `https://github.com/ckorzen/icecite`, 2016.

[2] Apache PDFBox 2.0.2. `https://pdfbox.apache.org/`, 2016.

[3] Ross Moore. Ongoing efforts to generate 'tagged PDF' using pdfTEX. In *Towards a Digital Mathematics Library*. Masaryk University Press, 2009.

[4] Min-Yen Kan et al. ParsCit: An open-source CRF reference string parsing package. LREC, 2008.

[5] John Lafferty, Andrew McCallum, Fernando C.N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *ICML '01 Proceedings of the Eighteenth International Conference on Machinen Learning*. Morgan Kaufmann Publishers Inc., 2001.

[6] Alexandru Constantin, Steve Pettifer, Andrei Voronkov. PDFX: fully-automated PDF-to-XML conversion of scientific literature. In *DocEng '13*. ACM New York, 2013.

[7] Caroline Sporleder, Mirella Lapata. Automatic Paragraph Identification: A Study across Languages and Domains. In *Proceedings of EMNLP 2004*. Association for Computational Linguistics, 2004.

[8] Dimitriy Genzel, Eugene Charniak. Variation of entropy and parse trees of sentences as a function of the sentence number. In *Proceedings of Empirical Methods in Natural Language*. Sapporo, 2003.

[9] Patrice Lopez. GROBID: Combining Automatic Bibliographic Data Recognition and Term Extraction for Scholarship Publications. Conference: Research and Advanced Technology for Digital Libraries, 13th European Conference, 2009.

[10] Edward Vanhoutte1. An Introduction to the TEI and the TEI Consortium. Association for Literary and Linguistic Computing, 2004.

[11] Karl Jonathan Ward. PDF-Extract. `https://github.com/GullyAPCBurns/lapdftext`, 2015.

[12] MK II: Wissensrepräsentation. Bauhaus-Universität Weimar, 2013.

[13] Dr. David Sabel. Einführung in die Methoden der Künstlichen Intelligenz. Goethe-Universität Frankfurt am Main, 2013.

[14] R. Bender, A. Ziegler. Logistische Regression. *Statistik-Serie in der DMW, Artikel Nr. 14*, 2002.

[15] Fabian Pedregosa et al. Scikit-learn: Machine Learning in Python. In *Journal of Machine Learning Research 12*. JMLR, 2011.

[16] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze. Support vector machines and machine learning on documents. In *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[17] OpenCV. Introduction to Support Vector Machines. http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html, 2016.

[18] http://i.stack.imgur.com/1gvce.png, 2016.

[19] Prof. Dr. Hannah Bast. Information Retrieval. Albert-Ludwigs-Universität Freiburg, 2015.

[20] Vladimir I. Levenshtein. 'Binary codes capable of correcting deletions, insertions, and reversals'. In *Doklady Akademii Nauk SSSR*. Soviet Physics Doklady, 1966.

[21] Jérôme Laurens. Direct and reverse synchronization with SyncTeX. In *TUGboat, Band 29, Nr. 3*. Tex Users Group (USA), 2008.