

# Entwicklung eines KI-basierten System zur Layout-Analyse und Ermittlung der Lese-Reihenfolge in gescannten Dokumenten

Jörn Bachmeier                            Matthias Lang  
joernbachmeier@freenet.de            matthias.lang@student.hs-rm.de

21. Mai 2023

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>3</b>
<b>2 Datensätze</b>	<b>5</b>
2.1 DocLayNet . . . . .	5
2.2 ReadingBank . . . . .	8
2.2.1 Preprocessing . . . . .	9
<b>3 Modelle</b>	<b>11</b>
3.1 Box-Class Predictor . . . . .	11
3.1.1 Training . . . . .	11
3.1.2 Evaluation . . . . .	12
3.2 Reading Order-Modell . . . . .	13
3.2.1 Modellierung . . . . .	13
3.2.2 Features . . . . .	14
3.2.3 Sampling . . . . .	17
3.2.4 Reading Order-Bestimmung . . . . .	18
3.2.5 Training . . . . .	19
3.2.6 Evaluation . . . . .	20
<b>4 Experimente und Ergebnisse</b>	<b>21</b>
4.1 Box-Class Prediction Experimente . . . . .	21
4.1.1 LR vs XGB . . . . .	21
4.1.2 LR+Font vs XGB+Font . . . . .	22
4.1.3 Finales Modell: XGB+Size . . . . .	23
4.1.4 Transferierung des Classifiers auf den Reading Bank Datensatz . . . . .	25
4.2 Reading Order-Modell . . . . .	26
4.2.1 Diskussion . . . . .	29

<b>5 Fazit</b>	<b>36</b>
5.1 Aussicht . . . . .	36
<b>A Anhang</b>	<b>39</b>
A.1 Bash-Script zum Sammeln und Mischen der Trainings-/Validierungsdaten . . . . .	39
A.2 Zusätzliche BLEU-Scores . . . . .	40

## 1 Einleitung

In den letzten Jahren wurden immer mehr Daten digitalisiert. Dennoch gibt es Dokumentarten, die weiterhin in Papierform ausgehändigt werden, wie Rezepte, Rechnungen, usw. Versicherungen sind ein Beispiel, bei denen jeden Tag unzählige gescannte Rezepte eingeschickt werden, wodurch sie das Dokument zwar digital besitzen, aber keine Metadaten zur Verfügung haben, um den Inhalt automatisch bestimmen zu können. Die meisten Unternehmen haben eine eigene Papiervorlage für ihre Dokumente und folgen keiner allgemein gültigen Norm. Das sorgt dafür, dass keine regelbasierten Systeme für die Inhaltsbestimmung eingesetzt werden können. Deswegen müssen Arbeitskräfte die Dokumente lesen und bearbeiten. Das ist teuer für die Unternehmen und ist schlecht skalierbar, falls das Unternehmen weiter wächst.

Heutzutage gibt es mit Hilfe von OCR (Optical Character Recognition)<sup>1</sup> bereits Möglichkeiten den Inhalt von eingescannten Dokumenten oder Bildern zu erfassen. Unternehmen wie Apple<sup>2</sup> haben bereits seit Jahren solche Technologien in ihren Produkten verbaut. Allerdings lässt sich damit keine **Reading Order** ableiten und damit auch keine Möglichkeit den Inhalt sinnvoll wiederzugeben. Eine Reading Order bestimmt in welcher Reihenfolge einzelne Layout-Elemente gelesen werden sollen und damit den Lesefluss eines Dokuments.

In diesem Projekt wird daher ein Modell entwickelt, welches die Reading Order von Dokumenten predicted und eine sinnvolle Bestimmung des Inhalts ermöglicht. Die Reading Order ist damit eine wichtige Wissensebene, um sie für verschiedene *Downstream Tasks* anzuwenden. So kann sie nicht nur für die oben erwähnten Beispiele eingesetzt werden, sondern auch für Systeme, die Dokumente für Blinde vorlesen oder um *Sentence Embeddings* für z. B. *Summary Tasks* zu erstellen. Dabei wird ein Satz im Kontext seiner ganzen Seite betrachtet. Die Anwendungsfelder sind weitreichend.

Aus dem beschriebenen Szenario heraus fokussiert sich das Projekt auf Dokumente, die nur wenig Textfluss zulassen. Es wird sich mit der Frage beschäftigt, wie das Layout eines Dokuments mit der Reading Order zusammenhängt und welche Features am wichtigsten zu betrachten sind. Durch den geringen Anteil an Fließtext in solchen Dokumenten werden keine Sprachmodelle (engl. Language Model, kurz: LM) eingesetzt, da es starke *semantische Breaks* zwischen einzelnen Zeilen gibt. Außerdem tauchen überall Abkürzungen und Zahlen auf, die sehr unverständlich für viele LMs sind. Ein weiterer Grund ist, dass viele Menschen das Layout und deren Reading Order anhand von Positionen der Boxen erkennen können und deshalb ein Teil der Forschung sein soll, inwiefern Machine Learning-Modelle diese reproduzieren können.

Für dieses Vorhaben werden Layout-Informationen gesammelt und verschiedene Feature getestet. Dafür werden **Bounding Boxen** um Layout Elemente eines Dokuments herum gezogen, die das Fundament für die Features darstellen. Mit Hilfe dieser werden unterschiedliche Features entwickelt und miteinander verglichen. Gleichzeitig wird ein **Classifier** eingesetzt, der die Bounding Boxen einer Klasse zuordnet, um neben den Layout Informationen ein weiteres Feature für die Reading Order zu

<sup>1</sup><https://aws.amazon.com/de/what-is/ocr/>, abgerufen: 22.03.2023

<sup>2</sup><https://www.maclife.de/ratgeber/ios-15-texte-bildern-erkennen-gehts-100119428.html>, abgerufen: 22.03.2023

formen. Die Idee ist es, dem Modell mit Klassen eine natürliche Reihenfolge anzutrainieren, weil z. B. „Text“ kommt meistens nach einer Überschrift („Title“) und eine „Caption“ meist vor oder nach einer Abbildung („Tabel“/„Picture“). Zusammen ergibt sich damit ein Modell, welches die Reading Order von Layout Elementen predicted und damit den Lesefluss des Dokuments bestimmt (Abbildung 1).

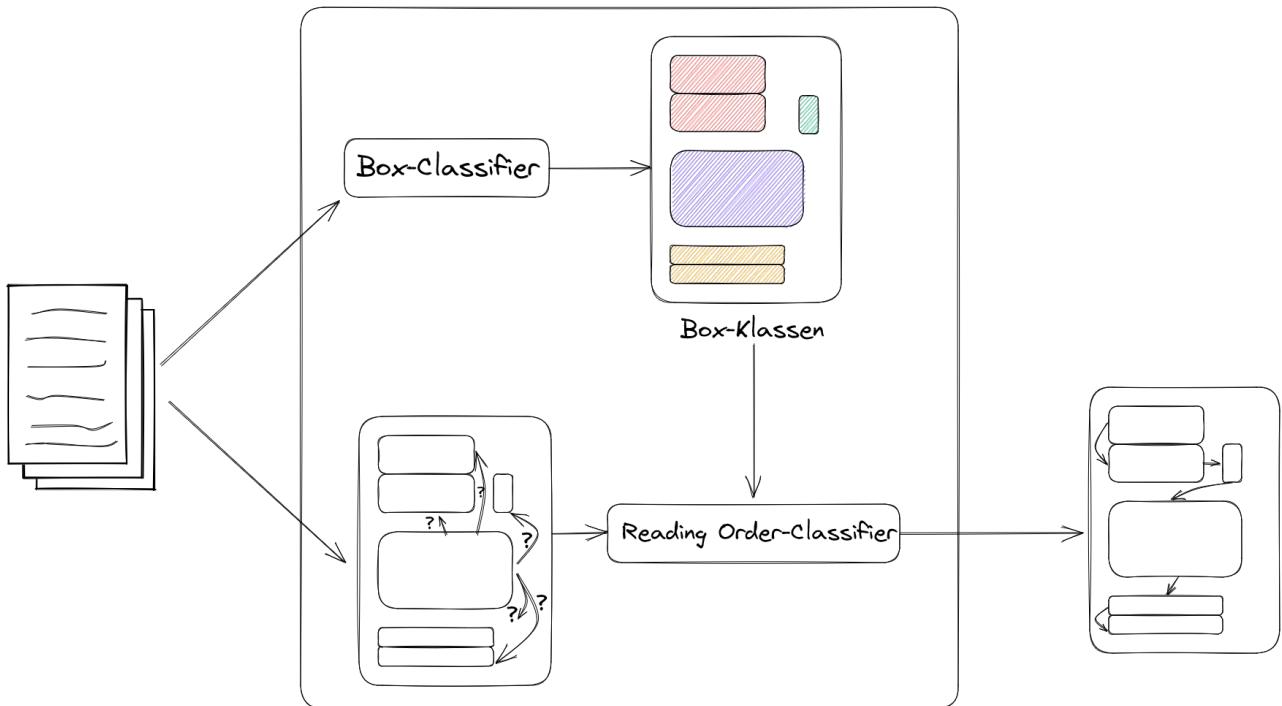


Abbildung 1: Systemdiagramm des Reading Order Systems. Erstellt mit Excalidraw. Das System erhält verschiedene Arten von Dokumenten, um dessen Layout Elemente, Bounding Boxen gezogen wurden. Mit Hilfe der Bounding Boxen werden Layout-Features entwickelt und eine Klassifizierung durchgeführt, um die Reading Order zu bestimmen.

## 2 Datensätze

Im folgenden Kapitel werden die Datensätze analysiert, mit denen im Anschluss der Box-Class Predictor (Kapitel 3.1) und das Reading Order-Modell trainiert werden. Der DocLayNet-Datensatz wird für das Klassifizieren der Layout-Elemente verwendet und der Reading Bank-Datensatz für die Reading Order-Bestimmung.

### 2.1 DocLayNet

Das Modell zur Erkennung der Reading Order erhält ein Teil der Features für sein Training aus einem Classifier. Dieser Classifier soll anhand weniger Features, Layout-Elemente klassifizieren und eine Wahrscheinlichkeitsverteilung über die Klassen erstellen. Dafür wird er auf verschiedenen Dokument-Layouts trainiert. Er bekommt eine Bounding Box und wenige Font-Informationen und klassifiziert damit das Layout-Element.



Abbildung 2: Beispiele aus dem DocLayNet Paper, die eine Vielfalt an unterschiedlichen Dokumenten zeigen.

Für dieses Vorhaben wurde sich für den DocLayNet-Datensatz entschieden, der 80.863 unterschiedlichen Dokumentseiten besitzt, die alle auf eine Größe von 1025 \* 1025 skaliert sind [3]. Eine Dokumentseite besteht aus mehreren Layout-Elementen, die alles darstellen was auf einer Seite zu sehen ist (Abbildung 2). Um die Layout-Elemente herum liegen die Bounding Boxen, die mit zwei Punkten, top\_left ( $x_1, y_1$ ) und bottom\_right ( $x_2, y_2$ ), ein Rechteck aufspannen. Außerdem besitzt der Datensatz Font-Informationen zu jedem Element, die aus Font-Size, -Name und -Color bestehen. Weitere Features, die für dieses Szenario uninteressant sind, sind die original Bilder in PDF und PNG Format und die Texte aus den Bounding Boxen in zufälliger Reihenfolge.

Der Datensatz enthält nur wenige gescannte Dokumente, vor allem Rezepte, Rechnungen und Handgeschriebenes, da die Annotation für diese schwieriger für die Entwickler ist. Das ist für die Aufgabe des Modells in Ordnung, da sich auf technische PDFs im Bereich „government offices, company websites, financial reports, etc.“ [3] mit mindestens 10 Seiten fokussiert wird und diese Art von Inhalt aus vielen Tabellen, Abbildungen, usw. besteht. Genau diese komplizierten Layout-Strukturen mit vielen Tabellen-ähnlichen Elementen sind der Grund, warum sich für diesen Datensatz entschieden wird. Der

wichtigste Punkt ist die Ground-Truth, die aus 11 verschiedenen Klassen (*Caption*, *Footnote*, *Formula*, *List-item*, *Page-footer*, *Page-header*, *Picture*, *Section-header*, *Table*, *Text* und *Title*) besteht, welche die Elemente voneinander trennen und wichtige Informationen beinhalten.

Die Schwierigkeit der Annotation in diesem Forschungsgebiet ist die subjektive Interpretation von Elementen, die von Person zu Person unterschiedlich sein kann. Dabei ist es nicht nur subjektiv zu welcher Klasse welche Box gehört, sondern auch welche Subobjekte ein übergeordnetes Layout Element zusammen bilden. Dafür wurden vereinzelt mehrere Labels pro Sample zugelassen. Dokumente, bei denen zum Beispiel erkannte Bounding Boxen überlappen, wurden nicht in den finalen Datensatz mit aufgenommen [3]. Dennoch bereitet genau diese Subjektivität das spätere Klassifizieren und Erkennen der Reading Order Schwierigkeiten.

Klasse	Anzahl Training	...in %	Anzahl Validation	...in %
Caption	19218	2.04	1763	1.79
Footnote	5619	0.59	312	0.31
Formula	21167	2.25	1894	1.89
List-item	161818	17.19	13320	13.34
Page-footer	61313	6.51	5571	5.58
Page-header	47973	5.10	6683	6.69
Picture	39667	4.21	2775	2.78
Section-header	118590	12.60	15744	15.77
Table	30070	3.19	2269	2.27
Text	431251	<b>45.82</b>	49186	<b>49.28</b>
Title	4437	0.47	299	0.29

Tabelle 1: Verteilung der Samples auf alle 11 Klassen. Die Klasse „Text“ ist stark überrepräsentiert mit 45.82% (Training) und 49.28% (Validation). Klassen wie „Footnote“ und „Title“ sind dafür unterrepräsentiert 0.59%/0.31% und 0.47%/0.29%.

Die Klassenverteilung des Datensatzes zeigt eine Überrepräsentation der „Text“-Klasse. Fast 50% der Klassen im Trainings- und Validierungsset bestehen aus Text-Elementen. Das macht bei einem Dokument-Layout natürlich Sinn, es wird sich aber zeigen, dass viele der anderen Klassen mit der Klasse „Text“ verwechselt werden. Dieser Bias hinsichtlich der Klasse „Text“ ist bekannt, wird dennoch als nicht problematisch angesehen, da der Reading Bank Datensatz hauptsächlich aus diesen besteht (siehe Kapitel 2.2).

Der DocLayNet-Datensatz beinhaltet mehreren Ordnern mit unterschiedlichen Files und Informationen. Im *Core*-Verzeichnis befinden sich bereits drei Splits des Datensatz für das Training, die Validierung und dem anschließenden Testen im *COCO*-Format. Aufgeteilt wurde der Datensatz in einem 80/10/10-Split und beinhaltet jeweils eine ähnliche Verteilung an Layouts und Klassen. In den Files befinden sich Informationen über die Ground-Truth(-Klassen) und alle Dokumente, die zu den Splits gehören. Jedes Dokument besteht aus einer Größe, die auf  $1025^2$  genormt ist, aus der Dokumenten Kategorie, die das Thema der Seite vorgibt und aus einem Filename, der zu dem Bild und den Feature-Informationen führt. Die Bilder sind im selben Verzeichnis unter PNG und die Dokumenten Informationen in einem weiteren Verzeichnis namens *Extra*. Dort befinden sich die originalen PDF-

```

def overlap(segment_bbox, bbox):
    if segment_bbox[0] <= bbox[0] and
       segment_bbox[1] >= bbox[1] and
       segment_bbox[2] <= bbox[2] and
       segment_bbox[3] >= bbox[3]:
        return True
    else: return False

x0, y0, w, h = c["bbox"]
bbox = [x0, x0+w, y0, y0+h]

```

Abbildung 3: Der Code auf der linken Seite berechnet die Bounding Box aus den gegebenen Segment-Boxen. Die rechte Seite überprüft, ob eine Wortbox in einer größeren Layout-Box vorkommt oder nicht.

Dokumente und die JSON-Files mit den wichtigen Features zu jedem Dokument. Jedes JSON-File ist unterteilt in die einzelnen Layout-Elemente und besteht aus den Bounding Boxen, der Klasse, dem Text (falls vorhanden) und den Font-Informationen jeder Zeile/jedes Wortes.

Der Datensatz für den Classifier wurde aus den COCO- und JSON-Files zusammengebaut. Er besteht aus den Dokumentenseiten, welche wiederum aus mehreren Layout-Elementen bestehen. Jedes Layout-Element besteht aus den Bounding Boxen einzelner Zeilen, Font-Informationen, Text und Klassen. Die Segment-Boxen aus den COCO-Files sind die für die Klassifizierung wichtigen Boxen. Da sie ausschließlich im Format  $(x_0, y_0, width, height)$  vorhanden sind, müssen die Element-Boxen zunächst in das Format  $(x_0, y_0, x_1, y_1)$  gebracht werden (Code auf der linken Seite von Abbildung 3).

Als nächstes wurden die Zeilen aus den JSON-Files auf die Segment-Boxen gemapped. Dies wird mit einer *overlap*-Methode3 realisiert, die überprüft, ob sich die Zeile in den Grenzen der Segment-Box befindet. Im selben Zug werden auch der Text und die Font-Informationen auf die Box gemapped, um eine Bounding Box, einen Text und eine durchschnittliche Font-Information pro Layout-Element zu erhalten.

Die Font-Color wird zur Farbe, die am häufigsten in der Box vorkommt. Sollte sie nicht als Information vorliegen, wird sie als schwarz interpretiert  $[0, 0, 0, 0]$ . Der Font-Name wird über ein One-Hot Encoding realisiert. Es wird im Vorhinein die 30 häufigsten Font-Namen gesammelt und daraus ein Vektor geformt mit 29 Nullen und einer 1 an der Stelle, an der der häufigste Font-Name aus der Bounding Box vorkommt. Sollte der Font-Name nicht vorhanden sein oder nicht zu den häufigsten 30 gehören, dann besteht der Vektor ausschließlich aus Nullen. Die Font-Size kann aus dem Datensatz direkt abgelesen werden und ist der Durchschnittswert aller Font-Sizes aus einer Bounding Box.

Dabei fällt auf, dass ein paar Samples keine Font-Informationen haben, die welche haben sollten, wie „Caption“ und „Text“. Die Klasse „Picture“, die erwartungsgemäß keine Font-Informationen haben sollte, hat dennoch zu fast 50% welche. Das führte in den ersten Tests des Classifiers zu Problemen in der Aussagekraft zu den Font-Features (Kapitel 4.1).

Der Datensatz beinhaltet weniger Dokumente, die klassischerweise dem ausgewählten Szenario entsprechen, verfügt aber dennoch über viele Tabellen-ähnliche Layouts, die auch für die Reading Order hilfreich sind. Der neue Datensatz beinhaltet nur noch die wichtigsten Layout-Informationen und wird in allen Bereichen, die mit Text zu tun haben gekürzt. Außerdem werden alle Lücken im Datensatz

Klasse	Trainingsset	Validationset
Caption	0.07	0.37
Footnote	0.0	0.0
Formula	0.0	0.0
List-item	0.0	0.0
Page-footer	0.0	0.0
Page-header	0.0	0.0
Picture	55.84	45.23
Section-header	0.0	0.0
Table	0.0	0.0
Text	0.01	0.02
Title	0.0	0.0

Tabelle 2: Die Samples der meisten Klassen sind mit Font-Informationen. Das macht Sinn, da die meisten Klassen Text-basiert sind. Nur einzelne Samples aus der Klassen „Caption“ und „Text“ sind ohne Font-Informationen. Nur die Klassen „Picture“ hat zu 55.84%/45.23% keine Font-Informationen, was zunächst nach einer großen Zahl aussieht, aber eigentlich sehr wenig ist, da davon ausgegangen wird, dass Bilder keine Font-Informationen haben.

geschlossen und durch Default-Werte ersetzt, um einen einheitlichen Featurevektor zu erschaffen mit dem der Classifier trainiert werden kann. Die Ground Truth bildet dabei eine feine Unterteilung, die viele Informationen beinhalten kann.

## 2.2 ReadingBank

Um eine Reading Order trainieren zu können, braucht es einen Datensatz, der diese als Ground Truth besitzt. Dafür wird in diesem Projekt der ReadingBank-Datensatz verwendet, welcher aus „500.000 Dokumenten besteht, mit einem weiten Spektrum an Dokumententypen“ [4]. Für diesen Datensatz wurden WORD-Dokumente aus dem Internet gesammelt, denn in den XML-Files der WORD-Dokumente findet sich alle nötigen Metadaten, so auch die Reading Order. Der Text einer Seite wird dabei als langer Fließtext in der richtigen Reihenfolge ausgegeben. Die Wortboxen hat der Datensatz, durch die von WORD in PDF umgewandelten Dokumente und dem anschließenden parsen durch einen PDF Parser. Außerdem beinhaltet der Datensatz die originalen Bilder (Abbildung 4 linke Seite).

Der tatsächlich verfügbare Inhalt der ReadingBank beinhaltet allerdings nur eine reduzierte Variante der Daten (Abbildung 4 rechte Seite). Er besteht aus einem Datensplit (Dev, Train, Test) mit jeweils 12 Files, in denen ausschließlich Wortboxen von tausenden Dokumenten enthalten sind. Jede Wortbox besteht aus einem Array von 4 Werten ( $\text{top\_left } (x_0, y_0)$  &  $\text{bottom\_right } (x_1, y_1)$ ). Ein einzelnes Dokument besteht aus  $\text{src}$ - und  $\text{tgt}$ -Boxen. Die  $\text{src}$ -Boxen beinhalten alle Wortboxen einer Seite in einer beliebigen Reihenfolge und die  $\text{tgt}$ -Boxen in der Richtigen. Die Reading Order lässt sich mit diesem Datensatz ausschließlich mit Features trainieren, die aus Layout-Informationen bestehen. Außerdem beinhaltet der ReadingBank-Datensatz einen *examples*-Ordner mit 100 PNG- und TXT-Files. Die TXT-Files haben neben der Wortbox auch das Wort selbst und den Color-Code. Damit lassen sich allerdings nur Tests durchführen und kein Training betreiben, aufgrund der geringen Datenmenge.



Abbildung 4: Auf der linken Seite befindet sich ein Dokument aus dem Reading Bank Datensatz. Auf der rechten Seite ist der Output eines Dokuments aus dem reduzierten Version, der bereits auf die spätere Größe von  $1025^2$  gestretched wurde.

Für dieses Projekt steht nur der reduzierte Datensatz zur Verfügung, weshalb Preprocessing Schritte vollzogen werden.

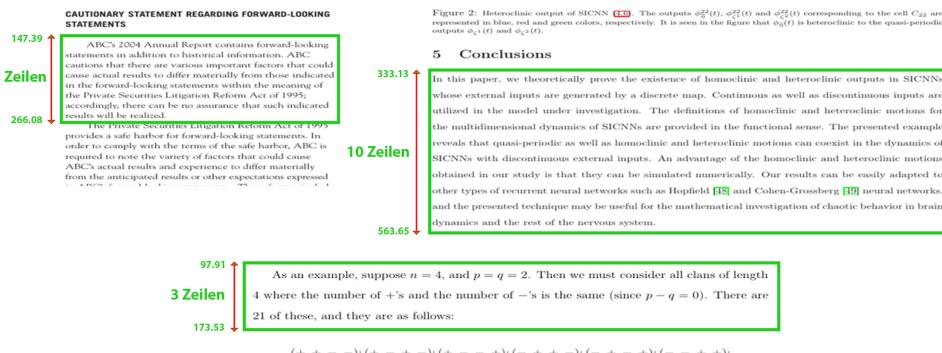
### 2.2.1 Preprocessing

Der ReadingBank-Datensatz beinhaltet weder die benötigten Bounding Boxen für den Classifier, noch die Font-Size des Textes in den Boxen. Daher muss vorab beides erstellt werden. Die Bounding Boxen werden mithilfe eines eigenen Algorithmus berechnet, der Wortboxen bekommt und daraus Layout-Boxen formt. Dafür werden zwei Thresholds für die X- und Y-Distanz zwischen einzelnen Boxen eingeführt, die überprüfen wie weit eine Box von der Anderen entfernt sein darf, bis sie zu einer neuen Bounding Box gehört. Beide Werte werden nach mehreren Iterationen auf 6 festgelegt.

Der Algorithmus bekommt alle Wortboxen einer Dokumente und baut im ersten Schritt daraus Zeilen. Dafür wird sich der obere linke Punkt der ersten Box gemerkt und solange der bottom\_right Punkt verschoben, wie es direkt nachfolgende Boxen mit dem selben Y-Wert gibt. Da es dabei zu minimalen Schwankungen in den Y-Werten der Zeile kommen kann, wie bei kursiv gedruckten Wörtern, gibt es hier eine Spanne von  $+/-3$  um den ersten Y-Wert herum. Sollte der bottom\_right X-Wert weiter weg von dem top\_left X-Wert sein, als der Threshold für die X-Distanz, wird die Zeile als abgeschlossen angesehen.

Hat der Algorithmus die Zeilen errechnet, wird ein **Agglomerative Clustering**<sup>3</sup> durchgeführt,

<sup>3</sup><https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html>, auf-



Font-Size	Segment Box Höhe	Anzahl Zeilen	Pixel pro Line	
1	118.69 px	8	14.84 px	Box links oben
1	27.25 px	2	13.63 px	-
1	58.19 px	4	14.55 px	-
11	75.62 px	3	25.21 px	Box rechts oben
9	230.52 px	10	23.05 px	Box unten

Abbildung 5: Die Abbildung zeigt drei Beispiele (Die Tabelle zeigt 5) aus dem DocLayNet-Datensatz und deren Box-Höhe. Der DocLayNet-Datensatz gibt dem Text eine Font-Size, die der Tabelle zu entnehmen ist. Die Höhe der Box geteilt durch die Anzahl der Zeilen ergibt eine durchschnittliche Zeilenhöhe (Pixel pro Line) für die Font-Size. Es fällt auf, dass die Font-Size=1 bis zu einer Zeilenhöhe von 15 px geht und alles darüber höhere Font-Sizes besitzt.

welches die Zeilen miteinander verbindet und größere Bounding Boxen erschafft, die Absätze darstellen. Das Clustering fügt dabei solange Boxen zusammen, bis ein zuvor fest gewählter Thrshold überschritten wird. Dies geschieht über die *Single-Linkage*-Methode. Als Abstandsmaß wird eine Manhattan-Distanz verwendet, welche den geringsten Abstand zweier Boxen zueinander berechnet. Sobald sich zwei Boxen in X- oder Y-Richtung überschneiden, wird die Distanz zu Null. Es entsteht eine Distanz-Matrix mit allen Manhattan-Distanzen zwischen den Clustern (Zeilenboxen), die Diagonale besteht aus Nullen. Nach wenigen Iterationen wird der Threshold mit dem Wert 6 belegt. Somit ergeben sich Cluster mit (Zeilen-)Boxen, die mindestens 6 Pixel voneinander entfernt sind.

Das zweite Feature für den Classifier, die Font-Size, wurde ebenfalls im Vorhinein geschätzt. Dafür wurden die Font-Sizes der DocLayNet-Samples mit ihren Bounding Box Höhen verglichen. Sollte eine Box über mehrere Zeilen gehen, wird die Höhe durch die Anzahl der Zeilen geteilt. Dabei kam heraus, dass die Font-Size bis zu einer Zeilenhöhe von 14/15 Pixel die Größe 1 hat und sie mit jedem Pixel mehr um eine Größe ansteigt. Für den ReadingBank-Datensatz wurde daher ein Wert von 15 px für die Font-Size=1 festgelegt. Außerdem wurden die Dokumente der ReadingBank auf die Größe der Bilder des DocLayNet ( $1025^2$ ) vergrößert. Wenn nun z. B. eine Textzeile eine Höhe von 25 px hat ( $25px - 15px = 10px$ ), dann beträgt die Font-Size:  $1 + 10 = 11$ . Sollte sie unter  $\leq 15$  px sein, so erhält sie die Font-Size=1.

gerufen: 22.03.23

<sup>3</sup><https://www.youtube.com/watch?v=RdT7bhm1M3E>, aufgerufen: 22.03.23

## 3 Modelle

In diesem Kapitel werden verwendeten Modell beschrieben und diskutiert.

### 3.1 Box-Class Predictor

Das Reading Order Modell bekommt als zusätzliches Feature eine Wahrscheinlichkeitsverteilung über typische Layout Klassen, die mithilfe eines Classifier erzeugt werden. Die Idee dahinter war es ein Feature zu erzeugen, das mehr über die einzelnen Boxen verrät, als nur die reinen Layout Informationen. Außerdem würden viele Personen ein „Header“ wahrscheinlich vor einem „Text“ lesen und eine „Caption“ direkt vor oder nach einem „Picture“.

#### 3.1.1 Training

Für den Classifier wurden typische Klassifizierungsmodelle verwendet, wie **Logistic Regression**<sup>4</sup> und **XGBoost**<sup>5</sup>. Logistic Regression ist genau wie XGBoost ein Standardmodell für Prediction- und Klassifizierungsprobleme, welche Wahrscheinlichkeiten über die möglichen Ergebnisse berechnen. Der XGBoost verwendet dafür den Gradient Boosting Algorithmen und ist dafür bekannt bessere und schnellere Ergebnisse hervor zu bringen, als viele andere Classifier im supervised learning Bereich. Da der Classifier nur als Modell dient, um ein Feature für das Reading Order-Modell zu erzeugen, wurde nicht weiter gefine-tuned. Beide Modelle wurden mit ihren Default Parametern aus der Sklearn-Bibliothek (XGBoost<sup>6</sup>) trainiert und keine Hyperparametersuche durchgeführt. Neuronale Netze wurden für dieses Projekt nicht ausprobiert, da sie den Umständen entsprechend nicht sehr ausdrucksstark gewesen wären und aufwendig hätten gefine-tuned werden müssen.

Insgesamt wurden für jedes Modell zwei Versionen trainiert, wobei das favorisierte Modell weiter gefine-tuned wurde. Das erste Modell (LR und XGB) bekommt ausschließlich die absoluten Werte der Bounding Boxen mit den Aufspannvektoren top\_left ( $x, y$ ) und bottom\_right ( $x, y$ ). Der Featurevektor besteht aus 4 Werten und errechnet, wie alle nachfolgenden Modelle auch, den Index der richtigen Klassen. Das zweite Modell (LR+Font und XGB+Font) erhält neben den Bounding Boxen auch alle Font-Informationen zu der Box. Wie in Kapitel 2.1 beschrieben, werden die Informationen Style, Size und Color über die Box zu jeweils einem Feature zusammengeführt. Der Featurevektor enthält neben den 4 Einträgen der Bounding Boxen nun auch 35 Werte aus den Font-Informationen. Insgesamt, bekommt das Modell einen Featurevektor von 39 Werten (Tabelle 3). Es wurde sich auf diese Features beschränkt, da sie auch in der Industrie über Hilfsmittel erhalten werden können.

Nachdem beide Modelle miteinander verglichen wurden (siehe Kapitel 4.1), wurde der XGB ein letztes Mal mit neuen Features trainiert. Das Modell XGB+Size bekommt neben den Bounding Boxen nur die Font-Size im Training (Featurevektor Länge: 5, Tabelle 3).

---

<sup>4</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html), aufgerufen: 22.03.23

<sup>5</sup><https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>, aufgerufen: 22.03.23

<sup>6</sup><https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html> und Logistic Regression<sup>7</sup>, aufgerufen: 22.03.23

Modelle	Features				Featurevektor
	Bounding Box (4)	Font-Style (30)	Font-Color (4)	Font-Size (1)	
LR & XGB	x				4
...+Font	x	x	x	x	39
XGB+Size	x			x	5

Tabelle 3: Diese Tabelle veranschaulicht welches Modell auf welche Feature trainiert wird und wie lang der daraus entstandene Featurevektor ist.

Zur Inferenzzeit bekommen die Modelle die selben Eingabevektoren und geben eine Wahrscheinlichkeitsverteilung über alle Klassen zurück, anstatt wie im Training nur die Klasse mit der höchsten Wahrscheinlichkeit.

### 3.1.2 Evaluation

Der erste Teil der Evaluierung, um die verschiedenen Test Durchläufe zu vergleichen, wurde mit dem AUROC Score umgesetzt. Der AUROC Score gibt eine Einschätzung darüber, ob das Modell am raten ist oder tatsächlich weiß, welche Klasse es auswählt. Dafür nimmt er einen zufälligen Classifier, der über alle Klassen hinweg gleichverteilt rät und vergleicht ihn mit dem eigentlichen Modell. Je höher der Score von AUROC ist, desto weniger rät das Modell und die Features sind von Relevanz. Ein Wert von 0,5 würde bedeuten, dass das Modell ausschließlich rät. Bei einem Wert unter 0,5 wäre das Modell schlechter als nur zu raten und alles über 0,5 beweist, dass das Modell wirklich versucht eine Lösung zu finden (Abbildung 6). Wurden erst einmal gute Features für das Modell gefunden, hat sich der AUROC Score schnell auf einen hohen Wert festgelegt und sich kaum verändert.

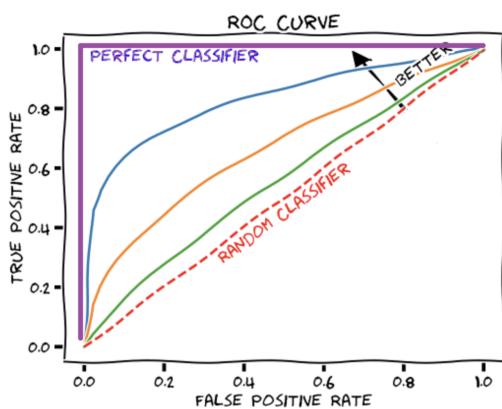


Abbildung 6: Der AUROC Score vergleicht einen zufälligen Classifier über alle Klassen mit dem eigentlichen Modell. Ist der Score über 0.5 bedeutet es, dass das Modell mehr als nur rät und wirkliche Ergebnisse erzeugt. Sollte der Wert unter 0.5 sein ist das Modell schlechter als gleichverteilt über alle Klassen zu raten (Bildquelle<sup>8</sup>).

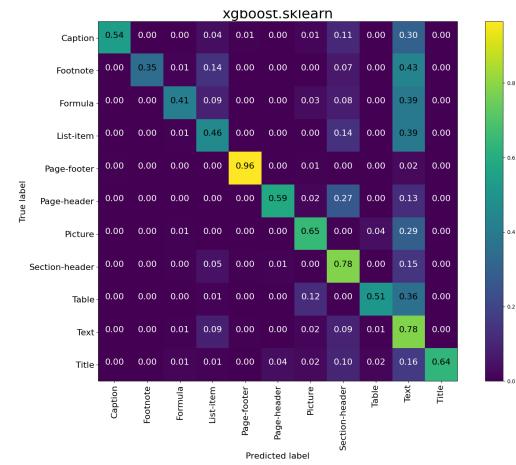


Abbildung 7: Die beispielhafte Confusion-Matrix zeigt eine Wahrscheinlichkeitsverteilung über alle 11 Klassen. Die Diagonale entspricht der Trefferquote für die Klassen und die restlichen Werte einer Zeile sind die Verwechslungsrate zu den anderen Klassen.

Der wichtigste Faktor zum Vergleichen der Modelle ist die eigen geschriebene „Hits\_at“-Evaluation. Dabei wird zwischen „Hits\_at\_1“ und „Hits\_at\_3“ unterschieden. Bei dem ersten Faktor wird geschaut wie sicher sich das Modell mit seinen Predictions ist, indem nur die Klasse verglichen werden, die das Modell mit der höchsten Wahrscheinlichkeit ausgegeben hat. Zusammen mit der Confusion Matrix lässt sich dann herausfinden, welche Klassen untereinander verwechselt werden (Abbildung 7). Die Diagonale einer solchen Matrix verrät den tatsächlich getroffenen Wert in Prozent. Eine ideale Confusion-Matrix würde 100% auf der Diagonalen zeigen. Die restliche Zeile einer Klasse gibt an, bei wie viel Prozent der Samples er die Klasse mit einer anderen Klasse verwechselt. Eine Zeile ergibt zusammen genau 100%. Darüber konnte später die Verwechslung zwischen „Picture“ und „Text“ herausgefunden werden, wodurch sich die Features final noch einmal änderten. Der zweite Faktor „Hits\_at\_3“ ist dafür da, um zu schauen, ob das Modell die richtige Klasse wenigstens in seiner Top 3 predicted. Da das Modell für das Reading Order System seine komplette Wahrscheinlichkeitsverteilung weitergibt, ist es ein guter Wert um herauszufinden, wie wertvoll das Modell für die spätere Reading Order ist.

### 3.2 Reading Order-Modell

Um eine Lesereihenfolge vorherzusagen, wird zunächst ein einfacher Fall betrachtet. Angenommen ein Dokument besteht aus genau zwei Boxen, die jeweils Absätze repräsentieren: Box A und Box B. Außerdem soll die korrekte Lesenreihenfolge gegeben sein durch: Box A → Box B (Auf Box A folgt Box B). Es wird weiter angenommen, dass alle Absätze in sich geschlossen bereits in der korrekten Lesereihenfolge stehen. Aus diesem Dokument mit lediglich zwei Absätzen lassen sich zwei Fälle formen:

1. Fall: Box A folgt auf Box B.
2. Fall: Box B folgt auf Box A.

Lediglich Fall 1 ist in diesem Szenario die korrekte Reihenfolge. Mehrdeutige korrekte Lesereihenfolgen werden nicht beachtet. Ein Beispiel dafür sind Bilder und ihre Beschreibungstexte, die sowohl vor als auch nach dem Bild gelesen werden können. Einen Bruch in der Lesereihenfolge gibt es dadurch nicht.

#### 3.2.1 Modellierung

Wie zuvor sind die Dokumente in Absätze gegliedert und stehen in der richtigen Lesereihenfolge. Ein Dokument besteht nun nicht mehr aus zwei Absätzen, sondern aus  $N_i$  vielen. Damit ergibt sich für das Dokument:

$$\mathcal{D} = \{A_1, A_2, \dots, A_{N_i}\} \subseteq \mathbb{R}^n.$$

wobei  $A_j$  sowohl Absätze als auch  $n$  Box-Features sind.

Das Modell soll nun vorhersagen, ob zwei gegebene Absätze direkt aufeinander folgen. Dementsprechend ist die Wahrscheinlichkeit gegeben als:

$$P(A_1 \rightarrow A_2 | x_{12}, \theta) = \begin{cases} 1 & \text{wenn } A_1 \rightarrow A_2 \\ 0 & \text{sonst} \end{cases},$$

---

<sup>8</sup><https://glassboxmedicine.com/2019/02/23/measuring-performance-auc-auroc/>, aufgerufen: 22.03.23

wobei  $x_{12}$  ein Kontext-Feature ist, das  $A_1 \rightarrow A_2$  in Beziehung setzt. Sollten die Boxen in umgekehrter Reihenfolge stehen oder zwischen sich eine weitere Box in der Lesereihenfolge haben, werden sie nicht erkannt und sind somit nicht aufeinanderfolgend:  $P(A_2 \rightarrow A_1 | x_{21}, \theta) = P(A_1 \rightarrow A_3 | x_{13}, \theta) = 0$ , wenn  $A_1 \rightarrow A_2 \rightarrow A_3$  gilt. Deswegen werden Absatz-Paare zur Bestimmung der Lesereihenfolge genutzt. Der Parameter  $\theta$  beinhaltet alle verwendeten Hyperparameter.

Allgemein wird dies mit beliebigen Absätzen  $A_i, A_j \in \mathcal{D}$  mit  $i, j \in \{1, \dots, N_i\}$ ,  $i \neq j$  und  $x_{ij} \in \mathbb{R}^m$  und einer zuvor festgelegten Lesereihenfolge wie folgt definiert:

$$P(A_i \rightarrow A_j | x_{ij}, \theta) = \begin{cases} 1 & \text{wenn } A_i \rightarrow A_j \\ 0 & \text{sonst.} \end{cases} \quad (1)$$

*Bermerkung:*

Gleiche Absätze sind kein Teil der Lesereihenfolge, da  $P(A_i \rightarrow A_i | x_{ii}, \theta) = 0$  für alle  $A_i \in \mathcal{D}$ . Die korrekte Lesereihenfolge eines Dokuments wird als gegeben vorausgesetzt.

Beispielhaft wird in Tabelle 4 die korrekte Lesenreihenfolgen drei fiktionaler Dokumente dargestellt. Es wird dabei vereinfacht angenommen, dass die jeweilige Lesereihenfolge der aufsteigenden Nummerierung der Absätze entspricht ( $A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow A_4$ ). In den Tabellen 4a – 4c sieht man die Ergebnisse aller möglichen Absatzkombinationen, wie sie sich aus Gleichung (1) ergeben.

	$A_1$	$A_2$
$A_1$	0	1
$A_2$	0	0

(a)  $A_1 \rightarrow A_2$

	$A_1$	$A_2$	$A_3$
$A_1$	0	1	0
$A_2$	0	0	1
$A_3$	0	0	0

(b)  $A_1 \rightarrow A_2 \rightarrow A_3$

	$A_1$	$A_2$	$A_3$	$A_4$
$A_1$	0	1	0	0
$A_2$	0	0	1	0
$A_3$	0	0	0	1
$A_4$	0	0	0	0

(c)  $A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow A_4$

Tabelle 4: Zeigt die Ergebnisse der Gleichung (1) angewandt auf fiktionale Dokumente, deren Lesereihenfolge durch die aufsteigende Nummerierung der Absätze gegeben ist. Zur Übersicht, sind die konkret abgebildeten Lesereihenfolgen unter den Tabellen 4a, 4b und 4c dargestellt.

Aus den Kombinationsmatrizen der Absätze in Tabelle 4 ergibt sich ein Klassifikationsproblem mit zwei Klassen. Die Klasse „1“ beschreibt hierbei die Situation, in der zwei Absätze direkt aufeinander folgen. Damit sind sie in der korrekten Lesereihenfolge. Die Klasse „0“ zeigt, dass die beiden Absätze nicht direkt aufeinander folgen. Anhand der Kombinationsmatrizen ist absehbar, dass eine große Imbalance zwischen den beiden Klassen existiert. Sie wird durch ein Sub-Sampling gelöst, aber dazu In Kapitel 3.2.3 mehr.

### 3.2.2 Features

Die hier verwendeten Features sind in zwei Kategorien unterteilt. Während die eine Kategorie sich auf einzelne Absätze bzw. deren Bounding Boxes bezieht, setzt die andere zwei Bounding Boxes in ein Verhältnis.

### Single-Box-Features

Diese Features beziehen sich auf einzelne Absätze mit dem Ziel, sie mit Informationen anzureichern. Dadurch ergibt sich eine aufgewertete Darstellung der Absätze.

Eine Bounding Box (kurz: Box) ist durch zwei sich gegenüberliegende Eckpunkte beschrieben. Dabei handelt es sich um die obere linke und untere rechte Ecke der Box. Diese Boxen werden durch den ReadingBank-Datensatz geliefert und anschließend die Eckpunkte normiert.

Die Fläche einer Box wird berechnet und durch die Wurzelfunktion regularisiert, um hohe Werte stärker zu regulieren als niedrige. Anzumerken ist weiterhin, dass es sich hierbei um eine nicht-lineare Regularisierung handelt. Sei  $a$  die Breite und  $b$  die Höhe einer Box, dann ist  $\sqrt{a \cdot b}$  der berechnete Flächeninhalt.

Das Seitenverhältnis wird durch den Bruch  $ratio = \frac{b}{a}$  (Breite geteilt durch die Höhe) einer Box ausgedrückt. Über das Seitenverhältnis erhält man Informationen über die Form der Box. Ist die  $ratio$  größer als eins, so handelt es sich um eine in die Breite gestreckte Box. Ist sie jedoch kleiner, dann handelt es sich um eine in die Höhe gestreckte Box.

Als letztes Feature werden die 11 Box-Klassen verwendet. Diese stammen aus dem Classifier aus Kapitel 3.1.

Zur besseren Übersicht werden die eben erwähnten Features in Tabelle 5 zusammengefasst. Dabei wird die Feature-Größe als Information ergänzt. Diese gibt die Anzahl von Einträgen pro Feature zu einem Absatz an.

Feature	Feature-Größe	Bemerkung
Box	4	Zwei sich diagonal gegenüberliegende Eckpunkte
Flächeninhalt	1	Regularisiert durch Wurzelfunktion
Seitenverhältnis	1	Indiz für Boxform
Box-Klassen	11	Klassen aus dem DocLayNet-Datensatz

Tabelle 5: Übersicht über die Single-Box-Features. In der Spalte **Feature-Größe** steht die Anzahl an Features, die durch das jeweilige Feature hinzugefügt wird. In der Spalte **Bemerkung** sind Hinweise für das Feature aufgeführt.

### Two-Box-Features

Die in diesem Abschnitt eingeführte Features beziehen sich auf zwei Bounding Boxes. Mit diesen Features werden Informationen über die Lage zweier Boxen zueinander in ein Kontext überführt.

Als Distanzen werden die Manhattan-Distanz und die euklidische Distanz eingesetzt. Erstere ist auch bekannt als Cityblock-Distanz. Der Einsatz dieser Distanz begründet sich in der natürlich vorliegenden Rasterung der Dokumente. Dabei wird die minimale Distanz der zwei betrachteten Boxen in x- sowie y-Richtung herangezogen. Überlappen die Boxen in einer Richtung, so wird die Distanz als Null angenommen. Abbildung 8 zeigt exemplarisch drei Boxen mit den aus der Anordnung resultierenden Manhattan-Distanzen. In dieser Abbildung sind keine Distanzen in x-Richtung für Box 1 zu 2 sowie Box 2 zu 3 eingezeichnet, da diese gleich Null sind. Weiterhin wird die euklidische Distanz zwischen

den beiden oberen linken bzw. unteren rechten Ecken als Feature eingesetzt. Durch dieses Feature erhält man Informationen über die direkte Nähe der beiden Ecken zueinander. In Summe entstehen vier Features: Manhattan-Distanz (nur x-Richtung), Manhattan-Distanz (nur y-Richtung), euklid. Distanz obere linke Ecken, euklid. Distanz untere rechte Ecken.

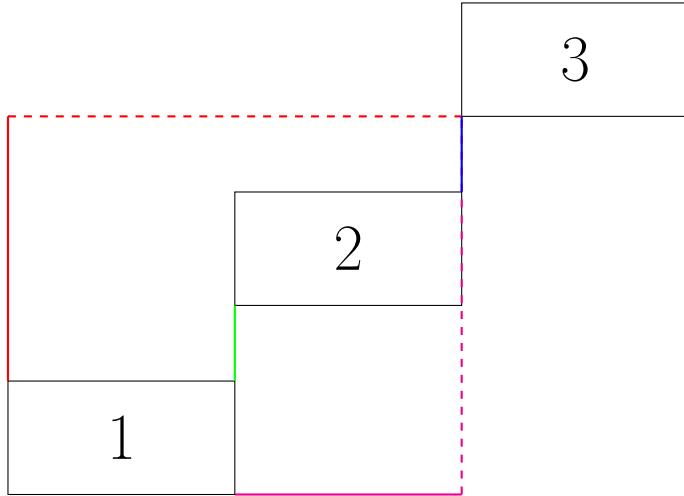


Abbildung 8: Zeigt die minimalen Manhattan-Distanzen aller Boxen zueinander. Durchgezogene Linien stellen die Manhattan-Distanz dar. Gestrichelte Linien sind Hilfslinien zur besseren Orientierung im Schaubild. Die Boxen 1 und 2 sowie 2 und 3 haben eine Manhattan-Distanz von Null, da deren linken bzw. rechten Seiten genau übereinander angeordnet sind.

Außerdem werden geographische Features eingesetzt. Diese geben Aufschluss auf die räumliche Lage der Boxen zueinander. Dazu werden die Zentren der Boxen bestimmt. Von dem Zentrum einer Box ausgehend wird überprüft, ob das Zentrum einer weiteren Box oberhalb, unterhalb, links oder rechts des Zentrums der ersten Box liegt. Dafür wird der Raum um eine Box herum in Quadranten geteilt und anschließend geprüft in welchem Quadranten die nächste Box liegt. Abbildung 9 illustriert die selben Boxen wie in Abbildung 8 nur mit eingezeichneten Zentren. In Box 2 sind zusätzlich noch die Entscheidungsgrenzen eingezeichnet. Überschreitet ein Zentrum eine dieser Grenzen, so verändern sich die geographischen Features. Insgesamt werden daraus vier Features. Das Feature setzt sich dann wie folgt zusammen: links der blauen gestrichelten Linie, oben der orangen gestrichelten Linie, rechts der blauen gestrichelten Linie und unterhalb der orangen gestrichelten Linie. Diese Einträge sind binär kodiert, sodass sich die geo. Features der Boxen in Abbildung 9 zu den folgenden ergeben:

- Box 2, Box 1: links=1, oben=0, rechts=0, unten=1,
- Box 2, Box 3: links=0, oben=1, rechts=1, unten=0.

In Tabelle 6 findet ist eine Übersicht über zuvor beschriebenen Two-Box-Features. Zusätzlich aufgeführt sind die Anzahl der Einträgen pro Feature.

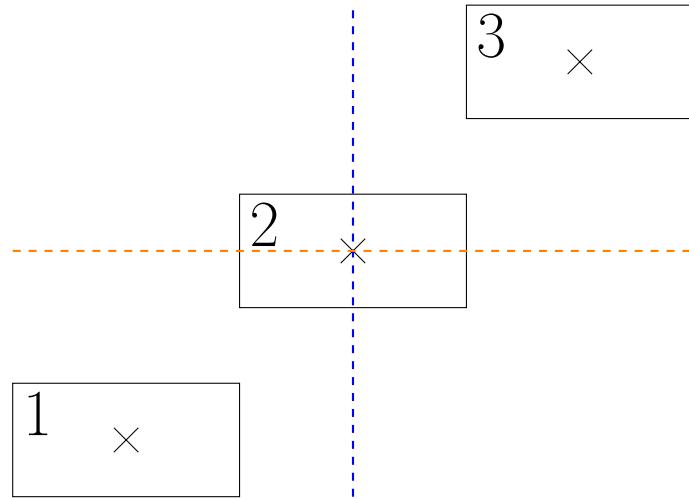


Abbildung 9: Zeigt exemplarisch an Box 2 das Finden der geographischen Features. Das Zentrum jeder Box ist durch ein  $\times$  markiert. Durch eine blaue gestrichelte Linie wird eine vertikale Entscheidungsgrenze dargestellt. Eine horizontale Entscheidungsgrenze ist durch die orange gestrichelte Linie abgebildet.

Feature	Feature-Größe	Bemerkung
Distanzen	4	euklid. und Hamming-Distanz
Geographische Features	4	Lage um eine Box herum

Tabelle 6: Übersicht über die Two-Box-Features. Die Spalte **Feature-Größe** zeigt die Anzahl an Einträgen, die pro Feature hinzukommen. In der Spalte **Bemerkung** sind zusätzliche Informationen für die Features aufgeführt.

### 3.2.3 Sampling

Aufgrund der starken Imbalance der Klassen „korrekte Absatzabfolge“ ( $P = 1$ , vgl. Gleichung (1)) und „inkorrekte Absatzabfolge“ ( $P = 0$ , vgl. Gleichung (1)) wird ein Sub-Sampling angewendet. Dabei wird die Imbalance auf ein Verhältnis von 1:2 (korrekt:inkorrekt) beschränkt. Das heißt, zu einem positiven Beispiel existieren zwei negative.

Die positiven Beispiele sind die Absatz-Paare, die direkt aufeinander folgen. Wohingegen die negativen Beispiele alle anderen Absatz-Paare darstellen. Als Beispiel dient das Dokument  $\mathcal{D}_1 = \{A_1, A_2, A_3\}$  mit der korrekten Dokumenten Lesenreihenfolge  $A_1 \rightarrow A_2 \rightarrow A_3$ . Es gibt lediglich zwei positive Beispiele:  $\{(A_1, A_2), (A_2, A_3)\}$ . Negative Beispiele sind entsprechend alle weiteren Paarungen, die keine positiven Beispiel sind:  $\{(A_1, A_3), (A_3, A_2), (A_3, A_1), (A_2, A_1)\}$ . Dies spiegelt sich auch in Tabelle 4b (aus Kapitel 3.2) wieder.

Alle positiven Samples werden deswegen für das Training verwendet. Aus dem oben genannten Beispiel sind dies die Absatz-Paare  $(A_1, A_2)$  bzw.  $A_1 \rightarrow A_2$  und  $(A_2, A_3)$  bzw.  $A_2 \rightarrow A_3$ . Dazu werden pro positives Beispiel zwei negative Beispiel zufällig aus der Menge aller negativen Beispiele gezogen.

### 3.2.4 Reading Order-Bestimmung

Es sind weitere Schritte notwendig, um eine Lesereihenfolge zu erhalten und diese zu evaluieren. Diese sollen nun erläutert werden.

#### Baseline

Als Baseline wird eine simple heuristische Methode eingesetzt. Dabei handelt es sich um die Sortierung von Bounding Boxes zuerst in y-Richtung und anschließend in x-Richtung aufsteigend. Dazu wird ein stabiler Sortieralgorithmus eingesetzt, sodass die Reihenfolge aus dem ersten Sortierschritt (y-Richtung) beim zweiten Sortierschritt (x-Richtung) erhalten bleibt.

#### Modell-basiert

Wie in Kapitel 3.2.1 definiert, erhält das verwendete Modell zwei Bounding Boxes, die zwei Absätze repräsentieren und mit verschiedenen weiteren Informationen sowie Kontext angereichert werden. Daraufhin bestimmt das Modell mit welcher Wahrscheinlichkeit die beiden Absätze direkt aufeinander folgen. Aus diesen Werten wird anschließend eine Lesereihenfolge konstruiert, sodass eine Abfolge aller Absätze bzw. Bounding Boxes entsteht. Es werden alle Absatz-Paare eines Dokuments in das Modell gegeben und die erhaltene Ausgabe in eine Matrix eingetragen. Die resultierende Matrix entspricht im Idealfall den Beispielen der Tabelle 4. Auf diese Matrizen werden zwei Algorithmen angewendet, welche eine Lesereihenfolge basierend auf den Indizes einer Matrix erzeugen.

Einer dieser Algorithmen heißt „Greedy“. Dieser geht von einer bestimmten frei gewählten Box als Startpunkt aus. Dann sucht der Algorithmus in einer Zeile den höchsten gefundenen Spalten-Wert als Nachfolger. Im Anschluss wird der höchste Spalten-Wert in der Nachfolger-Zeile bestimmt. Falls ein Nachfolger bereits besucht wurde, wird der jeweils nächst höchste Spalten-Wert betrachtet und der Nachfolger bestimmt. Der Algorithmus lässt bei der resultierenden Lesereihenfolge keine doppelten Absätze zu. So ergibt sich eine Lesereihenfolge von Absätzen über die Indizierung der Matrix. Abbildung 10 zeigt den Ablauf anhand eines Dokuments mit drei Absätzen. Dabei ist die Start-Box  $A_1$  fest gewählt.

	$A_1$	$A_2$	$A_3$	betrachtet Box	Lesereihenfolge
$A_1$	0	0.5	0.4	$A_1$	[ $A_1$ ]
$A_2$	0.3	0	0.5	$A_2$	[ $A_1, A_2$ ]
$A_3$	0.1	0.2	0	$A_3$	[ $A_1, A_2, A_3$ ]

$\Rightarrow$  alle Boxen gefunden

Abbildung 10: Zeigt den Ablauf des Greedy-Algorithmus anhand eines Beispiels. Links ist eine Matrix mit Wahrscheinlichkeitswerten zu sehen. Diese geben an, mit welcher Wahrscheinlichkeit bspw. Box  $A_1$  auf Box  $A_2$  bzw.  $A_3$  direkt folgt. Rechts ist der Ablauf des Greedy-Algorithmus dargestellt. Dabei wird die betrachtet Box sowie die bisher gefundene Lesereihenfolge aufgelistet. Zu beachten ist, dass Box  $A_1$  als Start-Box gewählt ist.

Beim dem zweiten Algorithmus, der auf die Matrix angewendet wird, handelt es sich um den Beam Search-Algorithmus. Beam Search ist ein heuristischer Suchalgorithmus, welcher einen Graphen

exploriert und dabei den vielversprechendsten Kandidaten bevorzugt [1]. Klassischerweise findet Beam Search in Anwendungen von maschinellen Übersetzungs- oder Zusammenfassungsaufgaben [1] statt. Um Beam Search anzuwenden, werden Absätze als Knoten gesehen, die über Gewichte miteinander verbunden sind. Die Gewichte zwischen den einzelnen Boxen entsprechen den Wahrscheinlichkeiten wie sie von dem Modell erzeugt werden. Abbildung 11 stellt dar, wie von der Absatz-Absatz-Matrix ein Graph erzeugt wird. Dabei ist anzumerken, dass der Graph nie explizit erzeugt wird, sondern nur zur Erläuterung hier aufgeführt ist. Auch dieser Algorithmus lässt, wie bei Greedy keine doppelten Boxen zu. Die sogenannte „Beam Width“ gibt die Anzahl der vielversprechendsten Kandidaten an, die schrittweise expandiert bzw. pro Iteration behalten werden. Diese wird auf 10 festgelegt.

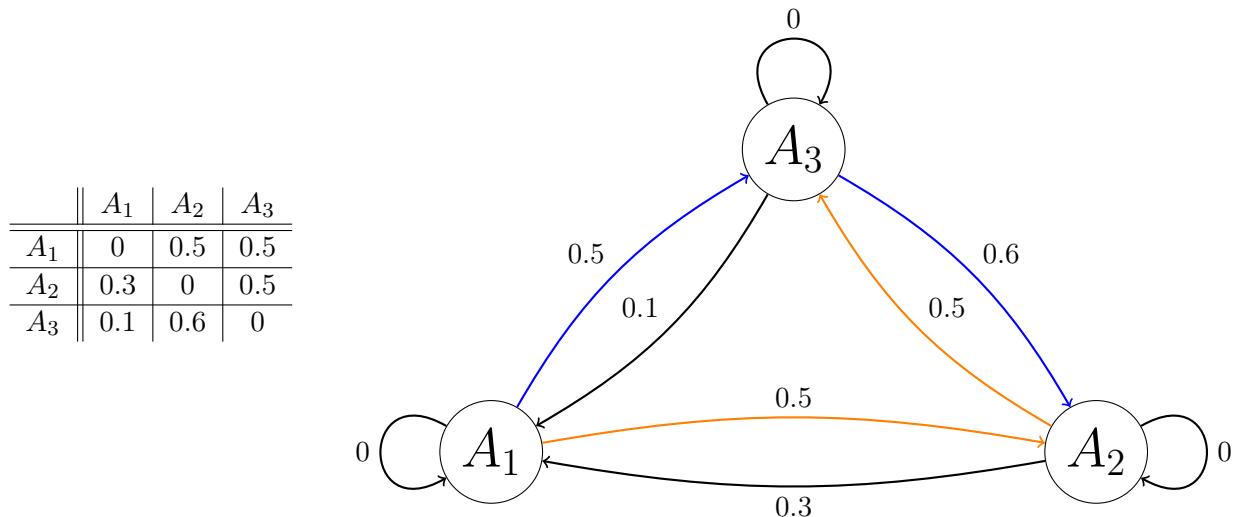


Abbildung 11: Zeigt die Anwendung des Beam Search-Algorithmus auf die Absatz-Absatz-Matrix (links). Auf der rechten Seite ist der Graph dargestellt, wie man ihn aus der Absatz-Absatz-Matrix abliest. In Orange ist der Weg eingezeichnet, den der Greedy-Algorithmus findet. In Blau sieht man den Weg, den der Beam Search-Algorithmus findet. Absatz  $A_1$  ist als Start-Absatz bzw. Start-Box fixiert.

### 3.2.5 Training

Um das Modell zu trainieren, werden die Layout-Dateien des ReadingBank-Datensatzes (vgl. Kapitel 2.2) zusammengefasst und zufällig gemischt.<sup>9</sup> Anschließend werden die gesamten Layout-Daten in Trainings- und Validierungsmenge aufgeteilt. Dies erfolgt in einem Verhältnis von 80/20 (Training/Validierung) in Prozent.

Es wird ein XGBoost-Modell<sup>10</sup> mit der in Kapitel 3.2.3 erläuterten Sampling-Methodik und den in Kapitel 3.2.2 eingeführten Features trainiert. Die Ground-Truth-Lesereihenfolge wird durch den ReadingBank-Datensatz geliefert. Das XGBoost-Modell wird mit den Default-Hyperparametern trainiert und evaluiert.

<sup>9</sup>Das Bash-Script hierfür befindet sich im Anhang A.1.

<sup>10</sup><https://github.com/dmlc/xgboost>, abgerufen: 17.03.2023

Zur Trainingszeit werden die Boxen aus dem Datensatz geladen und mit den entsprechenden Single-/Two-Box-Features angereichert. Zu beachten ist hierbei, dass die Single-Box-Features pro Box gelten und die Two-Box-Features pro Box-Paar. Single-Box-Features erweitern die jeweilige Box. Anschließend werden diese angereicherten Boxen konateniert und die Two-Box-Features angehängt. Tabelle 7 zeigt den beschriebenen Aufbau.

Box1	Single-Box-Features (Box 1)	Box 2	Single-Box-Features (Box 2)	Two-Box-Features
	angereicherte Box 1		angereicherte Box 2	Two-Box-Features
	4-15		4-15	0-8

Tabelle 7: Aufbau des Inputs für das XGBoost-Modell. Die erste Zeile zeigt den atomaren Aufbau; die zweite Zeile zeigt eine Vereinfachung von Zeile Eins. Die letzte Zeile zeigt die Einträge pro Feld.

### 3.2.6 Evaluation

Die Evaluation geschieht einerseits händisch und andererseits durch den sogenannten Bi-Lingual Evaluation Understudy-Algorithmus (BLEU-Algorithmus) [2]. Der BLEU-Score wiederum ist das Produkt des BLEU-Algorithmus und liegt zwischen Null und Eins.<sup>11</sup> Dies ist ein in der maschinellen Übersetzung eingesetzte Metrik. Die Metrik beruht auf dem Vergleich verschieden langer N-Gramme. Im Kontext dieser Arbeit werden dem BLEU-Algorithmus die originale Lesereihenfolge in Form von Indizes gegeben, sowie die Lesereihenfolge, die durch unseren Ansatz (vgl. Kapitel 3.2.4) produziert wird. Heraus kommt eine gemittelte Prozentzahl, die Aufschluss über die korrekt erkannte Lesereihenfolge gibt, aufgeteilt in N-Gramme bzw. N-Absätze. Der BLEU-Score wird dokumentenweise berechnet. Um eine Aussage über die gesamten Dokumente zu erhalten, wird der Mittelwert über alle Dokumenten-BLEU-Scores gebildet.

Es wird auf die Verwendung von Special-Tokens wie beispielsweise <START> oder <END> verzichtet. Damit ergibt sich jedoch das Problem mit welchem Absatz die Lesereihenfolge startet. Um dies zu umgehen, da diese Arbeit sich auf die Bestimmung der Lesereihenfolge fokussiert, wird die Start-Box durch die korrekt Start-Box vorgegeben.

---

<sup>11</sup>Es wird die NLTK-Implementation verwendet: [https://www.nltk.org/\\_modules/nltk/translate/bleu\\_score.html](https://www.nltk.org/_modules/nltk/translate/bleu_score.html), abgerufen: 19.03.2023.

## 4 Experimente und Ergebnisse

### 4.1 Box-Class Prediction Experimente

Im ersten Experiment wurden die beiden Classifier Logistic Regression und XGBoost miteinander verglichen. Beide wurden wie in Kapitel „Classifier“ beschrieben, einmal nur mit den Bounding Boxen und einmal mit den Bounding Boxen und den Font-Informationen trainiert. Das Ziel war es schnell gute Features zu finden, um ein Modell zu trainieren, welches ebenfalls schnell zu guten Ergebnissen kommt. Deswegen wurden beide Modelle so wie sie aus der Sklearn Bibliothek kommen getestet .

#### 4.1.1 LR vs XGB

In dem ersten Experiment wurden die beiden Modelle ausschließlich mit den Bounding Boxen trainiert. Sie bekommen einen Eingabevektor, der aus den zwei Aufspannvektoren der Bounding Box besteht  $[x_0, y_0, x_1, y_1]$  und ein Label, welches den Index  $[0 \dots 10]$ , der richtigen Klasse repräsentiert. Beide Modelle errechnen eine Wahrscheinlichkeitsverteilung über die möglichen 11 Klassen und geben den Wert aus, der die höchste Wahrscheinlichkeit besitzt. Der Loss ist der Abstand zwischen [1] (100%) und dem Prediction Value [0 bis 1]. Beide Modelle wurden nach dem Training mit dem Validationset des DocLayNet evaluiert.

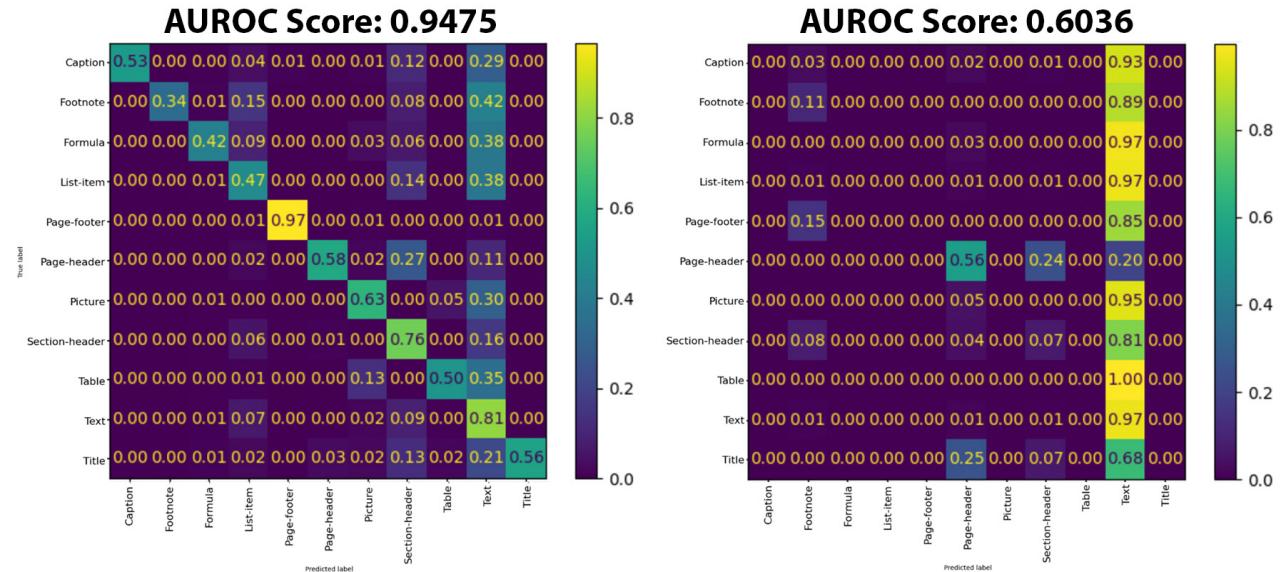


Abbildung 12: Beide Confusion Matrizen zeigen die Trefferquote der einzelnen Klassen und welche Klassen einer hohen Verwechslungsgefahr ausgesetzt sind. Verglichen werden alle 11 Klassen miteinander. Die linke Matrix ist das „hits\_at\_1“ Ergebnis des XGB Modells und zeigt erste gute Werte. Die rechte Matrix ist das Ergebnis des LR Modells und zeigt eine starke Verwechslungsgefahr mit der Klasse „Text“.

<sup>11</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html), aufgerufen: 22.03.23

<sup>11</sup><https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>, aufgerufen: 22.03.23

Heraus kommen die Trefferquoten der Modelle über die Klassen auf dem Validationset. Daraus lässt sich eine Confusion Matrix definieren, die zeigt wie hoch die Verwechslungen einer Klasse zu den anderen ist. Es zeichnet sich eine klare Tendenz Richtung dem XGBoost Modell ab, welches die Klassen gut auseinander halten kann und durchweg die ersten guten Ergebnisse erzeugt. Das LR Modell hingegen trifft kaum eine Klassen und schafft es nur die Klasse „Page-Header“ und „Text“ zuverlässig zu treffen. Den Rest der Klassen verwechselt es mit der Klasse „Text“ und erliegt damit dem Bias aus dem DocLayNet Datensatz. Das zeigt sich auch in dem AUROC Score: Das LR Modell zeigt mit einem Score von 0.6036, dass es eigentlich nur am raten ist, während das XGB Modell mit dem Feature gut arbeitet und sich sicher ist (0.9475).

#### 4.1.2 LR+Font vs XGB+Font

Um die Verwechslungsgefahr mit der Klasse „Text“ einzugrenzen und die Unterschiede besser herauszuarbeiten, wurde im nächsten Experiment, die Font-Informationen als Feature eingeführt. Dadurch wächst der Featurevektor von 4 Einträgen auf 39. Die Font-Informationen bestehen aus Font-Style, -Color und -Size. Wie der Featurevektor aussieht lässt sich in Kapitel 3.1 Abbildung 3 nachlesen. Die Modelle LR+Font und XGB+Font bekommen den Featurevektor mit den Font-Informationen, das selbe Label wie zuvor und werden ebenfalls auf dem Validationset evaluiert.

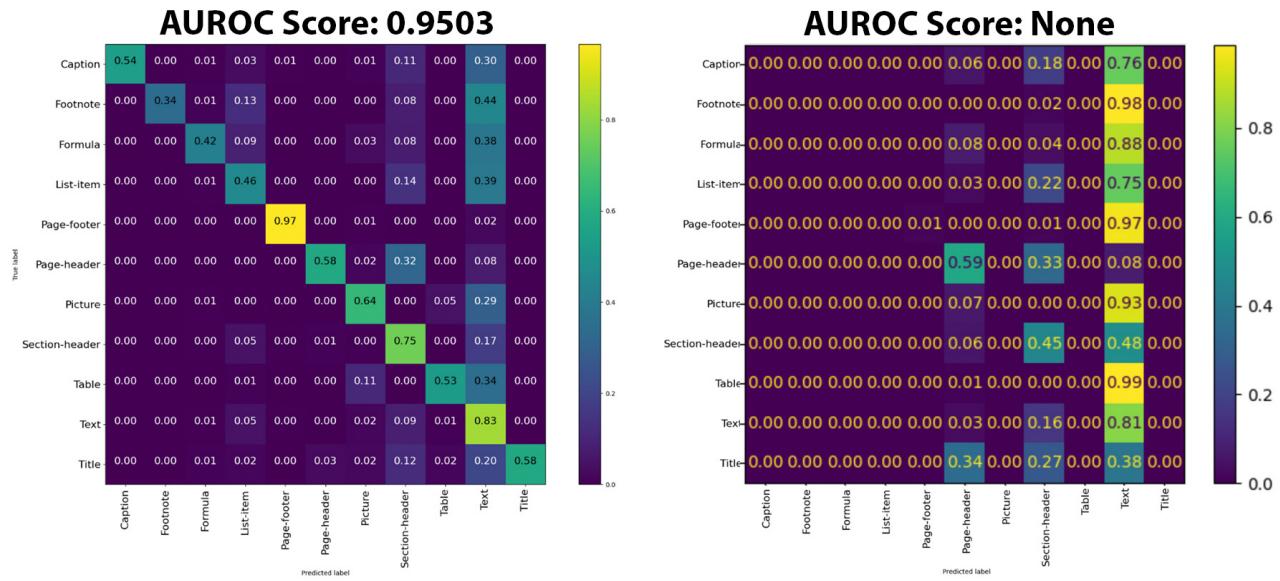


Abbildung 13: Wie in der Abbildung zuvor ist auf der linken Seite das XGB+Font Modell und auf der rechten Seite das LR+Font Modell. Der AUROC Score konnte für das LR Modell nicht berechnet werden und ist daher leer.

Auch hier zeigt die Confusion Matrix des LR+Font Modells keine klare Verbesserung zu dem LR Modell ohne die neuen Features und verwechselt wie zuvor die meisten Klassen mit „Text“. Ausschließlich die Klasse „Section-Header“ wurde weniger verwechselt und ist von 0.07 auf 0.45 gestiegen. Die Klasse „Page-Header“ ist ebenfalls weiter angestiegen von 0.56 auf 0.59, wurde aber auch zuvor nicht so oft mit der Klasse „Text“ verwechselt wie andere. Das XGB+Font Modell hingegen hat sich nur

minimal verändert. Insgesamt erfuhrn die Trefferquoten einen Anstieg von 0.0942 ( 9%), im Durchschnitt 0.0086 ( 1%). Hauptsächlich die Klasse „Title“ hat sich durch die neuen Features verbessert. Der AUROC Score hat sich ebenfalls nur minimal verbessert und lässt die Vermutung nahe, dass das Modell minimal sicherer geworden ist (0.9475 → 0.9503).

Durch beide Experimente wurde eindeutig klar, dass das XGB Modell besser mit den Daten und den Features umgeht, als das LR Modell. In der Folge wurde nur noch das XGB Modell verwendet und ein weiteres Mal getestet.

#### 4.1.3 Finales Modell: XGB+Size

Eine Layout Klassifikation nur anhand der Bounding Boxen zu vollziehen führte zu keinen guten Ergebnissen und sorgt für viele Verwechslungen zwischen den Klassen. Auch nachdem die Font-Informationen hinzugefügt wurden, haben sich die Confusion Matrizen über dem Validationset nur bedingt verbessert. Das ist für viele textbasierte Klassen weniger schlimm zu betrachten, aber die Verwechslung zwischen „Picture“ und „Text“ bei 31% der Fällen, bleibt ein kritischer Punkt, der behoben werden musste. Wie im DocLayout Datensatz Kapitel bereits erwähnt, gibt es viele Klassen wie „Picture“, die Font-Informationen besitzen, die für die Unterscheidung weniger förderlich sind. Daher wurde der Fokus nach dem letzten Experiment in diesem Feature Bereich beschränkt und auf die Font-Size gekürzt. Das hatte einen merklichen Anstieg zur Folge. Die Font-Size ist ein Feature aus dem Datensatz und musste nicht weiter errechnet werden. Dadurch wächst der Featurevektor von 4 (XGB) auf 5 (XGB+Size), bzw sinkt von 39 (XGB+Font) auf 5 (XGB+Size) Einträgen.

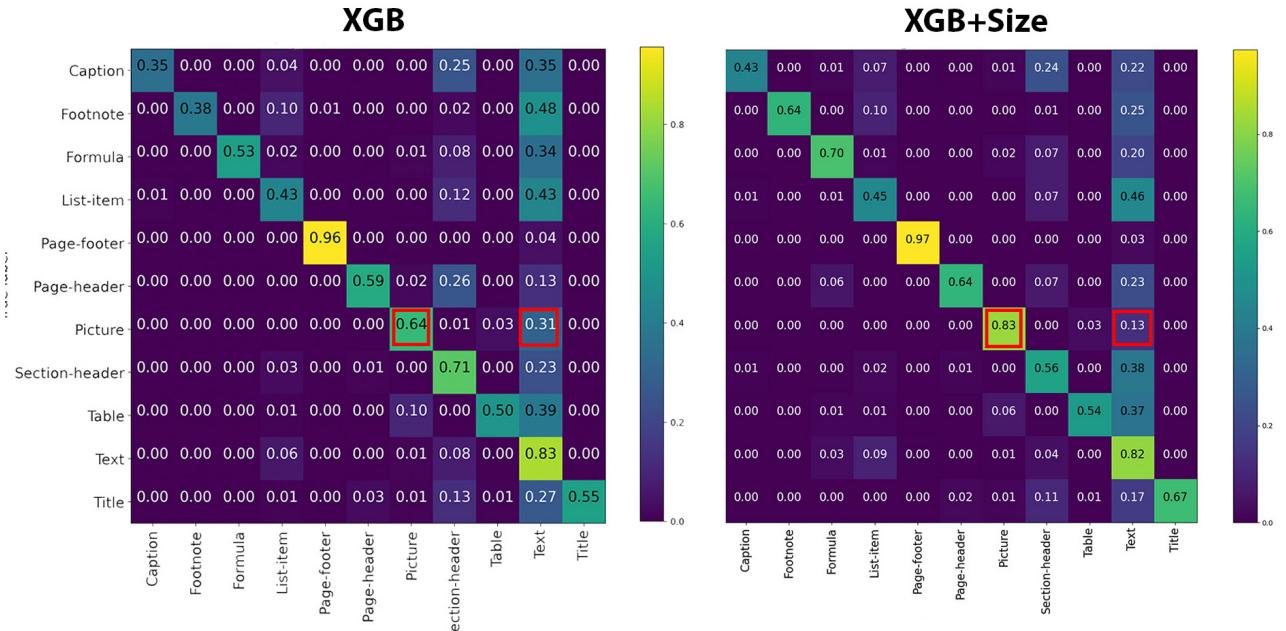


Abbildung 14: Zu sehen ist das Modell XGB und XGB+Size. Beide Matrizen zeigen eine gute Trefferquote über die Klassen. Das XGB+Size Modell schafft es die Verwechslung zwischen „Picture“ und „Text“ zu senken (rot markiert).

Klasse	H1(%)	H1(%)	Diff
Caption	34.54	42.99	8.45
Footnote	38.14	63.78	25.64
Formula	53.48	69.53	16.05
List-item	43.28	45.39	2.11
Page-footer	95.62	97.32	1.7
Page-header	58.67	63.98	5.31
Picture	64.21	82.63	18.42
Section-header	71.28	56.26	-15.02
Table	49.97	53.50	3.53
Text	83.30	81.65	-1.65
Title	54.51	67.22	12.71
Durchschnitt	58.82%	65.84%	7,02%

Klasse	H3(%)	H3(%)	Diff
Caption	69.65	76.12	6.47
Footnote	69.87	82.37	12.5
Formula	69.37	76.50	7.13
List-item	96.11	98.51	2.4
Page-footer	98.43	97.93	-0.5
Page-header	83.52	76.12	-7.4
Picture	92.03	94.95	2.92
Section-header	97.39	96.49	-0.9
Table	90.12	90.17	0.05
Text	99.96	99.94	-0.02
Title	71.23	77.25	6.02
Durchschnitt	85.24%	87.85%	2.61%

Abbildung 15: Zu sehen sind die Trefferquoten der Modelle für die „Hits\_at“-Werte über alle Klassen. Zusammen mit den Durchschnitts-Werten wird genauer auf die Zahlen eingegangen als bei der Abbildung 14.

Die Confusion Matrizen (Abbildung 14) und die Tabelle des H1 Scores (Tabelle 15) zeigen einen klaren Anstieg von 7.02% im Durchschnitt über alle Klassen verteilt. Ausschließlich die Klassen „Section-Header“ (-15.02%) und „Text“ (-1.65%) haben sich verschlechtert. Alle anderen Klassen haben sich teilweise bis zu 25% (Footnote) verbessert. Auch der H3 Score hat sich im Durchschnitt um 2.06% verbessert. Beide Scores zeigen eine klare Steigerung des Modells. Auch die Verwechslung zwischen „Picture“ und „Text“ ist von 31% auf 13% gesunken. Damit hat die Beschränkung des Featurevektors auf die Font-Size als einzige Font-Information sein Ziel erfüllt.

Klasse	Training (%)	Validation (%)	H1 (%)	H3 (%)
Caption	2.04	1.79	1.13	14.28
Footnote	0.59	0.31	0.22	1.63
Formula	2.25	1.89	3.54	17.79
List-Item	17.19	13.34	11.27	76.52
Page-Footer	6.51	5.58	5.68	6.50
Page-Header	5.10	6.69	4.60	7.79
Picture	4.21	2.78	2.86	8.65
Section-Header	12.60	15.77	12.89	50.81
Table	3.19	2.27	1.42	16.04
Text	45.82	49.28	56.15	99.30
Title	0.47	0.29	0.23	0.69

Tabelle 8: Zeigt die Verteilung der Samples über alle Klassen. Die Verteilung über dem Trainings- und Validationset kommt aus dem DocLayout. Die Verteilung über H1 und H3 sind die Anzahl der Predictions von dem Modell auf dem Validationset. Es ist ein klarer Bias in Richtung der Klasse „Text“ zu erkennen. Trainings- und Validationset sind sich sehr ähnlich. Auch die Verteilung der Prediction des XGB+Size Modells ist dem sehr ähnlich.

Dennoch ist die Verteilung der Predictions des Modells weiterhin sehr Text fokussiert. Sind sich die

Verteilungen der Klassen von Validation-/Trainingsset zu der Verteilung der Predictions der Klassen an erster Stelle (H1 Scores) noch ähnlich, so lässt sich doch in Betrachtung der ersten drei Plätze (H3 Score) ein großes Ungleichgewicht Richtung der Klasse Text erkennen. Denn das Modell hat in seiner Top 3 zu 99.3% die Klasse „Text“ predicted (Tabelle ). Das ist ein Bias, der in zukünftigen Forschungen behoben werden sollte.

Alles im allem bekommt das XGB+Size Modell auf den Testdaten einen guten Score, vor allem wenn die H3 Scores betrachtet werden. Das gilt aber nur für den Fall, wenn die Daten Ähnlichkeiten zu den gelernten haben. Die Features sind sehr einfach gehalten und sind die Voraussetzung für ein gutes Abschneiden des Modells. Die Bounding Boxen zu errechnen ist aber nicht einfach, falls sie nicht gegeben sind. Um die Transferleistung des XGB+Size Modells zu gewährleisten wurde ein weiteres Experiment vollzogen.

#### 4.1.4 Transferierung des Classifiers auf den Reading Bank Datensatz

Das Modell zum Predicten der Reading Order wird über den Reading Bank Datensatz trainiert und evaluiert, da sie eine Ground Truth beinhaltet. Der Classifier dient als zusätzliches Feature und muss deswegen auch auf diesem Datensatz eine vernünftige Klassifizierung hervorbringen. Da der Datensatz weder Bounding Boxen um einzelne Layout Elemente noch Font-Informationen besitzt, musste beides vorab berechnet werden (siehe Kapitel Reading Bank), was zu einigen Problemen führte.

Zum einen bildet der Datensatz nicht alle Klassen ab, die das Modell gelernt hat. Klassen wie „Picture“ oder „Table“ fehlen völlig. Der Reading Bank Datensatz ist dafür vorgesehen eine Reading Order anhand des Textes zu bestimmen und weniger anhand der Layout Informationen. Dadurch erhält der Datensatz z. B. nur den Inhalt der Tabellen, aber keine Box um die Tabellen selbst. Das liegt an dem zweiten Punkt: die Bounding Boxen. Auch dieses Features besitzt der Datensatz nicht und musste im Preprocessing Schritt algorithmisch gelöst werden. Dafür wurden die vorhandenen Boxen zunächst zu Zeilen zusammengesetzt, um danach mit einem agglomerativen Clustering Boxen zu formen (siehe Kapitel 2.2.1). Damit die daraus entstandenen Boxen nur ein einzelnes Layout Element abbilden, wurde der Ansatz passiv gewählt, wodurch ein Layout Element eher durch mehrere Boxen beschrieben wird als eine große, wie es bei dem DocLayNet der Fall gewesen wäre. Dadurch wirken viele Zeilen für den Classifier wie ein List-Item. Auch die einzelnen Tabellenzeilen sind unterschiedliche textbasierte Klassen für das Modell und keine Tabelle. Abbildung 16 zeigt zwei Beispiele aus dem Reading Bank Example Folder. Auf der linken Seite werden nur die Tabellenzeilen erkannt, welches zum Problem führt, da der Classifier diese Klasse nicht gelernt hat. Auf der rechten Seite werden die Zeilen nicht zu einer Textbox zusammengefasst, obwohl es an dieser Stelle angebracht wäre. Daher fällt es dem Classifier schwer eine einheitliche Aussage zu treffen, da er eher Textboxen gelernt hat, als Zeilen.

Da der Datensatz aus vielen dieser problematischen Beispiele besteht lässt sich der Classifier nicht händisch evaluieren. Die Ausgaben und Beispiele aus dem Reading Bank Datensatz lassen sich nicht mit dem DocLayNet Datensatz vergleichen, weshalb das Modell viele falsche Vorhersagen trifft. Das ist ein Grund warum der Classifier im finalen Reading Order Modell keine Relevanz mehr spielt.

**0. Prediction:** 1. 58.3 Section-header 2. 28.91 Text 3. 6.19 Caption  
 1. Prediction: 1. 57.74 Section-header 2. 22.75 Text 3. 16.08 List-item  
 2. Prediction: 1. 40.15 Section-header 2. 32.73 Caption 3. 21.08 Text  
 3. Prediction: 1. 60.69 Section-header 2. 20.57 Text 3. 15.68 Caption  
 4. Prediction: 1. 64.29 Caption 2. 21.38 Text 3. 13.4 Section-header  
 5. Prediction: 1. 68.14 Page-footer 2. 20.13 Caption 3. 8.33 Text  
 6. Prediction: 1. 41.93 Text 2. 30.87 Section-header 3. 24.43 Caption  
 7. Prediction: 1. 45.26 Text 2. 42.57 Caption 3. 11.2 Section-header  
 ...

**0. Box:** - 1. 44.14 Section-header 2. 42.95 Text 3. 6.37 Caption  
 1. Box: - 1. 43.02 Text 2. 23.63 List-item 3. 21.54 Section-header  
 2. Box: - 1. 42.8 Footnote 2. 29.51 Text 3. 14.29 List-item  
 3. Box: - 1. 40.7 Text 2. 31.31 Table 3. 13.4 Picture  
 4. Box: - 1. 50.35 Text 2. 20.71 List-item 3. 15.27 Section-header  
 5. Box: - 1. 54.35 Footnote 2. 27.27 Text 3. 11.08 List-item  
 6. Box: - 1. 79.66 Footnote 2. 11.67 Text 3. 4.95 List-item  
 7. Box: - 1. 55.01 Table 2. 26.54 Picture 3. 16.51 Text

Abbildung 16: Zeigt zwei Beispiele aus dem Reading Bank Example Folder und den Output, den der Classifier mit den grünen Boxen generiert.

## 4.2 Reading Order-Modell

Für die Experimente werden zwei Dokumentenmengen verwendet. Die kleinere Menge wird dazu verwendet neue Features zu erproben. Über die große Dokumentenmenge sollen die erprobten Features verifiziert werden. Die kleinere Menge besteht aus 10.000 Trainingsdokumenten und 1.000 Validierungsdokumenten, wohingegen die größere Menge aus 50.000 und respektive 5.000 Dokumenten besteht. Es wird sich auf die insgesamt 55.000 Dokumente als obere Grenze beschränkt, da diese zusammen ca. 5GB Arbeitsspeicher benötigen.

Insgesamt werden vier Experimente mit verschiedenen Feature-Kombinationen durchgeführt. In Tabelle 9 sieht man diese Feature-Kombinationen zusammen mit den Accuracy-Werten. Die Werte werden dabei über die Validierungsmenge erzeugt. Zusätzlich ist die Anzahl an insgesamt verwendeten Features pro Experiment in dieser Tabelle zu sehen. Die Accuracy-Werte für den kleinen (1.000) und großen (5.000) Dokumentensatz sind getrennt eingetragen. Die Ergebnisse, die in Tabelle 9 gezeigt werden, beziehen sich auf die Evaluation des binären Classifiers. Das heißt, die angezeigte Accuracy gibt an, wie viele Box-Paare korrekt als aufeinander folgend oder nicht aufeinander folgend erkannt werden. Auffällig ist, dass sich das Klassifikationsergebnis verschlechtert, sobald die Box-Klassen als Feature hinzugenommen werden (Zeile 1 und 2 in Tabelle 9). Ähnlich verhält es sich mit der Hinzunahme der Fläche und des Seitenverhältnisses einer Box als Feature (Zeile 1 und 3 in Tabelle 9). Besonders der Unterschied der Anzahl an verwendeten Features ist hierbei herauszustellen. Das beste Ergebnis auf dem kleinen und großen Dokumentensatz liefert die volle Feature-Konfiguration ohne die Box-Klassen (Tabelle 9 Zeile 4).

Im Folgenden werden ausschließlich die Modelle verwendet, die auf dem großen Dokumentensatz trainiert wurden. In Tabelle 10 enthalten sind die BLEU-Scores der einzelnen Feature-Kombinationen.<sup>12</sup>

<sup>12</sup>Die BLEU-Scores zum kleinen Dokumentensatz finden sich im Anhang A.2.

Features						#Feature	Accuracy	
Box	Box-Klassen	Fläche	SV	Distanzen	geo. Feature		klein	groß
x						8	91,15%	91,02%
x	x					30	90,96%	90,78%
x		x	x			12	90,77%	90,78%
x		x	x	x	x	20	<b>91,83%</b>	<b>91,77%</b>

Tabelle 9: Übersicht über die Evaluationsergebnisse des XGBoost-Classifiers. Die Features sind die selben wie in Kapitel 3.2.2. Die Spalte **#Feature** gibt die Anzahl der verwendeten Feature bzw. die Größe des Inputs in das Modell an. Die Accuracy wird durch das XGBoost-Python-Package bestimmt. Die Spalten „klein“ und „groß“ beziehen sich auf den kleinen (10.000/1.000) und den großen (50.000/5.000) Dokumentensatz. (SV: Seitenverhältnis)

In den Untertabellen (Tabellen 10a – 10d) aufgeschlüsselt sind die verwendeten Algorithmen zur Reading Order-Bestimmung (siehe Kapitel 3.2.4) und die BLEU-Scores einzelner N-Gramm-Längen. Dabei werden die gleichen 5.000 Dokumente verwendet, wie zur Erzeugung der Accuracy-Werte in Tabelle 9. In den Untertabellen ist zu erkennen, dass die Baseline in jedem Szenario das beste Ergebnis liefert. Weiterhin zeigt der Greedy-Ansatz verglichen mit dem Beam Search-Ansatz schlechtere BLEU-Scores. Eine weitere Auffälligkeit ist, dass sich die BLEU-Scores ähnlich zu den Ergebnissen aus Tabelle 9 verhalten. Beispielsweise sind die BLEU-Scores in dem Fall, dass nur die Boxen selbst als Feature verwendet werden (Tabelle 10a), höher als die BLEU-Scores, welche zusätzlich die Box-Klassen als Feature verwenden (Tabelle 10b).

Im Folgenden werden die durch die Feature-Kombinationen Box (Tabelle 9 Zeile 1), Box mit Box-Klassen (Tabelle 9 Zeile 2) sowie Box mit Fläche, Seitenverhältnis, Distanzen und geo. Features (Tabelle 9 Zeile 4) produzierten Lesereihenfolgen genauer betrachtet. Es werden nur einige wenige Beispiele gewählt. Die Abbildungen 17 – 20 stellen die erzeugte Lesereihenfolge (Schwarz) der Ansätze aus Kapitel 3.2.4 der korrekten Lesereihenfolge (Hellgrün) gegenüber. In den Spalten von links nach rechts sind Baseline, der Greedy-Ansatz und der Beam Search-Ansatz dargestellt. Zeilenweise sind auf verschiedenen Feature-Kombinationen trainierte Modelle gegenübergestellt.

Das in Abbildung 17 gezeigte Beispiel stellt die durchschnittliche Komplexität eines Dokuments aus dem ReadingBank-Datensatz dar. Es spiegeln sich hierbei die Accuracy-Werte (Tabelle 9) und BLEU-Scores (Tabelle 10) direkt in den gefundenen Lesereihenfolgen (Schwarz) wieder. So sieht man zum Beispiel, dass in Abbildung 17a (rechte Spalte) mehr aufeinander folgende Boxen korrekt gefunden werden als in Abbildung 17b (rechte Spalte). Außerdem zeigt die Abbildung 17, dass die Baseline (linke Spalte) im Mittel mehr Boxen in korrekter Reihenfolge abläuft als der Greedy-Ansatz (mittlere Spalte) und Beam Search-Ansatz (rechte Spalte).

Abbildung 18 zeigt ein Beispiel in dem der Beam Search-Ansatz eine bessere Lesereihenfolge als die Baseline oder der Greedy-Ansatz zeigt. Grundlegend wird das Dokument in einer Zeile von links nach rechts gelesen und es entstehen nur wenige Sprünge, die nicht zum korrekten Nachfolger führen. Wie schon in Tabelle 9 und 10 (bzw. 10b) erkennbar ist, wird durch die Hinzunahme der Box-Klassen weniger Box-Paare als direkt aufeinander folgend erkannt (Abbildung 18b).

Abbildung 19 ist ein Beispiel, in dem die Baseline die korrekte Lesereihenfolge des Dokuments er-

Ansatz	BLEU-Score			
	2-Gram	3-Gram	4-Gram	1- bis 4-Gram
Basline	<b>66,30%</b>	<b>55,90%</b>	<b>50,18%</b>	<b>60,65%</b>
Greedy	53,65%	39,16%	30,82%	45,82%
Beam Search	62,93%	47,76%	38,88%	54,29%

(a) Box

Ansatz	BLEU-Score			
	2-Gram	3-Gram	4-Gram	1- bis 4-Gram
Basline	<b>66,30%</b>	<b>55,90%</b>	<b>50,18%</b>	<b>60,65%</b>
Greedy	51,72%	36,78%	28,61%	43,44%
Beam Search	60,59%	44,93%	36,04%	51,60%

(b) Box, Box-Klassen

Ansatz	BLEU-Score			
	2-Gram	3-Gram	4-Gram	1- bis 4-Gram
Basline	<b>66,30%</b>	<b>55,90%</b>	<b>50,18%</b>	<b>60,65%</b>
Greedy	52,41%	37,69%	29,30%	44,38%
Beam Search	61,53%	46,09%	37,20%	52,73%

(c) Box, Fläche, Seitenverhältnis

Ansatz	BLEU-Score			
	2-Gram	3-Gram	4-Gram	1- bis 4-Gram
Basline	<b>66,30%</b>	<b>55,90%</b>	<b>50,18%</b>	<b>60,65%</b>
Greedy	58,74%	43,25%	34,30%	50,09%
Beam Search	66,12%	51,09%	42,03%	57,46%

(d) Box, Fläche, Seitenverhältnis, Distanzen, geo. Features

Tabelle 10: Übersicht der BLEU-Scores über den großen Dokumentensatz. In den Spalten zwei bis vier werden die 2- bis 4-Gramme einzeln dargestellt. Die letzte Spalte bildet eine BLEU-typische Mittelung der N-Gramm-Werte:  $\sqrt[4]{1\text{-Gram} \cdot 2\text{-Gram} \cdot 3\text{-Gram} \cdot 4\text{-Gram}}$ . Da jedoch jeder Algorithmus darauf abgestimmt ist, jede Box mindestens einmal auszugeben, ist der 1-Gram-Wert immer gleich Eins. Die dargestellten Ansätze entsprechen den in Kapitel 3.2.4 vorgestellten Algorithmen. In den Untertabellenunterschriften sind die verwendeten Feature-Kombinationen notiert.

kennt, jedoch die weiteren Ansätze nicht. Herauszustellen ist hierbei, dass der Greedy-Ansatz (mittlere Spalte) die obere Hälfte des Dokuments korrekte in Lesereihenfolge bringt, dagegen aber in der zweiten Hälfte Boxen überspringt. Diese Sprünge zeigen sich beim Beam Search-Ansatz bereits schon in der ersten Hälfte des Dokuments.

Hingegen ist in Abbildung 20 ein Beispiel dargestellt, bei dem ein Mensch bei händischen Evaluierungen Schwierigkeiten hat, die korrekte Lesereihenfolge der Absätze zu finden. Wie durch die hellgrünen Pfeile dargestellt ist, wird das Dokument weder in einer eindeutigen zeilenweisen oder spaltenweisen Richtung bevorzugt gelesen. Lediglich die Baseline (linke Spalte) zeigt einen nachvollziehbaren Weg durch das Dokument. Der Greedy- sowie Beam Search-Ansatz (mittlere und rechte Spalte) zeigen unerwartet Absatz-Sprünge.

Zuletzt werden die Ansätze aus Kapitel 3.2.4 auf den DocLayout-Datensatz (siehe Kapitel 2.1) an-

gewendet. Da auf dem DocLayNet-Datensatz keine Ground-Truth-Lesereihenfolge im Datensatz mitliefert wird, muss dies händisch durch das Betrachten des Dokuments vollzogen werden. In Abbildung 21 sind einige Beispiele dieser Evaluation dargestellt. Das Modell, welches zur Generierung der Lesereihenfolgen in dieser Abbildung verwendet wurde, setzt auf die Features: Box, Fläche, Seitenverhältnis, Distanzen und geo. Features. Es entspricht damit dem besten nach Tabelle 10 evaluierten Modell. Die Start-Box wird durch die Baseline bestimmt. In der ersten Zeile von Abbildung 21 zeigt sich, dass die Baseline (linke Spalte) für Dokumente, welche von oben nach unten zu lesen sind, die korrekte Lesereihenfolge erzeugt. Währenddessen finden für die restlichen Ansätze (mittlere und rechte Spalte) Sprünge von unten nach oben statt. In Zeile Zwei ist ein Dokument zu sehen, für das alle Ansätze keine korrekte Lesereihenfolge erzeugen. Die Baseline (linke Spalte) rekonstruiert nur streckenweise die korrekte Lesereihenfolge. Der Greedy- sowie Beam Search-Ansatz scheitern jedoch hierbei. Der DocLayNet-Datensatz besteht auch aus simplen Dokumenten, welche nur wenige Absätze bzw. Bounding Boxen besitzen. Ein solches Beispiel sieht man in der dritten Zeile von Abbildung 21. Hier können alle Ansätze die korrekte Lesereihenfolge erzeugen. Das Beispiel aus Zeile Vier soll nochmals verdeutlichen, dass das Beispiel aus der ersten Zeile die Regel darstellt. Allein der Baseline (linke Spalte) gelingt es die korrekte Lesereihenfolge zu bestimmen.

#### 4.2.1 Diskussion

Bereits die ersten Ergebnisse aus Tabelle 9 zeigen, dass die Hinzunahme der Box-Klassen als Feature zu einer Verschlechterung für die Vorhersage, ob zwei Absätze bzw. Bounding Boxes direkt aufeinander folgend sind oder nicht, führen. Besonders auffällig sind dabei die Anzahl an Einträgen für den Input des Classifiers. Durch die Box-Klassen werden insgesamt 22 weitere Feature-Einträge (11 Klassen aus dem DocLayNet-Datensatz pro Box), neben den Eckpunkten der Bounding Box selbst, für den Input verwendet. Dabei verbessern und verschlechtern sich die Accuracy des kleinen und großen Dokumentensatzes gleichermaßen. Verwendet man ausgewählte Features wie die Fläche, das Seitenverhältnis und die Kontext-Features (Distanzen und geo. Features), so erhält man eine Verbesserung der Accuracy auf dem kleinen und großen Dokumentensatz (Tabelle 9 Zeile Vier) gegenüber der Box als alleiniges Feature. Dabei werden 10 zusätzliche Features weniger verwendet, als bei der Verwendung der Box-Klassen.

Ein ähnliches Bild zeichnet sich durch die Tabelle 10 und der damit verbundenen Evaluation der Lesereihenfolge ab. Es zeigt sich, dass durch den Greedy- bzw. Beam Search-Ansatz ein BLEU-Score (für 1- bis 4-Gram) von 45,82% bzw. 54,29% erzielt wird mit lediglich der Bounding Box als einziges verwendetes Feature (Tabelle 10a). Durch die Hinzunahme der Box-Klassen verschlechtert sich wiederum der 1- bis 4-Gram BLEU-Score auf 43,44% respektive 51,6% (Tabelle 10b). Die Feature-Kombination aus Box, Fläche, Seitenverhältnis, Distanzen und den geographischen Features erzielt ein 1- bis 4-Gram BLEU-Score von 50,09% und 57,46% (Tabelle 10d). Damit handelt es sich um eine Verbesserung von ca. 5% und 3,5%. Hervorzuheben ist, dass der 2-Gram BLEU-Score für diese Feature-Kombination mit lediglich 0,18% am nächsten zu dem der Baseline liegt. Die BLEU-Scores der Baseline können jedoch von keinem der Ansätze mit keiner Feature-Kombination erreicht werden.

Insgesamt folgt aus den beiden Tabellen 9 und 10, dass die Einführung der Single-Box-Features zu einer Verschlechterung der Accuracy bzw. der BLEU-Scores führen. Setzt man jedoch die Boxen in einen Kontext, wie beispielsweise durch die Two-Box-Features, zeigt sich eine Verbesserung der Accuracy bzw. der BLEU-Scores. Somit folgt, dass der Kontextbezug einen Einfluss hat.

Die Abbildungen 17 – 20 zeigen die bisher aufgezeigten Ergebnisse an Beispielen. Die Unterabbildung (b) zeigt mehr und weitere Sprünge, als in den restlichen Unterabbildungen (a, c und d). Letztendlich zeigt sich bei der Anwendung des besten evaluierten Modells auf den DocLayNet-Datensatz, dass auch Dokumente, die für den Menschen einfacher erscheinen, nicht garantiert in eine korrekte Lesereihenfolge gebracht werden können. Insbesondere stellt sich heraus, dass die Baseline eine gute erste Schätzung bietet. Außerdem erzeugen der Greedy- bzw. Beam Search-Ansatz selbst auf einfach strukturierten Dokumenten abschnittsweise rückwärts lesende Lesereihenfolgen. Es ist zu Vermuten, dass die gewählten Features die einzelnen Box-Paare nicht stark genug unterscheiden können, sodass es zu der (streckenweisen) Umkehrung der Lesereihenfolge kommt.

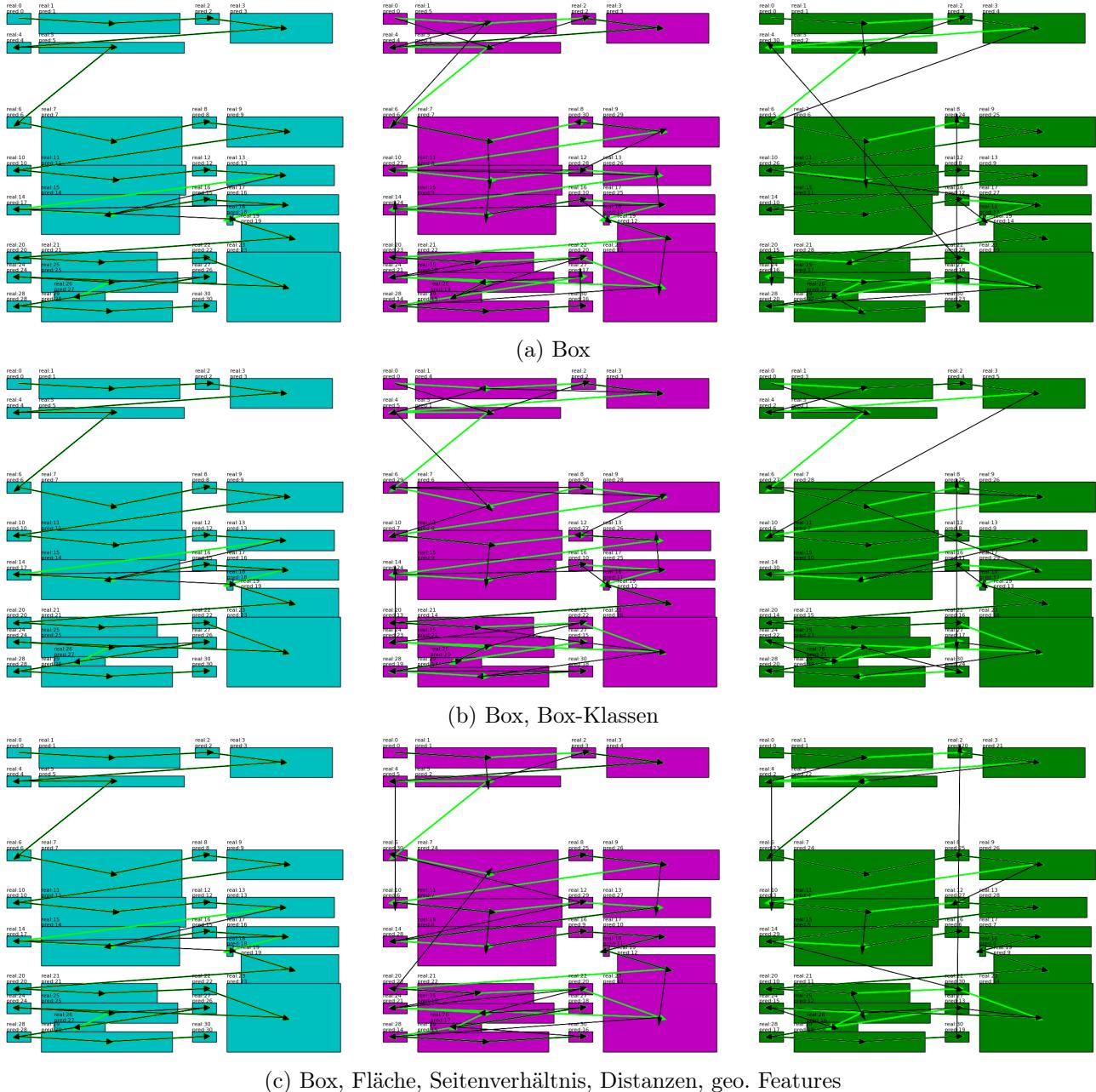


Abbildung 17: Gegenüberstellung der Reading Order-Bestimmung mit verschiedenen Feature-Kombinationen. Die Unterabbildungen sind mit der jeweiligen verwendeten Feature-Kombination unterschrieben. Von links nach rechts sind die Baseline-, Greedy- und Beam Search-Ansätze dargestellt. Die eingezeichneten hellgrünen Pfeile stellen die Ground-Truth-Lesereihenfolge. In schwarzen Pfeilen dargestellt ist die durch das Modell bzw. die Baseline bestimmte Lesereihenfolge zu sehen. An den oberen linken Ecke einer Box ist in „real“ und „pred“ dies nochmals in Zahlen ausgedrückt. Die Pfeilrichtung entspricht der Leserichtung.

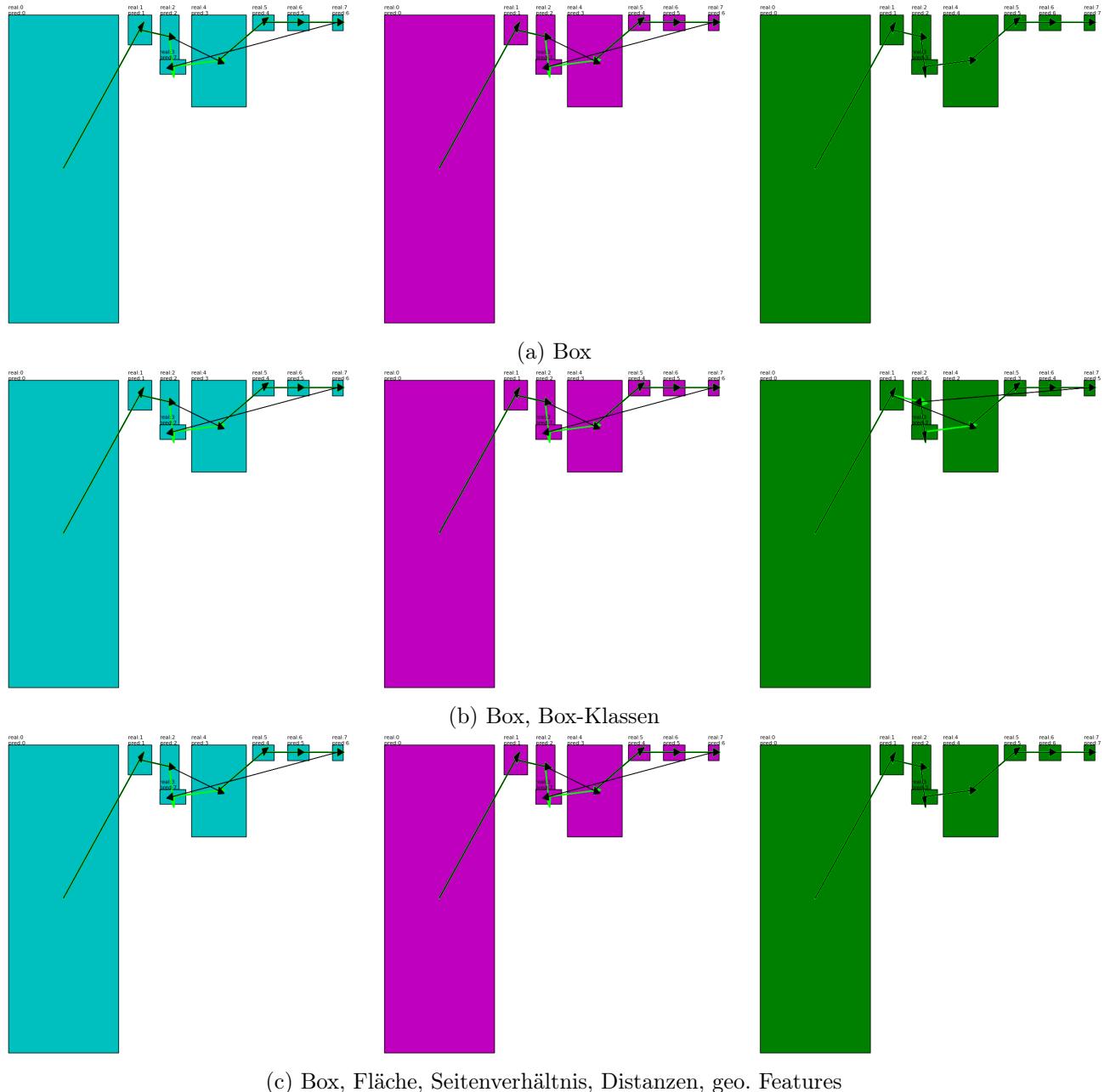


Abbildung 18: Für zusätzliche Erläuterungen siehe Abbildung 17.

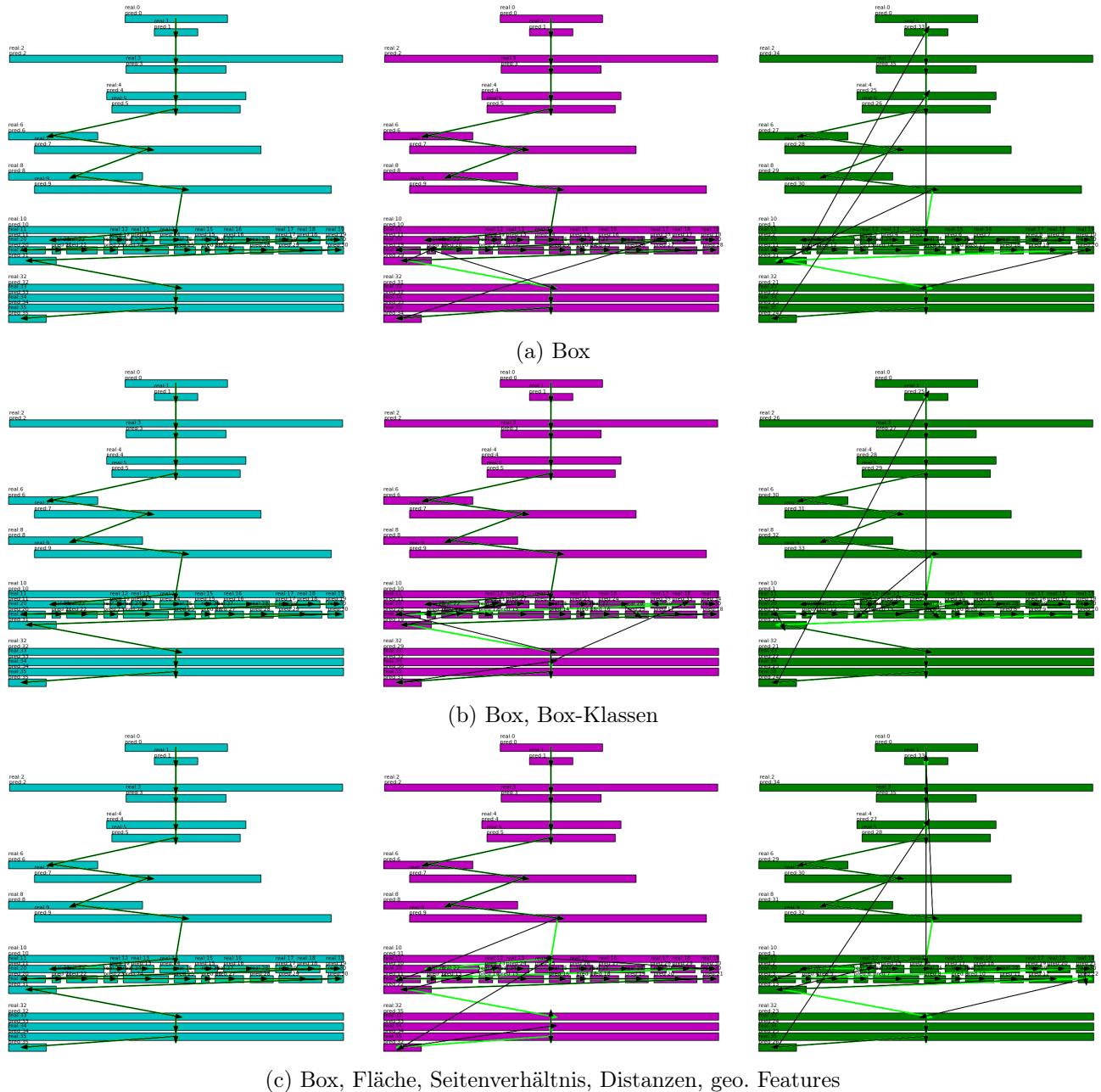


Abbildung 19: Für zusätzliche Erläuterungen siehe Abbildung 17.



Abbildung 20: Für zusätzliche Erläuterungen siehe Abbildung 17.

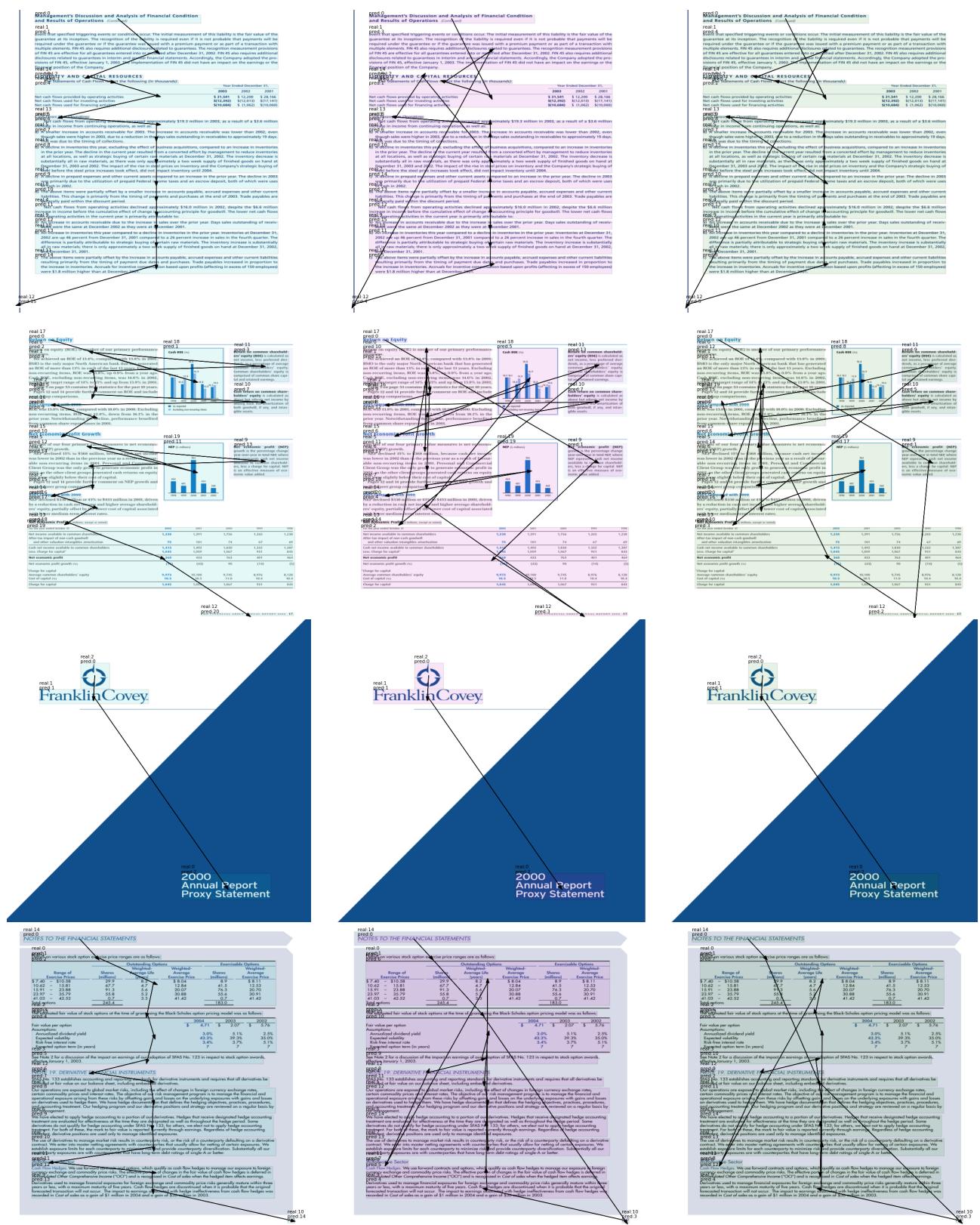


Abbildung 21: Anwendung der Ansätze aus Kapitel 3.2.4 auf den DocLayNet-Datensatz. Von links nach rechts sind Baseline, der Greedy-Ansatz und Beam Search-Ansatz zu sehen. In Schwarz eingezeichnet sind die Lesereihenfolgen, die durch die Ansätze erzeugt werden. Die Einträge „real“ und „pred“ haben hierbei keine weitere Bedeutung.

## 5 Fazit

In diesem Projekt wurde versucht eine Lesereihenfolge (Reading Order) auf Dokumenten, anhand von Layout-Informationen zu erzeugen, die als Wissensbasis für unterschiedliche Downstream Tasks dienen kann. Dabei lag der Fokus darauf, Features zu finden, die einen schnellen Erfolg der Modelle bringen. Dafür wurden unterschiedliche Features erstellt und miteinander verglichen. Die Evaluation lief dabei über einen Accuracy- und BLEU-Score. Die nach der Evaluation besten Feature Kombinationen, wurden mit einer einfachen Heuristik verglichen. Diese Heuristik diente gleichzeitig als Baseline für alle Experimente. Als Heuristik wurde die stabile Sortierung erst in y-Richtung (Höhe des Dokuments) und anschließend in x-Richtung (Breite des Dokuments) verwendet. Den größten Erfolg hatten Features, die zwei Boxen in einen Kontext rückten (vgl. Abbildung 17 – 21). Features die sich nur auf eine Box beziehen verschlechterten oft nur das Modell, wie bei dem Box-Klassen Feature. Im direkten Vergleich, waren alle trainierten Modelle schlechter als die Baseline. Außerdem ist die Anwendbarkeit auf dem DocLayoutNet-Datensatz nicht gegeben, wenn die Modelle auf dem ReadingBank-Datensatz trainiert wurden. Abstrakt zeigte sich jedoch, dass je besser der XGBoost-Classifier bereits auf der Validierungsmenge (Tabelle 9) agiert, der Downstream-Task der Reading Order-Bestimmung ebenfalls zu besseren Ergebnissen kam (Tabelle 10).

Die Wahrscheinlichkeitsverteilung über die 11 Klassen des DocLayoutNets, wurde zu einem zentralen Teil der Forschung in diesem Projekt, da diese Informationen im Bereich der Layout Analyse weit verbreitet sind und eine andere Information darstellt, als typische Layout Features. Die Idee war es den Gedanken „Nach einer Überschrift kommt der „Text“ in ein Feature umzuwandeln. Der Box-Class Predictor, der das Feature erzeugt, hat jedoch große Schwierigkeiten auf anderen Datensätzen zu transferieren. Außerdem ergaben die ersten Tests auf dem Reading Order Modell, mit einer ersten Version des Box-Class Predictors (XGB), eine Verschlechterung der Ergebnisse, weshalb der Box-Class Predictor ab diesem Punkt aus der Reading Order gestrichen wurde.

Die Kombination aus den beiden ausgewählten Datensätzen erwies sich als größtes Problem, wobei keine Alternativen zur Verfügung standen. Sie waren zu unterschiedlich und ließen keinen guten Transfer der Modelle zu. Die Bounding Boxen, um die Layout-Elemente herum und Dokumenttypen waren sich zu unterschiedlich und sorgten insgesamt für unterschiedliche Features. Die Verwechslung zwischen den Abbildungen („Table“ & „Picture“) und „Text“ konnte zwar gelöst werden, spielte aber im weiteren Verlauf für die Reading Order keine Rolle mehr, da der ReadingBank-Datensatz keine Bilder und keine Boxen um Tabellen herum enthält.

### 5.1 Aussicht

In zukünftigen Arbeiten könnten geeignete Datensätze gesucht werden. Außerdem könnten neue Features für den Box-Class Predictor ausprobiert werden, die mehr auf die Unterscheidung zwischen text-basierten Klassen abzielen. Die Idee eines Box-Class Predictor könnte verworfen und stattdessen Neuronale Netze in Form von MLPs eingesetzt werden. Viele der Features sind simpel gehalten und könnten gut mit einem MLP harmonieren.

Ein weiteres Forschungsgebiet wäre der Einsatz von CNNs, um ebenfalls auch optische Aspekte

der Dokumente und nicht nur Layout-bedingte Features zu berücksichtigen. Die Bounding Boxen der Elemente auf den Dokumenten könnten als eine Art Filter dienen, der die Interessanten Regionen zeigt. Dazu würde auch das ROI-Pooling passen. Die Bounding Boxen wären hier die “Region of Interest” und werden bis auf einen minimalen Featurevektor gepooled.

Mit diesem Ansatz wären die Layout-Informationen über die Bounding Boxen immer noch verfügbar, wodurch sich auch eine “Late Fusion” anbieten könnte. Dabei würde ein Modell die Layout-Features zusammen mit dem CNN erzeugten Bild Features lernen.

Letztendlich gäbe es noch die Möglichkeit typische Object Detection Modelle wie YOLOv5 auf die neue Aufgabe zu fine-tunen.

## Literatur

- [1] HuggingFace. How to generate text: using different decoding methods for language generation with Transformers. <https://huggingface.co/blog/how-to-generate>, 2023. [abgerufen 15.03.2023].
- [2] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.
- [3] Birgit Pfitzmann, Christoph Auer, Michele Dolfi, Ahmed S. Nassar, and Peter W. J. Staar. DocLayoutNet: A Large Human-Annotated Dataset for Document-Layout Analysis. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3743–3751, August 2022. arXiv:2206.01062 [cs].
- [4] Zilong Wang, Yiheng Xu, Lei Cui, Jingbo Shang, and Furu Wei. LayoutReader: Pre-training of Text and Layout for Reading Order Detection, August 2021. arXiv:2108.11591 [cs].

## A Anhang

### A.1 Bash-Script zum Sammeln und Mischen der Trainings-/Validierungsdaten

An dieser Stelle sei noch angemerkt, dass das folgende Bash-Script zu einer effektiven **Vervierfachung** Datengröße führt. Will man dies verhindern, so können nach Zeile 7 und 16 entsprechende **rm**-Befehle eingeführt werden.

Listing 1: Bash-Script, das zum Zusammenfassen und Mischen sowie Trennen in Trainings- und Validierungsdaten verwendet wird.

```
1 #! /bin/bash
2
3 # fuse all layout-files to one unified file
4 cat *layout* >> full_layout.jsonl
5
6 # random shuffle all
7 shuf -o shuffled_layout.jsonl full_layout.jsonl
8
9 # splitting the shuffled unified file into actually train- and validation-set
10 # -l 320000 are splitting the file shuffled_layout.jsonl into different files after
11 #      ↪ 320.000 lines
12 # this is equivalent to a 80/20 split for train- and validation-set
13 split -l 320000 shuffled_layout.jsonl
14
15 # renaming the output files of the splitting command
16 mv aa layout.train
17 mv ab layout.valid
```

## A.2 Zusätzliche BLEU-Scores

Die Beschriftungen sind hier nur verkürzt dargestellt. Die Orientierung ist jedoch die selbe, wie in Kapitel 4.2 Tabelle 10.

Ansatz	BLEU-Score			
	2-Gram	3-Gram	4-Gram	1- bis 4-Gram
Basline	<b>66,30%</b>	<b>55,90%</b>	<b>50,18%</b>	<b>60,65%</b>
Greedy	54,11%	39,72%	31,57%	46,37%
Beam Search	63,77%	49,00%	40,29%	55,41%

(a) Box

Ansatz	BLEU-Score			
	2-Gram	3-Gram	4-Gram	1- bis 4-Gram
Basline	<b>66,30%</b>	<b>55,90%</b>	<b>50,18%</b>	<b>60,65%</b>
Greedy	49,70%	34,73%	26,67%	41,20%
Beam Search	58,73%	42,85%	34,12%	49,45%

(b) Box, Box-Klassen

Ansatz	BLEU-Score			
	2-Gram	3-Gram	4-Gram	1- bis 4-Gram
Basline	<b>66,30%</b>	<b>55,90%</b>	<b>50,18%</b>	<b>60,65%</b>
Greedy	51,24%	36,36%	28,22%	43,01%
Beam Search	60,85%	45,38%	36,54%	51,90%

(c) Box, Fläche, Seitenverhältnis

Ansatz	BLEU-Score			
	2-Gram	3-Gram	4-Gram	1- bis 4-Gram
Basline	<b>66,30%</b>	<b>55,90%</b>	<b>50,18%</b>	<b>60,65%</b>
Greedy	58,32%	42,99%	34,08%	49,62%
Beam Search	65,86%	50,83%	41,56%	57,21%

(d) Box, Fläche, Seitenverhältnis, Distanzen, geo. Features

Tabelle 11: BLEU-Scores mit Evaluiert am kleine (1.000) Dokumentensatz.