Processing the Structure of Documents: Logical Layout Analysis of Historical Newspapers in French

Nicolas Gutehrlé, Iana Atanassova

Journal of Data Mining and Digital Humanities

Processing the Structure of Documents: Logical Layout Analysis of Historical Newspapers in French

Nicolas Gutehrlé[1] **and Iana Atanassova**[1, 2]

[1]Centre de Recherches Interdisciplinaires et Transculturelles (CRIT)
Université de Bourgogne Franche-Comté,
30 rue Mégevand, 25000, Besançon, France,
[2]Institut Universitaire de France (IUF)

Corresponding author: Nicolas Gutehrlé , nicolas.gutehrle@univ-fcomte.fr

Abstract

Background. In recent years, libraries and archives led important digitisation campaigns that opened the access to vast collections of historical documents. While such documents are often available as XML ALTO documents, they lack information about their logical structure. In this paper, we address the problem of Logical Layout Analysis applied to historical documents in French. We propose a rule-based method, that we evaluate and compare with two Machine-Learning models, namely RIPPER and Gradient Boosting. Our data set contains French newspapers, periodicals and magazines, published in the first half of the twentieth century in the Franche-Comté Region.

Results. Our rule-based system outperforms the two other models in nearly all evaluations. It has especially better Recall results, indicating that our system covers more types of every logical label than the other two models. When comparing RIPPER with Gradient Boosting, we can observe that Gradient Boosting has better Precision scores but RIPPER has better Recall scores.

Conclusions. The evaluation shows that our system outperforms the two Machine Learning models, and provides significantly higher Recall. It also confirms that our system can be used to produce annotated data sets that are large enough to envisage Machine Learning or Deep Learning approaches for the task of Logical Layout Analysis. Combining rules and Machine Learning models into hybrid systems could potentially provide even better performances. Furthermore, as the layout in historical documents evolves rapidly, one possible solution to overcome this problem would be to apply Rule Learning algorithms to bootstrap rule sets adapted to different publication periods.

Keywords

Logical Layout Analysis, Historical Newspapers, Natural Language Processing, Rule-based system, Rule-Learning, Machine-Learning

I BACKGROUND

One important challenge in digital humanities is the efficient exploitation and processing of scanned textual documents (archives, documentary funds, ...). For example, historical documents such as newspaper archives are prime resources for historians [Tibbo, 2007]. Thanks to the important digitisation campaigns led by libraries and archives, vast collections of historical documents have been made easily accessible. However, the majority of these documents are available only as scanned images (e.g. in PDF format) which makes them difficult to explore in a text processing perspective. Extracting the text content from such documents requires at

least the following three steps: Optical Character Recognition (OCR), physical layout analysis (PLA) and Logical Layout Analysis (LLA).

Physical layout analysis (PLA), which is also sometimes called *document layout analysis*, consists in identifying physical regions of the document, with their text content and boundaries. Such regions can correspond to sections and lines of text, but also to figures, tables, etc. PLA also defines the reading order of the document, which corresponds to the linear order in which the different regions appear. This is particularly important for documents that have multi-column layouts. One commonly used output format of PLA is the XML ALTO format (https://www.loc.gov/standards/alto/). *Logical layout analysis (LLA)*, sometimes called *logical structure derivation* and *structure understanding*, consists in identifying the document structure elements and their categories i.e. title, header, paragraph, table, etc. Such logical elements can integrate one or more regions in the document that have been identified by PLA. Physical and logical layout analyses are necessary steps in the processing of documents for a large number of applications, including information retrieval, information extraction, table of content extraction, text syntheses, and more broadly document understanding.

In this article we focus on the problem of Logical Layout Analysis (LLA). We describe a methodology for Logical Layout Analysis, where logical labels are assigned to physical layout entities. The input of our processing pipeline is the physical layout analysis of documents in the XML ALTO format.

An important body of research around physical layout analysis of printed documents has been produced in the end of the XXth century. Several algorithms have been proposed such as the X-Y Cut algorithm [Nagy et al., 1992], the Docstrum algorithm [O'Gorman, 1993] or the Voronoi diagram based algorithm [Kise et al., 1999]. Furthermore, the processing of handwritten documents requires specific techniques, such as the "droplet" technique to identify text line by Bulacu et al. [2007], or neural networks as in Chen and Seuret [2017], where each pixel is labelled as text or not.

Existing Logical Layout Analysis systems make use of various methods that go from heuristic systems to more recent architectures using neural networks. Some heuristic systems use grammars such as stochastic or attributed grammars, where the document is represented as a string of symbols, e.g. Namboodiri and Jain [2007]. In their work, the grammar describes multiple production rules, each associated with a logical label. The string of symbols is then parsed by the grammar in order to extract logical labels. Other systems, such as LA-PDFText [Ramakrishnan et al., 2012] or DeLoS [Niyogi and Srihari, 1995], use rules that state the condition a physical block must meet to be given a logical label. For instance, DeLoS system uses first-order predicates in order to infer the logical category of a physical block.

While heuristic systems provide good results, they are often dedicated to specific layouts, and need to be adapted to work on other layouts. To tackle this problem, Klampfl and Kern [2013] created a system for Logical Layout Analysis on scientific articles in PDF format that combines heuristic rules with unsupervised-learning models such as k-means or Hierarchical Agglomerative Clustering (HAC). This system is made up of several detectors, each learning geometrical and textual features from the document in order to identify a specific logical label. Some rules using text occurrences are also used to help the model, such as finding the keywords "Table" or "Fig." to identify table or figure blocks.

More recent works use neural networks for Logical Layout Analysis. As noted by Akl et al. [2019], Convolutional Neural Network (CNN) or Long Short-Term Memory (LSTM) architec-

tures work better than classical neural networks because of the sequential nature of the documents. This task also benefits from the use of word-embeddings such as GloVe [Pennington et al., 2014], fastText [Bojanowski et al., 2016] or Flair [Akbik et al., 2018] which give a better encoding of textual data than simple one-hot encoding, as in Zulfiqar et al. [2019]. Neural network systems can be trained on big data sets such as the Publaynet data set [Zhong et al., 2019] or the Medical Articles Record Groundtruth (MARG) for physical and Logical Layout Analysis purposes.

Considering the task of processing historical documents, several small data sets exist such as the DIVA-HISDB data set [Simistira et al., 2016] which contains 150 annotated pages of three different medieval manuscripts or the European Newspapers Project data set [Clausner et al., 2015] which contains 528 documents. Other data sets in non-European languages exist, such as the PHIBD data set [Ayatollahi and Nafchi, 2013], which contains images of 15 Persian historical and old manuscript, and the HJdata set [Shen et al., 2020], which contains 2271 Japanese newspapers published in 1953, which was generated in a semi-automatic way. All of these data sets are too small to be used for Machine Learning or neural network approaches.

The works of Hébert et al. [2014] deal with the task of article segmentation by a Conditional Random Field (CRF) model with heuristic rules to perform logical analysis. First the CRF model labels pixel as titles, text lines, or horizontal and vertical separators, then heuristics rules describing usual article layouts are applied to that classification. In both cases, bad results were caused by the quality of the scan or the quality of the OCR output. On the other hand, Riedl et al. [2019] deal with article segmentation by looking at the similarity between segments of texts. These segments are computed either by using the Jaccard coefficient and their word distribution or by computing the cosine similarity between word-embeddings. The similarity between blocks is then computed using the TextTiling algorithm [Hearst, 1997].

Most common approaches to LLA are not suited for historical documents because the document layout changes over time. For example, the layout and structure of an advertisement in the same newspaper can display important changes over several years. Logical layout analysis systems applied to historical documents must then account for the diachronic aspect of their layouts and adapt to the changes. Barman et al. [2020] propose a system that goes beyond usual logical labels by labelling physical block as either Serial, Weather Forecast, Death Notice and Stock Exchange Table. To do so, their system combines visual and textual features using the word-embedding representation of each word and its coordinates on the page. Their results show that combining textual and visual features provide better results in most cases than using just one of them. Textual features are also more efficient to deal with the diachronic aspect of documents because they are more stable over time than visual features.

The rest of the article is organised as follows: Section II presents our train and test data sets, the methodology that we propose and describe the three models compared in this article. Section III proposes an evaluation of the three models and compares their results. Finally, we propose our conclusion in Section IV.

II METHODS

Our aim is to attribute logical layout labels to both TextBlock and TextLine tags in documents. For our experiment, first we design a rule-based system to perform the Logical Layout Analysis. This system is created using ad hoc rules and observations on the documents of the data set. Then we compare the results of this rule-based system with those of a Rule Learning model and a Machine Learning model.

In the following section we present our data set. Section 2.2 presents the tagset that is used, and section 2.3 presents the features set that we have defined for the TextBlock and TextLine elements. Then, section 2.4 explains the general processing pipelines. Section 2.5 explains the construction of the rule-based system, while sections 2.6 and 2.7 present the Rule Learning and Machine Learning models that we used.

2.1 Data set

We have processed a data set of press and magazine documents published in the first half of the twentieth century from the "Fond régional: Franche-Comté" catalogue, available on Gallica, the digital archive of Bibliothèque Nationale de France (https://gallica.bnf.fr). The previous study by Gutehrlé and Atanassova [2021] uses the same data set. Figure 1 shows an example of the first page of a newspaper with complex physical layout. It contains the header of the first page and several articles with titles and text content. From this catalogue, we selected documents that had an OCR quality measure greater than 90%. This data set was then split into a train and a test data set.

Figure 1: Excerpt of the first page of the second issue of the communist newspaper *Le Semeur* published on the 23rd of April 1932


PREMIERE ANNEE. — N° 2. SAMEDI 23 AVRIL 1932.

Le Semeur

JOURNAL COMMUNISTE

Territoire de Belfort, Doubs, Jura, Haute-Saône

ABONNEMENTS

6 mois..... 8.50

1 an..... 16 fr.

Le Numéro : 30 CENTIMES

Contre la guerre et la misère Tous debout le Premier Mai!

Pour la vie et le développement du "Semeur"

Cette Semaine...

Coup d'œil sur la situation en Allemagne

The train and test data sets are designed to cover as much as possible the various physical layouts that exist in the "Fond régional: Franche-Comté" catalogue. We have divided them into three layout types:

- **1c :** documents where the text is displayed in one column, as in books;
- **2c :** documents where the text is displayed into two columns;
- **3c :** documents where there are at least 3 columns of text, as in newspapers.

As shown in Table 1, our train set contains 15 collections of documents, which amount to a total of 48 documents, whereas the test set contains 6 collections and a total of 6 documents. Table 2 shows the distribution of documents in the train and test data sets across the three layout types.

The documents in the corpus cover three general topics: Catholicism, Resistance and News. The documents of the Catholic topic were published between 1900 and 1918. Most of them, such as *Bulletin paroissial de Censeau* or *Petit Écho de Sainte-Madeleine*, are bulletins of small parishes. As such, they focus mainly on the local religious life, although they sometimes discuss

Table 1: Description of the train and test data sets

Data set	Newspapers	Issues	Text blocks	Text lines	Words	Pages
Train	15	48	4 608	51 815	338 583	368
Test	6	6	1 445	8 836	63 343	52

Table 2: Number of documents per layout in the train and test data sets

Data set / Layout	1c	2c	3c+	Total
Train	18	5	25	48
Test	2	2	2	6

national and international events such as WWI. The documents from the Resistance topic, such as *La Haute-Saône libre* or *La Franc-Comtoise*, were published between 1939 and 1945 by Resistance fighters. As such, their main goal is to relay information about the ongoing local and international events of WWII. Finally, the documents of the News topic were published in the 1930s and focus on local and national events. Some are apolitical such as *Le Franc-Comtois de Paris*, while others have a political label. For instance, *Le Semeur* and *Le Front Comtois* are left-wing newspapers whereas *Vers l'Avenir* is a right-wing Catholic newspaper.

The French language used in these documents is not very different from modern French. However, we notice some variations in the written styles between the three topics. The written style in the Catholic document is formal and literary and uses many religious metaphors. On the other hand, the written style in the News document is mostly standard, although sometimes formal. Sentences are shorter and use simpler tenses than the Catholic documents. This simplification of the writing style is even more prominent in Resistance documents. The difference in the writing style between documents can first be explained by their domain: religious text should be more literary than newspapers or Resistance periodicals. This difference can also be explained by the size of the documents. Catholic documents are the longest in the corpus, with more than 10 pages on average. As such, their text can be more elaborate. On the other hand, News and Resistance documents are respectively four and two pages long on average. Their text is factual and concise in order to convey a lot of information in the limited space they have.

All the documents are stored in the XML ALTO format, which contains descriptions of their physical layout and the text content obtained from OCR. As such, the files already provide the physical layout analysis and the reading order of the documents. The XML ALTO format provides the text content and physical layout of documents in the following manner: the OCR output for the whole document is available in a PrintSpace tag. Lines of text are contained in TextLine tags, which in their turn contain String tags for words and SP tags for spaces. TextLine tags are grouped into blocks in TextBlock tags. Sometimes, TextBlock tags are also grouped into ComposedBlock tags. TextBlock and TextLine tags have the following attributes:

Id : the tag's identifier;
Height, Width : the text height and width;
Vpos : the vertical position of the text on the page. The higher the value, the lower the word is on the page;
Hpos : the horizontal position of the text on the page. The higher the value, the further on the right the text is on the page;
Language : the language of the text (only for TextBlock tags).

Besides the attributes listed above, a small portion of the TextBlock tags also have a Type attribute. This attribute is useful, when present, as it contains the logical labels of the lines in the block. It appears most often for tables or advertisements. However, the Type attribute is rarely present in our data set. As shown on Table 3, nearly 98 % of the TextBlock tags in the train and the test data sets do not have a Type attribute.

Table 3: Type attribute distribution on TextBlock tags in the train and test data sets

	Train		Test	
Type attribute	**Count**	**Percentage**	**Count**	**Percentage**
No attribute	4 514	97.96	1 423	98.48
illegible	79	1.71	15	1.04
titre1	15	0.33	0	0
advertisement	0	0	4	0.28
table	0	0	2	0.14
textStamped	0	0	1	0.07

2.2 Logical Layout Tagset

To perform the Logical Layout Analysis of the documents, we define the following annotation tagset:

TextBlock labels : Text, Title, Header, Other
TextLine labels : Text, Firstline, Title, Header, Other

The label "Firstline" must be understood as "first line of the paragraph". Thus, any TextLine tag labelled Firstline will indicate the beginning of a paragraph. A small portion of the TextBlock and TextLine tags correspond to elements that are not relevant for our study, such as images, tables or advertisement. Those elements were labelled as "Other" and are ignored for the evaluation.

The whole data set was manually annotated by a single annotator, then split into a train and a test data set. Table 4 shows the label distribution in the data sets. The train set was used to produce the rule set of our rule-based system, and also to train the Rule-learning and Machine-learning algorithms. The test set was used for the final evaluation of the system.

Table 4: TextBlock and TextLine tags label distribution over the train and test data sets

		Train		Test	
	Label	**Count**	**Percentage**	**Count**	**Percentage**
TextBlock	Text	2 064	45.724	1 102	80.203
	Title	429	9.503	90	6.550
	Header	333	7.377	53	3.857
	Other	1 686	37.35	128	9.314
TextLine	Text	36 272	70.138	6 648	75.881
	Firstline	9 785	18.921	1 563	17.840
	Title	1 820	3.519	234	2.670
	Header	740	1.430	115	1.312
	Other	3 098	5.989	201	2.293

2.3 TextBlock, TextLine and Document features

The first step of our processing pipelines consists in extracting and calculating sets of features from XML ALTO documents. These features describe three levels: TextLine, TextBlock and

Document level. Table 5 presents all the features with their descriptions and levels. The information on these features for all document elements, in the form of matrices, is used as input for the three models presented in Sections 2.5, 2.6, 2.7.

Table 5: Features used by the algorithm. The features marked with an * are used exclusively by the rule-based algorithm

	Feature	Description	TextLine	TextBlock	Document
1	*page*	page number of the page containing the element	X	X	
2	*blockType*	type of the block	X	X	
3	*wordCount*	number of words	X	X	
4	*precedingSpace, followingSpace*	spaces above and below the element	X	X	
5	*height, width*	height and width values of the line	X		
6	*hpos, vpos*	coordinates of the line on the page, i.e. its horizontal and vertical position	X		
7	*diffHpos*	difference between *hpos* and the median *hpos* value in the block	X		
8	*firsthpos, firstvpos*	coordinates of the first line of the block		X	
9	*lasthpos, lastvpos*	coordinates of the last line of the block		X	
10	*linecount*	number of lines		X	
11	*wordRatio*	number of words by line		X	
12*	*medHeight, medWidth*	median line height and line width		X	X
13*	*medHpos, medVpos*	median *hpos* and *vpos* values in the block		X	
14*	*medWordCount, medLineSpace*	median number of words by line and the median space between lines in the block		X	X
15*	*medBlockHeight, medBlockWidth*	median line height and block height and width			X
16*	*medBlockSpace*	median space value between blocks			X
17*	*thirdQuartileLineSpace*	third quartile of line space values in the document			X
18*	*medWordRatio, medLineCount*	median number of words by line and median number of line by block in the document			X
19	*capitalProp, digitProp nonAlphaProp*	proportion of capital letters and digits proportion of non-alphanmeric characters	X	X	
20	*stwCapital, stwDigit*	True if the line starts either by a capital letter or a number, False otherwise	X		
21	*endsPunct*	True if the line ends with a punctuation, False otherwise	X		
22	*headerMark1*	True if the element contains the word "Page" or a dash sign. False otherwise.	X	X	
23	*headerMark2*	True if the element contains a date, a currency, an address. False otherwise.	X	X	
24	*simTitle*	similarity of the line with the title of the document, calculated by the Levenshtein distance	X		
25	*simHeaderSet*	highest similarity of the line with the words contained in the header words set, calculated by the Levenshtein distance	X		

The header words set is used for the calculation of the *simHeaderSet* feature. It has been created by observing the different types of headers in the data sets and consists of the following words or phrases: *Rubrique Locale, Gérant, Publicité, Abonnement, Envoyez les fonds, Conservez chaque numéro, Rédacteur, Directeur, Numéro, Chèque postal, Dépôt, Achat-Vente-Echange, Annonce, Imprimerie, En vente partout, Paraissant*.

2.4 Processing pipelines

In this section we present the two pipelines that we propose for the processing of XML ALTO documents for Logical Layout analysis. First, we present the rule-based system, and then we explain how it can integrate the Rule Learning or Machine Learning models.

The first step of the processing pipeline consists in the extraction of features from the XML ALTO document at the TextLine, TextBlock and Document levels, as described in Section 2.3. We store these features into two matrices for TextLine and TextBlock features and in a dictionary for Document features. Each row in the matrices represents either a TextLine or a TextBlock tag and each column is a corresponding feature.

2.4.1 Rule-based system

For the rule-based system, the second step attributes logical labels to TextBlock tags. Labelling TextBlock before TextLine is important because the presence of a Type attribute in TextBlocks can help label the lines inside these blocks. The goal of this step is to add a Type attribute to every TextBlock. To do so, we process the TextBlock feature matrix from the previous step by applying sets of annotation rules, one for each possible logical label. A TextBlock is only processed if it doesn't already have a Type attribute. Because the sets of rules are applied independently from each other, a same TextBlock can obtain multiple labels. Another set of rules is then applied to solve such conflicts and keep only one possible logical label for each TextBlock, which is then set as the value of the Block's Type attribute in the feature matrix.

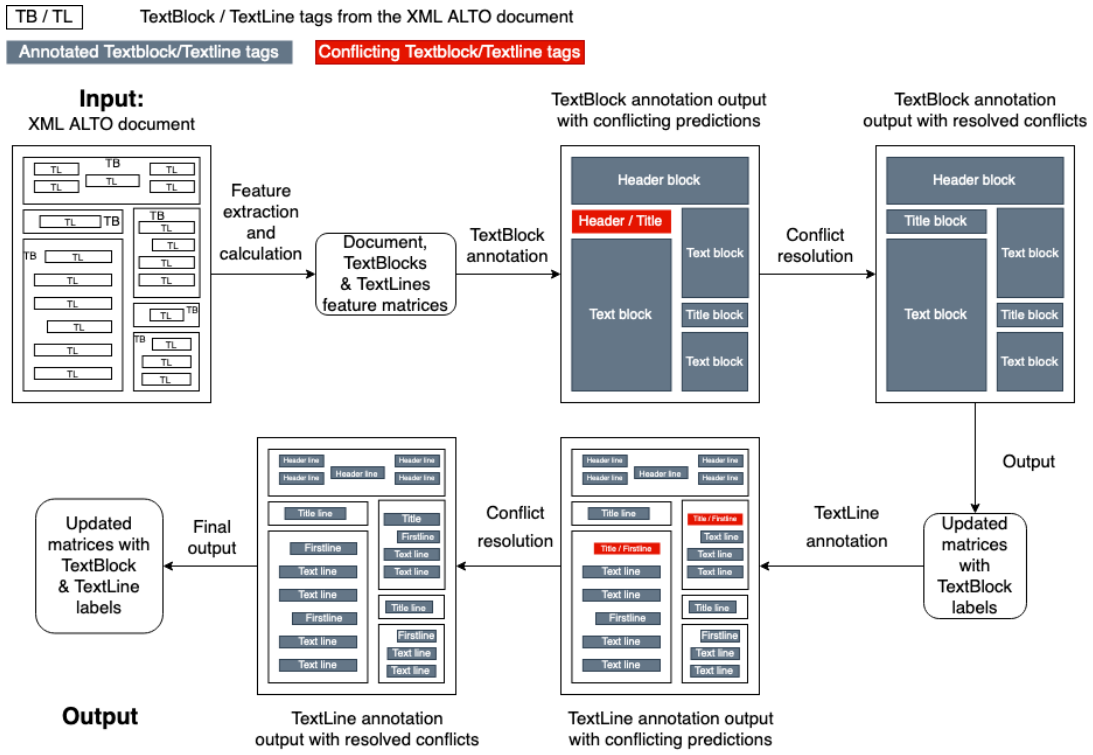
The third step attributes logical labels to TextLine tags. Every TextLine is by default labelled as Text. The system then applies rules to identify the other labels. First, any TextLine in a Title or a Header block inherits the same label. Then, any TextLine contained in a TextBlock is processed by a set of rules in order to identify Firstlines and possible missing Titles. Similarly to the previous step, rules are applied independently from each other, resulting sometimes in conflicting predictions. The TextLine feature matrix is processed a second time to solve conflicting predictions and keep only one possible label for each TextLine tags. This step also controls that any line that follows a Title is labelled as Firstline and that the first line of the document is labelled as Title if it not already labelled as Header.

The rule set of our rule-based system uses all features presented in Table 5. The algorithm finally outputs the three feature matrices, where the TextBlock and TextLine matrices are updated with the annotations of steps 2 and 3. Figure 2 shows the main steps of this pipeline.

2.4.2 Rule Learning and Machine Learning pipelines

The processing pipeline which integrates the Rule Learning and Machine Learning methods differs from the rule-based one as follows. The second step attributes logical labels to TextBlock tags and updates the feature matrices accordingly. The last step attributes logical labels to TextLine tags and outputs the final updated matrices. The steps 2 and 3 rely on Rule Learning or Machine Learning models. Unlike our rule-based system, however, there is no need for

Figure 2: Processing pipeline of the rule-based system



conflict resolution steps, as the Rule Learning and Machine Learning algorithms output the final prediction. Figure 3 shows the main steps of this pipeline.

To train the Rule Learning and Machine Learning algorithms, we use the same set of features, but we exclude features 12 to 18. These features contain values that are calculated using the data of the other features. They are used only by the rule-based system, and not by the other algorithms as they do not provide any new data.

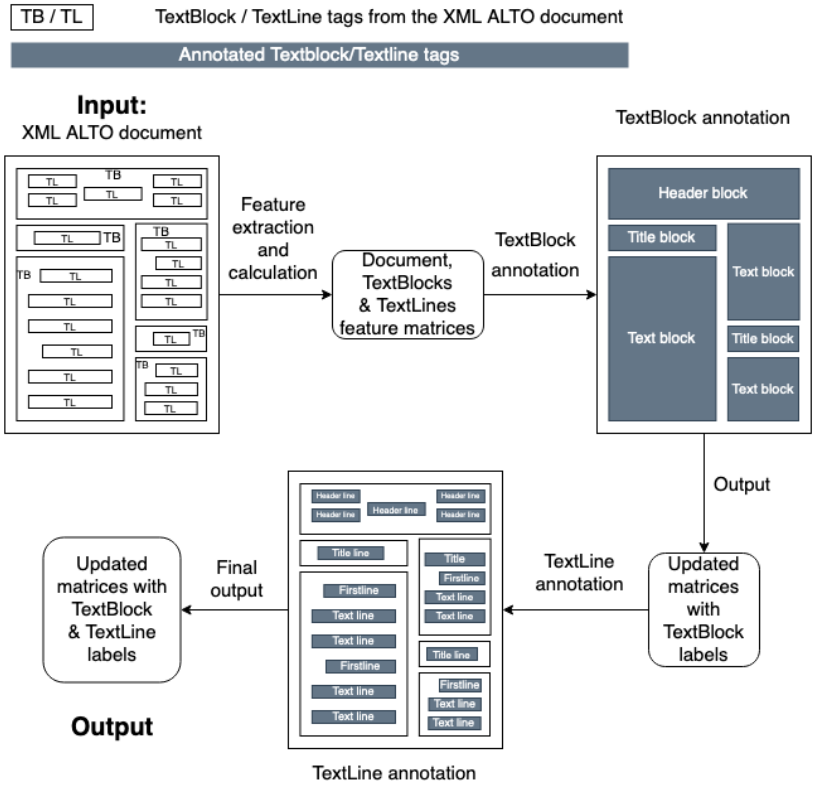
2.5 Rule-based system

To construct the rule-based system, we define sets of rules for the annotation of TextBlock and TextLine tags. To design the rules, we used heuristics and observations that we made on the train data set. For instance, we observed that the biggest titles in the documents start with a capital letter and are surrounded by important spaces. Then, we translated these patterns into rules in the form of conditions on the values of the features that must be that must be verified to trigger the annotation of the element with the corresponding label. The obtained set of rules is applied to documents regardless of the layout category they belong to.

Identifying Text and Title blocks relies on geometric and morphological features, whereas identifying Header blocks relies on semantic features. While designing the rules for the annotation of TextBlock elements we have taken into consideration the following observations:

- Text blocks contain relatively more lines and more words than other blocks (titles or headers) in the document.
- Title blocks are TextBlock tags that contain few lines, usually not more than 3. The role of a title is to introduce the topic of a text section, thus a Title block should be surrounded by Text blocks. The space around that block should also be important, in order to stand out with the surrounding blocks.

Figure 3: Processing pipeline of the Rule Learning and Machine Learning algorithms



- A Text block should have a smaller height than a Title block. As such, if there is a confusion between Text and Title block, we use the height of the block to distinguish between the two.
- Headers contain very specific information about the document, such as its title, its price, a date or the publisher's name. This information is displayed with keywords and sentences that are recurrent across multiple pages and documents.
- Header blocks are only located at the top of a page, generally in the first four lines of a page. Small blocks at the top of a page are most likely to be Headers.
- On the first page of a document, the header can be much longer because it contains more information. We consider that it can be up to 30 lines.

Naturally, TextLine tags that are contained in a Title or Header block inherit this annotation. TextLine tags that appear between two Header lines are also annotated as Header. Furthermore, to find Firstline and Title lines that have not yet been identified, we apply sets of rules that rely on geometric and morphological features, taking into account the following observations:

- The first line of a page or immediately after a Title should be labelled Firstline, if it starts with a capital letter.
- The first line of a paragraph always starts with a capital letter, and is often indented, i.e. with Hpos value greater than the other TextLines in the block.
- Firstlines that are not indented can be identified if the line that precedes them is shorter, indicating the end of the previous paragraph.
- Title lines are surrounded by relatively more space in order to stand out from other text sections. The smaller the title is, the less important the space around it is.
- Small titles (e.g. sections in an article) usually contain more capital letters then the rest of the text and are center-aligned.

Tables 6 and 7 present all annotation rules for TextBlock and TextLine tags and their corresponding annotation labels, where B is a TextBlock in a document D and L is a TextLine in B. The last two rules in each table solve conflicting annotations.

Table 6: TextBlock annotation rules and conflict resolution rules of the rule-based system

Rule	Condition	Label
1	*(B.linecount > D.medLineCount)* or *(B.wordCount > D.medWordCount/3)*	Text
2	Previous and next TextBlocks are Text and *(B.linecount < D.medLineCount)* and *(B.medHeight < D.medBlockHeight)*	Text
3	Previous and next TextBlocks are Text and B is not Text and *(B.linecount < 4)* and *(B.precedingSpace > D.medBlockSpace) or (B.followingSpace > D.medBlockSpace)*	Title
4	*B.page = 1* and for any of the first 30 lines of B: *simHeaderSet > 0.9* or *simTitle > 0.9* or *headerMark1* or *headerMark2* or *ctnTotal*	Header
5	*B.page > 1* and for any of the first 4 lines of B: *simHeaderSet > 0.9* or *simTitle > 0.9* or *headerMark1*	Header
6	Conflicting annotation: Header and (Text or Title):	
	(B.linecount < 15) and *(B.wordCount < 50)*	Header
	Otherwise	Text / Title
7	Conflicting annotation: Text and Title:	
	B.medHeight > D.medBlockHeight / 2	Title
	Otherwise	Text

Table 7: TextLine annotation rules and conflict resolution rules of the rule-based system

Rule	Condition	Label
1	L.*precedingSpace* = 0 and L.*followingSpace* > D.*medLineSpace* and L.*simTitle* < 60 and L.*simHeaderSet* < 60 and L.*stwCapital*	Title
2	L.*wordCount* < B.*medWordCount* and L.*precedingSpace* > D.*thirdQuartileLineSpace* and L.*followingSpace* > D.*thirdQuartileLineSpace*	Title
3	L.*capitalProp* > 10 and L.*wordCount* < B.*medWordCount* and L.*height* < B.*medHeight* and (L.*precedingSpace* > D.*thirdQuartileLineSpace* or L.*followingSpace* > D.*thirdQuartileLineSpace*)	Title
4	L.*diffHpos* > 104 and L.*capitalProp* > 0 and L.*precedingSpace* > D.*medLineSpace* and L.*followingSpace* > D.*medLineSpace*	Title
5	L.*hpos* > B.*medHpos* and L.*diffHpos* < 105 and (L.*stwCapital* or L.*stwDigit*)	Firstline
6	L.*width* < B.*medWidth* and L.*wordCount* < B.*medWordCount* and L.*hpos* < B.*medHpos*	Lastline
7	Previous TextLine is LastLine and L.*stwCapital* and L.*followingSpace* < B.*medLineSpace*	Firstline
8	Previous TextLine is not Lastline and L.*stwCapital* and L.*precedingSpace* > B.*medLineSpace* and L.*followingSpace* < B.*medLineSpace*	Firstline
9	Previous TextLine is not Lastline and L.*stwCapital* and L.*hpos* > B.*medHpos*	Firstline
10	None of the rules 1-9 above is True	Text
11	Conflicting annotation: Header and other label:	
	Previous TextLine is Header and next TextLine is Header	Header
12	Conflicting annotation: Title and FirstLine:	
	L.*followingSpace* < B.*medLineSpace* and L.*capitalProp* < 15	Title
	Otherwise	Firstline

2.6 Rule Learning

Rule Learning algorithms are Machine Learning algorithms that aim at identifying rules from a data set. In a classification task, a rule-learning model induces from the data set the rules that allow to classify a particular sample. For our experiment, we have selected the RIPPER algorithm for two reasons: firstly, it is considered to be the state of the art of rule-induction systems [Sammut and Webb, 2017], and secondly, for its ability to produce human-readable rule sets.

2.6.1 Description of the RIPPER algorithm

RIPPER (Repeated Incremental Pruning to Produce Error Reduction) [Cohen, 1995] is a rule induction algorithm which improves upon the Incremental Reduced Error Pruning algorithm (IREP) [Fürnkranz and Widmer, 1994]. RIPPER discovers rules in a sequential covering manner: for a positive class P, it identifies the rules that best cover the examples of P in the data set. Every element in the data set covered by the rule, be it positive or negative, is then removed. These operations are repeated until the rule set cannot grow any more or another condition is met.

In order to construct the rule set, RIPPER first randomly divides the training set into a *growing* and a *pruning* set. The growing set is used to identify rules and corresponds roughly to 2/3 of the training set. The pruning set is used to test the rules and prune them. Building a rule consists of the two following steps: *Growing* and *Pruning*.

Rules in RIPPER are sets of conditions combined using the AND operator. The growing step of a rule starts with an empty rule. The algorithm loops over every possible condition in the data set, i.e. every possible value for each given feature. It then adds the condition which provides the highest information gain to the rule. The information gain is calculated with the same metric used by the First Order Inductive Learner (FOIL) algorithm [Quinlan, 2005]. The algorithm keeps adding conditions to the rule until it only covers examples of the positive class. Once a rule has stopped growing, its quality is evaluated with the following metric:

$$ruleQuality = (P - N)/(P + N)$$

where P and N are respectively the number of positive and negative examples covered by the pruned rule in the pruning set.

The pruning step is then initiated. It consists in removing one by one every condition in the rule, from the newest to the oldest, while evaluating its quality with the same metric. The version of the rule having the best quality is kept.

When a rule is grown and pruned, RIPPER computes the description length (DL) of the new rule. The DL indicates the complexity of a rule and corresponds to the sum of the number of bits required to encode the rule and the examples of the positive class the rule fails to cover. Finally, every example in the growing set covered by the newly built rule is removed. These steps are repeated until one of the following conditions is met:

- the rule set covers every instance of the positive class;
- the error rate is above 50 % since the addition of the last rule;
- the rule has reached a specified complexity threshold.

The complexity threshold is reached when the DL of the last added rule is d bits bigger than the smallest DL in the rule set. By default, d is 64 bits. After creating the rule set, RIPPER iterates

from the newest to the oldest rule in the rule set and checks if the rule can be removed without increasing the total description length.

Finally, the optimisation step is initiated. It consists in creating two new versions of each rule: a replacement rule and a revision rule. The replacement is a completely new rule whereas the revision rule is created by adding new conditions to the existing rule. The algorithm then selects the version of the rule with the smallest description length. The optimisation step may be run k times, where $k = 2$ by default.

For our experiment, we used the python library *wittgenstein* which implements the RIPPER algorithm. The main hyperparameters of this implementation are:

prune_size :	the size of the prune set. The default value is .33
k :	the number of optimisations to run. The default value is 2
dl_allowance :	the allowed size for description length. The default value is 64
n_discretize_bins :	specific to the *wittgenstein* implementation. It automatically detects and discretises continuous features in the training set. This hyperparameter controls the size of each bin. The default value is 10.

As the *wittgenstein*'s implementation of RIPPER can only deal with binary classification, we trained one model for each label in our data set for both the TextBlock and TextLine annotation tasks. For each RIPPER model, we performed a GridSearch with the following combination to find the best hyperparameter values:

prune_size :	.25, .33, .50
k :	1, 2
dl_allowance :	32, 64, 128
n_discretize_bins :	5, 10, 20, 30

Table 8 shows the best hyperparameter combinations for each RIPPER model we trained for the TextBlock and TextLine annotation tasks.

For each logical label in the TextBlock or TextLine annotation task, we use the corresponding RIPPER model to predict the probability that the TextBlock or TextLine tag belongs to that logical label. The most probable label is then assigned to the tag.

Table 8: Best hyperparameter combinations for the RIPPER algorithm on each class for TextBlock and Textline annotation tasks

	Hyperparameter	Header	Title	Firstline	Text
TextBlock	prune_size	.5	.33		.25
	k	2	1		2
	dl_allowance	64	64		64
	n_discretize_bins	10	10		10
TextLine	prune_size	.33	.25	.33	.5
	k	1	1	1	2
	dl_allowance	64	64	64	64
	n_discretize_bins	10	10	10	10

2.7 Machine Learning

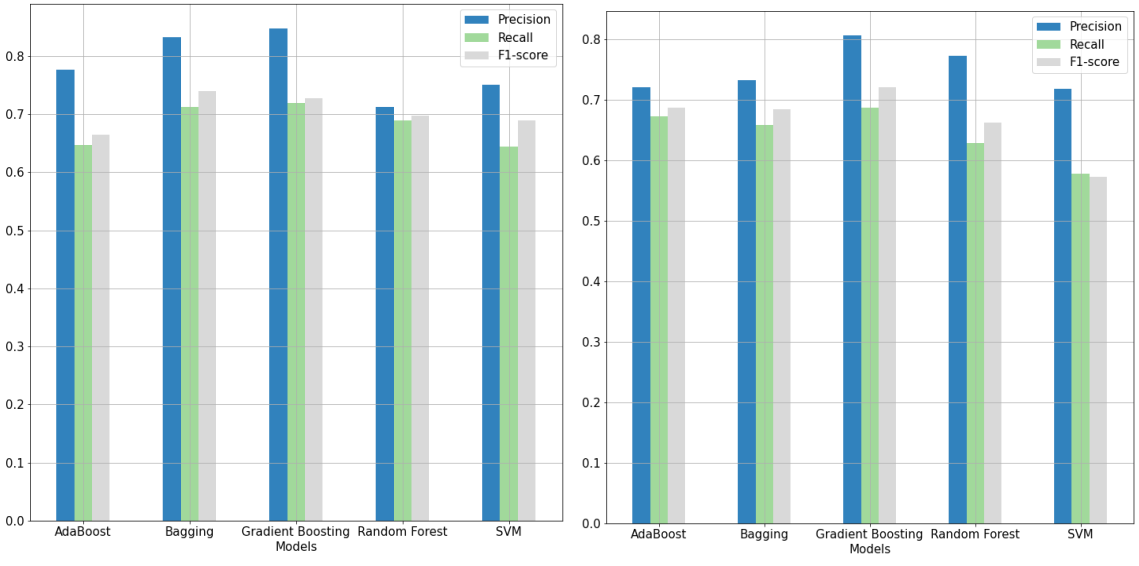
Machine Learning represents the family of algorithms such as Support Vector Machine or Gradient Boosting which are able to learn patterns from the data on their own. Machine Learning algorithms are often quicker to train than rule-based systems. However, they are also harder to interpret as they do not output rules, nor explain their reasoning to classify a specific sample.

In order to select the algorithm which is best suited for TextBlock and TextLine classification, we compared the performances of the following Machine Learning algorithms: *Support Vector Machine (SVM)*, *Bagging*, *Random Forest*, *AdaBoost* and *Gradient Boosting*. We used the *scikit-learn* implementation of these algorithms [Pedregosa et al., 2011], with their default hyperparameters on the same feature set that was used to train the RIPPER algorithm. Table 9 and Figure 4 compare the initial results obtained by these algorithms. The best result in each column is shown in bold.

Table 9: Results of Machine Learning models for TextBlock and TextLine annotation

	TextBlock			TextLine		
Algorithm	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
SVM	0.750	0.644	0.688	0.717	0.578	0.572
Bagging	0.833	0.712	**0.740**	0.732	0.657	0.683
RandomForest	0.712	0.689	0.697	0.773	0.628	0.662
AdaBoost	0.775	0.647	0.664	0.720	0.672	0.687
Gradient Boosting	**0.847**	**0.719**	0.727	**0.806**	**0.687**	**0.720**

Figure 4: Comparison of initial results of Machine Learning algorithms for TextBlock (left) and TextLine (right) annotations



This initial comparison shows that Gradient Boosting is the best algorithm for both TextBlock and TextLine annotation in every evaluation except one. From these results, we trained two Gradient Boosting classifiers, one for TextBlock annotation and another for TextLine annotation. We performed a GridSearch for each task with the following sets of hyperparameters in order to find the optimal combination:

n_estimators :	100, 150, 200
learning_rate :	0.01, 0.005, 0.001
max_leaf_nodes :	2,3,4,5,6,7
min_samples_split :	10, 20, 40, 60, 100
min_samples_leaf :	1,3,5,7,9
max_features :	2,3,4,5,6,7
subsample :	0.7,0.75,0.8,0.85,0.9,0.95,1

Table 10 shows the best hyperparameter combinations for the Gradient Boosting algorithm identified by GridSearch for both tasks.

Table 10: Best hyperparameter combinations for Gradient Boosting for TextBlock and TextLine annotation

Hyperparameter	TextBlock	TextLine
n_estimators	200	100
learning_rate	0.005	0.01
max_leaf_nodes	7	7
min_samples_split	40	100
min_samples_leaf	5	7
max_features	6	7
subsample	0.85	1

III RESULTS AND DISCUSSION

To evaluate our rule-based system, as well as the Rule Learning and Machine Learning algorithms, we have run the three models on the test data set. In this section we present the results in terms of Precision, Recall and F1-score. Considering the Rule learning algorithm, we also present the set of rules that were obtained by the algorithm and compare this set with the one that we have defined in our rule-based system. Finally, we compare the results of all systems.

3.1 Evaluation of the rule-based system

Table 11 shows the Precision, Recall and F1 scores for the TextBlock and TextLine classification of the rule-based system.

Table 11: Precision, Recall and F1 score for TextBlock and TextLine annotation with the rule-based system

		Text			Title			Firstline			Header		
	Cat	P	R	F1	P	R	F1	P	R	F1	P	R	F1
TextBlock	1c	0.947	0.938	0.942	0.312	0.357	0.333				0.679	0.373	0.476
	2c	0.973	0.989	0.981	0.899	1.000	0.947				1.000	0.271	0.411
	3c+	0.958	0.973	0.965	0.589	0.560	0.551				0.500	0.250	0.333
	Mean	0.959	0.966	0.962	0.600	0.639	0.610				0.726	0.298	0.406
TextLine	1c	0.979	0.986	0.983	0.354	0.720	0.473	0.943	0.854	0.895	0.909	0.598	0.721
	2c	0.961	0.995	0.978	0.746	0.765	0.747	0.955	0.859	0.902	1.000	0.118	0.197
	3c+	0.975	0.992	0.983	0.703	0.702	0.702	0.952	0.877	0.913	0.500	0.400	0.444
	Mean	0.969	0.991	0.979	0.595	0.733	0.639	0.949	0.861	0.902	0.803	0.348	0.435

TextBlock annotation rules perform best on documents from the 2c layout category. Title classification for TextBlocks performs with F1 score of 0.61 on average and 0.94 on documents from the 2c category. Header classification for TextBlocks provides a good Precision score (0.726) but with a low Recall (0.298).

Similarly to TextBlock annotation rules, TextLine annotation rules perform best on documents from the 2c category. Title identification performs worse on 1c documents, and obtains overall

F1 score of 0.639 for all layouts. Firstline identification performs fairly well with an F1 score above 0.9. Header identification obtains a good Precision score (0.803) but with a Recall of 0.348. This means that header identification rules are insufficient and need to be completed to capture the various types of headers.

A first type of error comes from errors in the Block classification step. As any line in a Title or Header block inherits that annotation, the Precision of TextBlock annotation is an important factor for the overall performance of the algorithm.

A second type of error is the confusion between Titles and First lines. Most Titles mislabelled as Firstline are short subsection titles. As such, they are similar to other text lines in terms of typography, and are hard to detect with the features we use. This confusion happens mainly in documents from the 2c and 3c+ categories. Other mislabelled Titles are one-line paragraphs such as greetings or signatures, or the beginning of a text section. Such lines have properties similar to Titles, being surrounded by important spaces and being either center or right-aligned. Extracting features about the font style of the line (bold, italics) and its alignment (left, center, right-aligned) could help solve this confusion.

3.2 Comparison of RIPPER's annotation rules and the rule-based system

In this section we present a comparison between the rules that are used in our rule-based system and the annotation rules that were learned by the RIPPER algorithm. We examine the sizes of the two rule sets and the overlap that exists between them.

3.2.1 TextBlock annotation rules

Table 12 presents all annotation rules discovered by the RIPPER algorithm for TextBlock tags and their corresponding annotation labels, where B is a TextBlock in a document D. Overall, RIPPER produced 26 rules, while in the rule-based system we produced 7, including the conflict resolution rules.

Table 12: TextBlock annotation rules produced by the RIPPER algorithm

Rule	Condition	Label
1	*B.capitalProp => 55* and *80 <= B.followingSpace <= 135* and *135 <= B.width <= 368.25*	Title
2	*B.height => 95* and *801.25 <= B.width <= 985* and *5 <= B.capitalProp <= 12.5*	Title
3	*B.capitalProp => 55* and *940 <= B.firstvpos <= 1545* and *B.height <= 30*	Title
4	*B.height => 95* and *485 <= B.firstvpos <= 940*	Title
5	*B.linecount <= 2* and *B.height => 95* and *940 <= B.firstvpos <= 1545* and *79 <= B.wordRatio <= 96*	Title
6	*B.capitalProp => 55* and *B.wordRatio => 149* and *B.digitProp <= 5* and *B.lastvpos <= 900*	Title
7	*B.capitalProp => 55* and *80 <= B.followingSpace <= 135* and *B.linecount <= 2* and *4 <= B.wordCount <= 9*	Title
8	*B.linecount <= 2* and *B.digitProp <= 5* and *B.height => 95* and *175 <= B.followingSpace <= 225*	Title
9	*B.linecount <= 2* and *B.digitProp <= 5* and *135 <= B.followingSpace <= 175* and *65 <= B.height <= 95*	Title
10	*B.linecount <= 2* and *B.digitProp <= 5* and *65 <= B.height <= 95*	Title
11	*B.linecount <= 2* and *B.capitalProp => 55* and *595 <= B.lasthpos <= 1130*	Title
12	*B.linecount <= 2* and *B.digitProp <= 5* and *B.capitalProp => 55* and *80 <= B.followingSpace <= 135* and *5849 <= B.firstvpos <= 7470*	Title
13	*B.linecount <= 2* and *B.digitProp <= 5* and *B.followingSpace => 225* and *B.height => 95* and *4 <= B.wordCount <= 9* and *B.precedingSpace <= 5*	Title
14	*B.height => 95* and *B.followingSpace => 225* and *940 <= B.firstvpos <= 1545* and *4 <= 4 B.wordCount <= 9*	Title
15	*B.linecount <= 2* and *B.digitProp <= 5* and *B.followingSpace => 225* and *B.height => 95* and *61 <= B.wordRatio <= 70*	Title
16	*B.linecount <= 2* and *B.digitProp <= 5* and *135 <= B.followingSpace <= 175*	
17	*B.linecount <= 2* and *B.digitProp <= 5* and *B.followingSpace => 225* and *B.height => 95* and *5 <= B.capitalProp <= 12.5*	Title
18	*B.lastvpos <= 900* and *5 <= B.digitProp <= 80*	Header
19	*B.lastvpos <= 900* and *3015 <= B.firstHpos <= 3640* and *B.capitalProp >= 55*	Header
20	*B.lastvpos <= 900* and *135 <= B.width <= 368.25* and *B.firstvpos <= 485*	Header
21	*B.lastvpos <= 900* and *B.digitProp => 80*	Header
22	*B.lastvpos <= 900* and *12.5 <= B.capitalProp <= 55* and *595 <= B.lasthpos <= 1130*	
23	*B.digitProp <= 5* and *B.capitalProp <= 5* and *B.wordCount => 272*	Text
24	*50 <= B.wordCount <= 272*	Text
25	*23 <= B.wordCount <= 50* and *B.capitalProp <= 5*	Text
26	*B.digitProp <= 5* and *B.capitalProp <= 5* and *45 <= B.precedingSpace <= 80*	Text

To identify Title blocks, RIPPER produced 17 different rules whereas our rule-based system only used one. We can see that $linecount$ and $height$ are the two most frequently used features, as they occur in 11 rules. A recurring condition is that $linecount$ must be inferior to three, which is similar to the condition in our rule-based system, where a block must have no more than four lines. This confirms our intuition that a small number of lines is an important factor to detect Title blocks. As in our system, $followingSpace$ is also an important feature, as it appears in nine rules and must always be above a certain threshold. However, RIPPER never uses the $precedingSpace$ attribute, suggesting that only one of them is necessary to identify

Title blocks. $capitalProp$ and $digitProp$ are also important features identified by RIPPER, as they respectively appear in eight and nine rules. The first one must always be above 50 % whereas the second one must always be inferior or equal to 5 %.

To identify Header blocks, RIPPER produced five rules, while we produced two. The condition $lastvpos <= 900$ is present in every rule, indicating that the block must be on the top part of the document. Other used features such as $digitProp$, which is always greater than 5 %, suggest that a Header blocks always contain numbers. To detect Headers, RIPPER relies exclusively on geometric and morphological features, unlike our system which mostly relies on semantic features such as $simHeaderset$, $simTitle$, $headerMark1$ or $headerMark2$.

Finally RIPPER produced four rules to identify Text blocks, while we produced two. The most important feature used is $wordCount$ which appears in three rules. Similarly to our system, this feature must be greater than a small threshold which is 23 words here. Unlike our system, RIPPER never uses the $linecount$ feature to detect Text blocks. Instead, it uses the $capitalProp$ and $digitProp$ features, which respectively appear in three and two rules. In every rule they appear, they must be lower or equal than 5 %. Finally, RIPPER uses in one rule the $precedingSpace$ feature, which must be lower or equal than 80 pixels, suggesting that the space before a Text block must be small.

3.2.2 TextLine annotation rules

Table 13 presents all annotation rules discovered by the RIPPER algorithm for TextLine tags and their corresponding annotation labels, where L is a TextLine in a document D and B is the TextBlock that contains L. Overall, RIPPER produced 19 rules whereas we produced 12, including the conflict resolution rules.

Table 13: TextLine annotation rules identified by the RIPPER algorithm

Rule	Condition	Label
1	*L.followingSpace => 75* and *L.stwCapital is True* and *L.blockType=title* and *745 <= L.width <= 970* and *L.vpos <= 985*	Title
2	*L.followingSpace => 75* and *L.stwCapital is True* and *L.blockType=title* and *L.capitalProp <= 5*	Title
3	*L.followingSpace => 75* and *L.stwCapital is True* and *L.blockType=title*	Title
4	*L.precedingSpace => 75* and *L.followingSpace => 75* and *335 <= L.width <= 645* and *20 <= L.simTitle <= 29*	Title
5	*L.followingSpace => 75* and *L.stwCapital is True* and *L.endsPunct is False* and *L.capitalProp <= 5* and *745 <= L.width <= 970* and *L.nonAlphaProp <= 5*	Title
6	*L.precedingSpace => 75* and *L.followingSpace => 75* and *L.stwCapital is True* and *335 <= L.width <= 645*	Title
7	*L.capitalProp => 10* and *335 <= L.width <= 645* and *L.height <= 35* and *505 <= L.hpos <= 1070* and *L.stwCapital is True*	Title
8	*L.precedingSpace => 75* and *L.followingSpace => 75* and *745 <= L.width <= 970*	Title
9	*L.precedingSpace => 75* and *L.followingSpace => 75* and and *L.stwCapital is True* and *645 <= L.width <= 745*	Title
10	*L.capitalProp => 10* and *L.vpos <= 985* and *L.blockType=header*	Header
11	*L.capitalProp => 10* and *L.simHeaderSet => 85.5* and *L.followingSpace => 75* and *335 <= L.width <= 645*	Header
12	*L.capitalProp => 10* and *L.vpos <= 985* and *54 <= L.simHeaderSet <= 60* and *L.precedingSpace <= 35*	Header
13	*L.capitalProp => 10* and *L.simTitle => 42.86* and *L.nonAlphaProp <= 5* and *L.precedingSpace <= 35* and *2550 <= L.hpos <= 3419*	Header
14	*L.capitalProp => 10* and *L.vpos <= 985* and *L.simTitle => 42.86* and *L.nonAlphaProp <= 5* and *335 <= L.width <= 645*	Header
15	*505 <= L.hpos <= 1070* and *L.simHeaderSet => 85.5* and *335 <= L.width <= 645*	Header
16	*L.stwCapital is True* and *970 <= L.width <= 1010*	Firstline
17	*L.stwCapital is True* and *L.capitalProp <= 5*	Firstline
18	*L.stwCapital is False* and *L.stwDigit is False*	Text
19	*1030 <= L.width <= 1050*	Text

Nine rules have been produced by RIPPER to identify Title lines, where we produced four. The most important feature is $followingSpace$, as it is used in eight rules. Similarly to our system, it must always be greater than a specific threshold, here 75 pixels. $precedingSpace$ seems to be a less important feature with RIPPER than with our system, as it is only used in four rules. This would suggest that, as for Title block annotation, the space following a block is more important than the one preceding it. Our rules relied mostly on the $blockType$, $precedingSpace$ and $followingSpace$ features to identify Titles, whereas RIPPER uses these features alongside others, especially $stwCapital$ and $width$ which are both present in seven rules. $stwCapital$ is always True whereas $width$ is always smaller or equal than a specific threshold, suggesting Title lines are shorter than most lines in the document.

RIPPER produced six rules to identify Header lines. In our system, a line is labelled as Header if it is contained in a Header block. The main feature used by RIPPER is $capitalProp$ which is present in 5 rules and is always greater or equal than 10 %. This suggests that Header lines have more capital letters than common lines. The other main features used are $vpos$, $simHeader$ and

$width$. $vpos$ is always below 985 pixels, which means that the Header must be on the highest part of the document. Depending on the rules, $simHeader$ must be greater than 55 % or 85 % percent, suggesting the importance of the header word set. The $width$ of the line is always set in a small range between 335 and 645 pixels, suggesting Header lines are smaller than most lines of text in the document.

Firstline annotation is the only case where RIPPER produced fewer rules than us: we produced five rules where RIPPER produced two. Similarly to our system, the main condition is that $stwCapital$ is True. The first line of a paragraph is also often indented. To detect that, our rule-based system used the $hpos$ feature whereas RIPPER uses the $width$ feature. Finally, RIPPER uses two rules to detect Text line whereas this is the default annotation in our system. In RIPPER, the main condition is that $stwCapital$ and $stwDigit$ are False or the $width$ of the line is average.

The use of the $blockType$ attribute is the main difference between our rule-based rule set and RIPPER's. In our system, the $blockType$ attribute is an important feature, as any line contained in a Header or Title block inherits this label. RIPPER also uses this feature but to a lesser extent. It is only used in three rules for Title annotation and in one rule for Header annotation, and is always used alongside other features.

3.3 Evaluation of the RIPPER system

An important question is which of the two sets of rules provides better results in terms of F1 score: RIPPER's rules or our manually designed rule set. To evaluate the rules produced by RIPPER, we have run the model through the test data set. Table 14 shows the results of TextBlock and TextLine annotation as performed by the RIPPER algorithm.

Table 14: Precision, Recall and F1 score for TextBlock and TextLine annotation with RIPPER

		Text			Title			Firstline			Header		
	Cat	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
TextBlock	1c	0.819	0.990	0.897	0.500	0.625	0.550				0.857	0.333	0.480
	2c	0.851	1.000	0.920	0.500	0.444	0.471				0.462	0.250	0.324
	3c+	0.852	0.999	0.920	0.679	0.297	0.413				0.000	0.000	0.000
	Mean	0.841	0.996	0.912	0.560	0.455	0.480				0.440	0.194	0.268
TextLine	1c	0.871	0.881	0.876	0.600	0.290	0.391	0.644	0.777	0.704	0.667	0.053	0.098
	2c	0.923	0.947	0.935	0.710	0.268	0.389	0.754	0.877	0.811	0.667	0.065	0.118
	3c+	0.876	0.908	0.892	0.610	0.207	0.309	0.614	0.709	0.658	0.000	0.000	0.000
	Mean	0.890	0.912	0.901	0.640	0.255	0.363	0.671	0.788	0.724	0.444	0.039	0.072

TextBlock annotation performs the best on the 1c layout category. The annotation of Text elements provides the best results with an F1 score of 0.912 on average. Title and Header annotations perform much worse with F1 scores of 0.48 and 0.26 respectively. The main cause of error is a confusion between Header and Title blocks. Every Header block mislabelled as Title contained fewer than three lines, and vice versa. This suggests that more conditions are required to distinguish between small Header blocks and Titles. Finally, many Header blocks were mislabelled as Text, suggesting a lack of rules to detect them.

TextLine annotation performs the best on the 2c layout category. Here again, the annotation of Text elements provides the best results with an F1 score of 0.901 on average. Firstline annotation comes second with an average F1 score of 0.724. Title and Header annotations are also disappointing, with respectively 0.36 and 0.07 of F1 scores on average. Like in TextBlock annotation, many Title lines were mislabelled as Header. Most Title and Header lines were mislabelled as Text because of their width. This suggests either that the rules to detect these

two labels are not precise enough, or that there are not enough rules in RIPPER's rule set. There is a confusion between Text lines and Firstline: 519 of Text lines were incorrectly labelled as Firstline because they started with a capital letter. A similar amount of Firstline was mislabelled as Text because of a low-quality OCR.

Overall, we can observe that RIPPER's TextLine annotation obtains lower F1 scores than its TextBlock annotation. This result is on the opposite of the evaluation of our rule-based system. TextBlock annotation is designed as an intermediary step that should facilitate the TextLine annotation, which is the final result of the algorithm. Unlike our rule-based system however, RIPPER uses only sparsely the TextBlock annotations to obtain the TextLine annotations. This would explain its poor final results on TextLine annotation.

3.4 Evaluation of the Gradient Boosting algorithms

Table 15 shows the results for TextBlock and TextLine annotation performed by the Gradient Boosting algorithms on the test set.

Table 15: Precision, Recall and F1 score for TextBlock and TextLine annotation with Gradient Boosting

		Text			Title			Firstline			Header		
	Cat	P	R	F1	P	R	F1	P	R	F1	P	R	F1
TextBlock	1c	0.863	0.962	0.910	0.500	0.375	0.429				0.667	0.222	0.333
	2c	0.891	0.989	0.937	0.706	0.667	0.686				0.625	0.208	0.312
	3c+	0.967	0.863	0.912	0.789	0.234	0.361				0.000	0.000	0.000
	Mean	0.907	0.938	0.920	0.665	0.425	0.492				0.431	0.144	0.215
TextLine	1c	0.778	0.983	0.868	0.667	0.194	0.300	0.641	0.207	0.312	1.000	0.211	0.348
	2c	0.905	0.949	0.926	0.758	0.305	0.435	0.727	0.756	0.741	0.875	0.113	0.200
	3c+	0.856	0.950	0.901	0.941	0.264	0.413	0.667	0.483	0.560	0.000	0.000	0.000
	Mean	0.846	0.961	0.898	0.788	0.254	0.383	0.678	0.482	0.538	0.625	0.108	0.183

For TextBlock annotation, the model performs best on the 2c category. The annotation of Text elements provides the best results with a F1-score of 0.92 on average. The performance for Title annotation is average with a mean F1-score of 0.49, but with a mean Precision of 0.66. Finally, Header annotation performs the worst with a mean F1 score of 0.21 and an average Precision of 0.43. Most errors are Header blocks mislabelled as Title (23 %) and Text blocks mislabelled as Header (17 %). Most mislabelled Title and Header blocks were labelled as Text, suggesting the model only covers a few instances of both labels.

For TextLine annotation, the model also performs the best on the 2c category, except for Header annotation where it performs the best on the 1c category. As for TextBlock annotation, Text lines annotation has the best performance with a mean F1 score of 0.89 and a Recall of 0.96. Firstline annotation is average, with a mean F1 score of 0.53 and an average Precision of 0.67. Both Header and Title annotation have bad performances with a mean F1 score of 0.18 and 0.38, but an average Precision score of 0.62 and 0.78 respectively. Every mislabelled Text line was labelled as Firstline. On the other hand, most mislabelled Firstline, Title and Header lines were labelled as Text, indicating the model only covers a few instances of these labels, as for TextBlock annotation.

3.5 Final comparison between the models

Table 16 presents the mean Precision, Recall and F1 scores of the three systems for TextBlock and TextLine annotation on the test set: the rule-based system, the RIPPER algorithm and the Gradient Boosting algorithm. Figure 5 shows the average F1 scores of the three models for TextBlock and TextLine annotations.

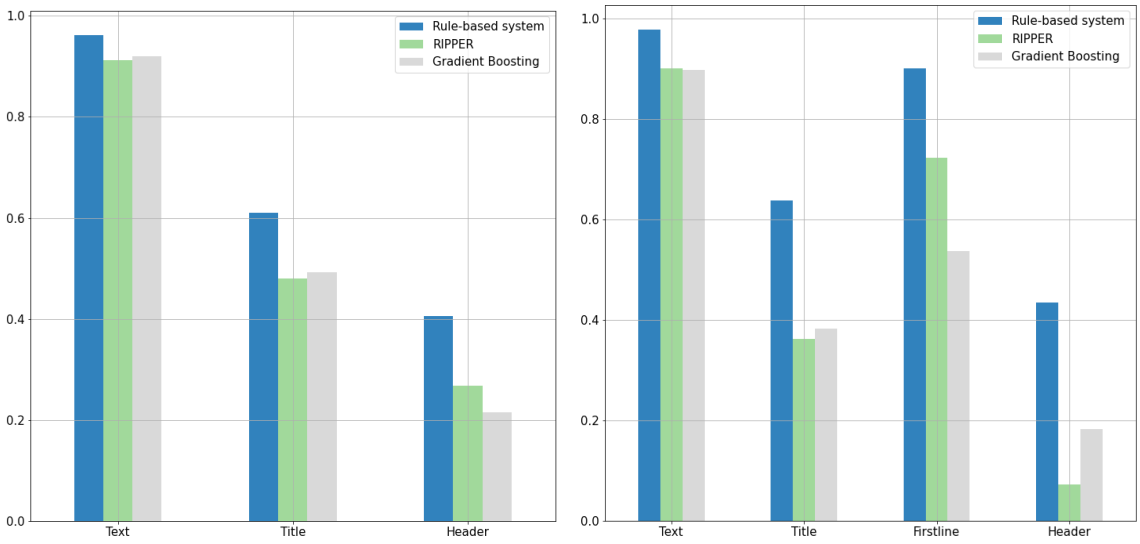
Table 16: Mean Precision, Recall and F1 scores of the three systems on the TextBlock and TextLine annotation tasks

		Text			Title			Firstline			Header		
	Cat	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
TextBlock	Rule-based	**0.959**	0.966	**0.962**	0.600	**0.639**	**0.610**				**0.726**	**0.298**	**0.406**
	RIPPER	0.841	**0.996**	0.912	0.560	0.455	0.480				0.440	0.194	0.268
	Gradient Boosting	0.907	0.938	0.920	**0.665**	0.425	0.492				0.431	0.144	0.215
TextLine	Rule-based	**0.969**	**0.991**	**0.979**	0.595	**0.733**	**0.639**	**0.949**	**0.861**	**0.902**	**0.803**	**0.348**	**0.435**
	RIPPER	0.890	0.912	0.901	0.640	0.255	0.363	0.671	0.788	0.724	0.444	0.039	0.072
	Gradient Boosting	0.846	0.961	0.898	**0.788**	0.254	0.383	0.678	0.482	0.538	0.625	0.108	0.183

All three models have good performances for Text blocks annotation, with the minimum reaching an average F1 score of 0.91. Our rule-based system has the best Precision and F1 score, whereas RIPPER has the best Recall score. The performances for Title block annotation are average for all three models. Gradient Boosting has the best mean Precision score with 0.66, while our rule-based system has the best mean Recall and F1 score. Furthermore, our system seems more stable, as Precision, Recall and F1 are nearly equal, unlike the other two models. Finally, the performances for Header block annotation go from bad to good. Every system has a much better Precision score than a Recall, suggesting that defining exhaustive rules to detect Header is a difficult task. However, our system has the best performances in Precision, Recall and F1 score once again.

Similarly to Text blocks annotation, all three models have very good performances for Text line annotation. Our rule-based system has the best results in each three metrics. The performances for Title line annotation go from bad to very good. Like Title block annotation, Gradient Boosting has the best mean Precision score. However, our system has the best mean Recall and F1 score by far. The performances for Firstline annotation go from average to very good. Our system has the best scores in every metric, followed by the RIPPER system. Surprisingly, Gradient Boosting only reaches an average F1 score of 0.53, suggesting that paragraphs are easier to detect with simple rules. Finally, the performance for Header line annotation go from very bad to very good. Similarly to TextBlock annotation, every system has a much better Precision

Figure 5: Mean F1 score of the three models for TextBlock (left) and TextLine (right) annotations



than Recall. In this case as well, our system reaches the best scores in each metric.

Our rule-based system outperforms the two other models in nearly all evaluations. It has especially better Recall results, indicating that our system covers more types of every logical label than the other two models. When comparing RIPPER with Gradient Boosting, we can observe that Gradient Boosting has better Precision scores but RIPPER has better Recall scores.

Despite its disappointing performances, RIPPER can produce very precise rules which are sometimes better than our manually crafted rules. Furthermore, RIPPER has better Precision than Recall scores on average, suggesting this algorithm is better at producing fine-grained rules than general ones. As producing rules by hand is a time-consuming task, it would be interesting in future works to first use RIPPER in an exploratory manner to produce a base rule set with high Precision scores. This rule set could then be manually updated in order to improve the performances of the rules.

Similarly, Gradient Boosting can reach very high Precision scores, as seen with Title lines annotation. Thus, it would be interesting to produce a hybrid system which would either use rules or Machine Learning algorithms to identify a specific logical label.

IV CONCLUSION

In this article, we have compared the performances of three systems for Logical Layout Analysis applied on XML ALTO: a rule-based system, the RIPPER Rule Learning algorithm and Gradient Boosting. All three of them are used in the same general pipeline: a set of features is first extracted from the document, and the system is then used to assign logical labels to TextBlock and TextLine elements.

The comparison between the performances of the three systems shows that our rule-based system outperforms the two other models in nearly all evaluations. Its higher Recall scores suggest that this system covers more types of every logical label than the other two models. The evaluation also confirms that our system can be used to produce annotated data sets that are large enough to envisage Machine Learning or deep learning approaches. However, both RIPPER and Gradient Boosting can reach very high Precision scores. Combining rules and Machine Learning models into hybrid systems could potentially provide even better performances. Thus, we plan in future works to produce such hybrid systems and evaluate them.

As stated earlier, the layout in historical documents evolves rapidly, especially in newspapers. It is then necessary to develop systems dedicated to a specific publication period. Although RIPPER's performances are disappointing, it is a valuable tool to explore a data set and quickly create a rule set, which can then be updated manually. As such, we plan in future works to use Rule Learning algorithms such as RIPPER to help creating rule sets adapted to specific publication periods.

COMPETING INTEREST

The authors declare that they have no competing interests.

LIST OF ABBREVIATIONS

OCR : Optical Character Recognition
PLA : Physical Layout Analysis
LLA : Logical Layout Analysis
HAC : Hierarchical Agglomerative Clustering

CNN : Convolutional Neural Network
LSTM : Long Short-Term Memory
MARG : Medical Articles Record Groundtruth
CRF : Conditional Random Field
RIPPER : Repeated Incremental Pruning to Produce Error Reduction
IREP : Incremental Reduced Error Pruning
FOIL : First Order Inductive Learner
DL : Description Length
SVM : Support Vector Machine

AUTHORS' CONTRIBUTIONS

All authors contributed equally to the study conception and design. Nicolas Gutehrlé carried out the background description, the creation and annotation of the data set, the construction of the rule-based system, the training of the Rule-Learning and Machine-Learning algorithm, the evaluation of the three systems and drafted the manuscript. Iana Atanassova supervised all of the above tasks and participated in improving the manuscript and the presentation of the results. All authors read and approved the final manuscript.

AUTHOR'S INFORMATION

Nicolas Gutehrlé is currently a PhD Student in Natural Language Processing at the CRIT laboratory at Université de Bourgogne Franche-Comté, under the supervision of Dr. Iana Atanassova.

Dr. Iana Atanassova is associate professor in Natural Language Processing at the CRIT laboratory at Université de Bourgogne Franche-Comté and at the Institut Universitaire de France (IUF). She supervises the EMONTAL project (Extraction and Ontology Modeling of Subjects and Places for the Exploitation of the Documentary Funds of Bourgogne Franche-Comté, 2020–2023) funded by the Région Bourgogne Franche-Comté, France.

ACKNOWLEDGMENTS

This research is supported by the Région Bourgogne Franche-Comté, France, as part of the EMONTAL project (Extraction and Ontology Modeling of Subjects and Places for the Exploitation of the Documentary Funds of Bourgogne Franche-Comté, 2020–2023).

References

Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018.

Hanna Abi Akl, Anubhav Gupta, and Dominique Mariko. FinTOC-2019 shared task: Finding title in text blocks. In *Proceedings of the Second Financial Narrative Processing Workshop (FNP 2019)*, pages 58–62, Turku, Finland, September 2019. Linköping University Electronic Press. URL https://www.aclweb.org/anthology/W19-6408.

S.M. Ayatollahi and Hossein Nafchi. Persian heritage image binarization competition (phibc 2012). pages 1–4, 03 2013. ISBN 978-1-4673-6204-7. doi: 10.1109/PRIA.2013.6528442.

Raphaël Barman, Maud Ehrmann, S. Clematide, S. Oliveira, and F. Kaplan. Combining visual and textual features for semantic segmentation of historical newspapers. *ArXiv*, abs/2002.06144, 2020.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

Marius Bulacu, Rutger van Koert, Lambert Schomaker, and Tijn van der Zant. Layout analysis of handwritten historical documents for searching the archive of the cabinet of the dutch queen. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 1, pages 357–361, 2007. doi: 10.1109/ICDAR.2007.4378732.

Kai Chen and Mathias Seuret. Convolutional neural networks for page segmentation of historical document images,

2017.

Christian Clausner, Christos Papadopoulos, Stefan Pletschacher, and Apostolos Antonacopoulos. The enp image and ground truth dataset of historical newspapers. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 931–935, 2015. doi: 10.1109/ICDAR.2015.7333898.

William W Cohen. Repeated incremental pruning to produce error reduction. In *Machine Learning Proceedings of the Twelfth International Conference ML95*, 1995.

Johannes Fürnkranz and Gerhard Widmer. Incremental reduced error pruning. In *Machine Learning Proceedings 1994*, pages 70–77. Elsevier, 1994.

Nicolas Gutehrlé and Iana Atanassova. Dataset for Logical-layout analysis on French historical newspapers, October 2021.

Marti A. Hearst. Texttiling: Segmenting text into multi-paragraph subtopic passages. *Comput. Linguist.*, 23(1): 33–64, March 1997. ISSN 0891-2017.

David Hébert, Thomas Palfray, Stéphane Nicolas, Pierrick Tranouez, and Thierry Paquet. Automatic article extraction in old newspapers digitized collections. *ACM International Conference Proceeding Series*, 05 2014. doi: 10.1145/2595188.2595195.

K. Kise, M. Iwata, and Keinosuke Matsumoto. On the application of voronoi diagrams to page segmentation. 1999.

S. Klampfl and Roman Kern. An unsupervised machine learning approach to body text and table of contents extraction from digital scientific articles. In *TPDL*, 2013.

G. Nagy, S. Seth, and M. Viswanathan. A prototype document image analysis system for technical journals. *Computer*, 25:10–22, 1992.

Anoop Namboodiri and Anil Jain. *Document Structure and Layout Analysis*, pages 29–48. 03 2007. ISBN 978-1-84628-501-1. doi: 10.1007/978-1-84628-726-8_2.

D. Niyogi and S.N. Srihari. Knowledge-based derivation of document logical structure. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 472–475 vol.1, 1995. doi: 10.1109/ICDAR.1995.599038.

L. O'Gorman. The document spectrum for page layout analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15: 1162–1173, 1993.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://aclanthology.org/D14-1162.

J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 2005.

Cartic Ramakrishnan, Abhishek Patnia, Eduard Hovy, and Gully Burns. Layout-aware text extraction from full-text pdf of scientific articles. *Source code for biology and medicine*, 7:7, 05 2012. doi: 10.1186/1751-0473-7-7.

Martin Riedl, Daniela Betz, and Sebastian Padó. Clustering-based article identification in historical newspapers. In *Proceedings of the 3rd Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 12–17, Minneapolis, USA, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-2502.

Claude Sammut and Geoffrey I. Webb, editors. *Encyclopedia of Machine Learning and Data Mining*. Springer, 2017. ISBN 978-1-4899-7685-7. doi: 10.1007/978-1-4899-7687-1.

Zejiang Shen, Kaixuan Zhang, and Melissa Dell. A large dataset of historical japanese documents with complex layouts. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2336–2343, 2020.

Fotini Simistira, Mathias Seuret, Nicole Eichenberger, A. Garz, M. Liwicki, and R. Ingold. Diva-hisdb: A precisely annotated large dataset of challenging medieval manuscripts. *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 471–476, 2016.

H. Tibbo. Primarily history in america: How u.s. historians search for primary materials at the dawn of the digital age. *American Archivist*, 66:9–50, 2007.

Xu Zhong, J. Tang, and Antonio Jimeno-Yepes. Publaynet: Largest dataset ever for document layout analysis. *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1015–1022, 2019.

Annus Zulfiqar, Adnan Ul-Hasan, and Faisal Shafait. Logical layout analysis using deep learning. In *2019 Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–5, 2019. doi: 10.1109/DICTA47822.2019.8946046.