Deep Understanding of a Document's Structure

Muhammad Mahbubur Rahman
University of Maryland, Baltimore County
Baltimore, Maryland 21250
mrahman1@umbc.edu

Tim Finin
University of Maryland, Baltimore County
Baltimore, Maryland 21250
finin@umbc.edu

Abstract

Current language understanding approaches focus on small documents, such as newswire articles, blog posts, product reviews and discussion forum discussions. Understanding and extracting information from large documents like legal briefs, proposals, technical manuals and research articles is still a challenging task. We describe a framework that can analyze a large document and help people to locate desired information in it. We aim to automatically identify and classify different sections of documents and understand their purpose within the document. A key contribution of our research is modeling and extracting the logical structure of electronic documents using machine learning techniques, including deep learning. We also make available a dataset of information about a collection of scholarly articles from the *arXiv* eprints collection that includes a wide range of metadata for each article, including a table of contents, section labels, section summarizations and more. We hope that this dataset will be a useful resource for the machine learning and language understanding communities for information retrieval, content-based question answering and language modeling tasks.

Keywords

Machine Learning; Document Structure; Natural Language Processing; Deep Learning

1 Introduction

Understanding and extracting of information from large documents such as reports, business opportunities, academic articles, medical documents and technical manuals poses challenges not present in short documents. State of the art natural language processing approaches mostly focus on short documents, such as newswire articles, email messages, blog posts, product reviews and discussion forum entries. One of the key steps in processing a large documents is sectioning it into its parts and understanding their purpose. For some large documents, this is relatively straightforward, but obtaining high precision results can be very challenging in many cases. Our initial work with collections of Requests for Proposals (RFPs) from a large range of U.S. Government agencies showed that simple approaches often failed for collections of documents that were large, complex, based on many different formats, had embedded tables, forms and lists, and lacked any useful metadata. The problems are significantly compounded for PDF documents produced by optical character recognition or lacking useful metadata.

Document understanding depends on a reader's own interpretation, where a document may structured, semi-structured or unstructured. Usually a human readable document has a physical layout and logical structure. A document contains sections. Sections may contain a title, section body or a nested structure. Sections are visually separated components by a section break such as extra spacing, one or more empty lines or a section heading for the latter section. A section break signals to a reader the changes of topic, purpose, concepts, mood, tone or emotion. The lack of proper transition from one section to another section may raise the difficulty of the reader's ability to understand the document.

Understanding large multi-themed documents presents additional challenges as these documents are composed of a variety of sections discussing diverse topics. Some documents may have a table of contents whereas others may not. Even if a table of contents is present, mapping it across the document is not a straightforward process. Section and subsection headers may or may not be present in the table of contents. If they are present, they are often inconsistent across documents even within the same vertical domain.

Most of the large documents such as business documents, health care documents and technical reports are available in PDF format. This is because of the popularity and portability of PDF-based files over different types of computers, devices and operating systems. But PDF is usually rendered by various kind of tools such as Microsoft Office, Adobe Acrobat and Open Office. All of these tools have their own rendering techniques. Moreover, content is written and formatted by people. All of these factors make PDF documents very complex with text, images, graphs and tables.

Semantic organization of sections, subsections and sub-subsections of PDF documents across all vertical domains are not the same. For example, a typical business document has a completely different structure from a user manual or a scholarly journal article. Even research articles from different disciplines, such as computer science and social science, have very different expected structures and use different terms to signal the role of elements that are similar. For example, social science articles have *methodology* sections where as computer science articles have *approach* sections. Semantically, these two sections are similar in that they both describe some important details of how the work was carried out.

We intend to section large and complex PDF documents automatically and annotate each section with a semantic and human-understandable label. Our *semantic labels* are intended to capture the general role or purpose that a document section fills in the larger document, rather than identifying any concepts that are specific to the document's domain. This does not preclude also annotating the sections with semantic labels appropriate for a specific

BDCAT'17: Big Data Computing, Applications and Technologies, December 5–8, 2017, Austin, TX, USA
 ISBN 978-1-4503-5549-0/17/12…$15.00
DOI: https://doi.org/10.1145/3148055.3148080

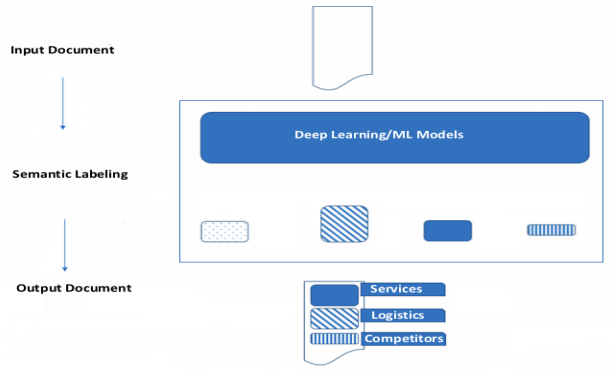


Figure 1: A High Level System Work-flow

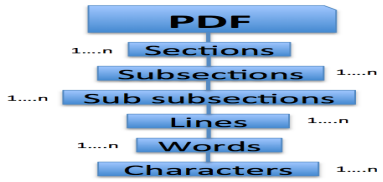


Figure 2: Logical Model of a PDF Document

class of documents (e.g., RFPs) or documents about a domain (e.g., RFPs for software services). In ongoing work we are exploring the use of embeddings, topic models and techniques to produce such annotations.

Figure 1 shows the high level system work-flow of our framework. The framework takes a document as input, extracts text, identifies logical sections and labels them with semantically meaningful names. The framework uses layout information and text content extracted from PDF documents. A logical model of a PDF document is given in Figure 2, where each document is a collection of sections and a section is a collection of subsections and so on.

Identifying a document's logical sections and organizing them into a standard structure to understand the shadow semantic structure of a document will not only help many information extraction applications but also enable users to quickly navigate to sections of interest. Such an understanding of a document's structure will significantly benefit and inform a variety of applications such as information extraction and retrieval, document categorization and clustering, document summarization, fact and relation extraction, text analysis and question answering. People are often interested in reading specific sections of a large document. It will help people simplify their reading operations as much as possible and save valuable time.

One might be confused that document sectioning and semantic labeling are the same as document segmentation [2], but these are distinct tasks. Document segmentation is based on a scanned image of a text document. Usually a document is parsed based on raw pixels generated from a binary image. We use electronic documents such as PDF versions generated from Word, LaTeX or Google Doc and consider different physical layout attributes such as indentation, line spaces and font information.

One might also confuse semantic labeling with rhetorical or coherence relations of text spans in a document. Rhetorical Structure Theory (RST) [17, 28] uses rhetorical relations to analyze text in order to describe rather than understand them. It finds coherence in texts and parses their structure. This coherence is helpful for identifying different components of a text block, but we aim to understand the text blocks in order to associate a semantic meaning.

2 Background

This section provides necessary background on our research and includes definitions required to understand the work.

2.1 Sections

A section can be defined in different ways. In our paper, we define a section as follows.
S = a set of *paragraphs*, P ; where number of paragraphs is 1 *to n*
P = a set of *lines*, L
L = a set of *words*, W
W = a set of *characters*, C
C = all character set
D = *digits* | *roman numbers* | *single character*
LI = a set of *list items*
TI = an entry from a table
Cap = *table caption* | *image caption*
B = characters are in *Bold*
LFS = characters are in *larger font size*
HLS = higher line *space*

Section Header = $l \subset L$ where l often starts with $d \in D$ **And** $l \notin$ {TI, Cap} **And** *usually* $l \in LI$ **And** generally $l \subset$ {B, LFS, HLS}

Section = $s \subset S$ followed by a *Section Header*.

2.2 Documents

Our work is focused on understanding the textual content of PDF documents that may have anywhere few pages to several hundred pages. We consider those with more than ten pages to be "large" documents. It is common for these documents to have page headers, footers, tables, images, graphics, forms and mathematical equation. Some examples of large documents are business documents, legal documents, technical reports and academic articles.

2.3 Document Segmentation

Document segmentation is a process of splitting a scanned image from a text document into text and non-text sections. A non-text section may be an image or other drawing. And a text section is a collection of machine-readable alphabets, which can be processed by an OCR system. Usually two main approaches are used in document segmentation, which are geometric segmentation and logical segmentation. According to geometric segmentation, a document is split into text and non-text based on its geometric structure. And a logical segmentation is based on its logical labels such as header, footer, logo, table and title. The text segmentation is a process of splitting digital text into words, sentences, paragraphs, topics or meaningful sections. In our research, we are splitting digital text into semantically meaningful sections with the help of geometrical attributes and text content.

2.4 Document Structure

A document's structure can be defined in different ways. In our research, documents have a hierarchical structure which is considered as the document's logical structure. According to our definition, a document has top-level sections, subsections and sub-subsections. Sections start with a section header, which is defined in the earlier part of the background section. A document also has a *semantic structure*. An academic article, for example, has an abstract followed by an introduction whereas a business document, such as an RFP, has deliverables, services and place of performance sections. In both the logical and semantic structure, each section may have more than one paragraph.

3 Related Work

Identifying the structure of a scanned text document is a well-known research problem. Some solutions are based on the analysis of the font size and text indentation [5, 18]. Song Mao et al. provide a detailed survey on physical layout and logical structure analysis of document images [18]. According to them, document style parameters such as size of and gap between characters, words and lines are used to represent document physical layout.

Algorithms used in physical layout analysis can be categorized into three types: top-down, bottom-up and hybrid approaches. Top-down algorithms start from the whole document image and iteratively split it into smaller ranges. Bottom-up algorithms start from document image pixels and cluster the pixels into connected components such as characters which are then clustered into words, lines or zones. A mix of these two approaches is the hybrid approach.

The O'Gorman's Docstrum algorithm [21], the Voronoi-diagram-based algorithm of Kise [14] and Fletcher's text string separation algorithm [10] are bottom-up algorithms. Lawrence Gorman describes the Docstrum algorithm using the K-nearest neighbors algorithm [11] for each connected component of a page and uses distance thresholds to form text lines and blocks. Kise et al. propose Voronoi-diagram-based method for document images with a non-Manhattan layout and a skew. Fletcher et al. design their algorithm for separating text components in graphics regions using Hough transform [13]. The X-Y-cut algorithm presented by Nagy et al. [20] is an example of the top-down approach based on recursively cutting the document page into smaller rectangular areas. A hybrid approach presented by Pavlidis et al. [22] identifies column gaps and groups them into column separators after horizontal smearing of black pixels.

Jean-Luc Bloechle et al. describe a geometrical method for finding blocks of text from a PDF document and restructuring the document into a structured XCDF format [4]. Their approach focuses on PDF formatted TV Schedules and multimedia meeting note, which usually are organized and well formatted. Hui Chao et al. describe an approach that automatically segments a PDF document page into different logical structure regions such as text blocks, images blocks, vector graphics blocks and compound blocks [7], but does not consider continuous pages. Hervé Déjean et al. present a system that relies solely on PDF-extracted content using table of contents (TOC) [9]. But many documents may not have a TOC. Cartic Ramakrishnan et al. develop a layout-aware PDF text extraction system to classify a block of text from the PDF version of biomedical research articles into rhetorical categories using a rule-based method [26]. Their system does not identify any logical or semantic structure for the processed document.

Alexandru Constantin et al. design PDFX, a rule-based system to reconstruct the logical structure of scholarly articles in PDF form and describe each of the sections in terms of some semantic meaning such as title, author, body text and references [8]. They get 77.45 F1 score for top-level heading identification and 74.03 F1 score for extracting individual bibliographic items. Suppawong Tuarob et al. describe an algorithm to automatically build a semantic hierarchical structure of sections for a scholarly paper [29]. Though, they get 92.38% F1 score in section boundary detection, they only detect top-level sections and settle upon few standard section heading names such as ABS (Abstract), INT (Introduction) and REL (Background and Related Work). But a document may have any number of section heading names.

Most previous work focuses on image documents, which are not similar to the problem we are trying to solve. Hence, their methods are not directly applicable to our research. Some research covers scholarly articles considering only the top-level sections without any semantic meaning. Our research focuses on any type of large document including academic articles, business documents and technical manuals. Our system understands the logical and semantic structure of any document and finds relationship between top-level sections, subsections and sub-subsections.

4 System Architecture and Approach

In this section, we describe the system architecture of our framework. We explain our approaches and algorithms in detail. We also show the input and output of our framework.

4.1 System Architecture

Our system is organized as a sequence of units, including a Pre-processing, Annotation, Classification and Semantic Annotation units, as shown in figure 3.

4.1.1 Pre-processing Unit The pre-processing unit takes PDF documents as input and gives processed data as output for annotation. It uses PDFLib [23] to extract metadata and text content from PDF documents. It has a parser, that parses XML generated by PDFLib using the XML element tree (etree). The granularity of XML is word level, which means XML generated by PDFLib from PDF document has high level descriptions of each character of a word. The parser applies different heuristics to get font information of each character such as size, weight and family. It uses x-y coordinates of each character to generate a complete line and calculates indentation and line spacing of each line. It also calculates average font size, weight and line spacing for each page. All metadata including text for each line is written in a CSV file where each row has information and text of a line.

4.1.2 Annotation Unit The Annotation Unit takes layout information and text as input from the Pre-processing Unit as a CSV file. Our annotation team reads each line, finds it in the original PDF document and annotates it as a *section-header* or *regular-text*. While annotating, annotators do not look into the layout information given in the CSV file. For our experiments on *arXiv* articles, we extract bookmarks from PDF document and use them as gold

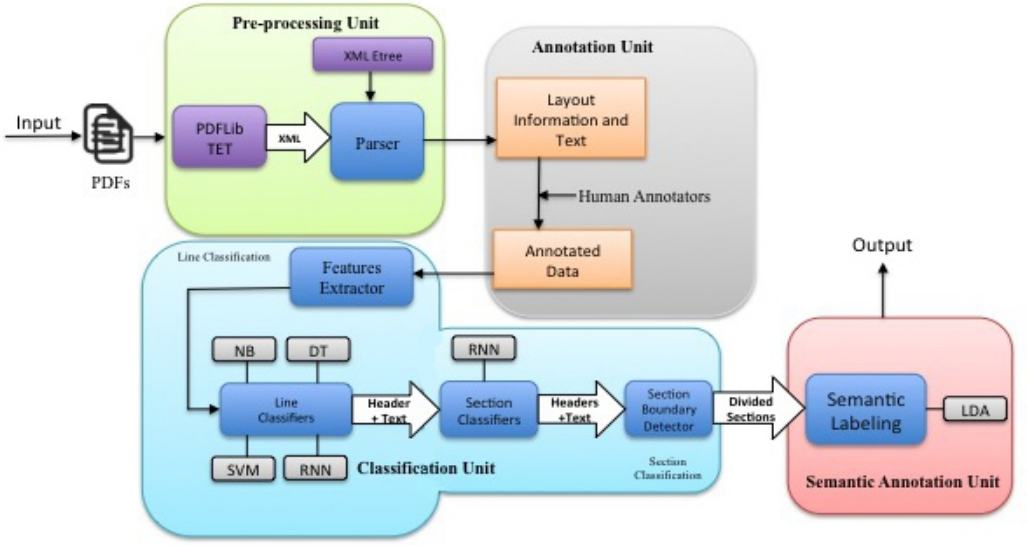


Figure 3: A High Level System Architecture

standard annotation for training and testing as described in the experiments section.

4.1.3 Classification Unit The Classification Unit takes annotated data and trains classifiers to identify physically divided sections. The Unit has sub-units for line and section classification. The Line Classification sub unit has Features Extractor and Line Classifiers module. The Features Extractor takes layout information and text as input. Based on heuristics, it extracts features from layout information and text. Features include text length, number of noun phrases, font size, higher line space, bold italic, colon and number sequence at the beginning of a line. The Line Classifiers module implements multiple classifiers using well known algorithms such as Support Vector Machines (SVM), Decision Tree (DT), Naive Bayes (NB) and Recurrent Neural Networks (RNN) as explained in the Approach section. The output of the Line Classifiers module are *section-header* or *regular-text*. The classified section header may be *top-level*, *subsection* or *sub-subsection* header. The Section Classifiers module of the Section Classification sub unit takes section headers as input and classifies them as *top-level*, *subsection* or *sub-subsection* header using RNN. The Section Classification sub unit also has a Section Boundary Detector which detects the boundary of a section using different level of section headers and regular text. It generates physically divided sections and finds relationship among *top-level*, *subsection* and *sub-subsection*. It also generates a TOC from a document based on the relationship among different levels of sections, as explained further in the Approach section.

4.1.4 Semantic Annotation Unit The Semantic Annotation Unit annotates each physically divided section with a semantic name. It has a Semantic Labeling module, which implements Latent Dirichlet Allocation(LDA) topic modeling algorithm to get a semantic concept from each of the sections and annotates each

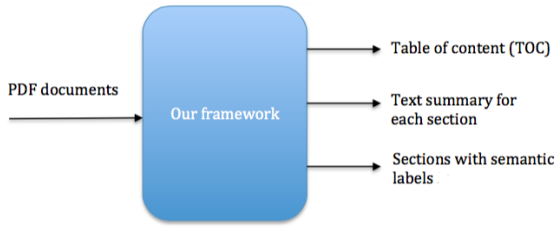


Figure 4: Overall input and output of our framework

section with a semantic concept understandable to people. It also applies document summarization technique using NTLK to generate a short summary for each individual section. The output are a TOC, semantic labels and a summary from each PDF document. The overall input and output of our framework are shown in figure 4.

4.2 Approach

In this section, we present powerful, yet simple approaches to build classifiers and models using layout information and text content from PDF documents in detail.

4.2.1 Line Classification The Line Classification unit identifies each line of text as a *section-header* or *regular-text*. We explain our approaches for the Line Classification below.

Features Extractor Given a collection of labeled text and layout information on a line, the Features Extractor applies different heuristics to extract features. We build a vocabulary from all section headers of *arXiv* training data, where a word is considered if the frequency of that word is more than 100 and is not a common English word. The vocabulary size is 13371 and the top five words

Table 1: Human generated features

Feature name	pos_nnp, without_verb_higher_line_space, font_weight, bold_italic, at_least_3_lines_upper, higher_line_space, number_dot, text_len_group, seq_number, colon, header_0, header_1, header_2, title_case, all_upper, voc

are "Introduction", "References", "Proof", "Appendix" and "Conclusions". The Features Extractor calculates average font size, font weight, line spacing and line indentation. It finds number of dot, sequence number, length of the text, presence of vocabulary and case of words (title case and upper case) in the text. It also generates lexical features such as the number of Noun or Noun Phrase, verb and adjective. It is common that a section header should have more Noun or Noun Phrases than other parts of speech. The ratio of verbs or auxiliary verbs should be much less in a section header. A section header usually starts with a numeric or Roman number or a single English alphabet letter. Based on all these heuristics, the Features Extractor generates 16 features from each line. These features are given in table 1. We also use the n-gram model to generate unigram, bigram and trigram features from the text. After features generation, the Line Classifiers module uses SVM, DT, NB and RNN to identify a line as a *section-header* or *regular-text*.

Support Vector Machines(SVM) Our line classification task can be considered as a text classification task where input are the layout features and n-gram from the text. Given a training data set with labels, we can train SVM models which learn a decision boundary to split the dataset into two groups by constructing a hyperplane or a set of hyperplanes in a high dimensional space. Suppose, our training dataset, $T = \{x_1,\ x_2, ..., x_n\}$ of text lines and their label set, $L = \{0, 1\}$ where 0 means *regular-text* and 1 means *section-header*. Each of the data points from T is either a vector of 16 layout features or a vector of 16 layout features concatenated with n-gram features generated from text using $TF - IDF\ vectorizer$. Using SVM, we can determine a classification model as equation 1 to map a new line with a class label from L.

$$f : T \rightarrow L \qquad f(x) = L \tag{1}$$

Here the classification rule, the function $f(x)$ can be of different types based on the chosen kernels and optimization techniques. We use LinearSVC from scikit-learn [24] which implements Support Vector Classification for the case of a linear kernel presented by Chih-Chung Chang et al. [6]. As our line classification task has only two class labels, we use linear kernel. We experiment with different parameter configurations for both the combine features vector and only the layout features vector. The detail of the SVM experiment is presented in the Experiments section.

Decision Tree(DT) Given a set of text lines, $T = \{x_1,\ x_2, ..., x_n\}$ and each line of text, x_i is labeled with a class name from the label set, $L = \{0, 1\}$, we train a decision tree model that predicts the class label for a text line, x_i by learning simple decision rules inferred from either 16 $layout features$ or 16 $layout features$ concatenated with a number of n-gram *features* generated from the text using $TF - IDF\ vectorizer$. The model recursively partitions all text lines such that the lines with the same class labels are grouped together.

To select the most important feature which is the most relevant to the classification process at each node, we calculate the $gini - index$. Let $p_1(f)$ and $p_2(f)$ be the fraction of class label presence of two classes 0: *regular-text* and 1: *section-header* for a feature f. Then, we have equation 2.

$$\sum_{i=1}^{2} p_i(f) = 1 \tag{2}$$

Then, the $gini - index$ for the feature f is in equation 3.

$$G(f) = \sum_{i=1}^{2} p_i(f)^2 \tag{3}$$

For our two class line classification task, the value of $G(f)$ is always in the range of (1/2,1). If the value of $G(f)$ is high, it indicates a higher discriminative power of the feature f at a certain node.

We use decision tree implementation from scikit-learn [24] to train a decision tree model for our line classification. The experimental results are explained in the Experiments section.

Naive Bayes(NB) Given a dependent feature vector set, $F = \{f_1,\ f_2, ..., f_n\}$ for each line of text from a set of text lines, $T = \{x_1,\ x_2, ..., x_n\}$ and a class label set, $L = \{0, 1\}$, we can calculate the probability of each class c_i from L using the Bayes theorem states in equation 4.

$$P(c_i|F) = \frac{P(c_i)\ .\ P(F|c_i)}{P(F)} \tag{4}$$

As $P(F)$ is the same for the given input text, we can determine the class label of a text line having feature vector set F, using the equation 5.

$$\left.\begin{aligned} Label(F) &= arg\ Max_{c_i}\{P(c_i|F)\} \\ &= arg\ Max_{c_i}\{P(c_i)\ .\ P(F|c_i)\} \end{aligned}\right\} \tag{5}$$

Here, the probability $P(F|c_i)$ is calculated using the multinomial Naive Bayes method. We use multinomial Naive Bayes method from scikit-learn [24] to train models, where the feature vector, F is either 16 features from layout or 16 layout features concatenated with the word vector of the text line.

Recurrent Neural Networks(RNN) Given an input sequence, $S = \{s_1,\ s_2, ..., s_t\}$ of a line of text, we train a character level RNN model to predict it's label, l ∈ L= {*regular-text* :0, *section-header* :1}. We use a many-to-one RNN approach, which reads a sequence of characters until it gets the *end of the sequence* character. It then predicts the class label of the sequence. The RNN model takes the embeddings of characters in the text sequence as input. For character embedding, we represent the sequence into a character level one-hot matrix, which is given as input to the RNN network. It is able to process the sequence recursively by applying a transition function to it's hidden unit, h_t. The activation of the hidden unit is computed by the equation 6.

$$h_t = \begin{cases} 0 & t = 0 \\ f(h_{t-1}, s_t) & otherwise \end{cases} \tag{6}$$

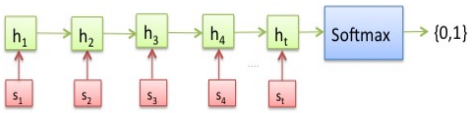


Figure 5: Many-to-one RNN approach for line classification

where h_t and h_{t-1} are the hidden units at time t and $t-1$ and s_t is the input sequence from the text line at time t. The RNN maps the whole sequence of characters until the *end of the sequence* character with a continuous vector, which is input to the $softmax$ layer for label classification. A many-to-one RNN architecture for our line classification is shown in figure 5.

We use TensorFlow [1] to build our RNN models. We build three different networks for our line classification task. In the first and second networks, we use only text and layout as input sequence respectively. In the third network, we use both 16 layout features and the text as input, where the one-hot matrix of characters sequence is concatenated at the end of the layout features vector. Finally, the whole vector is given as input to the network. Figure 6 shows the complete network architecture for layout and text input. The implementation detail is given in the Experiments section.

4.2.2 Section Classification The section classification module identifies different levels of section headers such as *top-level section*, *subsection* and *sub-subsection* headers. It also detects section boundaries. It has Section Classifiers module and Section Boundary Detector component, which are explained below.

Section Classifiers Like as the Line Classifiers module, the Section Classifiers module considers the section classification task as a prediction modeling problem where we have sequence of inputs $S = \{s_1, s_2, ..., s_t\}$ from a classified section header and the task is to predict a category from $L = \{$ *top-level section header*:1, *subsection header*:2 *sub-subsection header*:3$\}$ for the sequence. For this sequence prediction task, we use an RNN architecture similar to the architecture used for the line classification. The differences are input sequence and the class labels. The input and output of RNN for this task is shown in figure 7.

Section Boundary Detector After identifying different level section headers, we merge all contents (regular text, top-level section header, subsection header and sub-subsection header) with their class labels in a sequential order as they appear in the original document. The Section Boundary Detector splits the whole document into different sections, subsection and sub-subsections based on the given splitting level. By default, it splits the document into top-level sections. It returns output as a dictionary where the keys are text, title and subsections for each section. The subsection has the similar nested structure. The Section Boundary Detector finds the relationship among sections, subsections and sub-subsections using the dependency state diagram presented in figure 8. The high level algorithm to generate sections, subsections and sub-subsections using the dependency diagram and class labels is presented in algorithm 1.

4.2.3 Semantic Annotation Given a set of physically divided sections $D = \{d_1, d_2, ..., d_n\}$, the semantic annotation module assigns a human understandable semantic name to each section. We use Latent Dirichlet Allocation (LDA) [3] to find a semantic concept from a section. LDA is a generative topic model, which is used to understand the hidden structure of a collection of documents. In LDA, each document has a mixture of various topics with a probability distribution. Again, each topic is a distribution of words.

Using Gensim [27], we train an LDA topic model on a set of divided sections. The model is used to predict the topic for any test section and we select several terms having the highest probability values of the predicted topic are used to annotate the section as a semantic label. Using the Section Boundary Detector from Section Classification sub unit, the Semantic Annotation module generates a table of contents for any PDF document. It also summarizes each section of a document using the textrank algorithm [19] implemented in NLTK [16], where sections are detected by the Section Boundary Detector.

Algorithm 1 Section boundary detector

```
procedure SPLIT_DOC_INTO_SECTIONS(doc, split_level)
    sections =[]
    if split_level is top_level then
        for line in doc do
            Generate text_block based on class_label = 1
            Add {title, text_block} in sections
    else if split_level is subsection then
        for line in doc do
            Generate text_block based on class_label =1
        for block in text_block do
            Generate sub_block based on class_label =2
            Add {title, sub_block} in sections
    else
        for line in doc do
            Generate text_block based on class_label =1
        for block in text_block do
            Generate sub_block based on class_label =2
        for block in sub_block do
            Generate sub_sub_block based on class_label =3
            Add {title, sub_sub_block} in sections
    return sections
```

5 Experiments and Evaluation of Results

We evaluated the effectiveness of our approaches using scientific articles from arXiv Library [15] repository. This section describes data, experiments and evaluation of our results.

5.1 Data Construction

5.1.1 Data Collection We downloaded all *arXiv* articles from Amazon $S3$ cloud storage using arXiv Bulk Data Access option uploaded by arXiv for the time period of 2010 to 2016 December. The files were grouped into .tar files of $\sim 500MB$ each. The total size of all files is $743.4GB$. After downloading, we extracted all tar files and got 1121363 articles in PDF. Using open archives initiative protocol [12], we harvested metadata for each of the articles from the *arXiv* repository. The metadata includes title, publication date, abstract, categories and author names. Some of the *arXiv* articles have bookmarks. We also extracted bookmarks from each article.

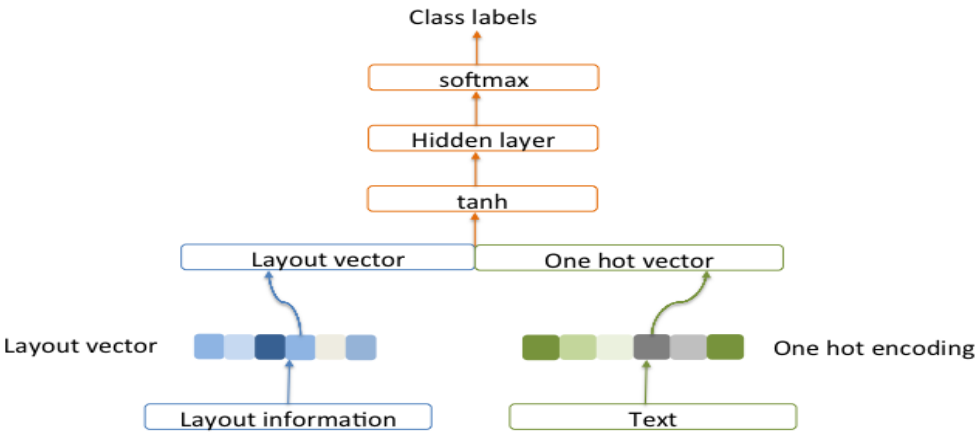


Figure 6: RNN architecture for layout and text

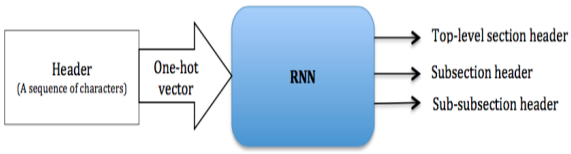


Figure 7: Input-output for section classification

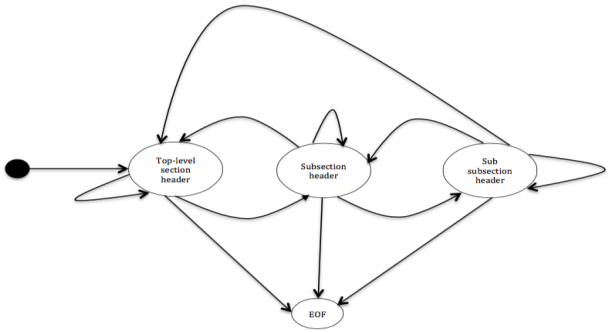


Figure 8: Top-level section, subsection and sub-subsection header dependency sequence

We kept the hierarchy in the bookmarks. We considered bookmarks as the table of contents (TOC). We combined metadata, the TOC and a downloadable link for each article and stored in a JSON file where *arXiv* file name is the key for each set of information.

5.1.2 Data Processing We converted each PDF article to an XML dialect called TETML (Text Extraction Toolkit Markup Language) using PDFLib. The granularity of the conversion was word level. After conversion, the total size of all TETML files was 5.1TB. The elements are organized in a hierarchical order in a TETML file. Each TETML file contains pages. Each page has annotation and content elements. The content element has all of the text blocks in a page as a list of para elements. Each para element has a list of words where each word contains a high level description of each character such as font name, size, weight, x-y coordinates and character width. Our parser reads the structure of the TETML file and parses it. The parser processes a description of each character and generates text lines and layout information from the description for each line by applying different heuristics. The layout information are the starting and ending of x and y positions of a line, font size, font weight, font-family, page number, page width and page height. It returns all lines of text with layout information.

5.1.3 Training and Test Data For our experiments on arXiv articles, we have a component, which processes bookmarks and each TETML file. After getting all lines of text with layout information from the parser, the component traverses the TOC for each file and maps each element of the TOC with text lines from the document. It finds a path for each element of the TOC and defines a class label for each line based on the mapping between the TOC element and text line. The class labels are regular-text:0, top-level section header:1, subsection header:2 and sub-subsection header:3. Finally, we generated a dataset in a CSV format where each row has text line, layout information, file name of that line and class label of that line. This dataset is used as gold standard data for our experiments. We took 60% as training and 40% as test out of 1121363 articles which have tables of contents sections. Our developed models identify sections and the TOCs for the rest of the data.

5.2 Experiment for Line Classification

As explained in the approach section, we used SVM, Decision Tree, Naive Bayes and RNN classifiers for our line classification. Table 2 shows the configurations of our classifiers. As a document has very few section headers with respect to regular text, our data is highly imbalanced and some of the layout features depend on the sequence of lines. After generating features, we balanced our dataset. We considered an equal number of samples for all the classes. As the *arXiv* dataset is very large, we only took a part of

Table 2: Classifiers configurations

SVM	DT	NB	RNN
kernel='linear'	criterion = 'gini'		max_doc_len = 100
regularization = 'l2'	algorithm = 'CART'	algorithm ='MultinomialNB'	hidden_size = 20
features = 'layout', 'layout and text'	features = 'layout', 'layout and text'	features = 'layout', 'layout and text'	encoding = 'one-hot'
vectorizer = TF-IDF vectorizer	vectorizer = TF-IDF vectorizer	vectorizer = TF-IDF vectorizer	optimizer = 'adam'
ngram= unigram, bigram and trigram	ngram= unigram, bigram and trigram	ngram= unigram, bigram and trigram	learning_rate =0.001
minimum doc frequency = 5%	minimum doc frequency = 5%	minimum doc frequency = 5%	function = 'Softmax'
maximum doc frequency = 95%	maximum doc frequency = 95%	maximum doc frequency = 95%	batch_size = 10

Table 3: Training and Test Data for Line Classification

	Training Data	Test Data
Regular-Text	121077	80184
Section-Header	121077	80184
Top-level Section Header	208430	166744
Subsection Header	208430	166744
Sub-subsection Header	208430	166744

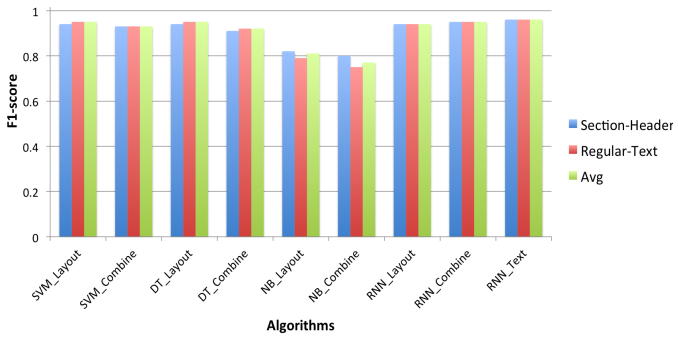


Figure 9: Performance Comparison for line classification

Table 4: For both Layout and Combine Features

Algorithms		Layout Features			Combine Features		
		Precision	Recall	F1 Score	Precision	Recall	F1 Score
SVM	Section-Header	0.97	0.92	0.94	0.93	0.92	0.93
	Regular-Text	0.93	0.97	0.95	0.92	0.93	0.93
DT	Section-Header	0.97	0.92	0.94	0.96	0.87	0.91
	Regular-Text	0.92	0.97	0.95	0.88	0.97	0.92
NB	Section-Header	0.76	0.90	0.82	0.73	0.89	0.80
	Regular-Text	0.88	0.72	0.79	0.85	0.67	0.75
RNN	Section-Header	0.94	0.94	0.94	0.95	0.95	0.95
	Regular-Text	0.94	0.94	0.94	0.95	0.95	0.95

Table 5: Text only using RNN for Line Classification

	Precision	Recall	F1 Score
Section-Header	0.97	0.96	0.96
Regular-Text	0.95	0.97	0.96

Table 6: For Section Classification using RNN

	Precision	Recall	F1 Score
Top-level Section Header	0.83	0.88	0.85
Subsection Header	0.81	0.81	0.81
Sub-subsection Header	0.78	0.73	0.75
Avg	0.81	0.81	0.81

the dataset to train and test our models. Table 3 shows the training and test dataset size for our experiments.

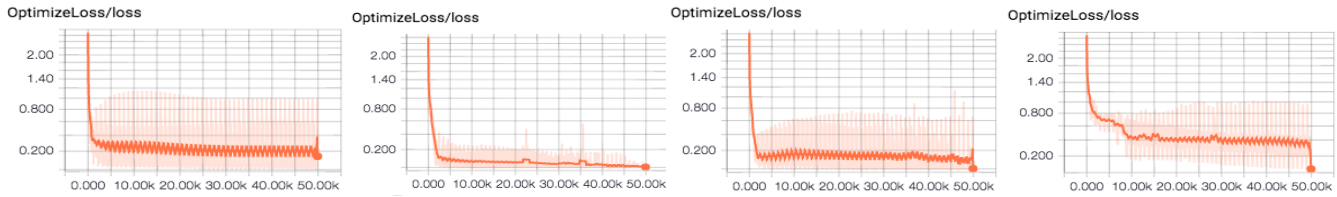
To evaluate our models, we used precision, recall and f-measure. Table 4 shows precision, recall and f1 scores for all of our approaches on the test dataset. We also trained a character level RNN model using only the text. Precision, recall and f1 scores for this model are shown in table 5. Figure 9 compares f1 scores for all of the algorithms we used for line classification. We achieved the best performance with character level RNN using only text as input. Figure 10a, 10b and 10c show the training losses over the number of steps for RNN with layout, text and combine input respectively where we got minimum loss for text input.

5.3 Experiment for Section Classification

We achieved the best result for the line classification using an RNN model. The reason is character level RNN model is able to learn varieties in the input sequence and automatically captures significant features. Analyzing of different levels of section headers such as top-level, subsection and sub-subsection, implies that RNN model works better when input sequence has varieties. So, we chose RNN model for the section classification. We also prepared a training and test dataset for section classification task. Table 3 shows the size of training and test datasets for section classification. Precision, recall and F1 scores for section classification are shown in table 6. From figure 10d, we can see that the training loss is higher in section classification than line classification. It is obvious that identifying *top-level*, *subsection* and *sub-subsection* headers are more complex than just identifying *section-header* or *regular-text*.

5.4 Experiment for Semantic Annotation

We trained an LDA model on 128505 divided sections through 50 passes for a different number of topics and evaluated the model on 11633 divided sections. While building the dictionary for the model, we ignored words that appear in less than 20 sections or more than 10% of all the sections. Our final dictionary size, after filtering, was 100000. Figure 14 shows inter topic distance map for ten topics where some of the topics overlap. This figure also shows the 30 most relevant terms for topic 4 where the relevance score is 80%. To annotate a section, we used the model to get the best topic for that section and chose a couple of terms with the highest probability. An example is shown in figure 11. To evaluate the LDA



(a) Layout **(b) Text** **(c) Combine** **(d) Sections**

Figure 10: Training Loss

```
Section
In this paper we discussed two possible ways to integrate 5G and LTE networks in order to improve the reliability of next generation mobile networks. We also presented the
implementation of a simulation framework that can be used to assess the performance of such systems, integrated in ns3,and showed that the level of detail of the simulation
thatcan be carried out with such a tool makes it possible to understand and evaluate which is the best solution among dual connectivity with switching and hard handover. We
showed some early results, for a particular choice of parameters, as an example of a possible simulation output. A more detailed description of the new software modules and
a more comprehensive set of preliminary results can be found in [13]. The application of the proposed framework to extensive simulation campaigns to fully characterize
performance trends and to gain key insights for system design is left for future work.

Topic
[(u'network', 0.0040563542608222309), (u'performance', 0.0031216212264198119), (u'error', 0.0029698130817764298), (u'optimal', 0.0026800514211577971), (u'power',
0.0023464491230957333), (u'channel', 0.0023200744860318368), (u'average', 0.0021789235410437004), (u'input', 0.0021466871428290238), (u'test', 0.0020625125401258155),
(u'control', 0.0020352192406846966)]
```

Figure 11: Semantic Annotation using top terms from LDA topic

```
[{'children': [], 'title': '1. Introduction'}, {'children': [], 'title': '2. Bayesian optimization'}, {'children'
: [], 'title': '3. Turbulent channel drag reduction'}, {'children': [], 'title': '4. Turbine blade shape design'}
, {'children': [], 'title': '5. Conclusion'}] link: https://arxiv.org/pdf/1410.8859.pdf
```

Figure 12: Only top-level section headers

```
[{'children': [], 'title': '1. Introduction'}, {'children': [], 'title': '2. Climate-Weathering Models'}, {'chil
dren': [{'children': [], 'title': '3.1. Steady-State Solutions'}, {'children': [], 'title': '3.2. Climate Cycles
'}], 'title': '3. Climate Solutions'}, {'children': [], 'title': '4. Conclusions'}, {'children': [], 'title': 'A
ppendix A. Energy Balance Climate Model.'}, {'children': [], 'title': 'Appendix B. Model Simplifications and Lim
itations.'}] link: https://arxiv.org/pdf/1411.5564.pdf
```

Figure 13: Top-level and subsection headers

model for sections, we considered perplexity and cosine similarity measures. The perplexity for a test chunk is -9.684 for tn topics. In our experiment, the perplexity is lower in magnitude, which means that the LDA model fits better for the test sections and probability distribution is good at predicting the sections. We split the test set into ten different chunks of test sections where each chunk has 1000 sections without repetition. We also split each section from each test chunk into two parts and checked two measures. The first measure is a similarity between topics of the first half and topics of the second half for the same section. The second measure is a similarity between halves of two different sections. We calculated an average cosine similarity between parts for each test chunk of sections. Due to the coherence between topics, the first measure should be higher and the second measure should be lower. Figure 15 shows these two measures for ten different chunk of test sections. We also generated TOCs from any scholarly article. Figure 12 and 13 show the TOCs from two different articles where each TOC represents the hierarchies of different section headers.

5.5 Comparison of Results and Discussion

We compared the performance of our framework in the previous sections with respect to different performance matrices. We also compared the performance of our framework against the top performing systems for scholarly articles in PDF form. The first comparison system is PDFX presented by Alexandru Constantin et al. in [8]. Our task is formalized in a different way and partially similar to their task. Their system identifies author, title, email, section headers etc. from scholarly articles. They reported an f1 score of 77.45% for top-level section headers identifying for a various articles. The dataset is not publicly available. We achieved an 85% f1 score for top-level section headers identifying along with a 96% f1 score for just section header identifying from *arXiv* repository which has various types of academic articles from thousands of different categories and subcategories. The second comparison system is a hybrid approach to discover semantic hierarchical sections from scholarly documents by Suppawong Tuarob et al. [29]. Their task is limited to a few fixed section heading name identifications whereas our framework can identify any heading name. Their dataset is not directly applicable to our system, but it is on scholarly articles. They

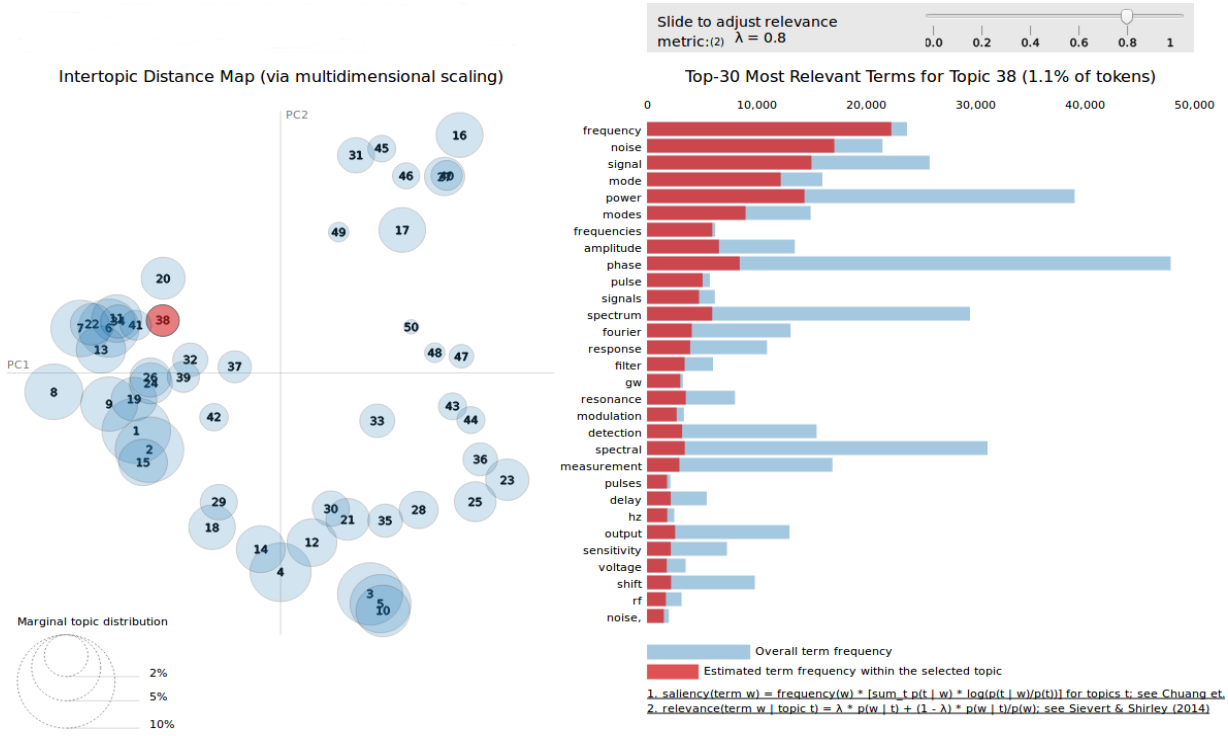


Figure 14: Inter topic distance map and top terms for a topic

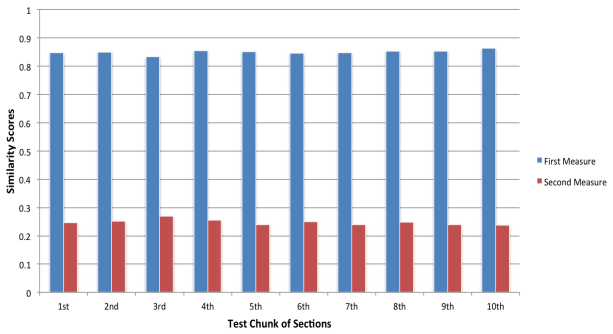


Figure 15: Similarity measures for LDA

got a 92.38% f1 score for section boundary detection where sections are of any level(from fixed names such as abstract, introduction and conclusions) and we got a 96% f1 score for any heading name identification. We also tried our framework on business documents such as a Request for Proposal (RFP) dataset collected from a startup company that works on business documents analysis. RFPs are usually large, complex and very unstructured documents. Due to the terms and conditions given by the company, we are not able to present results and that dataset in this research paper.

As we use PDFLib for PDF extraction, we depend on their system performance. Due to the different encoding of PDF documents, sometimes PDFLib divides the same block into two different blocks. This generates an error in our data when we map bookmarks in the original PDF for training and test data generation. To reduce this error, we used th SequenceMatcher function in Python's difflib module to calculate string similarity score. If the score is more than a threshold, we map the bookmark entry with a line of text from the original PDF. Due to the use of similarity score and threshold heuristic, we may still miss a few section headers. But the ratio is very low. We expect to overcome this error completely in our future work.

A complete dataset [25] is available with metadata including a table of contents, section labels, section summarizations, publication history, author names and downloadable *arXiv* link for each article from 1986 to 2016.

6 Conclusions and Future work

We presented a novel framework to understand academic scholarly articles by automatically identifying and classifying sections and labeling them with semantic topics. We applied different machine learning approaches. We also contributed to the community by releasing a large dataset from scholarly articles. For future work, we plan to develop an ontology to map semantic topics with standard names. We are also interested in developing a deep learning summarization technique for individual section summarization.

7 Acknowledgments

The work presented in this paper was partially supported by a grant number 1549697 from the National Science Foundation(NSF).

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, and others. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[2] Apostolos Antonacopoulos, Christian Clausner, Christos Papadopoulos, and Stefan Pletschacher. 2013. Icdar 2013 competition on historical newspaper layout analysis (hnla 2013). In *2013 12th International Conference on Document Analysis and Recognition*. IEEE, 1454–1458.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.

[4] Jean-Luc Bloechle, Maurizio Rigamonti, Karim Hadjar, Denis Lalanne, and Rolf Ingold. 2006. XCDF: a canonical and structured document format. In *International Workshop on Document Analysis Systems*. Springer, 141–152.

[5] Dan S Bloomberg and Francine R Chen. 1996. Document image summarization without OCR. In *Image Processing, 1996. Proceedings., International Conference on*, Vol. 1. IEEE, 229–232.

[6] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, 3 (2011), 27.

[7] Hui Chao and Jian Fan. 2004. Layout and content extraction for pdf documents. In *International Workshop on Document Analysis Systems*. Springer, 213–224.

[8] Alexandru Constantin, Steve Pettifer, and Andrei Voronkov. 2013. PDFX: fully-automated PDF-to-XML conversion of scientific literature. In *Proceedings of the 2013 ACM symposium on Document engineering*. ACM, 177–180.

[9] Hervé Déjean and Jean-Luc Meunier. 2006. A system for converting PDF documents into structured XML format. In *International Workshop on Document Analysis Systems*. Springer, 129–140.

[10] Lloyd A. Fletcher and Rangachar Kasturi. 1988. A robust algorithm for text string separation from mixed text/graphics images. *IEEE transactions on pattern analysis and machine intelligence* 10, 6 (1988), 910–918.

[11] Keinosuke Fukunaga and Patrenahalli M. Narendra. 1975. A branch and bound algorithm for computing k-nearest neighbors. *IEEE transactions on computers* 100, 7 (1975), 750–753.

[12] Open Archives Initiative. 2017. OAI Protocol. (2017). http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm

[13] Nahum Kiryati, Yuval Eldar, and Alfred M Bruckstein. 1991. A probabilistic Hough transform. *Pattern recognition* 24, 4 (1991), 303–316.

[14] Koichi Kise, Akinori Sato, and Motoi Iwata. 1998. Segmentation of page images using the area Voronoi diagram. *Computer Vision and Image Understanding* 70, 3 (1998), 370–382.

[15] Cornell University Library. 2017. arXiv e-print service. (2017). https://arxiv.org

[16] Edward Loper and Steven Bird. 2002. NLTK: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*. Association for Computational Linguistics, 63–70.

[17] William C Mann and Sandra A Thompson. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text-Interdisciplinary Journal for the Study of Discourse* 8, 3 (1988), 243–281.

[18] Song Mao, Azriel Rosenfeld, and Tapas Kanungo. 2003. Document structure analysis algorithms: a literature survey. In *Electronic Imaging 2003*. International Society for Optics and Photonics, 197–207.

[19] Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing order into texts. Association for Computational Linguistics.

[20] George Nagy, Sharad Seth, and Mahesh Viswanathan. 1992. A prototype document image analysis system for technical journals. *Computer* 25, 7 (1992), 10–22.

[21] Lawrence O'Gorman. 1993. The document spectrum for page layout analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15, 11 (1993), 1162–1173.

[22] Theo Pavlidis and Jiangying Zhou. 1992. Page segmentation and classification. *CVGIP: Graphical models and image processing* 54, 6 (1992), 484–496.

[23] PDFlib 2017. *PDFlib Text and Image Extraction Toolkit(TET)*. PDFlib. https://www.pdflib.com/products/tet/.

[24] 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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[25] Muhammad Rahman. 2017. Structural Metadata from ArXiv Articles. http://ebiquity.umbc.edu/resource/html/id/374. (September 2017).

[26] Cartic Ramakrishnan, Abhishek Patnia, Eduard Hovy, and Gully APC Burns. 2012. Layout-aware text extraction from full-text PDF of scientific articles. *Source code for biology and medicine* 7, 1 (2012), 1.

[27] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, 45–50. http://is.muni.cz/publication/884893/en.

[28] Maite Taboada and William C Mann. 2006. Rhetorical structure theory: Looking back and moving ahead. *Discourse studies* 8, 3 (2006), 423–459.

[29] Suppawong Tuarob, Prasenjit Mitra, and C Lee Giles. 2015. A hybrid approach to discover semantic hierarchical sections in scholarly documents. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*. IEEE, 1081–1085.