APPROVAL SHEET

Title of Dissertation: Understanding the Logical and Semantic Structure of Large Documents

Name of Candidate: Muhammad Mahbubur Rahman
Computer Science, 2018

Dissertation and Abstract Approved: _____

Tim Finin, PhD
Professor
Department of Computer Science and Electrical Engineering

Date Approved: _____

ABSTRACT

Title of Dissertation: Understanding the Logical and Semantic Structure of Large Documents

Muhammad Mahbubur Rahman
Doctor of Philosophy, 2018

Dissertation directed by: Prof. Tim Finin
Department of Computer Science and Electrical Engineering

Current language understanding approaches are mostly focused on small documents, such as newswire articles, blog posts, and product reviews. Understanding and extracting information from large documents like legal documents, reports, proposals, technical manuals, and research articles is still a challenging task. Because the documents may be multi-themed, complex, and cover diverse topics. The content can be split into multiple files or aggregated into one large file. As a result, the content of the whole document may have different structures and formats. Furthermore, the information is expressed in different forms, such as paragraphs, headers, tables, images, mathematical equations, or a nested combination of these structures.

Identifying a document's logical sections and organizing them into a standard structure to understand the 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 facilitate a variety of applications, such as information extraction, document summarization, and question answering.

We intend to section large and complex PDF documents automatically and annotate each section with a semantic, human-understandable label. Our *semantic labels* are intended to capture the general purpose and domain specific semantic in the large document. In a nutshell, we aim to automatically identify and classify semantic sections of documents and assign human-understandable, consistent labels to them.

We developed powerful, yet simple, approaches to build our framework using layout information and text contents extracted from documents, such as scholarly articles and RFP documents. The framework has four units: *Pre-processing* Unit, *Annotation* Unit, *Classification* Unit and *Semantic Annotation* Unit. We developed state-of-the-art machine learning and deep learning architectures. We also explored and experimented with the *Latent Dirichlet Allocation (LDA)*, *TextRank* and *Tensorflow Textsum* models for semantic concept identification and document summarization respectively. We mapped each of the sections with a semantic name using a document ontology.

We aimed to develop a generic and domain independent framework. We used scholarly articles from the *arXiv* repository and RFP documents from *RedShred*. We evaluated the performance of our framework using different evaluation matrices, such as precision, recall, and f1-score. We also analyzed and visualized the results in the embedding space. We made available a dataset of information about a collection of scholarly articles from the *arXiv* eprints that includes a wide range of metadata for each article, including a TOC, section labels, section summarizations, and more.

Understanding the Logical and Semantic Structure of Large Documents

by

Muhammad Mahbubur Rahman

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2018

Advisory Committee:
Tim Finin, PhD, Chair
Anupam Joshi, PhD
Tim Oates, PhD
Cynthia Matuszek, PhD
James Mayfield, PhD

To Dad and late Mom;

for their unwavering support, love and understanding

Acknowledgments

First and foremost I would like to express my deepest gratitude to my advisor Dr. Tim Finin for his consistent support, encouragement, careful guidance, and thoughtful input over these years. His sense of pragmatism and relentless empiricism have been well appreciated, and has availed my research enormously. Under his careful supervision, I have been able to pursue interesting research topics, and conduct the corresponding projects in this dissertation.

I would also like to thank my dissertation committee members: Drs. Joshi, Oates, Matuszek and Mayfield for their insightful comments and encouragement. Their critical inputs were vital for my research and helped me improve this dissertation.

I am grateful to Jim and Jeehye from RedShred for supporting me a lot to understand the research problem and providing the annotated data. It has been a great pleasure to work with the past and present members of Ebiquity Research Group at UMBC. Some of my friends like Prajit, Jenn, Abhay, Clare, Lisa, Ramin, Taneeya, Srishty, Arya, Ankur, Sandeep and Sudip have provided critical feedback on my work, for years. I am thankful to them for listening my speeches and presentations and providing insightful comments and feedback.

I would also like to thank Drs. Syed and Gupta for providing ideas and working with me over the years. Being a non-native speaker of a language comes with quirks that one might find difficult to avoid while writing. Therefore, I am extremely thankful to Dr. Mahmood for proof-reading this dissertation. He gave insightful and critical inputs, which helped me a lot to improve this dissertation.

I am also very thankful to the all staffs of the CSEE department for their hard work, constant support and kind words. In particular, I would love to thank Olivia, Keara and Dee Ann.

This dissertation would not have been completed without the persistent support of my family and friends. Thanks to all of them, and finally to my loving parents and sisters, who continue to encourage me to pursue my dreams.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

LSTM	Long Short Term Memory networks
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
TF	Tensorflow
BOW	Bag-of-words
IDF	Inverse Document Frequency
TF	Term Frequency
TF-IDF	Term Frequency-Inverse Document Frequency
SVM	Support Vector Machine
OWL	Web Ontology Language
RF	Random Forest
NB	Naive Bayes
DT	Decision Tree
RFP	Request for Proposal
OAI	Open Archives Initiative

Chapter 1

INTRODUCTION

This thesis presents the logical and semantic processing of large electronic documents in PDF format. The logical and semantic processing of a PDF document is a non-trivial research challenge due to characteristics and purposes of PDF documents. PDF documents are rendered in an optimized way for displaying and printing the content in a better visual representation, which makes it difficult for a machine to understand the content.

This introductory chapter describes the research problem along with the elements that motivate this work. It also describes necessary background knowledge, which is followed by a state of the art approach in this research domain. Then it has the thesis statement and contributions. At the end, it includes the overall structure of the thesis.

1.1 Motivation

The 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 news articles, dialogs, blog posts, product reviews and discussion forum entries. One of the key challenges in processing of large documents is sectioning different parts of a doc-

ument. The reason behind this challenge is that large documents are complex and may be unstructured and noisy with different formats.

Document understanding depends on a reader's own interpretation. A document can be structured, semi-structured or unstructured. Usually a human readable document has a physical layout and logical structure. Such documents contain title and sections. Sections may contain a header, section body or a nested structure. Sections are visually separated by a section break such as extra space, empty line or a section heading for the latter section. A section break gives indication to a reader regarding changes of concept, mood, tone and emotion. The lack of proper transition from one to another section may make the document more difficult to understand.

Understanding large multi-themed documents presents additional challenges since 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 large documents, such as business documents, academic articles and technical reports, are available in PDF format. This is because of the popularity and the portability of the PDF file over different types of machines. 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, contents are written and

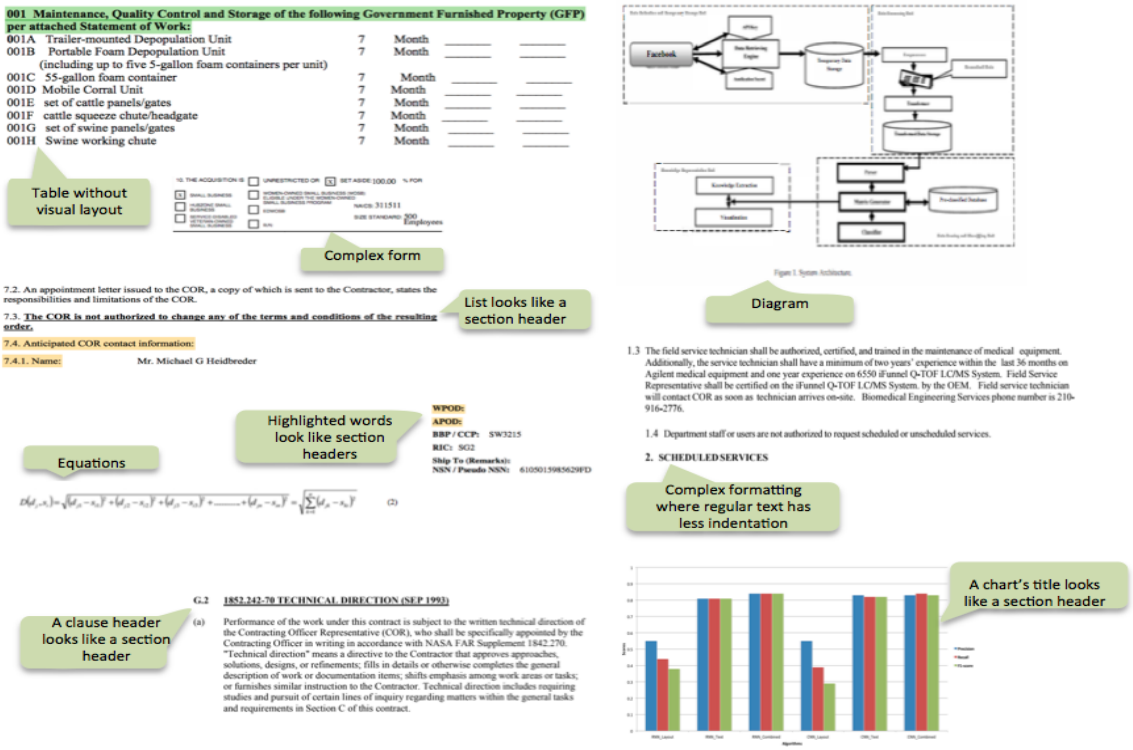


Figure 1.1: A Complex PDF View © 2012, IEEE
(Bubbles indicate different components, such as Tables, Diagrams, Equations and Forms)

formatted by human beings. All of these factors make PDF documents very complex with text, images, graphs, equations and tables. Figure **??** shows a sample of complex PDF view.

Semantic organization of sections, subsections and sub-subsections of PDF documents across all vertical domains is not the same. For example, a business document has a completely different structure from a user manual. Even research articles from computer science and social science have completely different structures. Social science articles have methodology sections where as computer science articles have approach sections. Semantically these two sections should be the same.

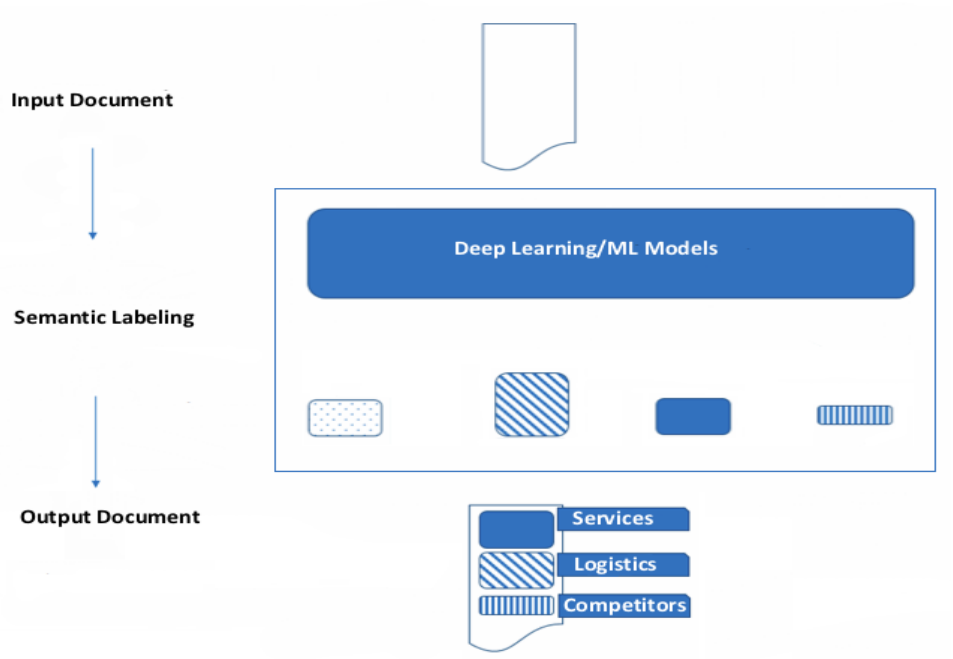


Figure 1.2: A High Level System Work-flow

We intend to sectionalize large and complex PDF documents automatically and annotate each section with a semantic and human-understandable label. Figure 1.2 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 1.3, where each document is a collection of n sections and a section is a collection of n subsections and so on.

Identifying a document's logical sections and organizing them into a standard structure, to understand the 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

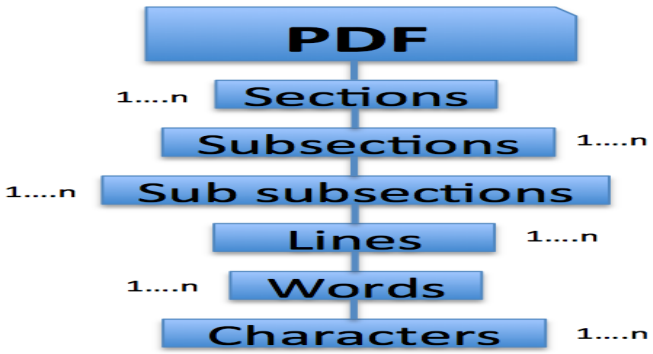


Figure 1.3: Logical Model of a PDF Document

and inform a variety of applications, such as information extraction and retrieval, document categorization and clustering, document summarization, text analysis and content based question answering. People are often interested in reading specific sections of a large document and hence will find semantically labeled sections very useful. It will help people to simplify their reading operations as much as possible and to save valuable time.

1.2 Background

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

1.2.1 Sections

A section can be defined in different ways. In this research, we define a section as follows.

$S =$ a sequence of *paragraphs*, P ; where number of paragraphs ranges from 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 preceded by a *Section Header*.

1.2.2 Documents

Our work is focused on understanding the textual content of PDF documents that may have a few pages to a few hundred pages. We consider those with more than ten pages to be "large" documents. It is common for such large documents to have page headers, footers, tables, images, graphics, forms and mathematical equations. Some ex-

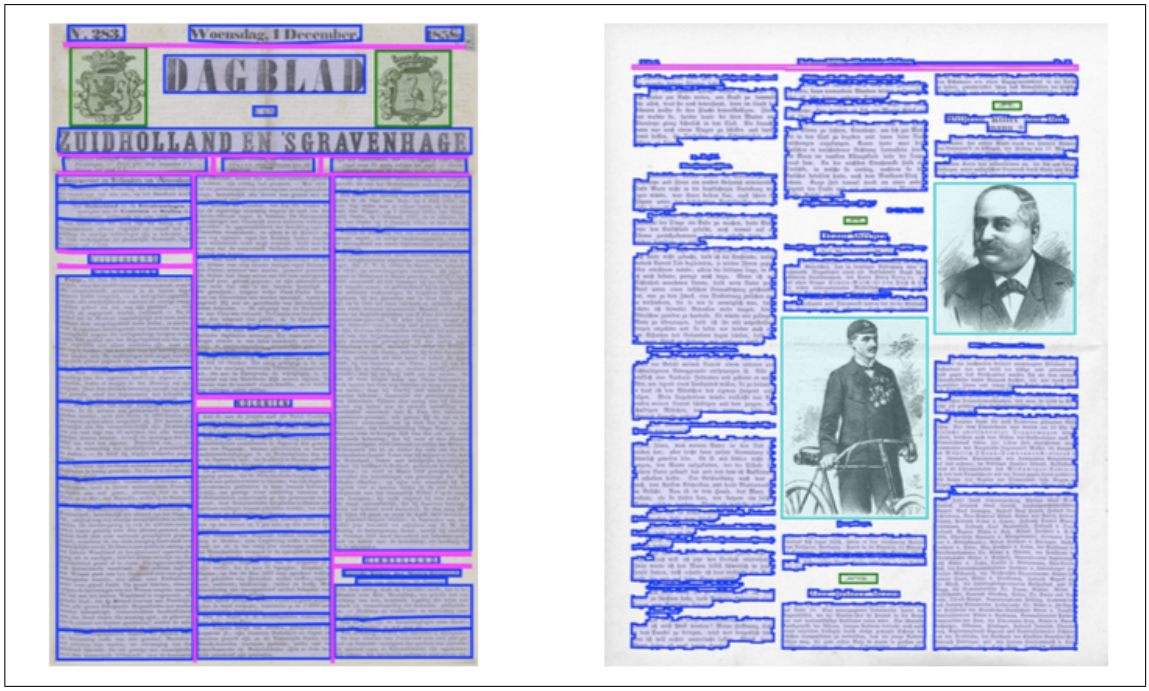


Figure 1.4: Geometric Segmentation of a Document © 2013, IEEE
(Texts are highlighted in blue, separators in magenta, graphics in green, images in cyan)

amples of large documents are business documents, legal documents, technical reports and academic articles.

1.2.3 Document Segmentation

Document segmentation is the process of splitting a scanned image of 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, geometric segmentation and logical segmentation. According to geo-

metric segmentation, a document is split into text and non-text based on its geometric structure. Figure **??** from Antonacopoulos et al. [6] shows geometric segmentation of a document. This type of segmentation can be done using top-down, bottom-up and hybrid approaches [60]. A logical segmentation is based on its logical labels, such as header, footer, logo, table and title. After reading [46, 47], a logical segmentation of an academic article [77] is drawn in Figure 1.5.

1.2.4 Text Segmentation

Text segmentation is a process of splitting digital text into words, sentences, paragraphs, topics or meaningful sections. This task differs from document segmentation and requires splitting text into meaningful sections, which is a non-trivial challenge. If the text is large, such as a document of 10 to few hundred pages, segmenting the text into meaningful semantic sections becomes more challenging. Reasons behind this challenge are large documents may be multi-themed and may cover diverse topics.

One might be confused that document sectioning and semantic labeling are the same as document segmentation [6], 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 PDFs 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) [59, 91] uses rhetori-

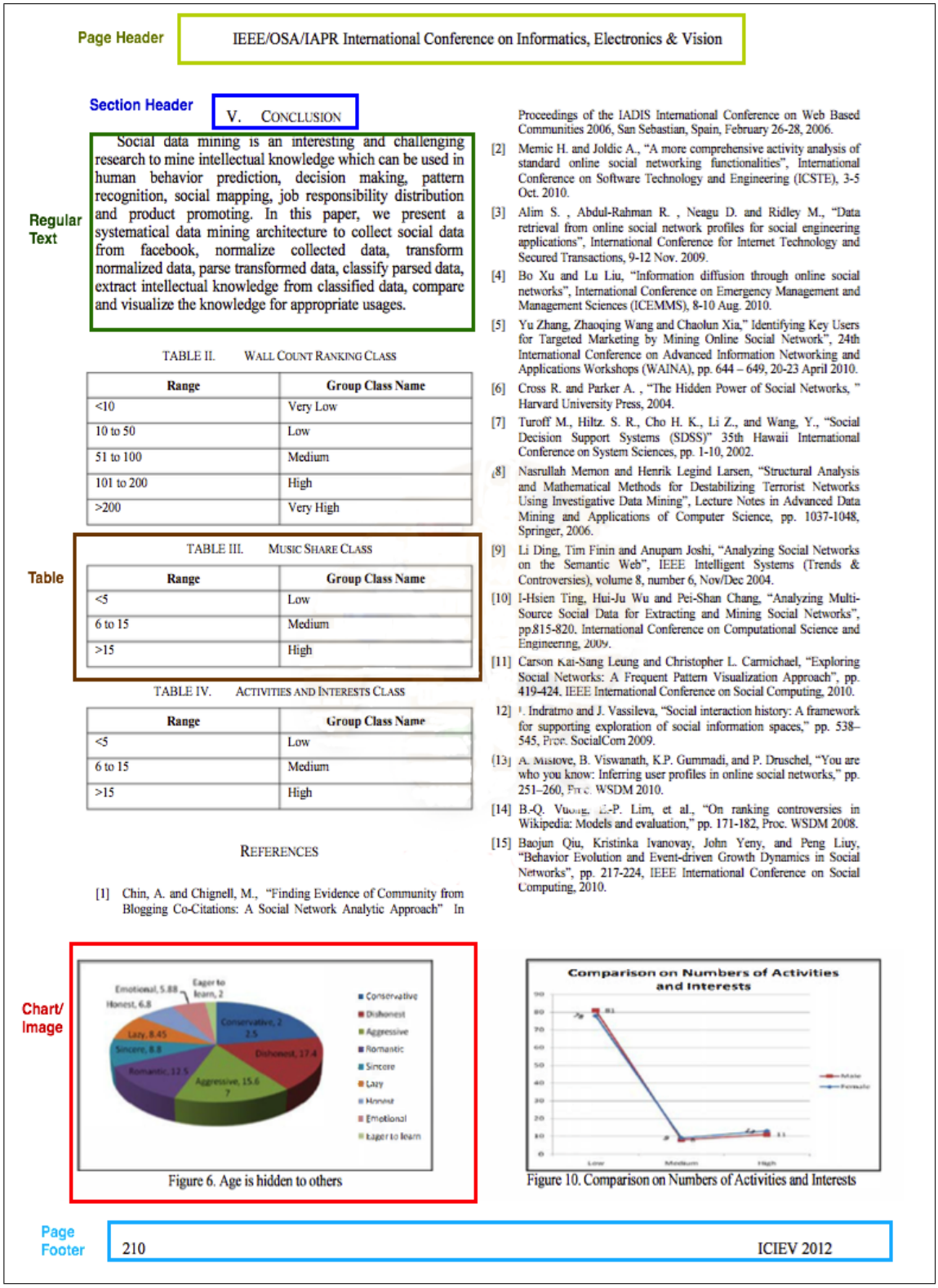


Figure 1.5: Logical Segmentation of a Document© 2012, IEEE

(boxes with different labels represent different logical components)

cal 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. Here, identifying is a process of recognizing components from text block, such as contrast and background. Understanding is a technique of finding semantic meaning for different text blocks, such as introduction and related work from an academic article.

1.3 State of the Art Approach

Several state of art methods have been proposed to tackle the problem. Constantin et al. developed a system called PDFX [22]. It is a rule-based system which reconstructs the logical structure of scholarly articles in PDF. It also describes each of the sections in terms of some semantic meaning, such as title, author, body text and references. There are few major limitations in this system. First of all, this system focuses on only selective top level headings. Secondly, this is a rule based system and considers only the logical structure. Thirdly, the system doesn't present any machine learning approach to cover any type of section headings.

Another system is presented by Tuarob et al. [92] to discover semantic hierarchical sections from scholarly documents. This system is also limited to a few fixed section headings identification. For classification algorithms, they only use Random Forest, Support Vector Machines (SVM), Repeated Incremental Pruning to Produce Error Reduction (RIPPER) and Naive Bayes. These are simple machine learning approaches to

recognize some standard sections.

For both systems, approaches are simple and they capture only limited section headings. If a document is large and complex, these simple approaches won't work well to learn all the top level, subsection and sub-subsection headings. To the best of my knowledge, none of the systems can identify the nested level of section boundaries. We also didn't find any system that annotates different sections of a document with human understandable semantic names.

To the best of my knowledge, there is no existing system, which understands different logical parts of a PDF formatted electronic document with a semantic meaning. Therefore, sectionalizing a large electronic document to identify the logical structure and annotating each of the sections with semantic names is still an open research problem. There is also a lack of general architecture, which capable of understanding the logical and the semantic structure of any sort of PDF document, such as academic articles and business documents.

Along with the traditional machine learning approaches, we also explore RNN/LSTM, CNN and seq2seq auto-encoder based deep learning models in extracting and understanding the logical and the semantic structure of PDF documents with the help of a document ontology.

1.4 Research Challenges and Contributions

Our thesis statement: it is possible to automatically identify a document's logical sections, infer their structure and assign human understandable and consistent semantic

labels that can help machines to understand large documents. The thesis statement along with the description above motivates two research challenges. These are given below.

- Understanding the logical structure of a document.
- Understanding the semantic structure of a document.

These two challenges provide substantial scientific and engineering contributions to understand a document. A general interest in document understanding is to identify a logical structure, and segment the document based on the structure. This structure is very important to split a document, so that different segments can be used to analysis the content for various purposes. The structure provides necessary information to a machine for understanding the general flow of the content. The structure also allows automatic document classification.

Another independent interest in text analysis is to understand the semantic meaning of a text segmentation, and hence improve the understanding capability of a machine in searching, information retrieval, text summarization, and question answering. Understanding the semantic structure of a sequence of sections in a document provides very useful insight in document indexing.

We solve two research challenges mentioned above and have the following contributions.

(i) Inferring the logical structure for a PDF document by a deep understanding of a low level representation of the document.

(ii) Modeling and extracting the semantic structure of a document by providing general purpose semantic and domain specific semantic concepts.

(iii) Making available a dataset of information about a collection of scholarly articles from the arXiv eprints collection that includes a wide range of meta-data for each article, including a table of contents, section labels, section summarizations and more.

(iv) Releasing the source code of different components of the system.

In the first contribution, we design and develop deep learning architectures to identify a document's logical structure from a low level representation of the document. Initially, the models generate text lines from the low level representation of the document and classify each of the lines into *regular-text* or *section-header*. Then, the models identify each *section-header* into different level of headings. Based on the output, our models generate logical sections and are able to find the relationships among them. Our deep learning models are reusable for any text classification tasks based on character level input sequences. The logical structure generated using our models, can segment any document into physically divided sections.

In the context of text summarization, the main concept is to find the most informative subset of data from the whole text where a key challenge is the length of the text. If state-of-the-art techniques of text summarization are applied on a large document, it is quite difficult to capture the most informative summaries over a variety of ideas from different parts of the document. Our models are able to capture section-wise summaries after splitting the document into different physically divided sections.

The models are applicable to a variety of document domains having document formatting, such as section headers, indentation and line spacing. Our models are also able to generate logical structure from a document of any language since people use document style parameters to format the content in most of the languages.

In the second contribution, we design and develop both character and word level deep learning architectures for understanding general purpose semantics of a document. The models are able to capture semantic patterns from the word representation of any document. We also design and develop a document ontology using deep learning. The ontology is able to describe different functional parts of a document, which can be used to enhance semantic indexing for a better understanding by human beings and machines. The ontology development architectures are adequate for any document type since we use auto-encoder and clustering approaches which are unsupervised techniques. We also build LDA based models to capture domain specific semantics from any document. The models are directly application to any academic article and are reproducible to any other document types.

The third contribution is releasing a dataset containing metadata, TOC, article categories, section summarizations and more from over one million *arXiv* articles. In the context of language modeling and topic modeling, the machine learning and NLP communities need to have a large number of clean sections or documents. Our dataset is very useful resource for these communities. The dataset can also be resourceful for content-based question answering, document indexing, and document similarity vector generation.

Finally, we make available the source code of different deep learning architectures of our system. Some of the source code can be used for any text classification and semantic concept identification. We also release the document ontology, which can be used for semantic indexing and labeling.

1.5 Structure of the Dissertation

This thesis has been structured into 6 chapters, including this introductory chapter. The rest of the chapters are as follows.

Chapter 2 narrates the previous work related to our research. It includes the necessary background knowledge to understand this thesis. It also explains the gaps in the existing work in this research domain.

Chapter 3 presents the technical approaches. It starts with a description of the developed framework. It clearly states each of the units of our system. It describes the detailed technical procedures of each of the tasks.

Chapter 4 illustrates input document processing. It explains different steps which are taken to attain clean data. It also discusses the difficulties we have faced during document processing. It demonstrates the training and test data preparation for each of the units. It also describes the annotation methods which were used to develop gold standard for our experiments.

Chapter 5 presents the experiments we have done through out this research. It narrates each of the models which different experimental setup. The results of our experiments are evaluated in this chapter. The comparative analysis among existing research,

baseline experiments and advanced experiments are also discussed in this chapter.

Chapter 6 concludes the thesis with giving limitations of the system along with some future research directions.

Finally, the last chapter includes the bibliographies and references of the existing research.

Chapter 2

BACKGROUND AND RELATED WORK

In this chapter, we will provide background knowledge to understand the research domain. We will also discuss related research work in this domain. At the end of this chapter, we will describe gaps in the existing research.

2.1 Background

2.1.1 Business Documents

A document that contains information related to business, such as business plan, goal, customer service and accounting, is known as a business document. Based on intended purposes, business documents can be of different semantic types. In our initial experiments, we used Request for Proposal (RFP) documents. An RFP is a document that solicits a proposal, often made through a bidding process, by an agency or company interested in procurement of a commodity, service, or valuable asset, potential suppliers to submit business proposals [23]. An RFP presents preliminary requirements for the commodity or the service. It often includes specifications of the item, project, or service for which a proposal is requested.

A Request for Quotation (RFQ) [85] is similar to an RFP, whereby the customer may simply be looking for a price quote. Another common format of business docu-

ments is Request for Information (RFI) [37] , which is also similar to an RFP, where a customer asks for more information from vendors before submitting an RFP.

2.1.2 Scholarly Articles

Documents that represent academic research, experiments, and results are known as academic or scholarly articles. These types of documents usually present research findings, analysis, comparisons, mathematical equations, graphics and tables. They usually have some standard sections, such as abstract, introduction, related works, approach and results. But, they may also have different standards based on the academic categories. For example, a scholarly article from computer science usually has an approach section whereas, social science has a methodology section. Semantically, these two sections are similar.

2.1.3 PDF Documents

Based on the visual representation of contents, documents may have different formats. In our research, we will use Portable Document Format (PDF) [11] documents. PDF is a file format which is commonly used to present documents in a manner independent of application software, hardware, or operating systems. It is an excellent visualization for presenting any kind of document in a platform-independent way for human readers. It is a standardized format, readable on various devices from mobile phones to personal computers.

However, automatic post processing of contents of a PDF document is not an easy

task, since PDF is essentially a graphical format. Texts in PDF are represented by text elements in page content streams. A text element specifies that the characters to be drawn in certain positions. The characters are specified using the encoding of a selected font resource. The PDF format allows numerous equally valid ways of producing the same visual result and therefore no structure can reliably be derived from how the text operators are used.

2.1.4 Ontology

According to Tom Gruber, an ontology is a specification of a conceptualization [36]. It describes a concept with the help of an instance, class and properties. It can be used to capture semantic meaning of different domains and annotate information. We need an ontology to understand the semantic structure of a document and reuse the structure in other documents. Detailed information of our ontology is given in the Technical Approach chapter.

2.1.5 Deep Learning

Deep Learning is a subfield of machine learning that allows computational models to learn important features from a large volume of data automatically. According to Lecun et al. [51], it discovers the complex structure in a large data set by using the back propagation algorithm [50]. A machine uses this structure to learn the important features by changing internal parameters from layer to layer in a neural network [27]. Some of the most popular deep learning algorithms are Convolutional Neural Networks

(CNN) [49], Recurrent Neural Networks (RNN) [62] and Long Short-Term Memory (LSTM) [39].

2.1.6 Semantic Annotation

Semantic annotation [93] can be described as a technique of enhancing a document with automatic annotations, which provides a human-understandable way to get the semantic meaning of an unstructured document. It also describes the document in such a way that the document is understandable to a machine.

2.2 Related Work

2.2.1 Document Sectioning

2.2.1.1 Scanned Document Segmentation

Identifying the structure of a scanned text document is a well-known research problem. Some solutions are proposed based on the analysis of the font size and text indentation [14, 60]. Mao et al. provided a detailed survey of physical layout and logical structure analysis of document images [60]. According to them, document style parameters, such as sizes of and gap between characters, words and lines, have been used to represent the physical document layout.

Algorithms used in physical layout analysis can be categorized into three main types: top-down approaches, bottom-up approaches 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 a hybrid approach.

The Docstrum algorithm of O'Gorman [69], the Voronoi-Diagram-based algorithm of Kise et al. [45] and the Text String Separation algorithm of Fletcher et al. [32] are bottom-up approaches. Gorman et al. described the Docstrum algorithm using the K-nearest neighbors algorithm [33] for each connected component of a page and used distance thresholds to form text lines and blocks. Kise et al. proposed the Voronoi-diagram-based method for document images with a non-Manhattan layout and a skew. This method is based on the connected component analysis. It uses an approximated area Voronoi diagram [89] to represent the neighborhood of connected components. Fletcher et al. designed an algorithm for separating text components in graphic regions irrespective of their orientation. This approach is based on the Hough transform [44].

The X-Y-cut based algorithm presented by Nagy et al. [64] is an example of the top-down approach. The X-Y-cut is a basic segmentation technique which recursively cuts the document page into smaller rectangular areas. The cuts are decided based on document pixels. The algorithm works only on a document that has a Manhattan layout [84]. In a Manhattan layout, the text, graphics and other details can be separated by horizontal and vertical line segments.

Pavlidis et al. [72] presented a hybrid approach. Their method identifies column gaps and groups them into column separators after the horizontal smearing of black pixels. A model-based logical structure analysis algorithm was presented by Yamashita

et al. [97]. It extracts character strings, lines and half-tone images [42] from a document image. Based on the extracted elements, it can detect vertical and horizontal field separators. Then it assigns labels to character strings.

2.2.1.2 Electronic Document Segmentation

Bloechle et al. described a method for finding blocks of text in a PDF document and restructuring the document into a structured XCDF format [13]. Initially, their approach trims all text primitives in order to remove extra white spaces. Then, it creates a layer for each text rotation and processes them individually in a horizontal position.

To get a basic text segment, all text primitives are merged horizontally using a dynamic distance threshold. The text segment is tokenized into words, numbers, punctuation signs and other textual primitives and then merged horizontally into lines. The lines are merged vertically into blocks by using a dynamic distance threshold. The idea of using a separate layer for each text rotation is useful, since it allows the process to concentrate only on basic left-to-right and up-to-down cases. This is a geometrical approach and focuses on PDF formatted TV Schedules, newspapers and multimedia meeting notes. Usually these types of documents are organized and have good formatting.

Chao et al. described an approach that automatically segments a PDF document page into different logical structured regions, such as text blocks, images blocks, vector graphics blocks and compound blocks [16]. Initially they converted a PDF page into different logical objects until there were no more compound objects on the page display

list. Then they formed new text-only, image-only and path-objects-only PDF documents while preserving the order of each object in the original objects list.

For text segmentation, they applied a bottom up approach starting with the smallest text: characters in PDF documents. Words were formed using Adobe's word-finder [70, 9]. Lines were constructed based on the alignment, style of the words and the distance between them. Text segments were formed based on the text line adjacency, style, and line gap. This approach works based on page outlines and doesn't consider continuous pages.

Déjean et al. presented a system that relies solely on PDF-extracted content [26]. They developed a word reconstruction component to extract text from PDF. They applied different heuristics, such as distance between two characters and geometrical positions of characters, to produce word and line segmentation. They also build a lexicon to correct any ill-formed words, but their lexicon is based on mainly technical documents. For segmenting text into paragraphs, an X-Y-cut algorithm [64] was used. In order to extract the logical structure of a document, they first detected the table of contents (TOC) from a document. Then they mapped the document according to the hierarchical information present in its TOC. However, many documents may not have a TOC.

Ramakrishnan et al. developed a layout-aware PDF text extraction system to enable accurate extraction of sections or bodies of text from PDF versions of research articles [80]. They used different heuristics to extract text blocks from the PDF and a rule-based method to classify these blocks into "rhetorical" categories (such as "Introduction", "Results" and "Discussion"). In their approach, the rules are specified by

a user for each different journal layout. Since different journals have their own styles and layout, their approach is not generic and may not be applicable to documents with various formatting. This is one of the limitations of their approach. Another limitation is that it may not capture varieties of categories, since their approach is rule-based.

2.2.2 Semantic Annotation and Labeling

2.2.2.1 Semantic Structure of Academic Articles

Constantin et al. designed PDFX, a rule-based system to reconstruct the logical structure of scholarly articles in PDF form and to describe each of the sections in terms of some semantic meaning, such as title, author, body text, or references [22]. The rule set relies on relative parameters derived from font and layout information of each article, rather than on a template-matching paradigm. PDFX has a two-stage process. In the first stage, it constructs a geometrical model of a PDF document to determine the spatial organization of textual and graphical units on a page. In the second stage, it identifies different logical units of discourse based on their discriminative features. They used the Utopia Documents PDF reader [7] to construct the geometrical model. They used the geometrical model, document-wise and page-wise statistics to determine the semantic roles of the newly created blocks.

Tuarob et al. described an algorithm to automatically build a semantic hierarchical structure of sections for a scholarly paper [92]. They defined a section as a pair of the section header and its textual content. They proposed a rule-based approach to recognize sections from scholarly articles. They applied a simple set of heuristics that

built a hierarchy of sections from the extracted section headers.

2.2.2.2 Semantic Annotation of Medical Documents

Monti et al. developed a system to reconstruct an electronic medical document with semantic annotation [63]. They divided the whole process into three steps. In the first step, they classified documents in one of the document categories specified in the Consolidated CDA (C-CDA) standard [29]. Examples of CDA templates are consultation notes, discharge summaries and operative notes. They used the Naive Bayes [53] classifier for the document classification.

They used PDFBox [95] to extract text from CDA standard medical documents. Later, they split the document into paragraphs by using the typographical features available in the PDF file. Finally, they identified key concepts from the document and mapped them to the most appropriate medical ontology. They didn't present any technical detail in their paper. They also didn't show any results. They only presented the concept of the solution. It is important to note that they only considered standard medical documents, which may not have complex document structure.

2.2.3 Deep Neural Networks

Zhang et al. applied temporal ConvNets [52] to understand text from character-level inputs all the way up to abstract text concepts [98]. Their ConvNets do not require any knowledge on the syntactic or semantic structure of a language to give good text understanding. They designed one large and one small ConvNet. Both of them are 9 layers

deep with 6 convolutional layers and 3 fully-connected layers, with different numbers of hidden units and frame sizes. They used an alphabet that consists of 70 characters, including 26 English letters, 10 digits, new line, and 33 other characters. They applied their models to DBpedia ontology classification, Amazon review sentiment analysis, Yahoo answers classification, and news categorization.

Ghosh et al. proposed a Contextual Long short-term memory (CLSTM) [39] for sentence topic prediction [34]. They developed a model to predict topic or intent of the next sentence, given the words and the topic of the current sentence. In their experiments with topic features, they considered supervised topic categories obtained from an extraneous source named Hierarchical Topic Model (HTM) [35] and also ”thought embeddings” that are intrinsically generated from the previous context. Their CLSTM is only for predicting the topic of the next sentence. It can be an interesting experiment to predict the topic for the next section, given the sentences and the topic of the current section.

Lopyrev et al. trained an encoder-decoder RNN with LSTM for generating news headlines using the texts of news articles from the Gigaword dataset [57]. They only used the first 50 words of a news article. Their approach works well on Gigaword datasets only. A bi-directional RNN can be used for better results on other texts.

Srivastava et al. introduced a type of Deep Boltzmann Machine (DBM) for extracting distributed semantic representations from a large unstructured collection of documents [88]. They presented a two hidden layer DBM model, which they called the Over-Replicated Softmax model. They used the Over-Replicated Softmax model

for document retrieval and classification. However, the model works well on short documents only.

2.2.4 Document Ontology Design

Over the last few years, several ontologies have been developed to understand a document's semantic structure and annotate it with a semantic name. Some of them deal with academic articles and some deal with other type of documents.

Ciccarese et al. developed an Ontology of Rhetorical Blocks (ORB) [20] to capture the coarse-grained rhetorical structure of a scientific article. ORB can add semantics to a new article and annotate an existing article. It divides a scientific article into three components, which are header, body and tail. The header captures meta-information about the article, such as publication's title, authors, affiliations, publishing venue and abstract. The body adopts the IMRAD structure from [87] and contains introduction, methods, results, and discussion. The tail provides additional meta-information about the paper, such as acknowledgments and references.

Peroni et al. introduced a Semantic Publishing and Referencing (SPAR) Ontologies [75] to create comprehensive machine-readable RDF meta-data for the entire set of characteristics of a document from semantic publishing. It is used to describe different components of books and journal articles, such as citations and bibliographic records. It has eight ontologies to cover all of the components for the creation of RDF meta-data.These are DoCO, FaBiO, CiTO, PRO, PSO, C4O, BiRO and PWO. Among all of the eight ontologies, DoCO describes the content of a document.

DoCO, the document components ontology [86, 21], provides a general-purpose structured vocabulary of document elements to describe both structural and rhetorical document components in RDF format. This ontology can be used to annotate and retrieve document components of an academic article based on the structure and content of the article. Some of the classes of DoCO ontology are chapter, list, preface, table and figure. DoCO also inherits another two ontologies: Discourse Elements Ontology (Deo) [25] and Document Structural Patterns Ontology[5].

Shotton et al. developed the Deo [25] ontology to study different corpora of scientific literature on different topics and publishers. It presents structured vocabulary for rhetorical elements within an academic document. The major classes of Deo are introduction, background, motivation, model, related work, methods, results, conclusion, and acknowledgements. This ontology is very intriguing and relevant to our semantic annotation. We can enhance the classes and the properties based on more than 1 million academic articles and a few hundred thousand business documents.

2.2.5 Document Summarization

2.2.6 Extractive Document Summarization

Kågebäck et al. introduced a multi-document summarization using continuous vector representations [41]. They used sentence embeddings along with phrase embeddings to maximize the cosine similarities among sentences, which significantly improved the performance of the summarization technique presented by Lin et al. [54].

Cheng et al. presented a data-driven approach to extractive document summariza-

tion based on neural networks and continuous sentence features [17]. Their model includes a neural network-based hierarchical document reader or encoder and an attention-based content extractor. They mentioned that the role of the reader was to derive the meaning representation of a document based on its sentences and words in those sentences. The attention-based extractor is directly used to select sentences or words from the input document to the output summary. These types of neural attention-based architectures were used for geometry reasoning in [94].

An RNN based sequence model, SummaRuNNer was presented by Nallapati et al. in [65]. They considered the summarization task as a sequence classification problem wherein, each sentence is visited sequentially in the original document and a decision was made whether the sentence should or should not be included in the summary. As the basic building block of their sequence classifier, they used GRU based RNN from the work presented in [19] by Chung et al. They claimed that their model had an additional advantage to visualize of its predictions by abstract features, such as information content and salience.

Nallapati et al. developed two architectures based on RNN for extractive summarization of documents [67]. The architectures are "Classify" and "Select". The Classify architecture sequentially accepts or rejects each of the sentences from the original input document for its membership in the final summary. The Select architecture has a generative model that sequentially generates sentences which should be included in the summary. They evaluated the performance of their models by deleting one abstract feature at a time from the model, with replacement.

Narayan et al. developed a framework for document summarization based on a hierarchical document encoder and an attention-based extractor with attention from the side information [68]. The encoder is based on the architectures presented in [17, 65]. They used RNN to read the sequence of sentences from the input document. The extractor selects sentences of the input document for the output summary using the architecture presented in [10].

2.2.7 Abstractive Document Summarization

An abstractive summary framework, based on Abstract Meaning Representation (AMR), was presented by Liu et al. [55]. The framework parses the source text into a set of AMR graphs. Then, the summary text is generated from a summary graph, which is transformed from AMR graphs. The framework focuses on the graph-to- graph transformation as a structured prediction problem. It assumes text documents as input and uses JAMR [31] to generate AMR graphs.

Inspired by the recent success of neural machine translation, Rush et al. presented Attention-Based Summarization (ABS), a data-driven approach for generating abstractive summaries [83]. ABS uses a neural language model with a contextual attention-based encoder from Bahdanau et al. [10]. But it also uses the Bag-of-Words Encoder and Convolutional Encoder for the input sentence. It incorporates a beam-search decoder [48] as well as additional features to model extractive elements.

Chopra et al. introduced a conditional RNN to generate a summary from input sentences [18]. The conditioning is achieved by a convolutional attention-based en-

coder. Their approach is similar to the model presented by Rush et al. [83] with the replacement of the feed-forward neural language model by an RNN. They claimed that their encoder was more sophisticated as it explicitly encodes the position information of the input words.

Nallapati et al. presented an abstractive text summarization using Attentional Encoder-Decoder Recurrent Neural Networks [66]. Their approach addresses modeling keywords, capturing the hierarchy of sentence-to-word structure, and emitting words that are rare or unseen at the training time. To identify the key concepts from the input document, similar to the word embeddings, it creates additional look-up based embedding matrices for the vocabulary of each tag-type. For mapping continuous features in an embeddings matrix, it converts them into categorical values. At the decoding stage, they used beam search to select a limited number of words for summary text.

2.2.8 Logical Structure Extraction Tools

Usually any PDF reader application implements some level of logical structure identification for rendering texts and images from a PDF document. Poppler [76], a PDF rendering library, uses an approach that starts from individual characters and creates blocks of text. Many other PDF processing tools use it. However, the algorithm used by Poppler is not described or presented anywhere in their documentation.

There are some tools available as open source or proprietary such as PDFBox [95], PDF2Text [96] and PDFlib [73]. None of these tools have technical details available online. Extracting text from PDFs using PDFBox, PDF2Text and PDFlib has some

limitations. PDF2Text doesn't work well when there is a table or image in the document. Sometimes it converts the whole table into one single line or a complete column into one single line. It also doesn't provide any layout information, such as font size, font family, or font weight. Similar to PDF2Text, sometimes PDFBox also gives an inconsistent output. PDFlib provides layout information and gives the position of each character in the document, but it doesn't always detect tables correctly. And it is not freely available.

2.3 Gaps in the Existing Research

To the best of our knowledge, many of the systems described above are not accessible. Most of the available systems focus on short articles or news articles. Some of the systems focus on scholarly articles within a limited scope. We also haven't seen any end-to-end system to understand large and complex unstructured PDF formatted documents.

Our proposed framework deals with large complex documents in electronic formats. In our experiments, we use business documents, such as RFPs and a wide varieties of scholarly articles from the arXiv repository. We applied machine learning approaches including deep learning for sectioning and semantic labeling. Our framework also understands the logical and semantic structure of scholarly articles as well as RFP documents. The details of our approaches are presented in the Technical Approach Chapter.

Chapter 3

TECHNICAL APPROACH

In this chapter, we will provide the system architecture of our framework. We will explain the approach for each of the parts of the framework. We will also describe the input and output of our proposed system architecture.

3.1 High Level System Architecture

Our proposed system is organized as a sequence of units, including a Pre-processing Unit, Annotation Unit, Classification Unit and Semantic Annotation Unit. A high level system architecture of our system is shown in Figure **??**. All of the units of our system are described below.

3.1.1 Pre-processing Unit

The Pre-processing Unit takes PDF documents as input and gives processed data as output for annotation. It uses PDFLib [73] 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 that the XML generated by PDFLib from a 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

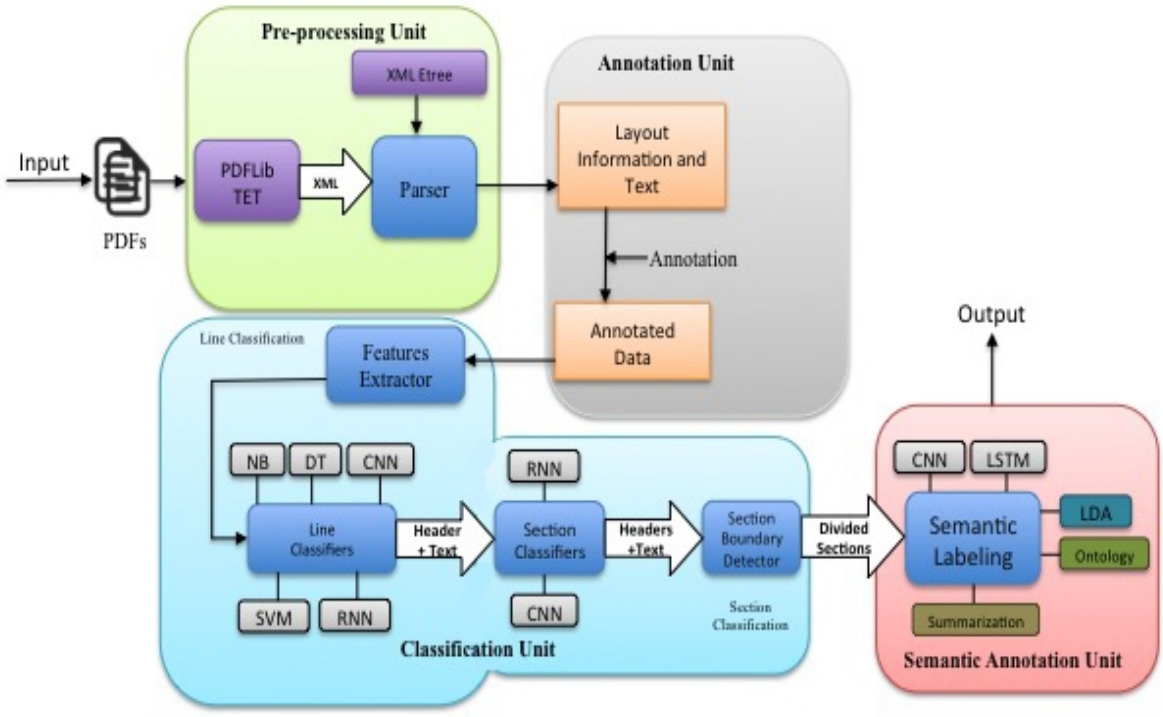


Figure 3.1: High Level System Architecture
(Ovals are software components; squares are data;components developed for this project are blue and light gray; algorithms are light gray; external components are purple.)

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 on layout and text of each line are written in a CSV file, where each row of the CSV has a line of text and layout information of the line.

3.1.2 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*. A *section-header* can be of different levels, such as *top-level*, *subsection* or *sub-subsection*. While annotating, annotators do not look into the layout information given in the CSV file. For our experiments on *arXiv* articles, we extracted bookmarks from PDF documents and used them as the gold standard annotation for training and test as described in the Input Document Processing chapter.

3.1.3 Classification Unit

The Classification Unit takes annotated data and trains classifiers to identify physically divided sections. The Unit has two sub-units for line and section classifications. The Line Classification sub unit has Features Extractor and Line Classifiers module. The Features Extractor takes layout information and texts as input. Based on heuristics, it extracts features from layout information and texts. 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), Convolutional Neural Network (CNN) and Recurrent Neural Networks (RNN) as explained in the Approach section. Based on the performance of different algorithms, the best one is selected for the operational purpose. The output of the Line Classifiers module is *section-header* or *regular-text*.

The classified section header can be *top-level*, *subsection* or *sub-subsection* header. The Section Classifiers module of the Section Classification sub unit takes section head-

ers as input and classifies them as *top-level*, *subsection* or *sub-subsection* headers using RNN and CNN. 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* headers. It also generates a table of contents (TOC) from a document based on the relationship among different levels of sections, as explained further in the Approach section.

3.1.4 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) [12] topic modeling algorithm, CNN for semantic classification for each of the divided sections, and LSTM for sequencing sections. LDA is used to capture domain specific semantic concept from each of the sections. We design and develop a document ontology to capture general purpose semantic annotation. The Semantic Annotation Unit also applies document summarization techniques using NTLK and Tensorflow to generate a short summary for each individual section. Detailed description is given in the approach section. The output of the Semantic Annotation Unit is a TOC, sections with semantic labels and section summarizations from each PDF document.

3.2 Input Data

The input documents for our framework are PDFs. Although our framework will be generic, we set the scope of our research by limiting the types of PDF documents. For the experiments in our research, we use RFP [23] documents and *arXiv* [1] scholarly articles.

The RFP documents and *arXiv* research articles are usually in PDF format. We have leveraged our existing collaboration with RedShred [2] to access large volumes of RFPs from different government and private sources. We collected more than 1 million scholarly articles from the *arXiv* repository. These PDF formatted RFPs and *arXiv* articles are the direct input to our framework. Our Pre-processing Unit extracts data from PDF documents. More details on input data processing are mentioned in the Document Processing chapter.

3.2.1 Input Data for the Classification Unit

The input for the Classification Unit is annotated data. The classification involves the Line Classification and the Section Classification modules. The Line Classification module takes a CSV file as input, where each row contains a text line and the layout information about that line. The layout information includes font size, font family, font weight, indentation of the line, $x - y$ coordinates of the line etc. The Section Classification module takes input from the Line Classification module. The input for the Section Classification module is also a CSV file, which contains classified text lines and layout information with their labels. The labels are *regular-text*, *top-level* section

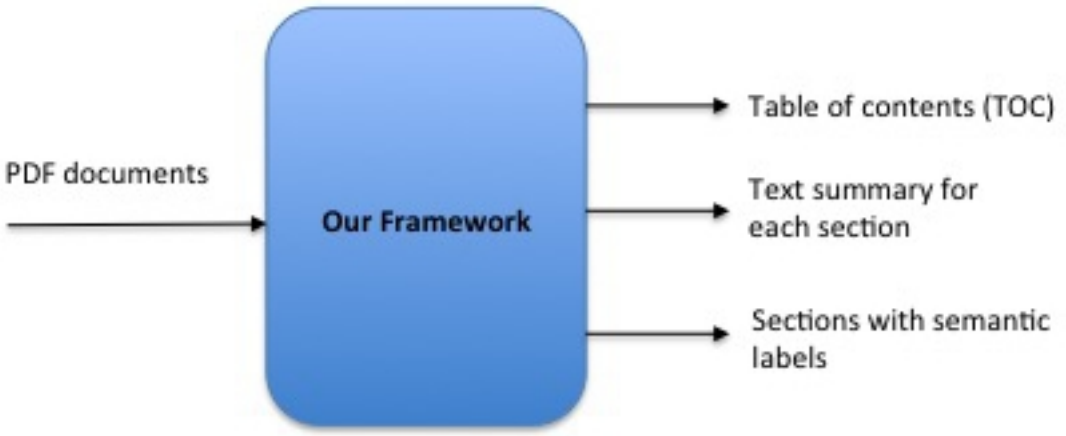


Figure 3.2: Overall Inputs and Outputs of Our Framework

header, *subsection* header and *sub-subsection* header.

3.2.2 Input Data for Semantic Annotation Unit

The input for the Semantic Annotation Unit are divided sections. Each divided section has a continuous text. For the training data, an annotator reads each section and gives a label to it. Consequently, the final input for training and test is a divided section with a class label. The class labels are introduction, background, services, deliverables, etc. A complete list of classes is given in the Approach section.

3.3 Output

Final outputs of our framework are individual sections with semantic labels, TOC and section-wise summaries of each of the documents. The overall inputs and outputs of our framework are given in Figure 3.2.

3.4 Our Approaches

We use powerful, yet simple approaches to build our framework using layout information and text contents. Layout information and text contents are extracted from PDF documents, such as RFP documents and *arXiv* articles. We use different machine learning algorithms, including deep learning, in our framework. The approaches are Support Vector Machines (SVM), Decision Tree (DT), Naive Bayes (NB), Recurrent Neural Networks (RNN), Long Short Term Memory (LSTM), Convolutional Neural Networks (CNN) and Latent Dirichlet Allocation (LDA) [12]. The details are given below based on each unit of our framework.

3.4.1 Line Classification

The Line Classification unit identifies each line of text as a *section-header* or *regular-text*. It has the Features Extractor and the Line Classifiers component. The approaches for the Line Classification are given below.

3.4.1.1 Features Extractor

Given a collection of labeled texts and layout information of 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 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 dots, 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 Nouns or Noun Phrases, Verbs and Adjectives. It is common that a section header should have more Nouns 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 numeral or single English alphabet letter. Based on all these heuristics, the Features Extractor generates 16 features from each line. These features are given in table 3.1. We also use the n-gram model to generate unigram, bigram and trigram features from the text. After feature generation, the Line Classifiers module uses SVM, DT, NB, RNN and CNN separately to identify a line as a *section-header* or *regular-text*.

3.4.1.2 Support Vector Machines (SVM)

Our line classification task can be considered as a text classification task where inputs are layout features and n-gram from the text. Given a training dataset with labels, we 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. Consider our training dataset, $T = \{x_1,\ x_2, ..., x_n\}$ 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*. Using SVM we can determine a classification model as Equation 3.1 to map a new data point with a class label from

Table 3.1: Human Generated Features

Feature name	**Description**
pos_nnp	The ratio of NN or NNP in the line is more than 50%
without_verb_higher_line_space	There is no Verb in the line and the line has higher line space
font_weight	Font weight of the line is higher than the average font weight of the page
bold_italic	the line is bold and italic
at_least_3_lines_upper	At least three consecutive lines are in upper case
higher_line_space	The line has higher line space
number_dot	There is a sequence of numbers followed by dot
text_len_group	The number of words in the line
seq_number	Numeric and roman numeral
colon	The line has a colon
header_0	Font size is more than the average font size of the page
header_1	Font size is more than the average font size of the page and font weight is more than the average font weight
header_2	Font size is more than the average font weight and bold
title_case	More than 50% of the words in the line are in title case
all_upper	All the words of the line are in upper case
voc	Vocabulary from section headers

L.

$$f : T \rightarrow L \qquad f(x) = L \tag{3.1}$$

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 [74] which implements Support Vector Classification for the case of a linear kernel as presented by Chang et al. [15]. Since our line classification task has only two class labels, we use a linear kernel. We experimented with different parameter configurations

using the combined features vector as well as the layout features vector. The detail of the SVM experiments is presented in the Experiments chapter.

3.4.1.3 Decision Tree (DT)

Given a set of lines, $T = \{x_1,\ x_2, ..., x_n\}$ where each line, 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 line, x_i by learning simple decision rules inferred from either only 16 $layout features$ or 16 $layout features$ concatenated with a number of n-gram *features* generated from the text using the *TF-IDF* $vectorizer$. The model recursively partitions all the 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 3.2.

$$\sum_{i=1}^{2} p_i(f) = 1 \tag{3.2}$$

Then, the $gini - index$ for the feature f is in equation 3.3.

$$G(f) = \sum_{i=1}^{2} p_i(f)^2 \tag{3.3}$$

For our two class line classification tasks, 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 [74] to train a decision tree model for our line classification. The experimental results are explained in the Experiments chapter.

3.4.1.4 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 calculate the probability of each class c_i from L using the Bayes theorem in equation 3.4.

$$P(c_i|F) = \frac{P(c_i)\ .\ P(F|c_i)}{P(F)} \tag{3.4}$$

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 3.5.

$$\left.\begin{aligned} Label(F) &= arg\ Max_{c_i}\{P(c_i|F)\} \\ &= arg\ Max_{c_i}\{P(c_i)\ .\ P(F|c_i)\} \end{aligned}\right\} \tag{3.5}$$

Here, the probability $P(F|c_i)$ is calculated using the multinomial Naive Bayes method. We use the multinomial Naive Bayes method from scikit-learn [74] 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.

3.4.1.5 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 its label, l $\in$ L= $\{$*regular-text* :0, *section-header* :1$\}$. We use a many-to-one RNN approach, which reads a sequence of characters until it gets to 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 its hidden unit, h_t. The activation of the hidden unit is computed by Equation 3.6.

$$h_t = \begin{cases} 0 & t = 0 \\ f(h_{t-1}, s_t) & otherwise \end{cases} \tag{3.6}$$

Here 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 3.3.

We use TensorFlow [4] to build our RNN models. We build three different networks for our line classification task. In the first and second network, we use text only and layout only as input sequence respectively. In the third network, we use both the 16 layout features and the text as input, where the one-hot matrix of characters sequence is

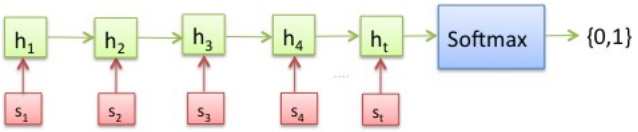


Figure 3.3: Many-to-one RNN Approach for Section Header Classification

concatenated at the end of the layout features vector. Finally, the whole vector is given as input to the network. Figure 3.4 shows the complete network architecture for combined layout and text input vectors. Implementation details are given in the Experiments chapter.

3.4.1.6 Convolutional Neural Network (CNN)

Given an input sequence, $S = \{s_1,\ s_2,\ ....,\ s_t\}$ of a line of text, we train a character level CNN model to predict its label, l $\in$ L= $\{$*regular-text* :0, *section-header* :1$\}$. The total vocabulary size is 256 different characters. We convert the input text into one hot encoding. If the input length is more than a threshold, we truncate it. And if it is less than the threshold, we pad it with zero at the end. Then the input sequence is passed through the embedding layer to represent each of the characters with a mapping in the embedding space. A convolution layer is used on the subsets of the input sequence with a filter to produce new features. Thus, c is a set of features from the input sequence, S. Then a feature map can be presented by equation 3.7 where each of the features, c_i can be generated by Equation 3.8.

$$c = [c_1, c_2,\ ....\ , c_i] \tag{3.7}$$

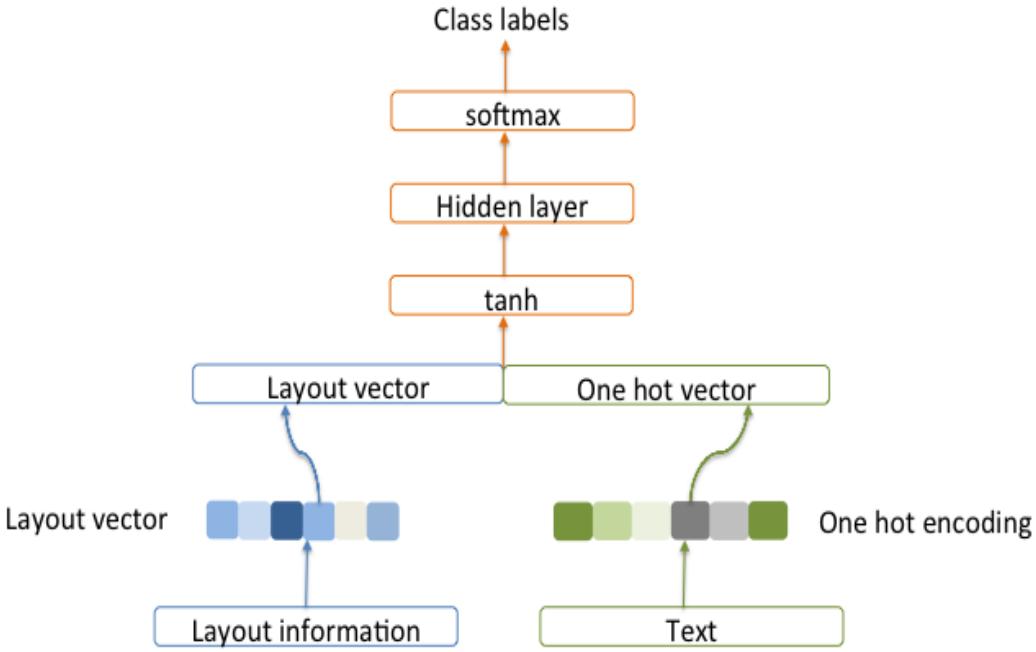


Figure 3.4: RNN Architecture for Layout and Text

$$c_i = f(w.X + b) \tag{3.8}$$

Here, w is a weight matrix, X is a subset from the input sequence S and b is a bias value.

After getting the feature set, we apply max pooling on c, to get the maximum feature values. These feature values are given to the fully connected hidden layer to get the sentence level embedding vector, which is then passed to the $Softmax$ layer for classification. We build CNN networks for three vectors; text only, layout only and combination of text and layout vectors. The network architecture for the text only input is shown in Figure 3.5. We apply $ReLu$ activation function at each convolution layer and max pooling layer.

For the combined text and layout input vectors, we have two parallel sequential layers. The first layer is for character level text input and the second layer is for the 16 layout features. The output from both layers are merged together and passed through the last fully connected hidden layer. Finally, the output from the last hidden layer is passed through the $Softmax$ for classification. The network architecture is given in Figure 3.6.

3.4.2 Section Classification

The section classification module takes section headers and section body text as input from the line classification module and identifies different levels of section headers, such as *top-level section*, *subsection* and *sub-subsection* headers. It also detects

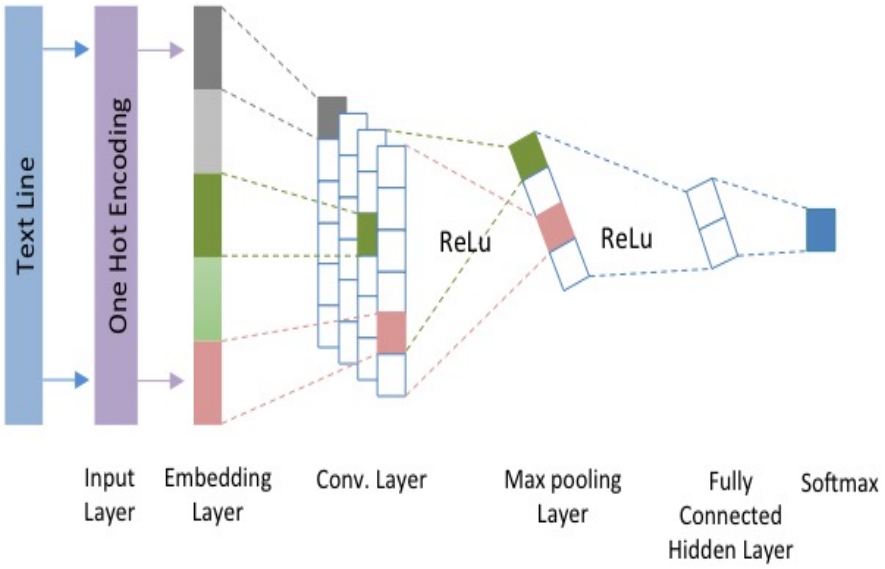


Figure 3.5: CNN Architecture for Text Only Input

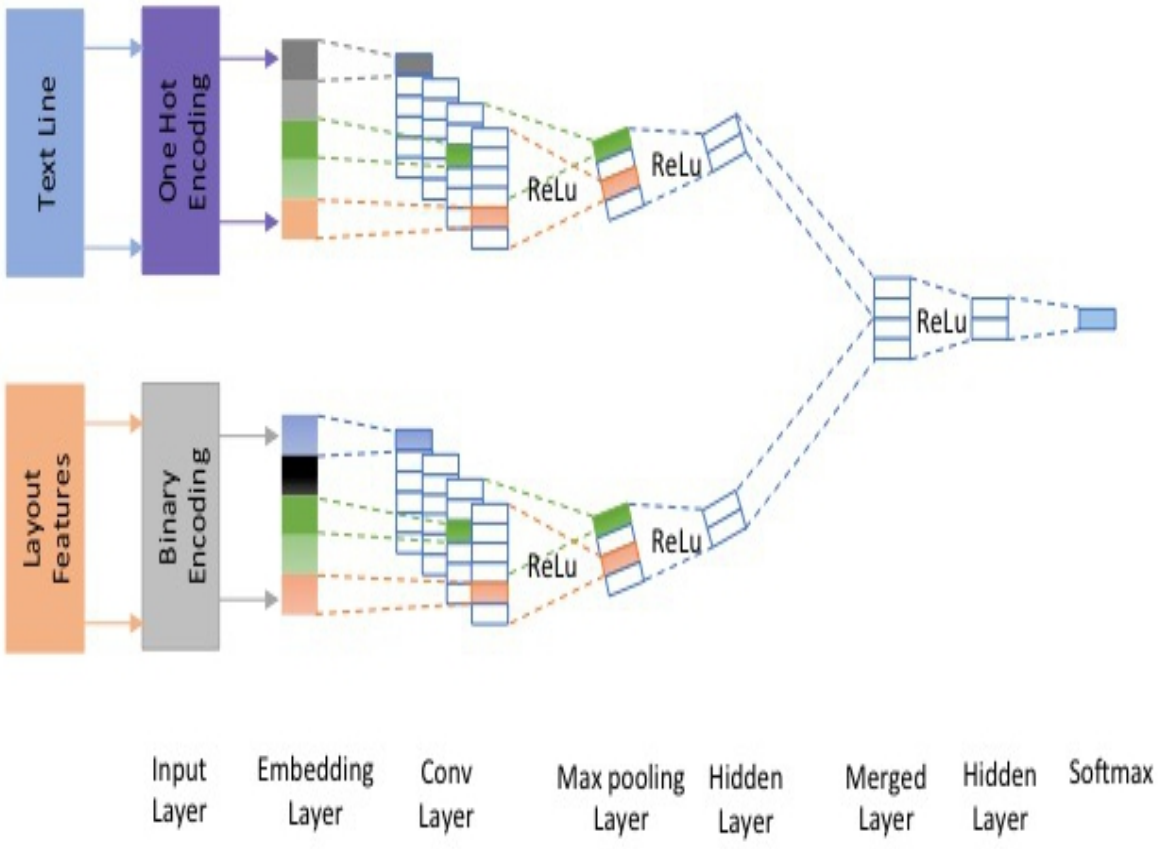


Figure 3.6: CNN Architecture for Combined Text Input and Layout Input

section boundaries. It has a Section Classifiers module and a Section Boundary Detector component, which are explained below. The output of this module are physically divided sections.

3.4.2.1 Section Classifiers

Like the Line Classifiers module, the Section Classifiers module considers the section classification task as a sequence classification problem, where we have a sequence of inputs, $S = \{s_1,\ s_2, ..., s_t\}$ from classified section headers 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 classification task, we use both RNN and CNN architectures similar to the architectures used for the line classification task. We also use text only, layout only and combined text and layout input vectors for the Section Classifiers. The input and output of RNN/CNN for the text only network is shown in Figure 3.7. The overall inputs and outputs for the section classification task with the Section Boundary Detector is shown in Figure 3.8, where the network has combined the text and layout input vectors.

There are few reasons for choosing RNN and CNN over other machine learning algorithms for the section classification. The first reason is that the section classification is more complex than the line classification due to the complexity of the nature of different types of section headers. The second reason is that we get better performance for line classification using RNN and CNN over other algorithms for our dataset. The third reason is that we worry less about feature generation. The fourth reason is that

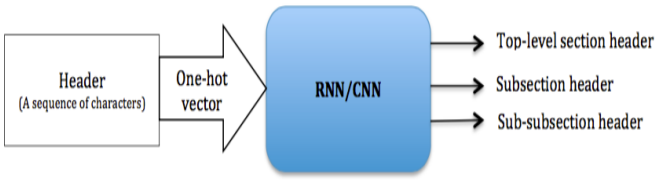


Figure 3.7: Inputs and Outputs of RNN/CNN for Section Classification with Text Only[Squares are data; ovals are software components.]

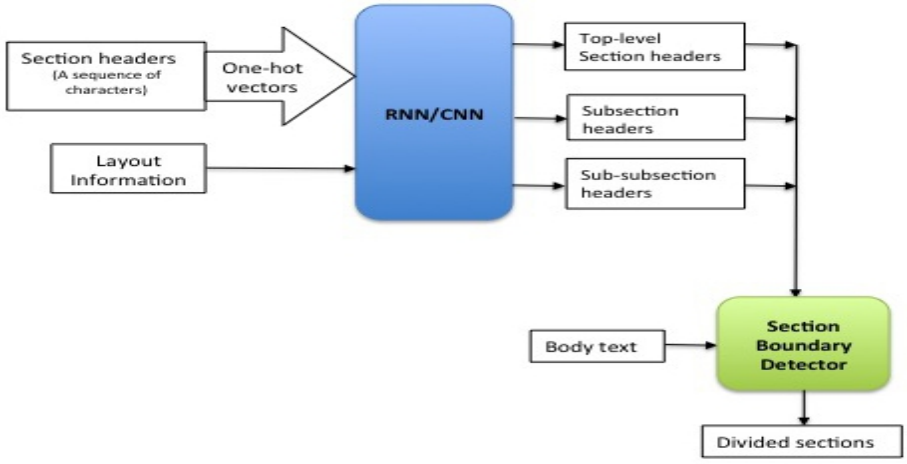


Figure 3.8: Overall Inputs and Outputs for Section Classification with Combined Text and Layout Vector

RNN and CNN learn the structure of our dataset more effectively than other machine learning algorithms. The last but not least reason is that RNN and CNN can generate new unknown features and find relationships between features for our dataset.

3.4.2.2 Section Boundary Detector

After identifying different levels of 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

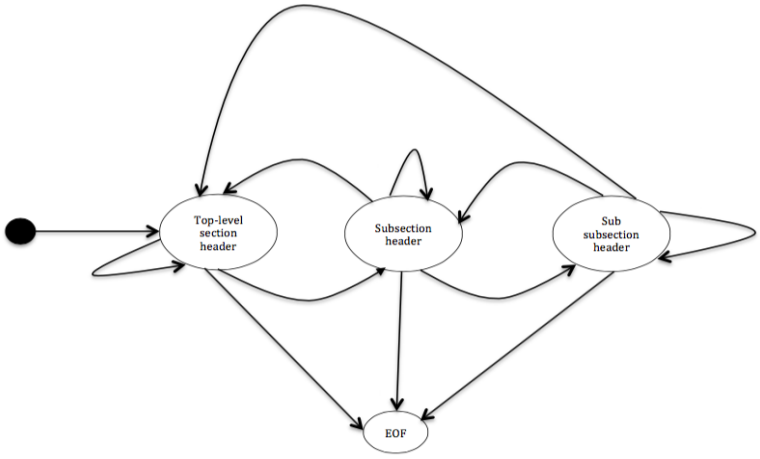


Figure 3.9: Top-level Section Header, Subsection Header and Sub-subsection Header Dependency Sequence

Section Boundary Detector splits the whole document into different *sections*, *subsections* 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, and the values are *content*, *header* and *nested sections* respectively. The subsections have a similar nested structure. The Section Boundary Detector finds the relationship among *sections*, *subsections* and *sub-subsections* using the dependency state diagram presented in Figure 3.9. The high level algorithm, that generates *sections*, *subsections* and *sub-subsections* using the dependency diagram and class labels, is presented in Algorithm 1.

3.4.3 Semantic Annotation

Given a set of physically divided sections, $D = \{d_1,\ d_2, ..., d_n\}$, the Semantic Annotation Unit assigns a human understandable semantic name to each section. It has

Algorithm 1 Section boundary detector

```
1: procedure SPLIT_DOC_INTO_SECTIONS(doc, split_level)
2:     sections =[]
3:     if split_level is top_level then
4:         for line in doc do
5:             Generate text_block based on class_label = 1
6:             Add {title, text_block} in sections
7:         end for
8:     else if split_level is subsection then
9:         for line in doc do
10:            Generate text_block based on class_label =1
11:        end for
12:        for block in text_block do
13:            Generate sub_block based on class_label =2
14:            Add {title, sub_block} in sections
15:        end for
16:    else
17:        for line in doc do
18:            Generate text_block based on class_label =1
19:        end for
20:        for block in text_block do
21:            Generate sub_block based on class_label =2
22:        end for
23:        for block in sub_block do
24:            Generate sub_sub_block based on class_label =3
25:            Add {title, sub_sub_block} in sections
26:        end for
27:    end if
28:    return sections
29: end procedure
```

a Semantic Labeling module, which implements different components to get the output of our framework.

3.4.3.1 Semantic Classifier

We built CNN and bidirectional LSTM models to classify each of the physically divided sections. At the end, word based CNN model was chosen as a Semantic Classifier. There are several reasons to choose word based CNN as a Semantic Classifier. First of all, we achieved the best performance using word based CNN for our dataset. Secondly, we chose word based architecture to capture the semantic meaning of each section based on words. Finally, CNN works better to capture the structure of sentences in different sections, which helps to identify the semantic meaning of different sections. For example, an introduction has sentences for explaining motivations, a related work section has lot of citations, a technical approach section usually has more mathematics and equations, and a result section has more graphics and plots.

The labels of the Semantic Classifier are classes from our document ontology. Detailed information about class selection is described in the ontology design section. We implemented a CNN architecture similar to the model presented by Kim [43] for sentence classification. The architecture is also similar to the architecture we presented in the Line Classification for CNN in a previous section of this chapter. In our implementation, we built a word embedding layer and each of the sections with its class label is considered as input to the network. The CNN architecture is given in Figure 3.10.

For the experimental purpose, we also built both word based and character based

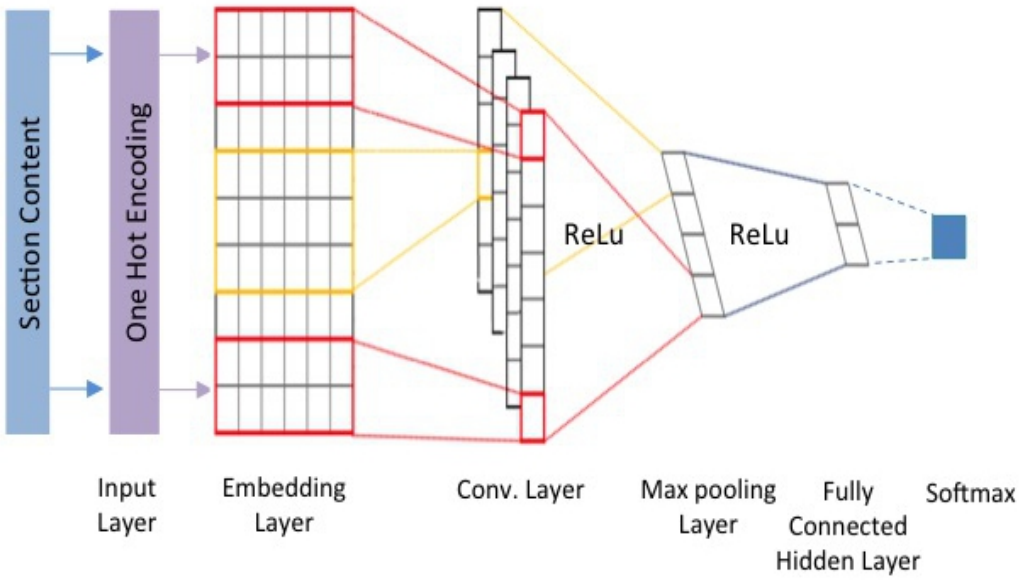


Figure 3.10: CNN Architecture for Semantic Section Classifier

bidirectional LSTM models for semantic section classification. The architecture is given in Figure 3.11. The detailed description is given in the Experiments and Evaluation chapter.

3.4.3.2 Sequence Prediction

After getting all of the sections with semantic names of a document, we may need to restructure the sequence of sections. The order of sections may differ from article to article. In some articles, an introduction is followed by a related work section, whereas in other articles, an introduction may also contain related work. It is important to reorganize the section after automatic section generation with semantic name.

For a sequence of section headers, $H = \{h_1,\ h_2, ..., h_n\}$ from a document, we built a Sequence Prediction model to reproduce the whole sequence of section head-

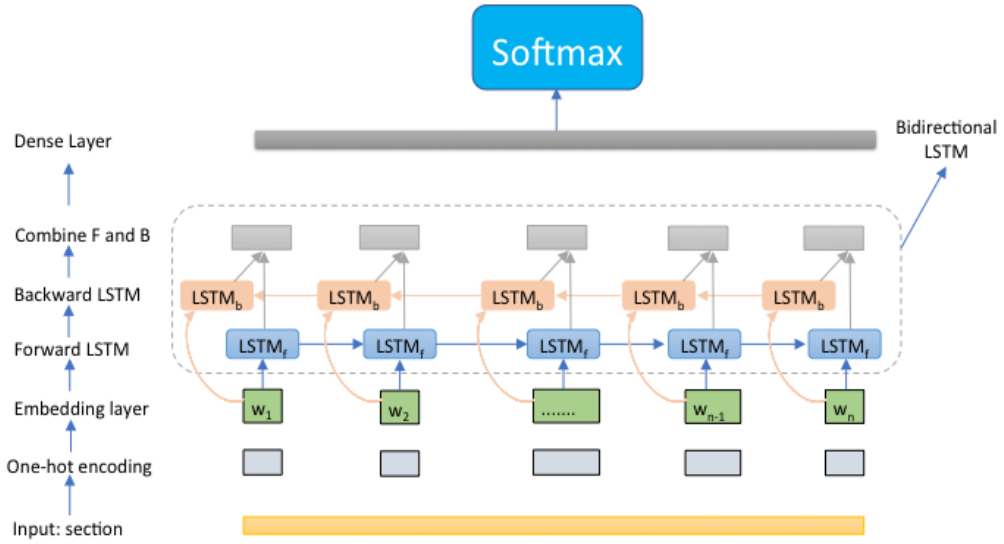


Figure 3.11: Bidirectional LSTM Architecture for Semantic Section Classifier

ers based on a given sequence of section headers. This prediction model predicts each section header using historical sequence information in the sequence. We used LSTM network for our sequence prediction task. The reason is that LSTM networks achieve state-of-the-art results in sequence prediction problems. We chose a many-to-many LSTM architecture for our section header prediction, since the whole sequence of section headers is predicted based on the given sequence of section headers. The section header prediction diagram is shown in Figure 3.12.

3.4.3.3 Sections Mapping and Ontology Design

After getting a list of section headers classified by the Semantic Classifier and sequencing them using the Sequence Prediction model, we map them in a semantic manner using an ontology. This mapping is basically the process of semantic annota-

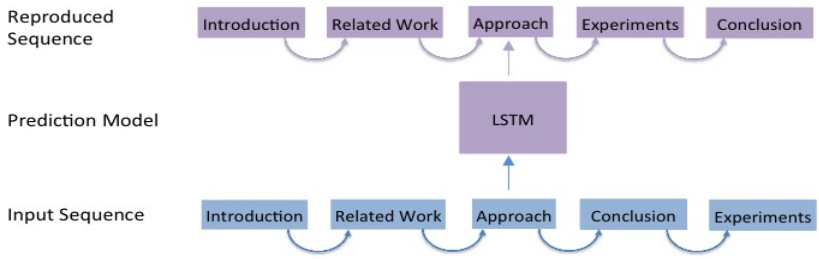


Figure 3.12: LSTM Sequence Prediction Diagram

tion in a document. An article from computer science may have an approach section, which is similar to a methodology section in a social science article. The semantic section mapping will help to map each of the sections of a document with human understandable names, which adds meaningful semantics by standardizing section names of the document.

In order to design a document ontology, we created a list of classes and properties. We followed the count based and the cluster based approaches. In the count based approach, we first took all section headers, including *top-level*, *subsection* and *subsubsection* which are basically headers from the table of contents of all *arXiv* articles. Then we removed numbers and dots from the beginning of each header and generated the count for each header and sorted them based on count. Then we manually analyzed all section headers, which might be a class for our ontology.

In the cluster based approach, we generated all section headers from the table of contents of all *arXiv* articles and developed a Variational Autoencoder(VAE) [28] to

represent each of the section headers in a sentence level embedding which is named as header embedding in our research. We applied Autoencoder to learn the header embedding in an unsupervised fashion so that we could get a good cluster. Then we dumped the embedding vector from the last encoding layer. This vector has high dimensions. Usually, clustering on high dimensional data doesn't work well. So we applied t-SNE [58] dimensionality reduction technique to reduce the dimensions of the embedding vector to 2 dimensions. After dimensionality reduction, we used k-means [38] clustering on the embedding vector to cluster the header embedding in semantically meaningful groups. We manually analyzed all clusters and all section headers from the count based approach and came up with the classes to design our document ontology. We also applied similar approaches for section headers from RFP documents. To understand the sections of an RFP, we read [3] and discussed with experts from RedShred [2]. Table 3.2 and 3.3 show classes from *arXiv* articles and RFPs respectively, which were used to design a simple document ontology. The architecture of the autoencoder is given in Figure 3.13.

A Variational Autoencoder is a type of autoencoder, which learns latent variable models [30] for the input data. As a result, instead of learning an arbitrary function, the autoencoder learns the parameters of a probability distribution of the input data. The encoder turns the input data into two parameters in a latent space, which are noted as $\bar{z}$ and $z \log \sigma$. Then randomly, a similar data point, z is selected from the latent normal distribution using equation 3.9. At the end, a decoder maps these latent space points

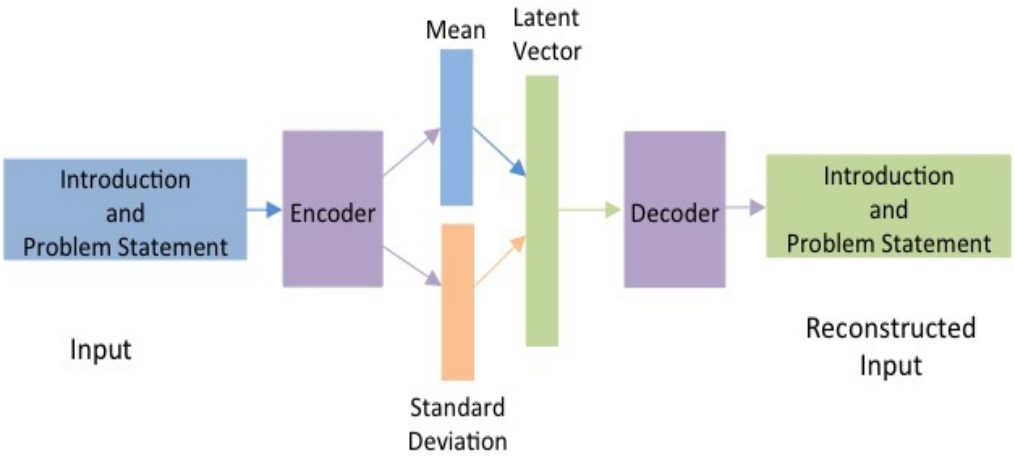


Figure 3.13: Variational Autoencoder for Ontology Class Selection

back to the original input data.

$$z = \bar{z} + e^{z \log \sigma} * \epsilon \tag{3.9}$$

After getting the classes from manual analysis of the count and the cluster based approaches, we designed an ontology for our input document. The classes represent concepts in our ontology. We also analyzed cluster visualization to get properties and relationship among classes. Detailed results are included in the Experiments chapter. Figure 3.14 shows our simple document ontology.

The document ontology builds the concepts from the *arXiv* academic articles and RFP documents. The top level "Document" class has two subclasses: "Academic Article" and "RFP". The Document class has "Category" that describes the type of document, such as Computer Science, Mathematics, Social Science, Networking, Biomedical and Software articles/RFPs. Both Academic articles and RFPs have contents, which

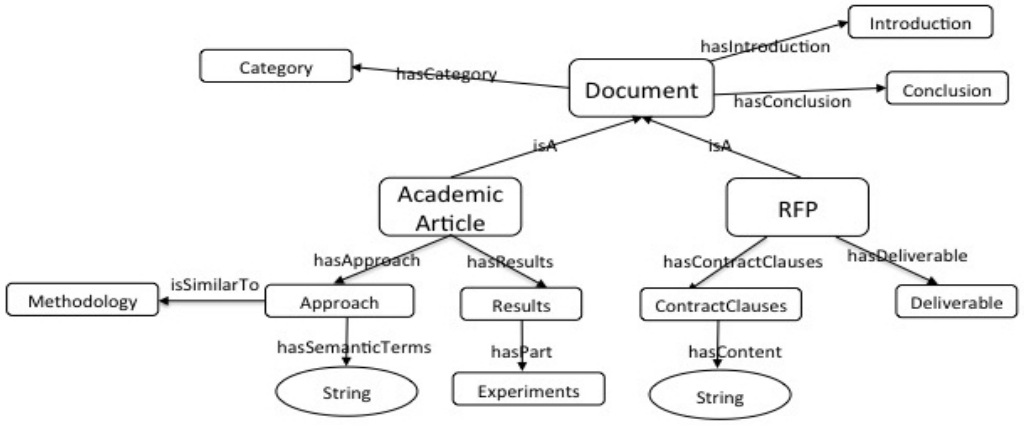


Figure 3.14: Document Ontology

are sections. These sections are the classes of different semantic concepts in a document. Both Academic articles and RFPs share some concepts, such as “Introduction”, “Conclusion” and “Background”. They also have their own concepts. For example, “Approach” and “Results” are available in Academic Articles whereas RFP has “ContractClauses” and “Deliverable” concepts/classes. Due to space constraint, the classes are shown in Tables 3.2 and 3.3. Each of the classes/concepts has two properties “SemanticTerms” and “Content” which are represented by the relationships “hasSemanticTerms” and “hasContent”. The data types for these two properties are String. The “hasSemanticTerms” property captures semantic topics applying Latent Dirichlet Allocation (LDA) to each section. Some concepts may have part, which is represented by a relationship “hasPart”. For example, a concept “Results” has another subconcept “Experiments”. Some of the concepts may be similar to another concepts, which is shown by a relationship “isSimilarTo”. For example “Approach” and “Methodology” are two similar concepts.

3.4.3.4 Semantic Concepts using LDA

We used LDA [12] to find semantic concepts from a section. LDA is a generative topic model which is used to understand the hidden structure of a collection of documents. In an LDA model, each document has a mixture of various topics with a probability distribution. Again, each topic is a distribution of words. A graphical LDA model is shown in Figure 3.15 presented by Blei et al. [12], where a circle represents a random variable and an observed variable is shaded. An arrow indicates the dependency of value from one random variable to another. A repetitive set of variables is surrounded by a rectangular plate.

Using Gensim [81, 82], we trained an LDA topic model on a set of divided sections. The model is used to predict the topic for any test section. A couple of terms, which have the highest probability values of the predicted topics, are used as semantic concepts for a given section. These semantic concepts are also used as property values in the document ontology.

3.4.3.5 Sections Summarization

Given a set of sections, $D = \{d_1,\ d_2, ..., d_n\}$ from an *arXiv* article or an RFP, the semantic labeling module implements a summarization component to generate automatic summary for each section. The summarization component uses state of the art approaches to generate both extractive and abstractive summaries. For an extractive summary, it uses the $Textrank$ algorithm [61] implemented in Gensim [81].

The summarization component also trains the $Tensorflow\ Textsum$ [71] model

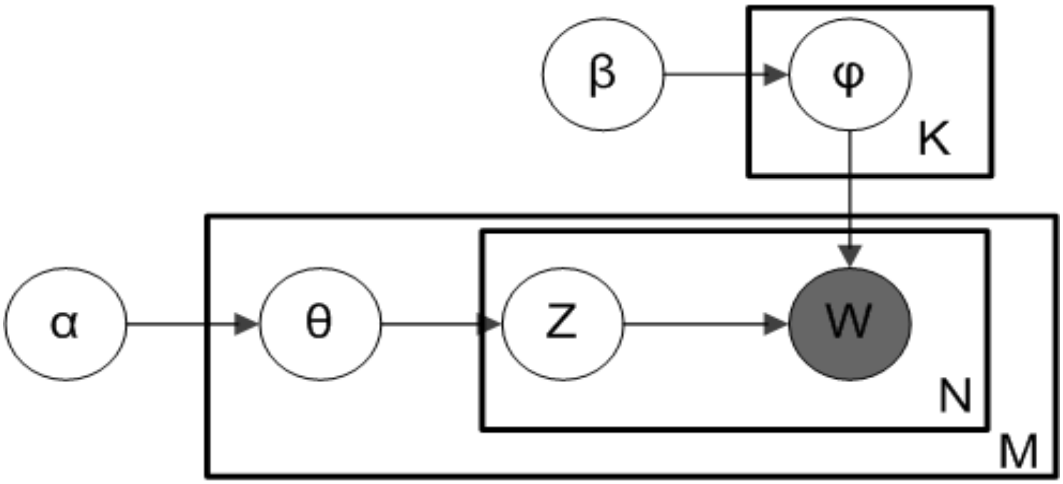


Figure 3.15: LDA Topic Model

using Sequence-to-Sequence with Attention Model [90] to generate an abstractive summary from each section. The model is based on LSTM architecture. The model works well for short articles, assuming that the first 2 sentences have enough information, which may play a significant role in summarization. During training, the model uses the first two sentences from the article as an input. And during decoding, the model uses beam search [48] to find the best sentence from candidate sentences generated by the model. The two sentence assumption may not always be true; hence we modified some of the network parameters, including number of sentences, words, and layers, to fit the model in our dataset. The details of training and test datasets are described in the Input Document Processing chapter and results are explained in the Experiment chapter.

Table 3.2: Classes for Ontology from arXiv Articles

Class Name	**Description**
Introduction	It is an initial description about the work. It may contain the main research idea and contributions.
Conclusion	This is the ending section of an article. Usually it summarizes the work with evidences. It may also include some future work.
Discussion	This is a section that describes and interprets the results of the experiments.
References	This section includes citations on related works.
Acknowledgments	This section presents an acknowledgement of the people and agencies that supported the work by giving funds and ideas.
Results	This section describes all the experimental results.
Abstract	A very initial description about the work is presented in this section.
Appendix	This section includes additional information, such as extra proofs and detailed information.
Related Work	This section often follows the introduction. The related work section may also be called a literature review. It includes related previous research.
Experiments	This section represents detailed procedures that were carried out to support, refute, or validate a hypothesis.
Methodology	This is the systematic, theoretical analysis of the methods applied to the research to solve the problem.
Proof of Theorem	This section evaluates mathematical proofs of hypotheses.
Evaluation	This the section that evaluates the methodology.
Future Work	This is the future direction of the current research.
Datasets	This section describes the dataset used for the research.
Contribution	This section includes all of the contributions of the research. Sometimes, this section is merged with introduction.
Background	This section includes the information that is essential for understanding the research problem.
Implementation	This section describes the detailed implementation of the proposed approach.
Approach	This is the technical approach section, which is similar to the methodology section.
Preliminary	This section includes the preliminary research work in the proposed research.

Table 3.3: Classes for Ontology from RFPs

Class Name	**Description**
Introduction	It is an initial description about the work. It may contain the main research idea and contributions.
Requirement	The requirement section of an RFP provides suppliers with technical requirements and enough information to enable them to understand the issues and write a proposal.
General Information	This section of an RFP has a solicitation form. The form gives basic information about the project.
Conclusion	This is the ending section of an article. Usually it summaries the work with evidences. It may also include some future work.
Statement of Work	In this section, the main concept of an RFP is presented. It describes the necessaries that should be submitted by the contractor. It also includes the process of submission.
Contract Administration	This is a section that describes administrative information that is required from the agency.
Appendix	This section includes additional information such as extra proofs and detailed information about any aspect of a paper.
Background	This section includes the information that is essential for understanding the research problem.
Deliverable	This section presents all of the deliverables for a specific proposal call, such as software as a deliverable
Contract Clauses	This section presents all of the clauses, which are required to understand an RFP submission. Usually this section describes boilerplate.

Chapter 4

INPUT DOCUMENT PROCESSING

In this chapter, we will describe input documents, data collection, data processing, training data and test data. We will also explain the nature of input data.

4.1 Data Types

In this research, we focused on PDF documents. The reason to choose PDF documents as input documents is the popularity and portability of PDF files over different types of devices, such as personal computers, laptops, mobile phones and other smart devices. PDF is also compatible with different operating systems, such as Windows, Mac OS and Linux. We mostly focused on large PDF documents and those may be of different domains, such as academic articles, business documents, medical reports and user manuals. Due to resource limitation, we confined our scope to input document types to academic articles and business documents. We chose *arXiv* e-prints as academic articles and RFPs as business documents.

4.1.1 arXiv Articles

arXiv is a repository for a large number of scholarly articles from different scientific fields, such as computer science, mathematics, statistics and quantitative finance. It has more than one million scholarly articles over different categories. We collected

1,121,363 *arXiv* articles during or before 2016. Detailed description is given in the Data Collection section.

4.1.2 RFP Documents

An RFP is a type of bidding document that is used to announce available funding for a project, where an agency or a company can bid to get the business opportunity. The announcing party can get potential vendors for a desired project. An RFP usually has different sections, such as statement and scope of work, requirements, terms and conditions, technical goals and administrative contract. Each of the sections has a specific purpose. The bidding agencies read an RFP carefully, analyze it, and take a "go" or "no go" decision. We leveraged our existing collaboration with RedShred [2] to get a wide range of RFPs for our experiments.

4.2 Data Collection

We used the *arXiv* bulk data access option to collect *arXiv* articles available from Amazon S3. The available access mechanisms are grouped into two different services: metadata access and full-text access services. Both are explained below.

4.2.1 Full Text Access

We downloaded a complete set of *arXiv* articles available in PDF format using requester pay buckets [8] from Amazon S3 cloud. The buckets are located in the Eastern US (N. Virginia) region. The PDFs are grouped into .tar files as a chunk. The size of

```
<?xml version='1.0' standalone='yes'?>
<arXivPDF>
  <file>
    <content_md5sum>1852974c8570cdafd91522ee93719ee5</content_md5sum>
    <filename>pdf/arXiv_pdf_0001_001.tar</filename>
    <first_item>astro-ph0001001</first_item>
    <last_item>hep-th0001208</last_item>
    <md5sum>4b5eeb603fd68bb05b9dd3341e9067fb</md5sum>
    <num_items>1751</num_items>
    <seq_num>1</seq_num>
    <size>526080000</size>
    <timestamp>2009-12-23 14:41:24</timestamp>
    <yymm>0001</yymm>
  </file>
  <file>
    <content_md5sum>650da80f3bcd1f4cd3d994b572ecdbb9</content_md5sum>
    <filename>pdf/arXiv_pdf_0001_002.tar</filename>
    <first_item>hep-th0001209</first_item>
    <last_item>quant-ph0001119</last_item>
    <md5sum>eedc2d7c09cf11fda188d8600c966104</md5sum>
    <num_items>565</num_items>
    <seq_num>2</seq_num>
    <size>139560960</size>
    <timestamp>2009-12-23 14:42:52</timestamp>
    <yymm>0001</yymm>
  </file>
  ...
</arXivPDF>
```

Figure 4.1: Manifest File Structure

each tar file is approximately $500MB$. We downloaded an XML formatted manifest file from *arXiv*, which contains the complete list of all chunks. The manifest file structure is given in Figure 4.1. We parsed the manifest file and took the filename element for each tar file. Then we installed and configured the "s3cmd" tool with our amazon account and downloaded all tar files using the s3cmd.

After downloading all tar files, we used the "xopf" tar command to unzip each of the tar files. In total, we received $1,121,363$ PDF articles over all *arXiv* categories. The total size of all PDF files was $743.4GB$. We organized all the PDF files according to publication years. Detailed statistics of all files are given in Table 4.1.

Table 4.1: arXiv Statistics of All Files

Name	Value
Tar file size	500MB
Number of total PDFs	1,121,363
Size of all PDFs	743.4GB
Size of all TETML files	5.1TB
Number of categories	37966
Years	1986 to 2016

4.2.2 OAI Call

After unzipping all the PDF files, we used the name of the files as *arXiv* article identifier and used Open Archives Initiative (OAI) protocol to harvest metadata for each article. The $arxiv.org$ returns an XML file for each OAI request. We parsed the XML response and generated metadata for each article. A list of metadata is shown in table 4.2. We generated a JSON file to store all meta information for each article, where the key of each article was the *arXiv* file identifier. We also retrieved month and year of the publication from the date metadata.

Table 4.2: arXiv Article Metadata

Metadata Name	Type
Title	String
Publication Date	Date
Summary	String
Category Name	List
Authors Name	List
Link To arXiv	String

4.2.3 TOC Extraction

Some of the *arXiv* articles have bookmarks. These bookmarks are generated from the latex source when *arXiv* generates PDF version from the source submitted

by the authors. For each of the *arXiv* identifiers from the JSON file, we extracted bookmarks from the original PDF file. The hierarchy, which had up to several level of sub-subsections, was kept intact. We considered bookmarks as the TOC for each article. The TOCs were used as section header annotation in our experiments. Later, We merged metadata and TOC for individual *arXiv* articles and stored them in the same JSON file.

4.2.4 Convert to TETML

For each of the *arXiv* articles in PDF format, we applied PDFLib TET [73] to extract their contents from the PDF. The PDFLib converts PDF to special type of XML called Text Extraction Toolkit Markup Language (TETML). While converting into TETML, we used word level granularity which generated TETML with a detailed description of each character of every word in a PDF file. The description includes $x-y$ coordinates, font information, and page number for each character. The description is used for feature generation. After converting all *arXiv* PDF files into TETML files, the total size of all TETML files was $5.1TB$.

4.3 TETML Processing

The elements in a TETML are organized in a hierarchical order. Each TETML file contains pages, and 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.

We developed a parser to read the structure of the TETEML file. The parser also reads and processes the description of each character. We applied different heuristics to process the description. Based on the heuristics, the parser generates the text on each line, font size, font weight, and font family for that line. It also calculates the starting and ending position of each line by generating $x - y$ coordinates for the very first character of the first word, and the last character of the last word in a line. All of the generated attributes from the TETML description are given in Table 4.3. All of these attribute values were used to generate layout features.

Table 4.3: Generated Attributes from the TETML

Attribute's Name	**Description**
Text Line	A complete text line based on heuristics.
Font Size	Font size is selected based on the maximum occurrence of font size from a line.
Font Family	Font family is selected based on the maximum occurrence of font family from a line.
Font Weight	Font weight is selected based on the maximum occurrence of font weight from a line.
Page Number	Page number is taken from TETML attribute "number".
X Position Left	X coordinate of the first character of the first word in a line.
X Position Right	X coordinate of the last character of the last word in a line.
Y Position Left	Y coordinate of the first character of the first word in a line.
Y Position Right	Y coordinate of the last character of the last word in a line.
Page Width	Page width is taken from TETML attribute "width".
Page Height	Page height is taken from TETML attribute "height".

We also processed a table from the TETML file. TETML has a tag called "table" to store content from a table. A table element has a row, a row has a cell, a cell has a para, a para has words, and finally a word has a text tag to store content from a table. We also captured content from all tables.

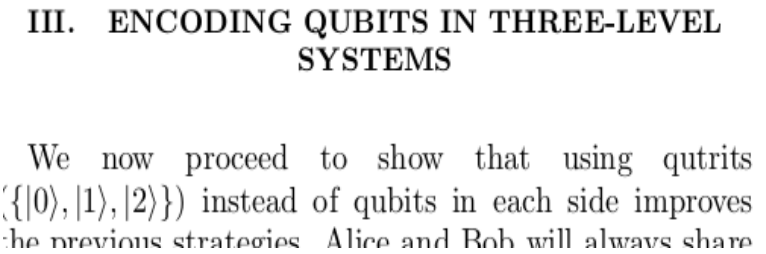
For each article, we mapped TOC with original text lines from the document.

This mapping was used to generate class labels for each of the text lines. If a line is not in the TOC, it was considered a regular text and the class label was 0. If a line was in the TOC, we searched the path of that line from the root to leaf. And for each hierarchical hop, we added a level in such a way that the top-level element from a TOC had label 1, the next level had 2 and so on. A complete list is show in table 4.4. To find the path of a text line from the TOC, we wrote a recursive function. Due to the limitation of PDFLib TET, sometimes a text line doesn't match with an element from a TOC, although it exists in the TOC for an article. This happens when the PDFLib tool mislabels a para element. To overcome this challenge as much as possible, instead of exactly matching a line in the TOC, we used string similarity matching based on a threshold value. If the similarity score is more than a threshold value, we considered that partial match as a match. We used the SequenceMatcher python library to calculate the string similarity.

Sometimes PDFLib processes a section header into multiple para elements. It is very challenging to keep track of or monitor this issue in a document, as there is no identical difference between such para. An example is shown in Figure 4.2. In such cases, our parser may fail to map a line with an element from a TOC. PDFLib splits "III. ENCODING QUBITS IN THREE-LEVEL SYSTEMS" into two different text blocks, which are two different para elements in the TETML. One para element contains "III. " and another one contains "ENCODING QUBITS IN THREE-LEVEL" and "SYSTEMS". Our parser can handle the second para, as two lines are in the same para element. Due to the PDF encoding technique, PDFLib makes this type of error.

Table 4.4: Class Labels with Text hierarchy

Hierarchical Level	Class Label
Regular Text (not from a TOC)	0
Top-level from TOC	1
2nd level from TOC	2
3rd Level from TOC	3
4th Level from TOC	4
5th Level from TOC	5


III. ENCODING QUBITS IN THREE-LEVEL SYSTEMS

We now proceed to show that using qutrits ($\{|0\rangle, |1\rangle, |2\rangle\}$) instead of qubits in each side improves the previous strategies. Alice and Bob will always share

Figure 4.2: PDFLib Processing Issue with Multiple Lines

After mapping each line with elements from the TOC for a document, we wrote all the lines with attribute values shown in Table 4.3 in CSV files. We obtained a total of 762 CSV files containing all *arXiv* articles. The size of each CSV file was around $115MB$. Each row of a CSV file contained a line with layout information and a class label. A complete process diagram is shown in Figure 4.3.We split the whole dataset for training and test data. A Complete set of actions for processing a TETML file is shown in the list below.

- Read each TETML file
- Generate lines based on heuristics
- Generate layout information for each line

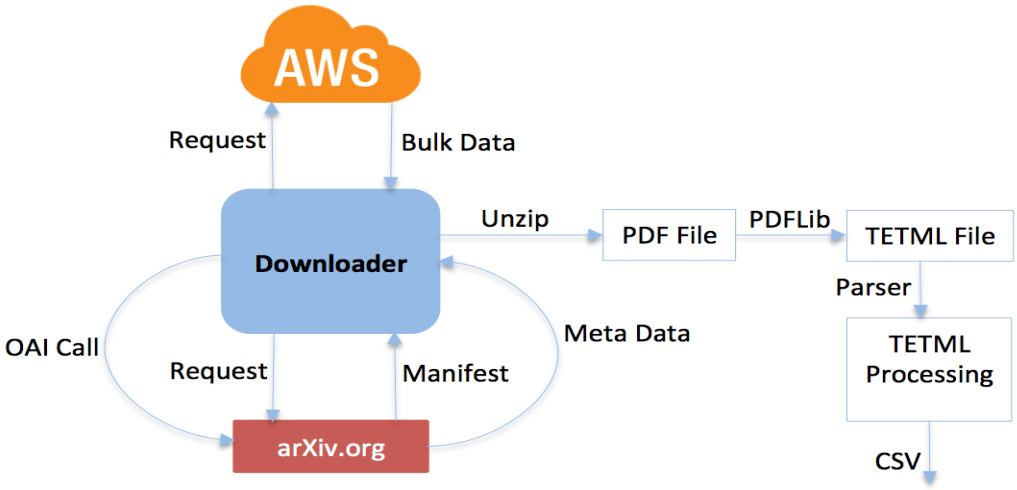


Figure 4.3: Flow Diagram for Input Document Processing

- Map each line with the elements from the TOC
- While mapping, find the path from the TOC to obtain a class label
- Write each line to a CSV file

4.4 RFP Processing

We collected a wide range of RFPs from different sources through the collaboration with RedShred. The total number of RFPs was three hundred fifty thousand. Most of the RFPs were in PDF format, though some of the RFPs were in DOC or Excel format. We converted non-PDF formatted RFPs into PDF using a standard PDF conversion tool. Then we chose 250 random RFPs over the total RFPs to ensure diverseness in the chosen RFPs. We generated TETML files for each of the chosen RFPs.

For each of the TETML files, we processed it as described in the TETML Pro-

cessing section above and stored them in a CSV file. Unlike *arXiv* articles, there was no bookmark available in RFPs. As a result, we had to follow a manual annotation process to obtain annotated data from RFPs. We took help from the RedShred expert annotation team in the annotation process. We followed standard annotation techniques for annotating each line of the CSV file as *regular-text*, *top-level section* header, *subsection* header and *sub-subsection* header. We used this annotated data for our experiments on RFPs. Detailed description is given in the Experiments chapter.

4.5 Training and Test Data

For each of the units of our system architecture, we created a training and test dataset. The training set was used to build models in each unit and the test set was used to evaluate the performance of the models. We took 60 random files among 762 CSV files for training and test dataset preparation.

4.5.1 Data for Line Classifiers

We developed the training and test dataset from 60 randomly chosen CSV files. For each of the data points we had two feature vectors: layout vector and text vector. The Features Extractor presented in the Technical Approach chapter was used to generate the layout feature vector. The text vector was a one hot encoding vector from a text line. After generating both vectors, we split the whole dataset into training and test sets using a $5fold$ cross validation with balanced class labels. Then we randomized each dataset using stratified sampling so that the classifiers learned from random input data.

Table 4.5 shows the training and test dataset for the Line Classifiers having two classes.

Table 4.5: Dataset for Line Classifiers Having Two Classes

Dataset	**Class Label**	**Sample Number**
Training	Regular-Text	389229
	Section-Header	389229
Test	Regular-Text	80184
	Section-Header	80184

4.5.2 Data For Section Classifiers

Similar to the Line Classifiers, we also developed training and test dataset for Section Classifiers. In our section classification task, we experimented with three and four classes. The three classes are *top-level*, *subsection* and *sub-subsection* section headers. The four classes experiment uses an additional *regular-text* class. The dataset was prepared using stratified sampling to balance all of the class samples. Table 4.6 and 4.7 show the training and test dataset for the Section Classifiers having three and four classes respectively.

Table 4.6: Dataset for Section Classifiers Having Three Classes

Dataset	**Class Label**	**Sample Number**
Training	Top-level	37650
	Subsection	37650
	Sub-subsection	37650
Test	Top-level	9003
	Subsection	9003
	Sub-subsection	9003

Table 4.7: Dataset for Section Classifiers Having Four Classes

Dataset	Class Label	Sample Number
Training	Regular-Text	37650
	Top-level	37650
	Subsection	37650
	Sub-subsection	37650
Test	Regular-Text	9003
	Top-level	9003
	Subsection	9003
	Sub-subsection	9003

4.5.3 Data For Semantic Section Classifier

For each physically divided section and corresponding section header, we applied some heuristics to group them based on the classes defined for our document ontology. For the ontology of *arXiv* articles, we had 20 different classes. We mapped those classes with the section headers of each section to generate training and test datasets for the Semantic Section Classifier. We stratified the dataset to balance class samples, where each class had 5000 sample. Later we split the dataset into training and test using $5fold$ cross validation approach to ensure the randomness in training and test dataset. After developing the training and test datasets, we gave special emphasis on test dataset. We manually went through the test dataset, checked the class labels and corrected if there was any wrong class label. This implies that our test dataset has good quality.

4.5.4 Data For Section Sequencing

We took all section headers for each document in a sequential order. Then we mapped them to the classes of our document ontology for *arXiv* articles. As we processed section headers in a sequential order from a document, we could predict the

next section header based on the previous sequence using the approach explained in the Technical Approach chapter. We developed training and test datasets for sequence prediction in a section sequence. After data generation, the whole dataset was split into training and test dataset with a stratified sampling. The training and test dataset sizes are 86991 and 16439 respectively. We manually went through the test dataset and checked each sample and, corrected if it was needed. This implies a good test dataset for evaluation.

4.5.5 Data For Section Summarization

After detecting the section boundaries, we used state of the art techniques to generate extractive summarization for each of the sections. To generate abstractive summarization using the Tensorflow TextSum model, we developed training and test datasets from extractive summarization. Extractive summarization for each section was considered an annotated summary and was used for training and test. We also used $5fold$ cross validation for splitting the dataset. The total training and test sections for summarization are shown in table 4.8.

Table 4.8: Training and Test Sections for Summarization

Dataset	**Sample Number**
Training	618276
Test	117876

Subsequently we prepared the dataset for Tensorflow Textsum models and we read each section and its summary. Both section and summary were split into sentences. Each sentence was included inside $< s >< /s >$ tag. A list of $< s >< /s >$ is

generated based on all sentences, which was inserted in $< d >< p >< /p >< /d >$ tags sequences. Finally, we encoded each section and summary as article and title respectively in a binary formatted file. We also built a vocabulary list from the whole dataset. The total vocabulary size was 241657.

4.5.6 Data For Ontology Design

We retrieved all section headers from TOCs of all `arXiv` articles and applied some heuristics to remove numbers, dots, etc from all headers. Later we lowered the case for all headers. The total number of unique section headers was 3364668 for all the categories of articles. We used these section headers to get classes for ontology design as explained in the Technical Approach chapter. We also retrieved section headers from only Computer Science articles. After applying the similar approach, we found 666877 unique section headers from the category of Computer Science articles. The results for all categories, as well as Computer Science, are described in the Experiments chapter.

Chapter 5

EXPERIMENTS AND EVALUATION

In this chapter, we will discuss experimental setup followed by the detailed procedures. We will describe the results and the findings of each experiment and illustrate the results using comparative analysis with the baseline system and other existing researches. We will explain the performance of the developed models together with critical discussion and evaluation.

5.1 Experiments for Line Classification

As explained in the Technical Approach chapter, we used SVM, DT, NB, RNN and CNN algorithms for our line classification. Since a document has very few section headers with respect to regular texts, after layout features generation we balanced our datasets. The detailed information is given in the Input Document Processing chapter. The initial experiments and results are presented in our work [78]. More detailed experiments using each of the algorithms are described below.

5.1.1 Using DT

We used the Decision Tree Classifier from scikit-learn [74] to implement DT models for the line classification. During training, we used $gini-impurity$ for splitting data points into branches. The first model was trained on layout feature vector, the

second model was trained on the combined layout and text feature vectors. As explained in the Technical Approach chapter, layout features were generated from layout meta information and text features were generated using n-gram language models, such as unigram, bigram and trigram. We used the $TF - IDF$ vectorizer for *vectorizing* the texts. The configuration for DT is shown in table 5.1.

Table 5.1: Configuration: Decision Tree Algorithms

Attribute	**Value**
criterion	gini
algorithm	CART
features	only layout, layout and text
minimum doc frequency	5%
maximum doc frequency	95%
vectorizer	TF-IDF vectorizer
ngram	unigram, bigram and trigram

While generating text features, we considered a word or a phrase as a feature if it occurred in at least 5% and at most 95% of the training data. For the combined layout and text features, we made a feature union of layout vector and text vector using the scikit-learn pipeline module. We also removed English stop words before applying the vectorizer to the texts.

5.1.1.1 Results and Evaluation

To evaluate the performance of DT models, we used precision (positive predictive value), recall (sensitivity) and f1-score (harmonic mean of precision and recall) using the test dataset. Table 5.2 shows the precision, recall, and F1-score for the model trained using only the layout feature vector. The precision, recall and f1-score for the model

trained using the combined layout and text feature vectors are shown in table 5.3.

Table 5.2: Precision, Recall and F1-score for DT Using Only Layout Features

Class Label	Precision	Recall	F1-score
Regular-Text	0.92	0.97	0.95
Section-Header	0.97	0.92	0.95
Avg	0.95	0.95	0.95

Table 5.3: Precision, Recall and F1-score for DT Using Combined Layout Features and Text Features

Class Label	Precision	Recall	F1-score
Regular-Text	0.88	0.97	0.92
Section-Header	0.96	0.87	0.91
Avg	0.92	0.92	0.92

These two tables show that a better performance was achieved for the model trained using only the layout feature vector. This is because the DT works better for a rule based system, and the layout feature vector was generated based on heuristic rules. For the layout feature vector, we also generated feature importance, which is shown in table 5.4.

Table 5.4: Top Five Features for DT Model Using Layout Feature Vector

Feature	Rank	Score
number_dot	1	0.759395
voc	2	0.080806
colon	3	0.060793
header_0	4	0.034286
text_len_group	5	0.020130

5.1.2 Using SVM

Similar to the DT, we built models using the SVM algorithm for the layout feature vector, and the combined layout and text feature vector. We produced both vectors using

the same way as the DT model. The configuration of the SVM implementation is shown in table 5.5. In this experiment, we chose linear kernel, since we had two class labels for the Line Classification.

Table 5.5: Configuration: SVM Algorithms

Attribute	Value
kernel	linear
regularization	l2
features	only layout, layout and text
minimum doc frequency	5%
maximum doc frequency	95%
vectorizer	TF-IDF vectorizer
ngram	unigram, bigram and trigram

5.1.2.1 Results and Evaluation

We used the precision, recall and f1-score for the evaluation of the models generated using SVM. Precision, recall and f1-score for the test dataset using the layout feature vector are shown in Table 5.6. Table 5.7 shows the precision, recall and f1-score for the combined layout feature and text feature vector. Table 5.6 and 5.7 show that we accomplished a better performance using only the layout feature vector.

Table 5.6: Precision, Recall and F1-score for SVM Using Only Layout Feature Vector

Class Label	Precision	Recall	F1-score
Regular-Text	0.93	0.97	0.95
Section-Header	0.97	0.92	0.94
Avg	0.95	0.95	0.95

Table 5.7: Precision, Recall and F1-score for SVM Using Combined Layout Feature and Text Feature Vectors

Class Label	Precision	Recall	F1-score
Regular-Text	0.92	0.93	0.93
Section-Header	0.93	0.92	0.93
Avg	0.93	0.93	0.93

5.1.3 Using NB

We also trained line classification models using the NB algorithm. Similar to DT and SVM models, we used the layout feature vector, and the combined layout feature and text feature vectors. Using scikit-learn, we implemented the Multinomial Naive Bayes architecture for the NB models. The complete configuration for the NB is displayed in Table 5.8.

Table 5.8: Configuration: Naive Bayes Algorithms

Attribute	Value
algorithm	MultinomialNB
features	only layout, layout and text
minimum doc frequency	5%
maximum doc frequency	95%
vectorizer	TF-IDF vectorizer
ngram	unigram, bigram and trigram

5.1.3.1 Results and Evaluation

We also evaluated the performance of the NB models using precision, recall and f1-score. Table 5.9 and Table 5.10 show the precision, recall and f1-score for the models trained using the layout feature vector, and the combined layout feature and text feature vector respectively. Similar to DT and SVM, the NB model trained using the layout

feature vector had better performance over the combined layout feature and text feature vectors.

Table 5.9: Precision, Recall and F1-score for NB Using Only Layout Feature Vector

Class Label	Precision	Recall	F1-score
Regular-Text	0.88	0.72	0.79
Section-Header	0.76	0.90	0.82
Avg	0.82	0.81	0.81

We obtained a poor performance using NB models compared to DT and SVM. The reason is that NB makes a strong assumption that any two features from the feature vectors are independent. But features might not be independent since a section header could be bold while it had dots.

Table 5.10: Precision, Recall and F1-score for NB Using Combined Layout Feature and Text Feature Vectors

Class Label	Precision	Recall	F1-score
Regular-Text	0.85	0.67	0.75
Section-Header	0.73	0.89	0.80
Avg	0.79	0.78	0.77

5.1.4 Using RNN

We used the Tensorflow [4] deep learning framework to build models using RNN. Three different networks were developed for text-only, layout-only, and combined layout and text input vectors. The texts were converted into a character level *one-hot* vector. We used a character level RNN model to capture character patterns in an input sequence. The *one-hot* vector was passed into the $tanh$ layer and the output of the $tanh$ layer was passed through hidden layers. Finally, the output of the last fully connected

hidden layer, which we called header embedding, was passed through the $softmax$ layer for classification. The network configuration for the RNN network is given in Table 5.11.

Table 5.11: Configuration of RNN Models for Line Classification

Attribute	Value
max_doc_len	100
hidden_size	20
encoding	one-hot
optimizer	adma
learning_rate	0.001
objective_function	softmax
batch_size	100

For layout only RNN network, the layout vector, having 16 features for each line, was converted into a multi-label *one-hot* vector. Later the vector was given to the network for classification using the same architecture designed for text-only input vector. For the third RNN network, we concatenated text-only and layout-only vectors. The final vector was passed into the $tanh$ layer followed by a hidden layer. The output of the last fully connected hidden layer was passed through the $softmax$ layer for classification. The architecture was presented in Figure 3.4 in the Technical Approach chapter.

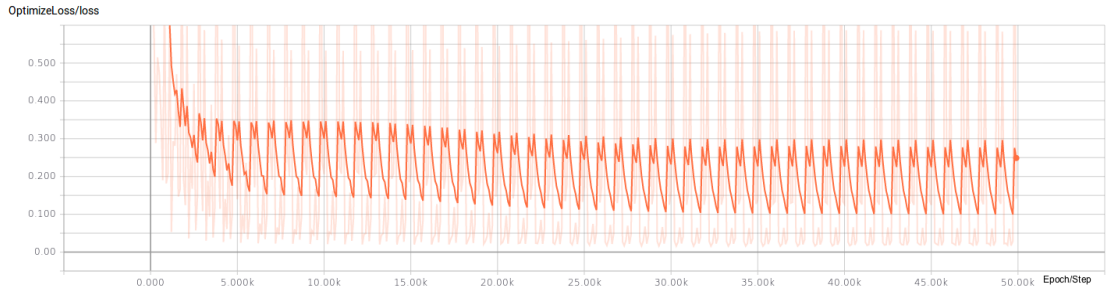
5.1.4.1 Results and Evaluation

To evaluate the performance of the RNN models, we calculated precision, recall, and f1-score for the test data using the models trained on the training data. Table 5.12 shows the performance of RNN models using three different input vectors. We observed that an RNN model trained using the combined text and layout vectors had the best

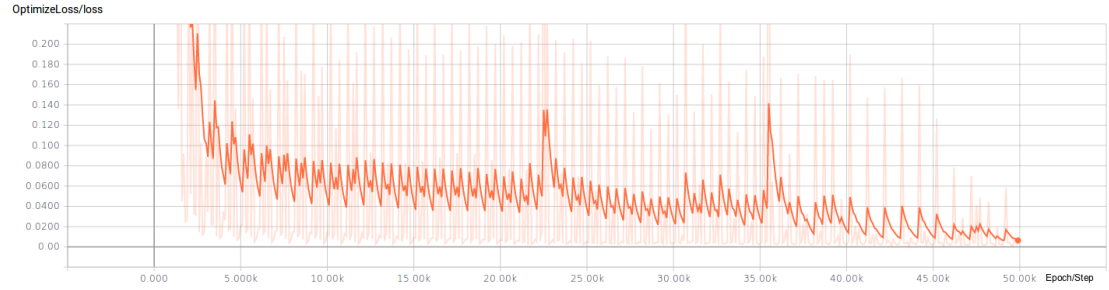
performance. An intuitive reason for having the best performance using the combined feature vectors is that the network captures more information from two different vectors. Figure 5.1 shows the training losses for layout-only, text-only, and combined layout and test input vectors using RNN models.

Table 5.12: Precision, Recall and F1-score for RNN Models for Line Classification

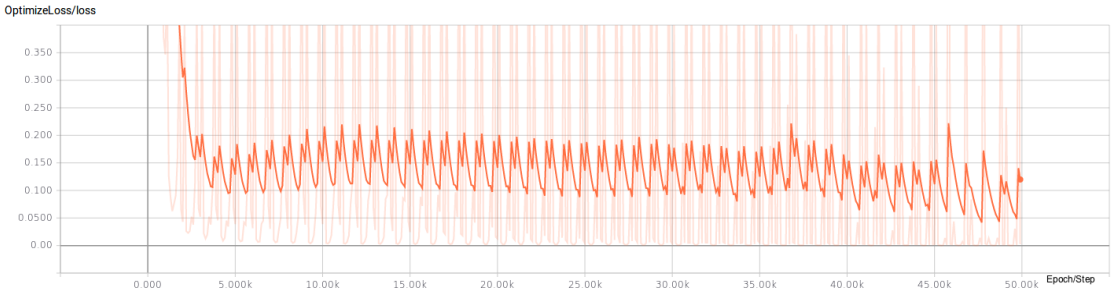
RNN Model	Class Label	Precision	Recall	F1-score
Text Features	**Regular-Text**	0.94	0.95	0.95
	Section-Header	0.95	0.94	0.94
	Avg	0.94	0.94	0.94
Layout Features	**Regular-Text**	0.94	0.94	0.94
	Section-Header	0.94	0.94	0.94
	Avg	0.94	0.94	0.94
Combined Text and Layout Features	**Regular-Text**	0.95	0.95	0.95
	Section-Header	0.95	0.95	0.95
	Avg	0.95	0.95	0.95



(a) Only Layout



(b) Only Text



(c) Combined Layout and Text

Figure 5.1: Training Losses for Line Classification using RNN Models

5.1.5 Using CNN

We trained CNN models as described in the Technical Approach chapter for text-only, layout-only, and combined text and layout vectors. Texts were converted into character level *one-hot* vector where the maximum input length was 100 characters. If the length was more than 100, it was truncated. If the length was less than 100, it was

padded with 0. The hyperparameters for the CNN models are given in Table 5.13. Since we used *categorical_crossentropy* as a loss function, we converted the class vector into a categorical vector using *keras* deep learning utility.

Table 5.13: Configuration of CNN Models for Line Classification

Attribute	Value
max_doc_len	100
hidden_dim	100
encoding	one-hot
optimizer	adma
learning_rate	0.001
objective_function	softmax
batch_size	128
filter_size	250
kernel_size	3
loss_function	categorical_crossentropy

We mapped the *one-hot* input vector into the embedding vector. Before passing the embedding vector through the $conv1D$ layer, we applied dropout to reduce $overfitting$ in CNN models. Later we applied $MaxPooling$ to get the maximum value as output from the previous layer. This technique was applied to reduce the number of features effectively, which helped to prevent $overfitting$ in our models. Thereafter, we had a fully connected hidden layer, which was followed by a $softmax$ layer to classify each line.

In a similar way, we trained the CNN model for the layout-only input vector. In this case, the layout vector was converted into a multi-label binary encoding vector, which was passed into the embedding layer. Later a $conv1D$ layer was applied on the output of the embedding layer. A fully connected hidden layer was applied on the output of the $conv1D$ layer after passing through a $MaxPooling$ layer. Finally, we

used a $softmax$ layer for the classification of the output of the last fully connected hidden layer.

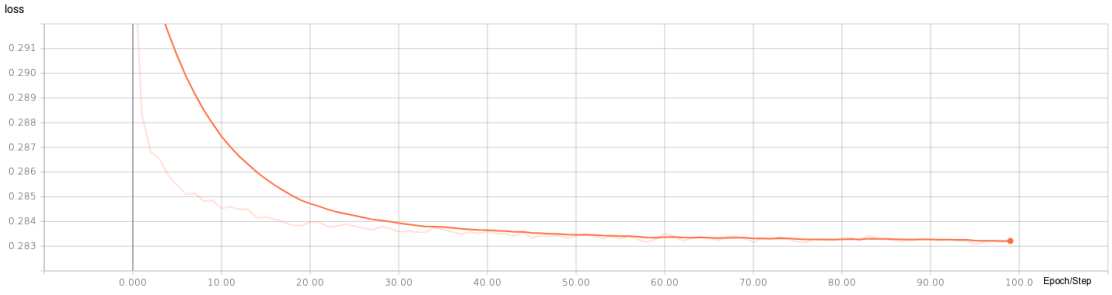
Using two parallel sequential layers, we trained a CNN model for combined text and layout input vectors, where both layers were merged before the last fully connected hidden layer. One sequential layer had the text vector and another one had the layout vector as input. Due to the merging of two parallel, sequential layers, the last hidden layer had two hundred dimensions. The output of the last hidden layer was passed through a $softmax$ layer for line classification.

5.1.5.1 Results and Evaluation

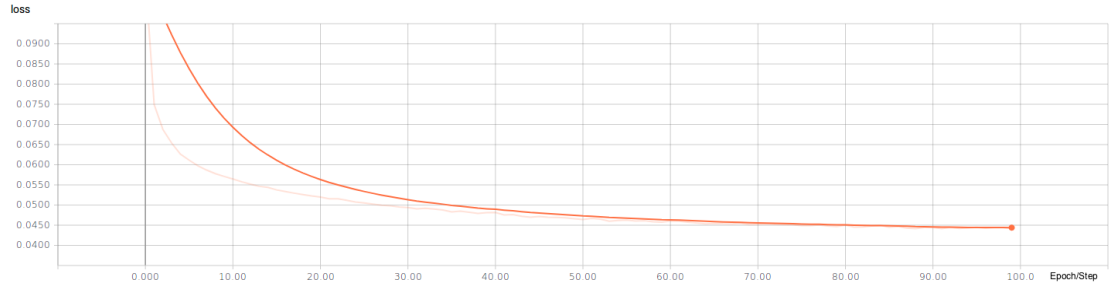
To evaluate the performance of the CNN models, precision, recall, and f1-score were calculated using the test dataset. Table 5.14 shows the average precision, recall, and f1-score for all three CNN models. We also calculated test accuracy for each of the models. Table 5.15 shows accuracy of each of the CNN models for line classification. From the results presented in Table 5.14 and 5.15, we observed that the CNN model using combined text and layout vectors had the best performance. A CNN model trained using only the layout vector had poor performance compared to other two models. We achieved better performance for the CNN model using the text-only input vector since CNN was able to learn important patterns from character sequences. Figure 5.2 shows the training losses for line classification using CNN models.

Table 5.14: Avg. Precision, Recall and F1-score for CNN Models for Line Classification

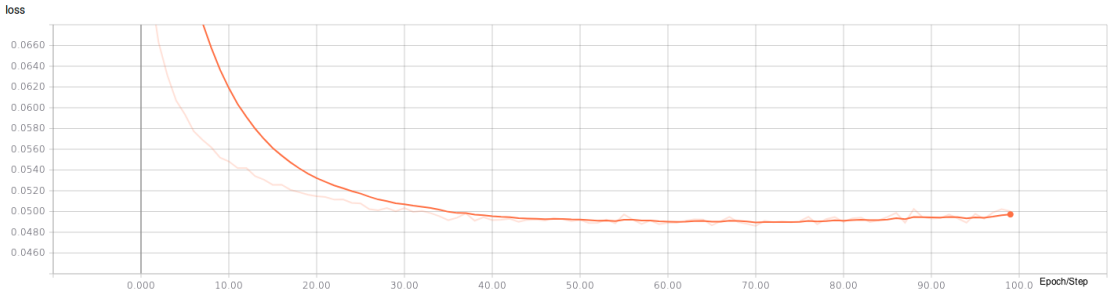
Model	Precision	Recall	F1-score
Text	0.97	0.96	0.96
Layout	0.91	0.84	0.87
Combined Text and Layout	0.98	0.95	0.97



(a) Layout



(b) Text



(c) Combined Layout and Text

Figure 5.2: Training Losses for Line Classification using CNN Models

Table 5.15: Test Accuracies for CNN Models for Line Classification

Model	Accuracy
Text	0.96
Layout	0.87
Combined Text and Layout	0.97

5.1.6 Discussion

From all of the experiments presented above for line classification, it is observed that we achieved the best performance using the CNN model with combined text and layout input vectors. We can also conclude that deep learning models had better performance over regular machine learning models for line classification. Deep learning models had the best performance because both RNN and CNN were able to learn important and complex features automatically. Figure 5.3 shows the performance comparison over all of the models for line classification.

5.2 Experiments for Section Classification

As explained in the Technical Approach chapter, we used RNN and CNN algorithms for our section classification. We also explained the reasons for choosing RNN and CNN for section classification in section 3.4.2.1 of the Technical Approach chapter. Training and test data processing were described in the Input Document Processing chapter. Experiments using each of the algorithms are described below.

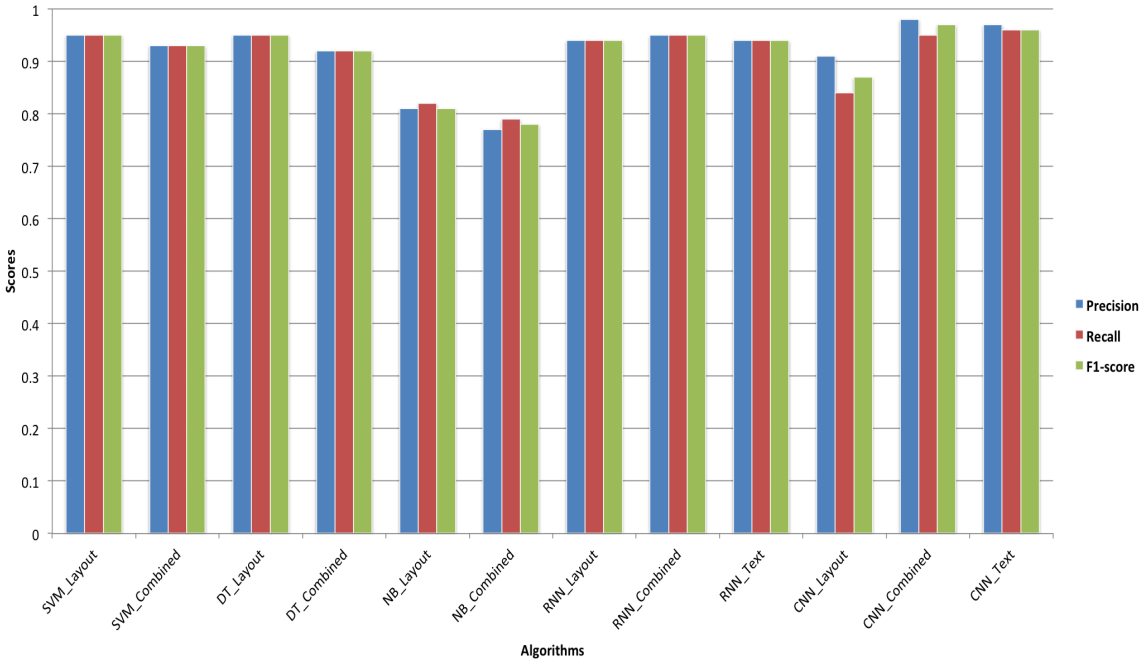


Figure 5.3: Performance Comparison for Line Classification

5.2.1 Using RNN

After identifying each line as a *Regular-Text* or *Section-Header*, we built RNN models to classify each *Section-Header* as a *Top-level*, *Subsection* or *Sub-subsection* header. We used text-only, layout-only, and combined text and layout as input vectors to train three different RNN models. Similar to the Line Classifiers, we converted each section header into character level *one-hot* vector. The layout vector was represented using 16 layout features.

For text-only input vector, initially the *one-hot* vector was passed into the $tanh$ layer. Thereafter, the output of the $tanh$ layer was passed into a fully connected hidden layer followed by a $softmax$ layer to classify section headers into *Top-level*, *Subsection* and *Sub-subsection* headers. For the layout-only input vector, the layout vector was

mapped in the embedding space followed by a $tanh$ layer. The output of the $tanh$ layer was passed through a fully connected dense layer followed by a $softmax$ layer to classify each input vector into any of the classes.

For the combined text and layout input vectors, we had two parallel sequential layers using the section header and layout. Section headers were converted into a *one-hot* vector, which was passed through an embedding layer with a $tanh$ activation function and dropout regularization technique. The output of the embedding layer was passed through a fully connected hidden layer with a $ReLu$ activation function. After applying an embedding layer on the layout vector, it was passed into a fully connected hidden layer. The output from both hidden layers were merged together and passed through the last fully connected hidden layer, which was later classified by the $softmax$ layer. We used *categorical_crossentropy* as a loss function and *adam* as an optimizer.

5.2.1.1 Results and Evaluation

Using precision, recall, and f1-score, we evaluated the performance of three RNN models for section classification. Table 5.16 shows the performance of RNN models. We achieved the best performance using the combined text and layout input vector. We observed that RNN models using layout-only input vector had poor performance compared with the two other models. Figure 5.4 shows a *T-SNE* visualization of embedding vector generated by RNN model using combined text and layout input vectors. After analyzing the visualization, we found same level of section headers to be grouped to-

gether. We also observed that similar section headers are plotted near by each other. For example, result and observation sections were close by in the embedding space.

Table 5.16: Precision, Recall and F1-score for Section Classification using RNN

Model	Class	Precision	Recall	F1-score
Text	**Top-level**	0.81	0.89	0.85
	Subsection	0.84	0.79	0.82
	Sub-subsection	0.77	0.74	0.76
	Avg	0.81	0.81	0.81
Layout	**Top-level**	0.39	0.94	0.55
	Subsection	0.62	0.15	0.24
	Sub-subsection	0.63	0.22	0.33
	Avg	0.55	0.44	0.38
Combined Text and Layout	**Top-level**	0.85	0.95	0.89
	Subsection	0.82	0.84	0.83
	Sub-subsection	0.85	0.74	0.79
	Avg	0.84	0.84	0.84

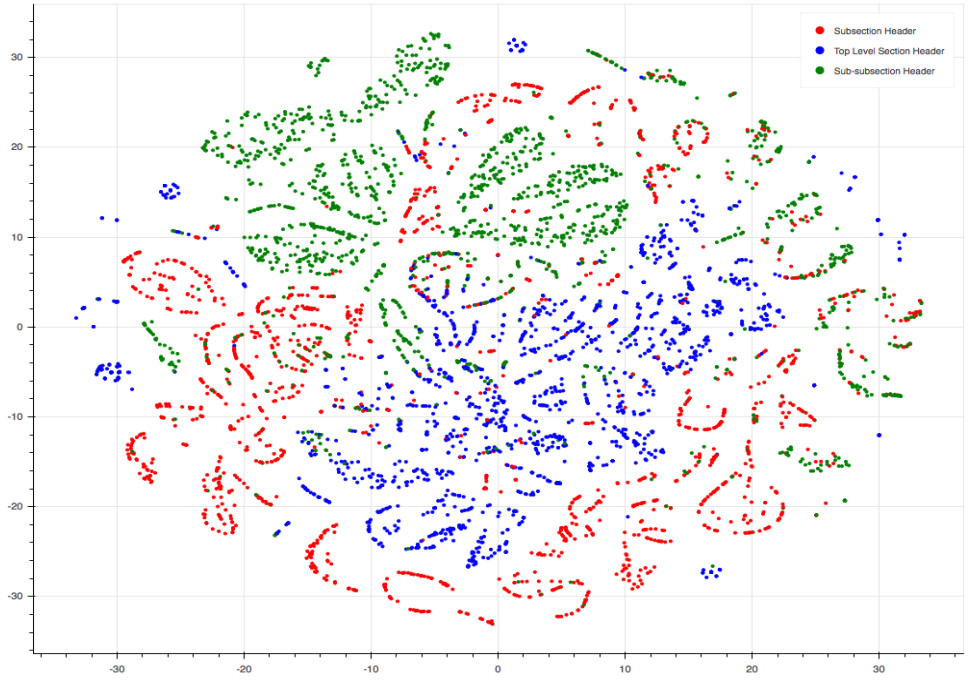


Figure 5.4: T-SNE Visualization of Embedding Vector using RNN for Section Classification

5.2.2 Using CNN

Similar to RNN models, we trained CNN models for section classification using text-only, layout-only, and combined text and layout input vectors. We also converted section headers into a *one-hot* vector and layout information into a multi-level *one-hot* vector using binary encoding. For the text-only CNN model, we built the network as explained in the Technical Approach chapter. *One-hot* vector was given as input to the CNN model, where the embedding layer mapped it to automatic feature set. The output of the embedding layer was passed through a $conv1D$ layer with a $ReLu$ activation function and dropout regularization technique. A $MaxPooling$ layer was applied to get the most important features. Finally a $softmax$ layer was used to classify each section header after applying a fully connected hidden layer on the output of the $MaxPooling$ layer.

For the layout-only input vector, the embedding layer was mapped into a multi-level *one-hot* vector in a feature space, on which a $conv1D$ layer was applied to generate automatic features. Thereafter, a $MaxPooling$ layer was applied followed by a fully connected hidden layer. Finally the $softmax$ layer classified each of the section headers into three different classes.

We built two parallel sequential layers using CNN for the combined text and layout input vectors, where we followed the procedures described above for text-only and layout-only vectors individually. The output of the two parallel layers were merged together and passed through a fully connected hidden layer, which was followed by a $softmax$ layer for classification.

5.2.2.1 Results and Evaluation

We evaluated the performance of CNN models for section classification using precision, recall, and f1-score. Table 5.17 shows the performance of each model, where we achieved the best performance using combined text and layout input vectors. We also observed that model using layout input vector has poor performance compared to the other two models. Figure 5.5 shows the *T-SNE* visualization for the embedding vector generated by the CNN models using combined text and layout input vectors for section classification. Similar to the RNN models, we also obtained that similar level of section headers are grouped together in the embedding space.

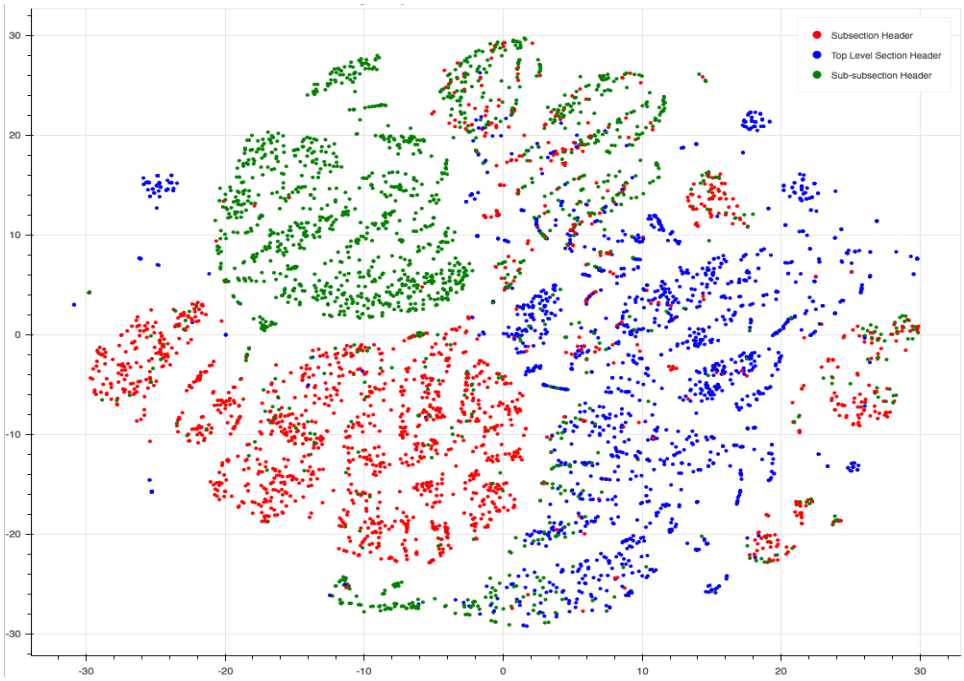


Figure 5.5: T-SNE Visualization of Embedding Vector using CNN for Section Classification

Table 5.17: Precision, Recall and F1-score for Section Classification using CNN

Model	Class	Precision	Recall	F1-score
Text	**Top-level**	0.81	0.90	0.85
	Subsection	0.84	0.82	0.82
	Sub-subsection	0.80	0.73	0.76
	Avg	0.83	0.82	0.82
Layout	**Top-level**	0.36	0.98	0.53
	Subsection	0.71	0.08	0.15
	Sub-subsection	0.59	0.11	0.18
	Avg	0.55	0.39	0.29
Combined Text and Layout	**Top-level**	0.82	0.94	0.88
	Subsection	0.83	0.84	0.83
	Sub-subsection	0.86	0.72	0.78
	Avg	0.83	0.84	0.83

5.2.3 Using CNN for Four Class

We also trained a CNN model for section classification as four class classification problem, where the classes are *Regular-Text*, *Top-level*, *Subsection* and *Sub-subsection* headers. The model was trained based on text-only, and combined text and layout feature vectors. The network architecture and experimental procedures were similar to the three class experiments using CNN.

5.2.3.1 Results and Evaluation

For the evaluation purposes, we assessed the CNN model with the text-only input vector. Table 5.18 shows the average precision, recall, and f1-score for the four class CNN model for section classification. Compared with Table 5.17 and 5.14, we observed that a pipeline approach of line and section classifiers performed better than the single four class classifier. Figure 5.6 shows the *T-SNE* visualization of the embedding vector generated by the CNN model for the four class section classification using the text-only

input vector. We analyzed the embedding visualization compared to the three class section classification presented in Figure 5.5 and observed that we achieved better class separation in the pipeline approach.

Table 5.18: Precision, Recall and F1-score for Four Class CNN for Section Classification

Model	**Precision**	**Recall**	**F1-score**
Text	0.8430	0.8442	0.8415

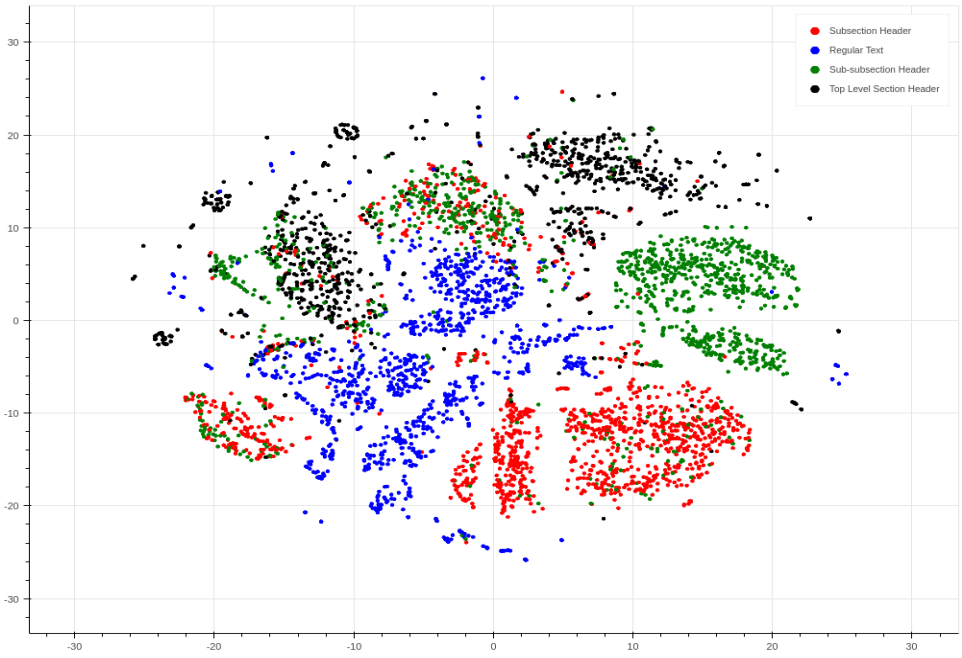


Figure 5.6: T-SNE Visualization of Embedding Vector using CNN for Four Class Section Classification

5.2.4 Discussion

After analyzing the performance of the models trained by RNN and CNN using text-only, layout-only, and combined text and layout input vectors, we achieved the best performance using combined input vectors. Figure 5.7 shows the performance comparison of all of the models trained by RNN and CNN for section classification. Models trained by CNN and RNN using the combined text and layout input vectors had almost similar performance. We achieved poor performance using the layout-only vector. The reason is that many section headers have similar layout information though they are from different classes. For example, a *top-level* section header and a *subsection* header might be bold with the same indentation. From Figure 5.4 and 5.5, we observed that some *sub-subsection* headers were plotted near by *top-level* section headers because in some articles, *sub-subsection* headers started with a single number or letter. The CNN and RNN models sketched those in the same embedding space.

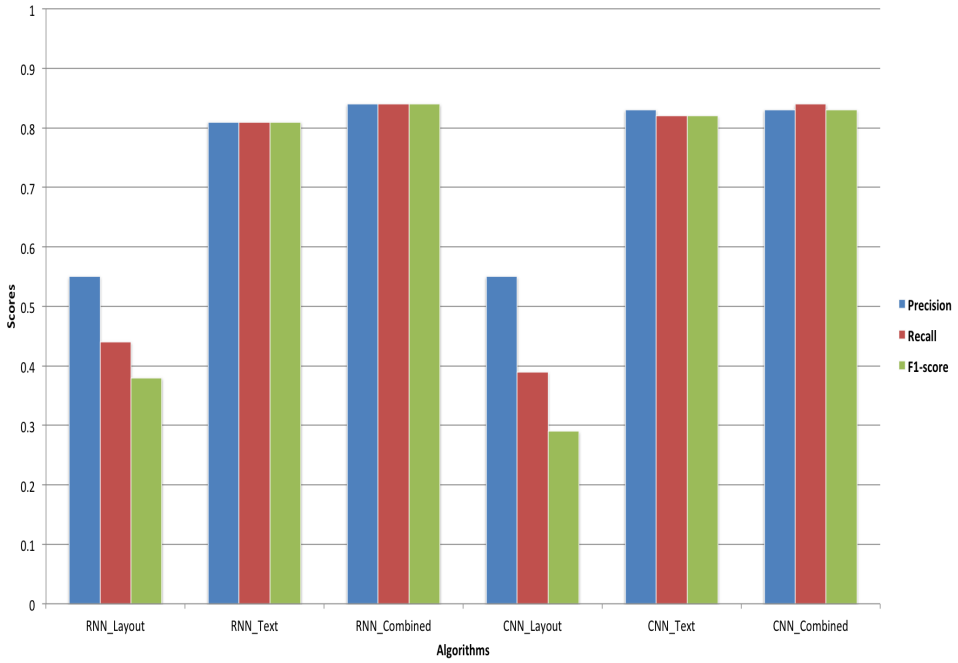


Figure 5.7: Performance of RNN and CNN for Section Classification

5.3 Experiments for Semantic Section Classification

After identifying the different levels of section headers, we applied the Section Boundary Detector 1 algorithm in order to split a document into different sections, sub-sections, and sub-subsections. We used semantic section classifiers to assign a human understandable semantic label for each physically divided section. We then built semantic section classifier models using CNN and bidirectional LSTM algorithms based on both word and character level inputs.

5.3.1 Using CNN

As explained in the Technical Approach chapter, we trained CNN models using sections obtained from section classifiers. We had 20 classes which were mentioned in the Table 3.2 of the Technical Approach chapter. The models were trained based on both word and character level inputs. For the word based CNN model, we considered the first two hundred words for each section. The total vocabulary size was 111084 words generated from all training samples with a minimum frequency of 150. The input texts were converted into a multi-label *one-hot* vector, which was passed into the embedding layer to map each word in the embedding space. A dropout layer was applied to the output of the embedding layer to avoid overfitting. Later, a $conv1D$ layer was applied with $ReLu$ activation function, and followed by a $maxpooling$ layer and a fully connected hidden layer. Finally, a $softmax$ layer, with an $adam$ optimizer and a $categorical\_crossentropy$ loss function, was used to classify the output of the last fully connected hidden layer.

For the character based CNN model, we considered the first 600 characters from each section and converted them into a multi-label *one-hot* vector, which was input to the embedding layer. The rest of the layers were similar to the layers of the word based CNN model. In this model, the total vocabulary size was 256.

5.3.1.1 Results and Evaluation

To evaluate the performance of the CNN models trained for semantic section classification, we used precision, recall, and f1-score, which are shown in Table 5.19.

We achieved better performance using the word based CNN model. This is because the word based model was able to capture the semantic meaning of different words. For some classes, the model wasn't able to classify any instance, such as background, datasets, and implementation. We analyzed the results and obtained that sections of these classes usually describe various concepts, and hence the model was unable to get the semantic meaning from those sections. We also achieved very high precision and recall for some classes, such as acknowledgements, references, abstract, and introduction. After analyzing sections for these classes, we found that sections for these classes have semantic patterns.

Figure 5.8 shows the *T-SNE* visualization of section embedding with different classes. After analyzing the visualization we can say that some of the sections were well separated and surrounded by semantically similar sections. We also achieved 0.79 and 0.77 test accuracy using word based CNN and character based CNN models for semantic section classification respectively.

Table 5.19: Precision, Recall and F1-score for Semantic Section Classifier using CNN

Model	Class	Precision	Recall	F1-score
Word Based	Avg	0.72	0.75	0.73
Character Based	Avg	0.69	0.72	0.70

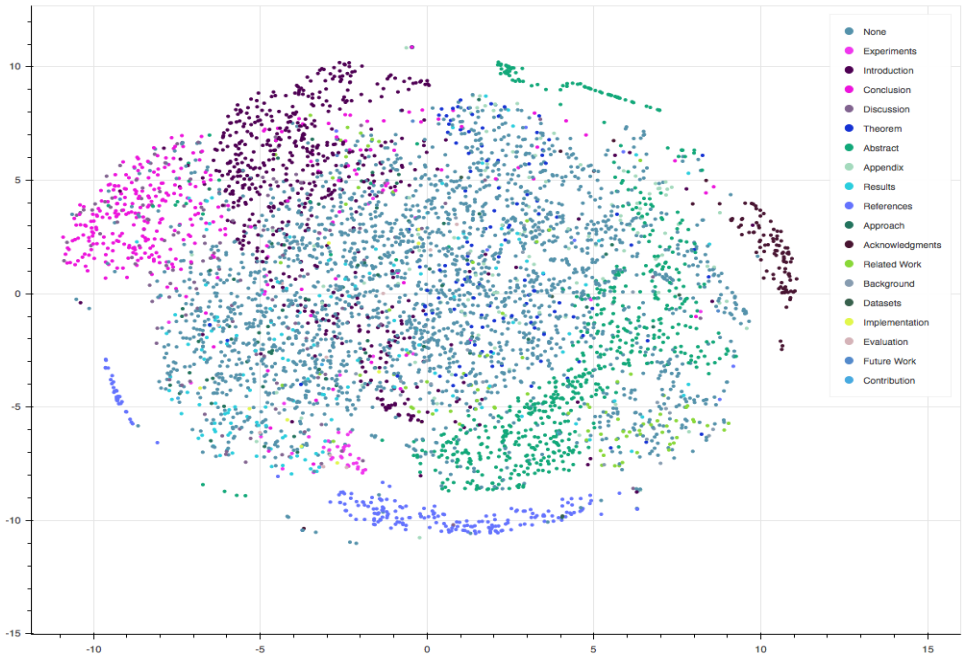


Figure 5.8: T-SNE Visualization of Semantic Section Embedding using Word Based CNN

5.3.2 Using Bidirectional LSTM

In the bidirectional LSTM model, we converted texts from each section into a multi-label *one-hot* vector, which was input to the embedding layer. The word based network had the word level *one-hot* vector and the character based network had the character level *one-hot* vector generated from texts. The output of the embedding layer was input to the forward LSTM units in a sequential order and a reversed sequence was input to the backward LSTM units. The outputs from both forward and backward LSTM units were concatenated and passed into a fully connected hidden layer. Finally, the output of the last fully connected hidden layer was passed into the $softmax$ layer for classifi-

cation. The $softmax$ layer had $adam$ as an optimizer and $categorical\_crossentropy$ as a loss function. We also applied dropout at each LSTM unit to avoid $overfitting$.

5.3.2.1 Results and Evaluation

The precision, recall, and f1-score for bidirectional LSTM models are given in Table 5.20. Figure 5.9 shows the *T-SNE* visualization of semantic section embedding using word based bidirectional LSTM. From this embedding visualization, we observed that most of the sections were scattered all over the embedding space. For example, sections from "acknowledgement" were spread into different regions of the embedding space. From this observation, we could infer that bidirectional LSTM wasn't able to capture the semantic meaning of each section.

Table 5.20: Precision, Recall and F1-score for Semantic Section Classifier using Bidirectional LSTM

Model	Class	Precision	Recall	F1-score
Word Based	Avg	0.71	0.72	0.71
Character Based	Avg	0.68	0.70	0.69

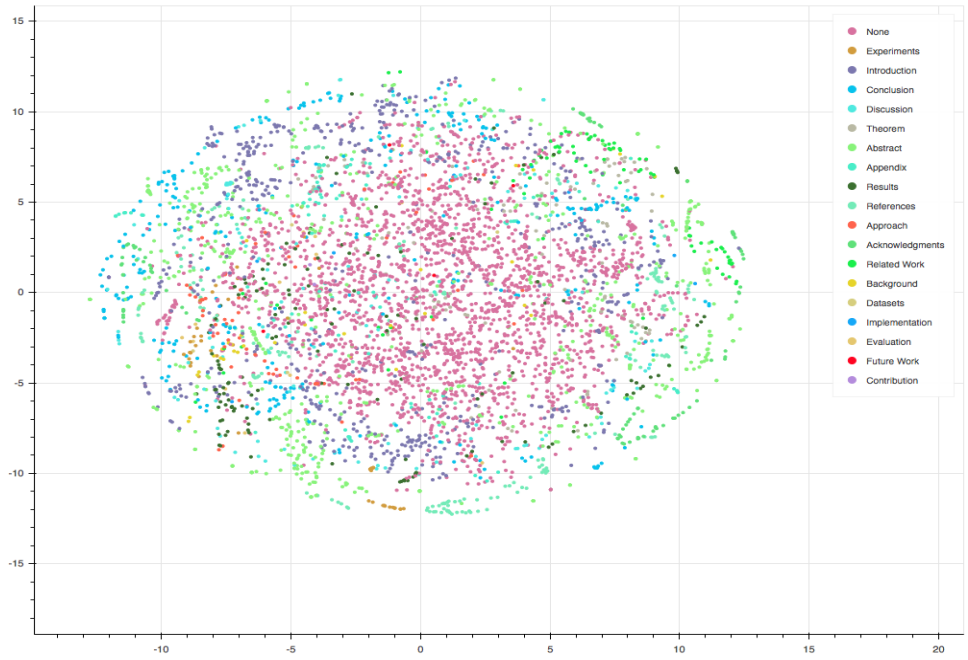
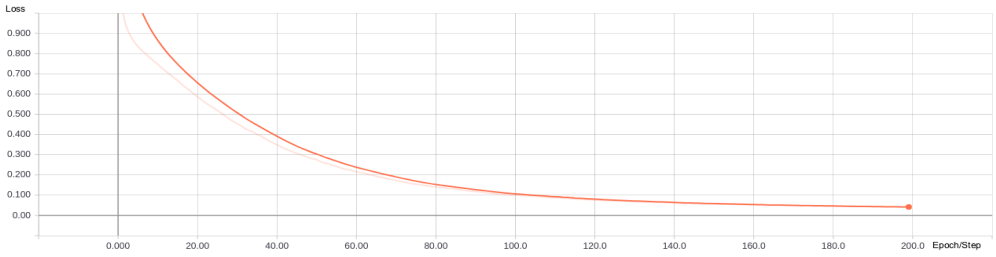


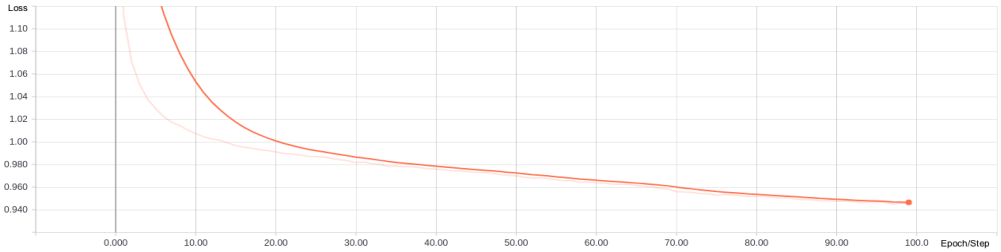
Figure 5.9: T-SNE Visualization of Semantic Section Embedding using Word Based Bidirectional LSTM

5.3.3 Discussion

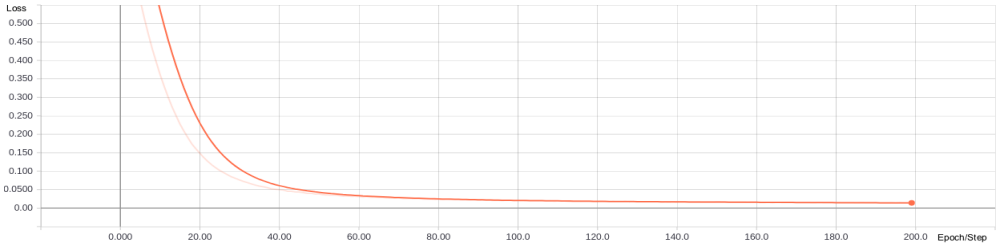
Figure 5.10 shows training losses of CNN and Bidirectional LSTM models for semantic section classification. Although we noticed that the word level bidirectional LSTM model had the lowest training loss among all of the models, we achieved poor performance in the embedding visualization. From this observation, we can infer that the bidirectional LSTM model was overfitted for our training dataset. Figure 5.11 shows the precision, recall, and f1-score comparison for the CNN and Bidirectional LSTM models trained at word and character level input.



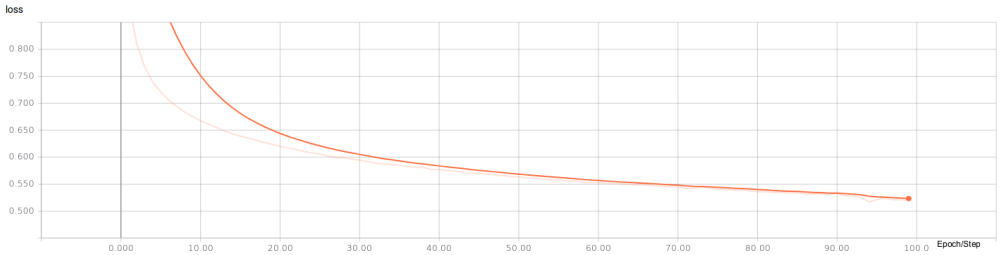
(a) Word Level CNN



(b) Character Level CNN



(c) Word Level Bidirectional LSTM



(d) Character Level Bidirectional LSTM

Figure 5.10: Training Losses for Semantic Section Classification using CNN and Bidirectional LSTM Models

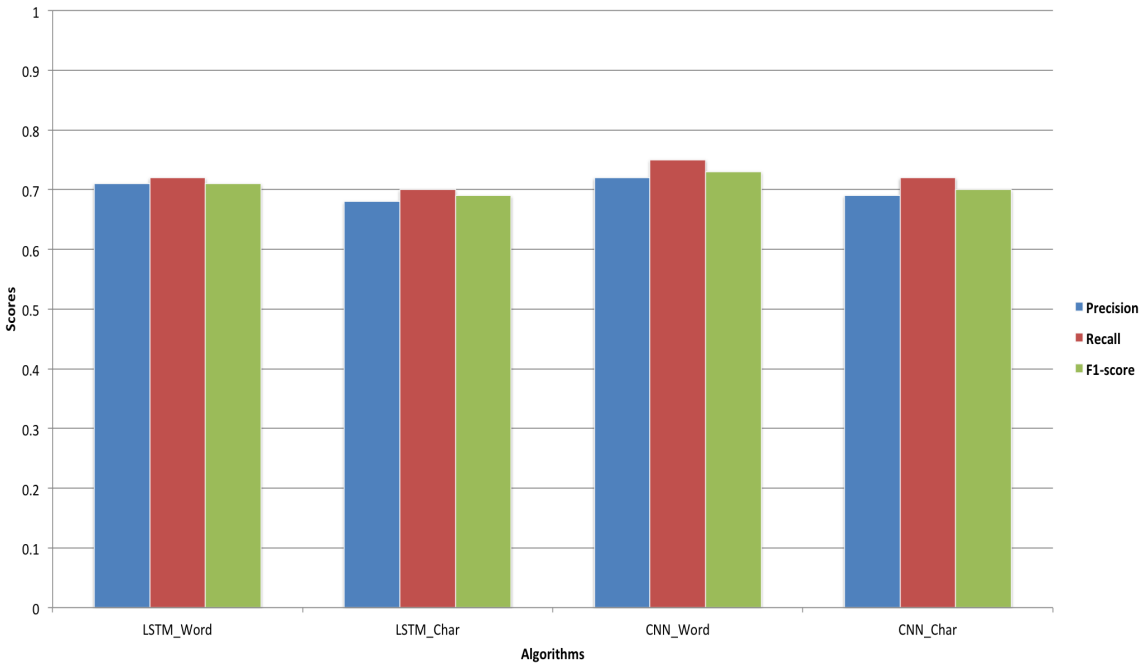


Figure 5.11: Performance Comparison for Semantic Section Classifiers

5.4 Experiments for Section Sequencing

After obtaining all of the semantic annotated sections, we restructured the sequence of sections with semantic names. We trained a sequence prediction model to reproduce the sequence of section headers or semantic names. As a sequence prediction model, we chose LSTM encoder-decoder architecture as sequence prediction model. The detailed experiment is described below.

5.4.1 Using LSTM

We chose an LSTM encoder-decoder architecture since LSTM can remember a long sequence of observations and its encoder-decoder approach can be trained in an

unsupervised way. The model took a sequence of section headers and reproduced the input sequence. We built the model using *Tensorflow* and *Keras* deep learning framework. A document may have any number of sections. Since we were using *arXiv* scholarly articles, we set the threshold for the number of sections to be 15. We truncated the sequence if the length was more than 15 and padded if the length was less then 15. Then we used a $LabelEncoder$ from the *scikit-learn* preprocessing module to encode each of the sequences into a sequence of integer numbers.

In order to feed the input sequence into the LSTM encoder-decoder, we transformed the sequence into a *one-hot* binary vector representation. As a result, our input sequence was converted into a vector of $15x20 = 300$ dimensions, where 15 was the input sequence length and 20 was the number of unique semantic section headers. Thereafter we built a sequential model where we had 20 LSTM memory units followed by a $TimeDistributed$ fully connected $Dense$ layer. We used a $TimeDistributed$ layer since each of the terms in the sequence was considered as an input at a time and the output layer predicted one term at a time. The fully connected $Dense$ layer had $softmax$ activation function. The model was complied using an $adam$ optimizer and a $categorical\_crossentropy$ loss function.

5.4.2 Results and Evaluation

Since we trained our model in an unsupervised way, we don't have the precision, recall and f1-score to evaluate the model. The loss in the validation dataset was 0.000000119, a very low test loss. Figure 5.12 shows the *T-SNE* visualization of sec-

tion sequences in an embedding space. From the Figure 5.12 and the validation loss, we inferred that the LSTM performed very well in our section sequencing and grouped similar section sequences together. The model was tested on the test dataset and a few random sequence of section headers prediction is given in Table 5.21.

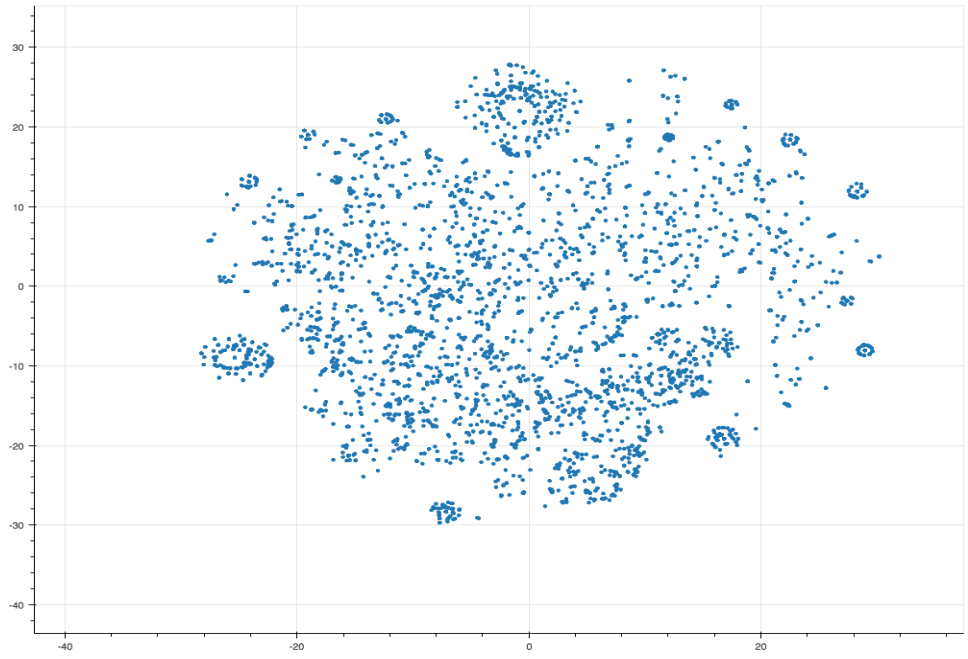


Figure 5.12: T-SNE Visualization of Section Sequencing using LSTM Encoder-Decoder

5.5 Experiments for Section Summarization

One of the outputs of our framework is document summarization. We generated summaries for each of the sections from a document using the $Textrank$ algorithm [61] and $Tensorflow\ Textsum$ [71] model. As described in the Technical Approach chapter, we used $Textrank$ for extractive summarization and $Textsum$ for abstractive

Table 5.21: Sample Sequence of Section Headers Prediction on Test Data using LSTM Encoder-Decoder

Original Sequence	Predicted Sequence
[0, 13, 13, 3, 3, 8, 8, 5, 5, 5, 5, 14, 14, 14, 14]	[0, 13, 13, 3, 3, 8, 8, 5, 5, 5, 5, 14, 14, 14, 14]
[0, 13, 3, 17, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19]	[0, 13, 3, 17, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19]
[0, 13, 4, 14, 14, 14, 14, 14, 14, 10, 16, 5, 15, 19, 19]	[0, 13, 4, 14, 14, 14, 14, 14, 14, 10, 16, 5, 15, 19, 19]
[0, 13, 3, 14, 14, 14, 14, 14, 16, 9, 8, 14, 14, 14, 14]	[0, 13, 3, 14, 14, 14, 14, 14, 16, 9, 8, 14, 14, 14, 14]
[0, 13, 16, 14, 5, 2, 13, 16, 14, 14, 14, 14, 9, 14, 14]	[0, 13, 16, 14, 5, 2, 13, 16, 14, 14, 14, 14, 9, 14, 14]
[0, 13, 18, 15, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19]	[0, 13, 18, 15, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19]
[0, 13, 16, 10, 5, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19]	[0, 13, 16, 10, 5, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19]

summarization.

5.5.1 Extractive Summarization

For each of the sections from a document, we generated an extractive summary using the $Textrank$ algorithm. The algorithm is available in Gensim. We set the ratio at 0.2 to return 20% of the original content as summary. The summary would consist of the most representative sentences from the original texts. As a result, we obtained a short version of the original document with the most informative sentences in each of the sections.

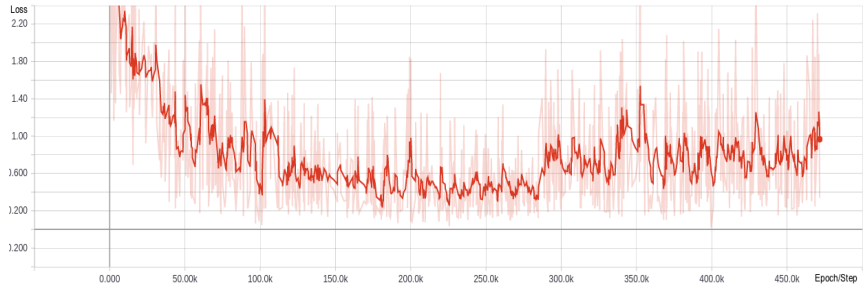
5.5.2 Abstractive Summarization

As explained in the Technical Approach chapter, we used Sequence-to-Sequence with an Attention model implemented in $Tensorflow$ for abstractive summarization.

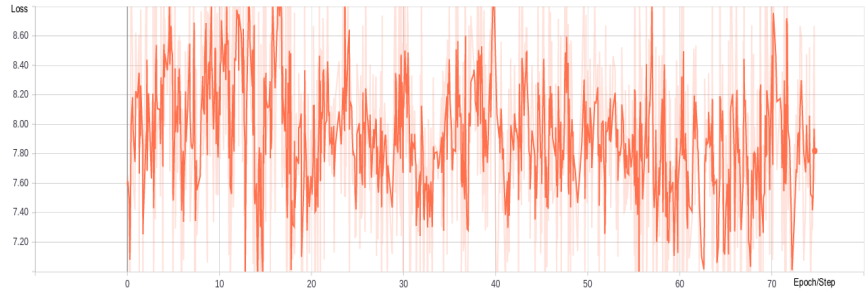
To train a deep learning model for abstractive summaries, we needed human annotated summaries. Due to resource constraints, we treated the extractive summary generated in the previous section as annotated data to train a $Textsum$ model. The training and test dataset development were described in the Input Document Processing chapter.

5.5.3 Results and Evaluation

Usually text summarization is evaluated by comparing automated summaries to human generated reference summaries. Due to resource constraints, we did not have access to human generated reference summaries. We also were not able to achieve evaluation scores by human judgments using a manual evaluation of automated summaries. Figure 5.13a shows the training loss of sequence-to-sequence learning for the $Textsum$ model. After analyzing the loss graph, we observed that the $Textsum$ model had high training loss. This is because the $Textsum$ model works well for short texts such as new headline generation from a few lines of a news article. Evaluation loss for the $Textsum$ model is shown in Figure 5.13b. We noticed that the evaluation loss was oscillating between 6 and 9, which inferred that the model didn't perform well for the test dataset.



(a) Training Loss



(b) Evaluation Loss

Figure 5.13: Loss for sequence-to-sequence TextSum Model

5.6 Experiments for Ontology Design

This section shows the experiments and results analysis for document ontology design and development. Some of the results are presented in our work [79]. A more detailed are described below.

5.6.1 Using Variational Autoencoder

As described in the Technical Approach chapter, we trained a Variational Autoencoder (VAE) to learn the header embedding for ontology design. We clustered the header embedding matrix into semantically meaningful groups and identified different

classes for ontology. The VAE was trained with different configurations and hyperparameters to achieve the best results. We experimented with different input lengths, such as 10, 15 and 20 word length section headers. All section headers were converted into a multi-level *one-hot* vector.

We used 100 embedding dimensions, 100 hidden layers and 1.0 ϵ to learn latent variables. The *one-hot* vector was the input to the network, which was followed by an embedding layer with $ReLu$ activation function. Then we had a dense layer to capture input features in a latent space. The model parameters were trained using two loss functions, which were a reconstruction loss to force the decoded output to match with the initial inputs, and a KL divergence between the learned latent and prior distribution. The decoder was used with a $sigmoid$ activation function and the model was complied with an $rmsprop$ optimizer and KL divergence loss function.

5.6.2 Results and Evaluation

The VAE models were trained in an unsupervised way to capture the semantic meaning of each section header. The output of the VAE embedding layer was dumped and clustered after *T-SNE* dimensionality reduction. Figure 5.14 shows the visualization of *k-means* clustering with $k = 50$ and $inputlength = 15$ for VAE embedding after *T-SNE* dimensionality reduction. Similar visualization with $inputlength = 20$ is shown in Figure 5.15. After analyzing both the Figures, we observed that VAE models learned very well and were able to capture similar section headers together. We noticed that semantically similar section headers were plotted nearby. We also realized that

semantically similar section headers were constructed gradually from one concept to another. For example, we noticed a pattern in the graph where a sequence of concepts from “methods” gradually moved to “data construction”, “results”, “discussion”, “remarks” and “conclusion”. From this analysis, we could infer that VAE learned concepts over section headers in a semantic pattern.

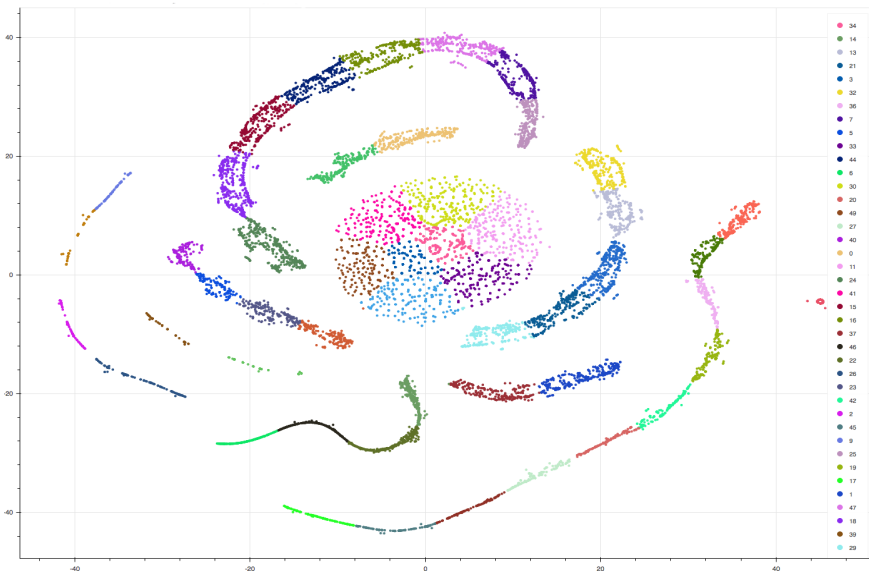


Figure 5.14: T-SNE Visualization of VAE Embedding Matrix Clusters with Input Length 15

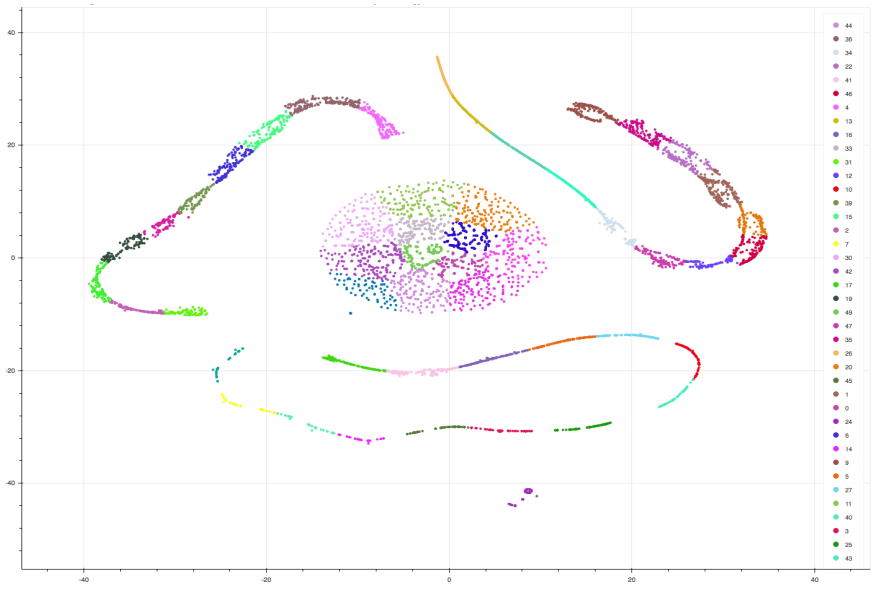


Figure 5.15: T-SNE Visualization of VAE Embedding Matrix Clusters with Input Length 20

We also trained VAE models for section headers from Computer Science articles and RFP documents individually. The performance for Computer Science articles was similar to the performance we achieved for all *arXiv* articles. Due to a fewer number of section headers collected from RFP documents, we obtained different patterns, where most of the section headers where scattered all over the embedding space.

5.7 Experiment for Semantic Concepts

5.7.1 Using LDA

We trained an LDA topic model on a large number of sections to capture a semantic concept for each of the sections. The model was trained using the Gensim frame-

work. The approach was described in the Technical Approach chapter and the input was divided into sections generated from *arXiv* articles. The total number of training and test sections for LDA are given in Table 5.22.

To build the LDA models, we applied different experimental approaches, based on word, phrase and bigram dictionaries. The word-based dictionary contains only unigram terms where as the bigram dictionary has only bigram terms. The phrase based dictionary contains combination of unigram, bigram and trigram terms. All three dictionaries were developed from the training dataset by ignoring terms that appeared in less than 20 sections or in more than 10% of the sections of the whole training dataset. The final dictionary size, after filtering, was $100,000$. Different LDA models were trained based on various number of topics and passes. We ran the trained model to identify a topic for any section, which was used to retrieve top terms with the highest probability. The terms with the highest probability were used as a domain specific semantic concepts for a section.

Table 5.22: Training and Test Dataset for LDA

Dataset	**Number**
Training	128505
Test	11633

5.7.2 Results and Evaluation

To evaluate LDA models, we used test dataset. Figure 5.16 shows the inter topic distance map for ten topics where some of the topics were overlapped. This Figure also shows 30 of the most relevant terms of the topic number 4, where the relevance score is

80%. For performance evaluation of LDA models, we considered perplexity and cosine similarity measures. The perplexity for a test chunk was -9.684 for ten topics. In our experiment, the perplexity was lower in magnitude, which meant that the LDA model fit better for test sections and probability distribution fit better for predicting sections.

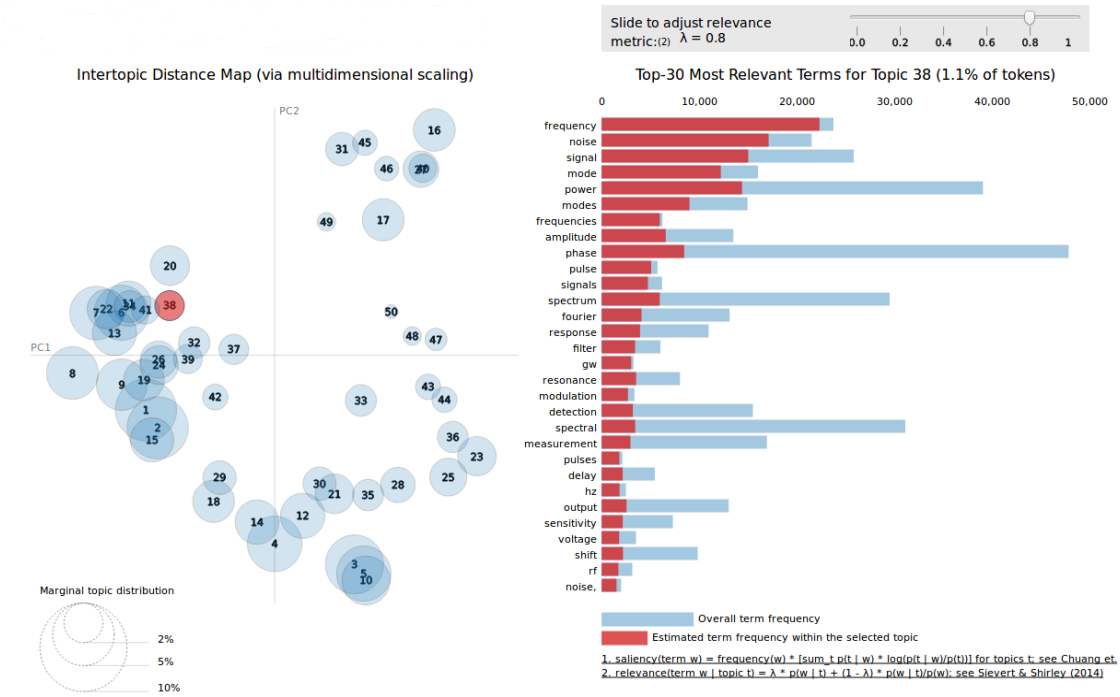


Figure 5.16: Inter Topic Distance Map and Top Terms for a Topic

For cosine similarity measurement, we split the test dataset into ten different chunks of test sections where each chunk had 1000 sections without repetition. We also split each section from each test chunk into two parts and checked two measures. The first measure was a similarity between topics of the first half and topics of the second half for the same section. The second measure was a similarity between halves of two different sections. We calculated an average cosine similarity between parts for each test chunk. Due to coherence among topics, the first measure would be higher

and the second measure would be lower. Figure 5.17 shows these two measures for ten different chunk of test sections.

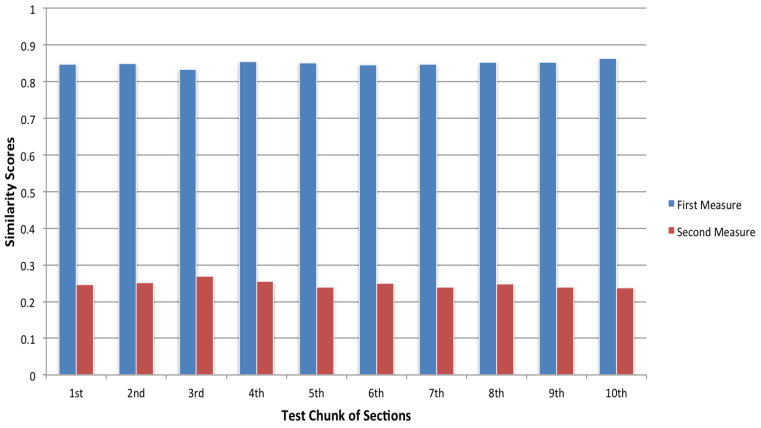


Figure 5.17: Similarity Measures for LDA

Table 5.23: Comparative analysis of LDA models for semantic concepts

arXiv Category	Word based LDA	Bigram based LDA	Phrase based LDA
Mathematics - Algebraic Topology, Mathematics - Combinatorics	algebra, lie, maps, element and metric	half plane, complex plane, real axis, rational functions and unit disk	recent, paper is, theoretical, framework, and developed
Nuclear Theory	phase, spin, magnetic, particle and momentum	form factor, matrix elements, heavy ion, transverse momentum and u'energy loss	scattering, quark, momentum, neutron move and gcd
Computer Science - Computer Vision and Pattern Recognition	network, performance, error, channel and average	neural networks, machine learning, loss function, training data and deep learning	learning, deep, layers, image and machine learning
Mathematical Physics	quantum, entropy, asymptotic, boundary and classical	dx dx, initial data, unique solution, positive constant and uniformly bounded	stochastic, the process of, convergence rate, diffusion rate and walk
Astrophysics - Solar and Stellar Astrophysics	stars, emission, gas, stellar and velocity	active region, flux rope, magnetic reconnection, model set and solar cycle	magnetic ray, the magnetic, plasma, shock and rays

For the evaluation, we also loaded the trained LDA models and generated domain specific semantic concepts from 100 *arXiv* abstracts, where we knew the categories of the articles. We analyzed their categories and semantic terms. We noticed a very interesting correlation between the *arXiv* category and the semantic terms from LDA topic models. We found that most of the top semantic terms are strongly co-related to their original *arXiv* categories. A comparative analysis is shown in Table 5.23. After

manual analysis of the results, we noticed that a bigram LDA model is more meaningful than two other models.

5.8 Experiment on RFP Dataset

Since our initial proposal was on the RFP dataset, we evaluated our models using the RFP dataset. We manually annotated RFP documents as explained in the Input Document Processing chapter. Later, we processed annotated data to prepare a test dataset for text-only, layout-only, and combined text and layout input vectors. The models which had the best performance for line and section classification for the *arXiv* dataset, were used to test the RFP dataset. Since we achieved the best performance using CNN models, we loaded the CNN models for combined text and layout input vectors. The models were tested for both line and section classifications.

5.8.1 Results and Evaluation

To evaluate the performance of the RFP dataset, we calculated precision, recall, and f1-score. Table 5.24 shows the precision, recall, and f1-score for the line classification of the RFP dataset using the CNN model for the combined text and layout input vectors. The precision, recall, and f1-score for section classification of the RFP dataset using the combined text and layout input vectors are given in Table 5.25. The models did not perform as well as they performed for *arXiv* datasets. This is because we developed few features from *arXiv* section headers which are not similar to the RFP section headers, such as “Experiments”, “Dataset” and “Contribution” usually exist in *arXiv*

articles whereas “Requirement”, “Deliverable” and “Contract Clauses” generally exist in RFP documents.

Table 5.24: Precision, Recall and F1-score for Line Classification on RFP Dataset using CNN

Model	Class	Precision	Recall	F1-score
Combined Text and Layout	**Regular-Text**	0.88	0.92	0.92
	Section-Header	0.91	0.90	0.91
	Avg	0.90	0.91	0.91

Table 5.25: Precision, Recall and F1-score for Section Classification on RFP Dataset using CNN

Model	Class	Precision	Recall	F1-score
Combined Text and Layout	**Top-level**	0.75	0.77	0.76
	Subsection	0.79	0.73	0.76
	Sub-subsection	0.82	0.78	0.80
	Avg	0.79	0.76	0.77

5.9 Discussion

We compared the performance of our framework in this chapter with respect to different performance matrices and with the help of different visualization techniques. We also compared the performance of our framework against top performing systems developed for scholarly articles. The first system to be compared was $PDFX$ presented by Constantin et al. in [22]. Our task is partially similar to their task. Their system identifies author, title, email, section headers, etc. from scholarly articles. They reported an f1-score of 0.77 for top-level section headers identified from various articles. We could not evaluate our framework using their dataset since the dataset was not publicly available.

The second system, which we would like to compare our results with, had a hybrid approach by Tuarob et al. [92] to discover semantic hierarchical sections from scholarly documents. Their task was limited to a few fixed section header names whereas our framework identifies any section header. Hence, their dataset may not not directly applicable to our system. They attained a 0.92 f1-score for the section boundary detection where sections were from fixed names, such as abstract, introduction and conclusion.

Chapter 6

CONCLUSION

In this dissertation, we have explored a variety of machine learning and deep learning architectures to understand the logical and semantic structure of large documents. Our framework was able to automatically identify logical sections from an unstructured document, infer their structure, capture their semantic meaning, and assign a human understandable and consistent semantic label to each section that could help a machine understand a large document. The framework used *arXiv* scholarly articles and RFP business documents to extract and identify logical and semantic structure. This thesis also contributed to a new dataset of information about a collection of academic articles from *arXiv* eprints repository, which included a wide range of meta-data for each article, including TOCs, section summarizations, and more.

To wrap up this dissertation, we recapped key points from each of the chapters. Afterwards, some limitations of our work and future direction in this research were briefly described.

In the Background and Related work chapter, we described the current state of the art research and applications which exist in this research domain. However, we weren't able to find a complete solution for the proposed research challenge described in the Introduction chapter. At the end of the Background and Related Work chapter, we mentioned the gaps in the existing research and hence were motivated to get a complete

solution using a variety of deep learning architectures.

As seen in the Technical Approach chapter, we explained a detailed system architecture for our proposed framework. We propounded the approaches with powerful deep learning techniques using text-only, layout-only, and combined text and layout input vectors. The key points were identifying logical sections and inferring their structures. We also described important features which were used for learning and capturing both logical and semantic structures of a document which resulted in a novel approach to understand large document.

The Input Document Processing chapter described detailed procedures of input document collection and processing. The types and semantics of input documents were also explained in the Input Document Processing chapter. The training and test datasets for each of the units of our proposed framework were also narrated in that chapter. We used *arXiv* scholarly articles and RFP business documents to develop our training and test datasets. In order to train models for capturing most of the patterns from the dataset, we stratified the training and test datasets.

In order to prove our proposed approaches, a complete set of experiments with detailed evaluation for each unit of our framework was presented in the Experiments and Evaluation chapter. The chapter relates each of the units of our system by establishing connections among them. Each of the experiments was followed by a results and evaluation section, which showed the performance of a trained model using precision, recall, and f1-score matrices. The results and evaluation section also plotted graphs and visualizations of evaluation and embedding matrices. To visualize embedding matrices

of CNN, RNN, LSTM and Auto-encoder, we used *T-SNE* dimensionality reduction on the original data. At the end of the experiments of each unit, we had a discussion section to assess the overall performance of that unit.

6.1 Discussion and Summary of Contributions

The general goal of document understanding as a research endeavor is to form models and inferring relationship on those models in a variety of document types, such as academic articles, business documents and user manuals. The key challenges to achieve this goal are forming the logical segmentations, inferring the logical structure from these segmentations, deriving a high-level semantic description about document content and evolving the semantic structure from the semantic description.

This dissertation explores the above mentioned challenges to understand a large and complex document. While exploring the first and second challenges, we realized that the most difficult and important tasks are identifying different level of headings from the low level representation of a document. Because the documents are prepared by human beings and the models are confused due to document's formatting and style parameters. To get a good logical segmentation, initially we formed each line and identified it as a *regular-text* or *section-header*. Later we classified level of headings, such as *top-level section*, *subsection* and *sub-subsection* heading. Then we segmented the document based on level of section headings. Finally, we inferred the relationship among different segments of a document.

The granularity of the logical segmentation in our research is line level which

enables our machine learning approaches to learn the hidden structures of sections, subsections and sub-subsections. We designed the models using document layout parameters and text content. These designed attributes of our models are able distinguish the learning approaches in our experiments. For example, our deep learning approaches, such as RNN and CNN perform better than SVM and Decision Tree techniques. Because both RNN and CNN approaches were able to learn the character sequences and found distinguishable features whereas SVM and Decision Tree were not able to capture sequential features from the text content.

While focusing the third and fourth challenges, the most important tasks are capturing a semantic from a block of text and evolving a semantic structure from a document. To capture a general purpose semantic, we chose word level CNN, RNN and LSTM approaches. The word level granularity enables our approaches to understand the meaning of the context and to find the semantic patterns from a sequence of text. This attribute of our approaches has confirmed a lower computational cost than a character-based approach because the character-based approach for a large text block needs much higher hidden layer to capture long-term dependencies.

To evolve a semantic structure, we designed and developed a document ontology using both Variational and Convolutional Auto-encoder. A big challenge in semantic annotation is to understand and map semantically similar labels. Developing a document ontology helped to capture this semantic over a large number of different documents. The unsupervised nature of our approaches was able to choose the Variational Auto-encoder over the Convolutional Auto-encoder.

The output of this research are logically divided sections with semantic labels, section summaries and table of content of a document. A content-based question answering platform can use our system to generate sections with semantic names and develop a document indexing system for efficient information retrieval. NLP communities working in topic modeling, language modeling and co-reference resolution find out system very useful. Any agencies which read thousands of documents to take business decision, will find section summaries to save their valuable time and money. In a larger problem space, any person who works in a different problem domain, can use this thesis to understand deep learning architectures in text classification, semantic concept identification and ontology concepts development. People who are interested in document analysis can also get benefits from this thesis.

6.2 Limitations of the System

The input of this framework is TETML which is generated by PDFLib Text Extraction Toolkit. Hence the framework heavily depends on PDFLib TET. Sometimes PDFLib generates multiple blocks from a single text line and assigns them into different paragraph tags. While parsing a TETML file, the parser may consider these paragraphs separately since the post-processing of TETML file depends on rules and schema of a TETML.

Due to resource limitation for the annotation process of RFP documents, models trained for *arXiv* articles are used for RFP document sectioning and semantic labeling. Though it helps generalization, models cannot predict semantic labels for RFP sections.

This is because RFP is domain dependent and has its own sections, such as deliverable and Contract Clauses, which are not seen in any scholarly article. Moreover, other sections, such as background and requirement of an RFP document, describe different concepts than an academic article.

The Semantic Section Classifiers become confused with contribution sections since different articles describe different contributions. The classifiers also confuse future work and conclusion sections. Though many articles have two different sections for future work and conclusion, some articles merge them into one section.

6.3 Future Research Directions

Understanding a large and complex document is a very challenging research problem since there is a long road ahead for a machine to read and understand like a human reader. So far we have focused on the design of a general architecture to cover academic articles and RFP documents. But there are a lot of other domains where further research could be done with real world applications. A list of possible future work related to the research is explained below.

6.3.1 Improvement of Abstractive Summarization

As explained in the Experiments and Evaluation chapter, we observed that we were not able to achieve a good performance for abstractive summarization using the *Tensorflow Textsum* algorithm. We noticed that the model did not work well for large sections. So, we should further study the abstractive summarization techniques of deep

learning architectures and come up with a good architecture that could handle longer texts. We plan to develop a new variational encoder-decoder architecture with an attention mechanism which can capture abstractive summaries for longer texts.

6.3.2 Domain Adaptation

This research work can be extended to include other domains, such as Medical Reports, US Patents and Business Reports. A prospective approach to include other domains in our framework could be the domain adaptation approach [40, 24, 56]. The learning and cost functions in models presented in this research work can be modified to enhance the capabilities of current models for training in one domain and applying them in other domains.

Both unsupervised and supervised domain adaptation techniques can be applied to the current models. In the case of unsupervised domain adaptation, the models will be tested on unlabeled datasets from other domains. For the supervised domain adaptation, the models will be trained on a small amount of training data from other domains and tested on a large amount of unlabeled datasets.

6.3.3 Releasing a Complete System for Public Use

We hope to quickly develop a complete end-to-end system to release the product for public use. We will make the system open source. We also plan to deploy our system in the cloud. The final system will have navigations in order to access the sections of interest. The system will also have semantic indexing for sections, which would be

useful for content-based question answering systems.

6.3.4 Extracting Information from Scanned Documents

We may consider scanned documents, extract information from them and apply the techniques we presented in this dissertation. However, in order to extract information from scanned documents, we would need to include Optical Character Recognition (OCR) techniques in our framework. We plan to further explore our approach to include scanned documents in our system to section annotate and label them with semantic names.

6.3.5 Generating Document from a Structure

Another potential future work is implementing a generative deep learning approach to construct a document given the structure of the document. The model will be trained on a large number of documents with their semantic structures. Thereafter, the model will be used to generate contents for each semantic section and to combine the contents as a new document.

6.4 Concluding Remarks

We end this dissertation by repeating the thesis statement: " It is possible to automatically identify a document's logical sections, infer their structure and assign human understandable and consistent semantic labels that can help machines to understand large documents."

Bibliography

[1] Arxiv repository of electronic preprints, 2017. [Online; accessed 16-October-2017].

[2] Redshred, 2017. [Online; accessed 16-October-2017].

[3] Sections of an rfp, 2017. [Online; accessed 22-October-2017].

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

[5] Silvio Peroni Angelo Di Iorio, Fabio Vitali. Document structural patterns ontology, 2017. [Online; accessed 09-October-2017].

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

[7] Terri K Attwood, Douglas B Kell, Philip McDermott, James Marsh, SR Pettifer, and David Thorne. Utopia documents: linking scholarly literature with research data. *Bioinformatics*, 26(18):i568–i574, 2010.

[8] Amazon AWS. Requester pays buckets.

[9] Robert M Ayers. Method and apparatus for identifying words described in a page description language file, November 3 1998. US Patent 5,832,531.

[10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[11] Tim Bienz, Richard Cohn, and Calif.) Adobe Systems (Mountain View. *Portable document format reference manual*. Citeseer, 1993.

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

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

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

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

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

[17] Jianpeng Cheng and Mirella Lapata. Neural summarization by extracting sentences and words. *arXiv preprint arXiv:1603.07252*, 2016.

[18] Sumit Chopra, Michael Auli, and Alexander M Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, 2016.

[19] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[20] P Ciccarese and T Groza. Ontology of rhetorical blocks (orb). editor's draft, 5 june 2011. *World Wide Web Consortium. http://www. w3. org/2001/sw/hcls/notes/orb/(last visited March 12, 2012)*, 2011.

[21] Alexandru Constantin, Silvio Peroni, Steve Pettifer, David Shotton, and Fabio Vitali. The document components ontology (doco). *Semantic Web*, 7(2):167–181, 2016.

[22] Alexandru Constantin, Steve Pettifer, and Andrei Voronkov. Pdfx: fully-automated pdf-to-xml conversion of scientific literature. In *Proceedings of the 2013 ACM symposium on Document engineering*, pages 177–180. ACM, 2013.

[23] Joseph P Coyne, Harold A Daub, Judy J Kogut-O'Connell, Teresa G Fiore, Michael A Fortine, Pamela K Lowe, Aldo Morandin, Yoshiaki Ohsumi, Jose de Jesus Michel Rodriguez, Anne C Ten Dyke, et al. Process for determining an auction methodology, January 6 2009. US Patent 7,475,034.

[24] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. *arXiv preprint arXiv:1702.05374*, 2017.

[25] Silvio Peroni David Shotton. Discourse elements ontology(deo), 2017. [Online; accessed 09-October-2017].

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

[27] Howard B Demuth, Mark H Beale, Orlando De Jess, and Martin T Hagan. *Neural network design*. Martin Hagan, 2014.

[28] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[29] Robert H Dolin, Lawrence Garber, and Icode Solutions. Hl7 implementation guide for cda® release 2: Consolidated cda templates for clinical notes (us realm) draft standard for trial use release 2.

[30] Jacob Eisenstein, Brendan O'Connor, Noah A Smith, and Eric P Xing. A latent variable model for geographic lexical variation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1277–1287. Association for Computational Linguistics, 2010.

[31] Jeffrey Flanigan, Sam Thomson, Jaime G Carbonell, Chris Dyer, and Noah A Smith. A discriminative graph-based parser for the abstract meaning representation. 2014.

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

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

[34] Shalini Ghosh, Oriol Vinyals, Brian Strope, Scott Roy, Tom Dean, and Larry Heck. Contextual lstm (clstm) models for large scale nlp tasks. *arXiv preprint arXiv:1602.06291*, 2016.

[35] Justin Grimmer. A bayesian hierarchical topic model for political texts: Measuring expressed agendas in senate press releases. *Political Analysis*, 18(1):1–35, 2010.

[36] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[37] Awad S Hanna, Eric J Tadt, and Gary C Whited. Request for information: benchmarks and metrics for major highway projects. *Journal of Construction Engineering and Management*, 138(12):1347–1352, 2012.

[38] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[40] Jing Jiang. A literature survey on domain adaptation of statistical classifiers. *URL: http://sifaka. cs. uiuc. edu/jiang4/domainadaptation/survey*, 3, 2008.

[41] Mikael Kågebäck, Olof Mogren, Nina Tahmasebi, and Devdatt Dubhashi. Extractive summarization using continuous vector space models. In *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)@ EACL*, pages 31–39, 2014.

[42] D Kermisch and PG Roetling. Fourier spectrum of halftone images. *JOSA*, 65(6):716–723, 1975.

[43] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

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

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

[46] Stefan Klink, Andreas Dengel, and Thomas Kieninger. Document structure analysis based on layout and textual features. In *Proc. of International Workshop on Document Analysis Systems, DAS2000*, pages 99–111. Citeseer, 2000.

[47] Stefan Klink and Thomas Kieninger. Rule-based document structure understanding with a fuzzy combination of layout and textual features. *International Journal on Document Analysis and Recognition*, 4(1):18–26, 2001.

[48] Philipp Koehn. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. *Machine translation: From real users to research*, pages 115–124, 2004.

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[50] Yann Le Cun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

[51] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[52] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[53] David D Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *European conference on machine learning*, pages 4–15. Springer, 1998.

[54] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 510–520. Association for Computational Linguistics, 2011.

[55] Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A Smith. Toward abstractive summarization using semantic representations. 2015.

[56] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning*, pages 97–105, 2015.

[57] Konstantin Lopyrev. Generating news headlines with recurrent neural networks. *arXiv preprint arXiv:1512.01712*, 2015.

[58] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

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

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

[61] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. Association for Computational Linguistics, 2004.

[62] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[63] Diego Monti and Maurizio Morisio. Semantic annotation of medical documents in cda context. In *International Conference on Information Technology in Bio-and Medical Informatics*, pages 163–172. Springer, 2016.

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

[65] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. *hiP (yi= 1— hi, si, d)*, 1:1, 2017.

[66] Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

[67] Ramesh Nallapati, Bowen Zhou, and Mingbo Ma. Classify or select: Neural architectures for extractive document summarization. *arXiv preprint arXiv:1611.04244*, 2016.

[68] Shashi Narayan, Nikos Papasarantopoulos, Mirella Lapata, and Shay B Cohen. Neural extractive summarization with side information. *arXiv preprint arXiv:1704.04530*, 2017.

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

[70] Mohammad Daryoush Paknad and Robert M Ayers. Method and apparatus for identifying words described in a portable electronic document, November 3 1998. US Patent 5,832,530.

[71] Xin Pan and Peter Liu. Tensorflow textsum, 2015. [Online; accessed 23-October-2017].

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

[73] PDFlib. Pdflib tet, 2016. [Online; accessed 25-October-2016].

[74] 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.

[75] Silvio Peroni. The semantic publishing and referencing ontologies. In *Semantic Web Technologies and Legal Scholarly Publishing*, pages 121–193. Springer, 2014.

[76] Poppler. Poppler pdf rendering library, 2016. [Online; accessed 30-October-2016].

[77] Muhammad Mahbubur Rahman. Intellectual knowledge extraction from online social data. In *Informatics, Electronics & Vision (ICIEV), 2012 International Conference on*, pages 205–210. IEEE, 2012.

[78] Muhammad Mahbubur Rahman and Tim Finin. Deep understanding of a document's structure. In *Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, pages 63–73. ACM, 2017.

[79] Muhammad Mahbubur Rahman, Tim Finin, et al. Understanding and representing the semantics of large structured documents. In *Proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4, ISWC)*, 2018.

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

[81] R Rehurek and P Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 2011.

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

[83] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015.

[84] Georg Sander. A fast heuristic for hierarchical manhattan layout. In *International Symposium on Graph Drawing*, pages 447–458. Springer, 1995.

[85] Arne Schulze. Methods and computer readable storage medium for conducting a reverse auction, May 20 2008. US Patent 7,376,593.

[86] David Shotton and Silvio Peroni. Doco, the document components ontology. 2011.

[87] Luciana B Sollaci and Mauricio G Pereira. The introduction, methods, results, and discussion (imrad) structure: a fifty-year survey. *Journal of the medical library association*, 92(3):364, 2004.

[88] Nitish Srivastava, Ruslan R Salakhutdinov, and Geoffrey E Hinton. Modeling documents with deep boltzmann machines. *arXiv preprint arXiv:1309.6865*, 2013.

[89] Kokichi Sugihara. Approximation of generalized voronoi diagrams by ordinary voronoi diagrams. *CVGIP: Graphical Models and Image Processing*, 55(6):522–531, 1993.

[90] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

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

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

[93] Victoria Uren, Philipp Cimiano, José Iria, Siegfried Handschuh, Maria Vargas-Vera, Enrico Motta, and Fabio Ciravegna. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *Web Semantics: science, services and agents on the World Wide Web*, 4(1):14–28, 2006.

[94] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[95] Wikipedia. Apache pdfbox — wikipedia, the free encyclopedia, 2016. [Online; accessed 20-September-2016].

[96] Wikipedia. Pdftotext — wikipedia, the free encyclopedia, 2016. [Online; accessed 27-September-2016].

[97] A Yamashita, T Amano, I Takahashi, and K Toyokawa. A model based layout understanding method for the document recognition system. In *Proceedings of International Conference on Document Analysis and Recognition*, pages 130–138, 1991.

[98] Xiang Zhang and Yann LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.