Extracting Specific Text From Documents Using Machine Learning Algorithms

by

Sahib Singh Budhiraja

Lakehead University

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTERS

in the Department of Computer Science

Lakehead University

Extracting Specific Text From Documents Using Machine Learning Algorithms

by

Sahib Singh Budhiraja
Lakehead University

Supervisory Committee

Dr. Vijay Mago, Supervisor

(Department of Computer Science, Lakehead University, Canada)

Dr. Salimur Choudhury, Departmental Member

(Department of Computer Science, Lakehead University, Canada)

Dr. Philippe J. Giabbanelli, External Member

(Department of Computer Science, Furman University, USA)

ABSTRACT

Increasing use of Portable Document Format (PDF) files has promoted research in analyzing the files' layout for text extraction purpose. For this reason, it is important to have a system in place to analyze these documents and extract required text. The purpose of this research fulfills this need by extracting specific text from PDF documents while considering the document layout. This approach is used to extract learning outcomes from academic course outlines. Our algorithm consists of a supervised leaning algorithm and white space analysis. The supervised algorithm locates the relevant text followed by white space analysis to understand document layout before extraction. The supervised learning approach used for detecting relevant text does so by looking for relevant headings, which mimics the approach used by humans while going through a document.

The data set used for this research consists of 500 course outlines randomly sampled from the internet. To show the capability of our text detection algorithm to work with documents other than course outlines, it is also tested on 25 reports and articles sampled from the internet. The implemented system has shown promising results with an accuracy of 81.8% and remediated the limitation shown by the current literature by supporting documents with unknown format. The algorithm has a wide scope of applications and takes a step towards automating the task of text extraction from PDF documents.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

DEDICATION

"What we find changes who we become."

- Peter Morville

I would like to dedicate this thesis to,

My parents, who gave me the foundation of something they always enjoyed, education. For inspiring me and always believing in my ability to accomplish anything I set my mind to. For making sure I had access to all the resources I need to succeed no matter the circumstances. My sister, who always took care of my chores when I was busy studying and made me laugh whenever I would feel low. They are my driving force, what keeps me going.

My uncle, who always thought of me as a son. Watching futurist documentaries on the Discovery channel with him when I was little is what got me interested in science and technology in the first place. Without his support and encouragement, I would not be where I am today.

I would also like to dedicate this thesis to all the teachers and mentors who inspired, empowered, taught, and shaped me into the person I am today.

Chapter 1

Introduction

1.1 Overview

The Portable Document Foment (PDF) format is portable, thus allowing its users
to open these documents independent of the underlying platform. PDF is the one
of the most commonly used document formats for storing text based data. A PDF
document is visually an exact digital copy that displays text by drawing characters
on a specific location, so the extracted text will contain the whole text from the
document extracted in a left to right and top to down flow ignoring any formatting
details like multiple columns or header/footer text [4].

The shift from analog to electronic storage of text based documents is one of the
most significant accomplishments which has lead to led to an exponential increase in
the number of electronic documents used in both professional and non-professional
contexts. With the rising number of electronic documentation, the demand for text

based analysis and the extraction of meaningful and specific information from digital documents / e-documents is also increasing. Software applications with the ability to automatically extract specific text from e-documents can save a lot of time and human effort when dealing with numerous documents. While extracting text from a PDF is straightforward if the intention is to extract the entirety of the text, given that it is neither embedded in images nor present in an unknown font. But, extraction gets complex when only a certain part of that text is needed. Decisions related to selecting the appropriate sections of the whole text while maintaining the integrity and flow of the information present challenges that are addressed in this research.

1.2 Motivation

Systematic extraction of textual structure and text is increasingly necessary and useful as demonstrated in El-Haj et al.'s work involving 1500 financial statements [10]. El-Haj performed an analysis of the document structure and extracted specific sections in a structured manner. It's application is not just limited to one field as it can be used to extract information about a company's performance from financial statements, in healthcare sector to extract specific notes from patient records, in legal sector to extract procedural history or judgement from case briefs and, for the purpose of this research, extracting post-secondary learning outcomes from course outlines for analysis and review [10, 53]. Proposing a method for efficiently extracting *learning outcomes* from course outlines is the main motivation behind this thesis.

Learning outcomes for a course define what a student will gain from taking a specific class [21]. Specifically, learning outcomes detail the skills and knowledge the instructor of the course expects the students to gain from the course [50]. This makes learning outcomes a useful means of comparing the objectives of multiple courses for the purpose of credit transfer between institutions and also for measuring an educational credential against accreditation or Quality Assurance standards. Extracting learning outcomes is not a straightforward task because there is no standardized format for course outlines and varies across institutions and professors/instructors. The purpose of extracting learning outcomes for this research is to support the automation of university/college transfer credit agreements in Ontario using semantic similarity algorithms to assess the similarities in learning outcomes between post-secondary cre-

dentials [35, 36, 37].

1.3 Problem Description

Automating the process of extracting learning outcomes from course outlines requires the program to identify which text to extract and then extract it while accounting for the document's layout and formatting. The approach followed here is to isolate the text required to be extracted by looking for headings that are relevant to what needs to be extracted, which in this case are learning outcomes. We assumed that learning outcomes are mentioned under a separate heading and not in a continuous form where the outline consists of a series of paragraphs with no headings to specify about the content. A supervised learning approach is necessary to train an algorithm to detect, label, and classify learning outcomes from the ambiguous textual context of non-standardized post-secondary course outlines.

1.4 Brief Description of Methodologies

1.4.1 Supervised Learning

Supervised learning is a branch of machine learning that deals with algorithms learning from labeled training data, therefore it mimics the process of learning under a supervising teacher [31]. A supervised learning algorithm uses the input variables (X), and its corresponding output variable (Y) to train a mapping function [46]. The algorithm keeps on learning until an acceptable performance level is attained. Mathematically, it can be represented using the Equation 1.1.

$$Y = f(X) \tag{1.1}$$

The purpose of training the algorithm is to make it capable enough to make predictions on unseen situations. Fig 1.1 shows the framework for the whole process. Once trained, the algorithm can generalize based on the training data and make reasonable predictions on new instances. Supervised learning algorithms can be further divided

into two categories: classification and regression.
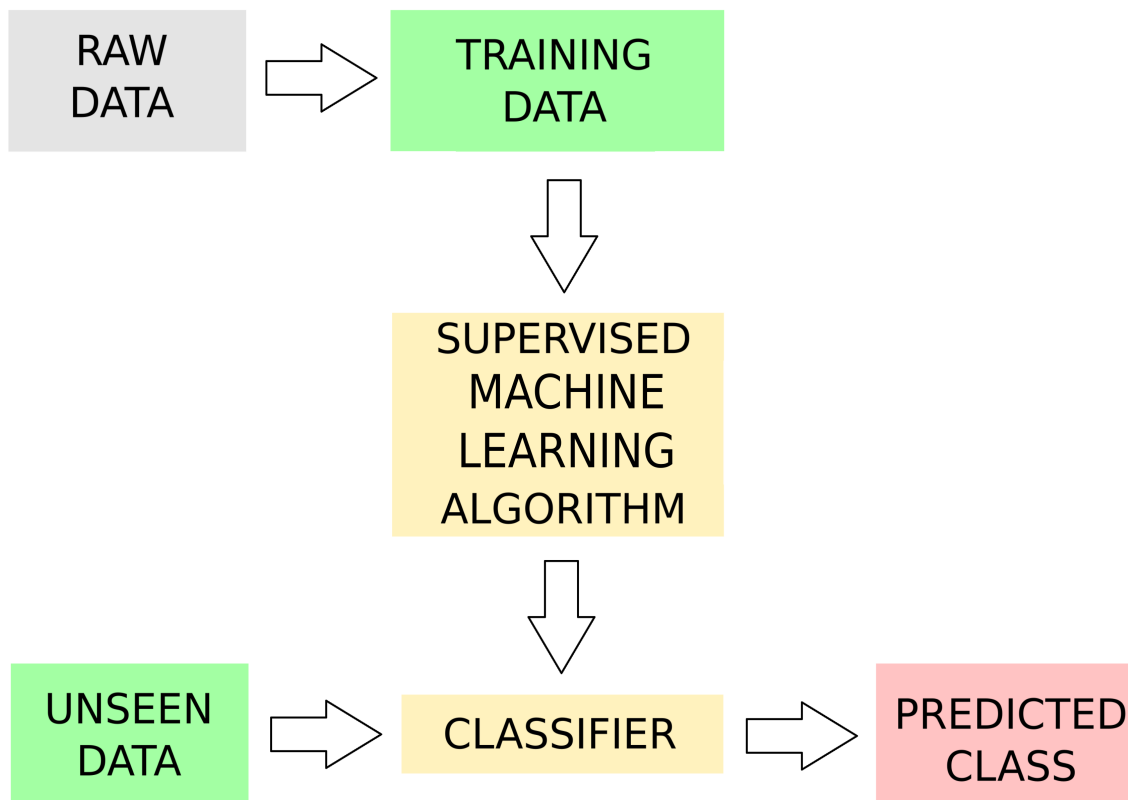
Figure 1.1: Supervised Learning Method

Classification

The final output of classification algorithms is discrete and are usually labels or categories. The purpose of the algorithm is to use a data point to predict the category or class for that observation [1]. An example of this is the classification of text as a heading or non-heading. These algorithms can be multi-class, two-class or binary, depending on the nature and number of output classes. These algorithms usually generate a continuous value representing the probability of a given class, and this value represents the confidence level of the decision. This probability is used to make the final choice regarding the final predicted class. The example in Box 1.1 illustrates the process.

Box 1.1: Classification Example

Lets say an algorithm is classifying text as either heading or non-heading.

If P(heading) = 0.75,

Therefore, P(non-heading) = 0.25

So, the class chosen is *heading* for this observation.

This example uses a default threshold of of 0.5, which we can change to different values to adjust how the algorithm behaves.
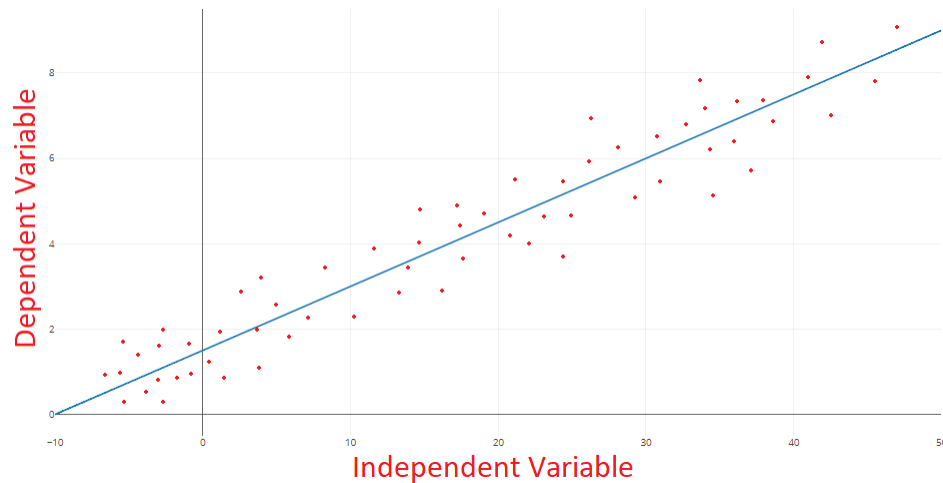
Figure 1.2: Simple Linear Regression

Regression

Regression algorithms use a set of inputs to generate a continuous variable as output, which is a real value such as float or double value. In simple linear regression as illustrated in Fig 1.2, the aim is to find a linear function that can predict the dependent variable as accurately as possible based on the independent one [13]. An example relating to this thesis would be an algorithm estimating the size of the header

or footer region using any one of page layout characteristics as input because both these regions can range from 0 to 50% of the page height, so the output is continuous variable dependent on one independent variable.

1.4.2 Document Layout Analysis

The purpose of performing layout analysis on a document is to identify elements like text columns, header/footer, and tables in it. Both methods discussed below have their advantages and disadvantages, depending on the nature of the document being analyzed.
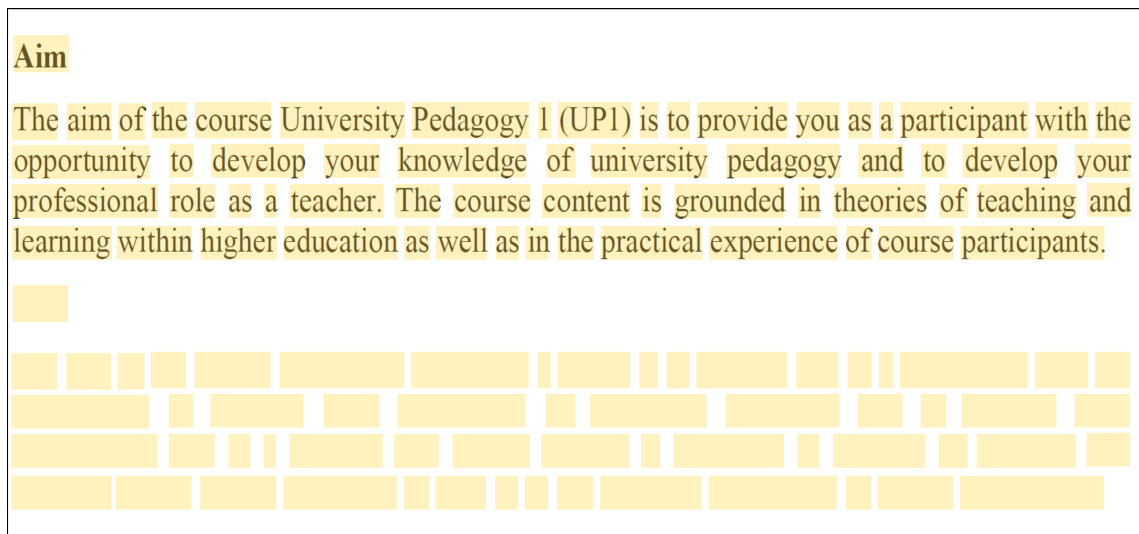
Figure 1.3: Bounding Box Based Approach

Examining Text blocks

This approach analyzes the text content in the document to comprehend its layout and involves using complex data structures to accomplish it. Ramakrishnan et al. proposed an approach for examining text blocks using the bounding boxes that surround each word, as shown in Fig 1.3 [42]. The structural information gathered is aggregated to understand the structure of the document. The bounding box data can be extracted from a document using XML conversion of a document which is

Syllabus

This is a description of the material to be examined. On registration, students will receive a detailed subject guide which provides a framework for covering the topics in the syllabus and directions to the essential reading

The course covers basic topics in microeconomics such as supply and demand, consumer theory, labour supply, asymmetry of information, neo-classical view of the firm, production, costs, factor demands, perfect competition, monopoly, monopolistic competition, oligopoly, cartels and tacit collusion. We also analyse some newer material regarding alternative theories of the firm, internal organisation of the firm, market structure, efficiency wages, incentive structures, corporate governance as well as some industrial organisation theories of commonly used pricing practices.

The following topics also form part of the course syllabus:

- individual (one person) decision making under uncertainty, attitudes towards risk and the value of information

- theory of games and strategic decision making, including its applications to oligopoly, collusion among firms, product differentiation, entry deterrence and other market practices

- the effects of asymmetric information in areas such as bargaining, bidding and auctions, situations of moral hazard and adverse selection

- corporate governance in modern organisations.

Some knowledge of constrained maximisation and lagrangian functions would be helpful for students taking this subject, although this is not a prerequisite.

Students should consult the appropriate *EMFSS Programme Regulations*, which are reviewed on an annual basis. The *Regulations* provide information on the availability of a course, where it can be placed on your programme's structure, and details of co-requisites and prerequisites.

Figure 1.4: White Space Analysis

discussed further in chapter 3.

White Space Analysis

White space refers to the part of the document that is completely empty. The white space separates paragraphs, text columns, and header/footer text. Therefore, it can be used to analyze the layout. As shown in Fig 1.4, the white spaces represented by the pink shaded area divide different elements of the page. The idea behind separating all the elements is to eliminate anything that is not required and to stitch the required text in the right order as in the case of multiple text columns. Detecting the white space can be done in a number of ways, for example by checking pixel color and looking for large empty rectangles.

An example of white space analysis is presented by Berg et al. based on the work originally presented by Breuel, which works by looking for the largest empty rectangle [3, 6]. The largest empty rectangle works by finding the largest possible rectangle that can be placed amongst all the text blocks in the document without overlapping with them. There are many other approaches and some of them are discussed in the next chapter.

Overview To The Thesis

Introduction to the problem and motivation is presented in this chapter. Chapter 2 discusses the merits and limitations of the available literature. Preprocessing PDFs to extract data for analysis is discussed in chapter 3. For extracting this data the PDF is converted to HTML for supervised learning models and images to perform white space analysis. Chapter 4 presents a supervised learning approach to detect headings, which provides information about the contents of the PDF. Chapter 5 deals with identifying the relevant heading(s) using the headings detected earlier and also presents an approach for layout analysis used for extracting the text in its proper flow. Chapter 6 discusses the whole system included the main contributions, current limitations and future work.

Chapter 2

Related Work

2.1   HTML Conversion

While PDF format is convenient as it preserves the structure of a document across platforms, extracting textual layout information is required for detecting headings and further analysis. One solution to extracting layout information is to convert the PDF into HTML and use the HTML tags for further analysis, the other one being conversion to XML discussed later in chapter 3. Once converted to HTML all the information related to text formatting required for the analysis, like font size and boldness of text, can be easily extracted.

Jiyang and Yang proposed an approach to perform the conversion from PDF to HTML using text detection [20]. The output is a HTML file that preserves the font information and the layout of the whole document. The process works by detecting text fragments and merging them. The conversion is made possible by using the coordinate information gathered using PDFBox [38]. They claim good results but did not include the results. Rahman and Alam [41] also use conversion to HTML to perform

their analysis as discussed in Section 2.3 of this chapter. It is clear from the available literature that HTML conversion is already an established approach for analyzing a PDF's textual content. The justification for the choice of tool to make the conversion is discussed in Section 3.3 of the next chapter.

2.2   Heading Detection

Previous research provides insight into processes related to extracting the heading layout of a HTML document [29]. In Manabe's work, headings are used to divide a document at certain locations that indicate a change in topic. Document Object Model(DOM) trees are used to sort candidate headings based on their significance and to define blocks. They made a binary judgement by classifying a candidate as a heading or a block. A recursive approach is applied for document segmentation using the list of candidate headings. They computed the Fleiss Kappa coefficient which measures the degree of agreement between the results and the manually labelled dataset, for headings it came out to be 0.693 and for blocks, it is 0.583. The evaluation based on the manually labelled dataset shows good results, but still not enough for our objective. Heading detection is the first step in the extraction process, and its accuracy needs to be near perfect to get good overall results. The reason for this claim is the fact that this application will be divided into multiple small modules, and the accuracy of the each module will contribute towards the applications overall accuracy.

El-Haj et al. provide a practical application of document structure detection through the analysis of a large corpus of UK financial reports including 1500 annual reports from 200 different organizations [10]. They use the content page to get the list of sections for the report. The generated list of synonyms for section headings is cross-referenced against a list of 'gold standard' section names created from 50 randomly selected reports. The structure is then extracted using this list and evaluated, in addition to being reviewed by a domain expert for accuracy.

Table 2.1: Evaluation results for the approach proposed by El-Haj et al. [10]

	Count	**%**
# of PDFs	105	-
Headers in PDFs	2473	-
Extracted Headers	2502	-
Extract Matches	2202	88.01%
Partial Matches	105	4.2%
Wrong Headers	195	7.8%
Missing Headers	166	6.6%
Correct Headers	2307	93.3%
Detected Page Number	94	89.5%
Detected Contents Pages	97	92.4%

Table 2.1 shows the evaluation results from both the evaluation stages. These are the motivation behind using a keyword-based approach for detecting relevant headings in our proposed approach. One major drawback is that for this approach to work the document under review needs to have a table of contents page. In the case of course outlines and many other types of documents, a table of contents is not used. To make this a generic approach that works on all kinds of documents, we need a mechanism to look for section headings before using the keyword-based approach to select the relevant headings.

The literature discussed in this section has taken steps towards developing a system which analyses a document's textual structure. This served as the motivation behind using headings as a means to locate relevant text, but there is still a need to have an approach that can efficiently and accurately analyse the textual layout of a document and divide it into content sections to automate the process of extracting text from a PDF documents.

2.3   Layout Analysis

Analysing the layout of a document is an essential part of text extraction as it ensures the consistency and flow of the text. Ramakrishnan et al. propose using contiguous

text blocks to analyze the layout and determine the text flow in scientific articles for text extraction. The proposed method uses the GPL version of *JPedal* which is an open-source Java PDF library, to get bounding boxes of words from the document and then performs aggregation to build up bigger blocks while taking the document layout into account [42].

Various page formats are supported by computing proximity of the boxes automatically. Once the blocks have been aggregated together, they are classified into predefined categories using a rule-based system. Some of the attributes used for this rule-based classification are alignment, page number, and last section. After classification, the blocks are stitched together in the correct sequence using the section they are classified under. Stitching them together ensures that the text is extracted in the correct sequence. Although the provided approach shows good accuracy it only works with documents that have a defined format and does not present a generic solution [42].

Gao et al. uses an approach originally presented by Lin et al. for detecting the header and footer text of PDF documents [14, 24]. The proposed approach uses page association and looks for repeating characters across different pages. Using the relationship between header/footer text spread across multiple pages their system compares one page with the other for detecting headers/footers. The algorithm needs text and their bounding boxes to initialize the process by which they directly extract from the PDF documents.

The whole process is divided into 4 steps:

- The text lines are constructed using text bounding boxes.

- Candidate header/footer text lines are selected by choosing the top five lines as candidates for the header and bottom three lines as candidates for the footer.

- Each candidate is then evaluated quantitatively to determine if they qualify as a header/footer by assigning it a score.

- All the candidates that have scores above a certain threshold are chosen as header/footer text.

Even though this approach has a precision rate of 98% and recall of 92.7%, it was not chosen as it only presents a solution for detecting header and footer text, not for detecting multiple text columns. We discuss more about this choice in the conclusion section.

Finally, Rahman and Alam [41] use an approach that looks for the largest rectangle made up of white space and converts the file into Hypertext Markup Language (HTML) to perform their analysis. The algorithm identifies the white and black space in the document and creates boxes that only contain white spaces. This process is performed both in vertical and horizontal directions to come up with these rectangles which they refer to as *White Space Rectangles (WSR)*. After that, the algorithm detects columns by looking for maximal WSR in the vertical direction. Rows are detected using the same algorithm, but first, the document is rotated by 90 degrees. Once all WSRs in vertical and horizontal direction have been detected, they are recursively merged together into *White Space Polygons (WSP)* until all of the WSRs have been merged together. WSPs are used to define the document segment, which is then classified into distinct categories. Once classification is done, the next step involves rendering (or re-ordering in the correct content flow) by using the information from segmentation and recognition. Even though they provide a novel approach for analysing document layout, no testing results validating the approach are currently available and the system is confined to detecting multiple text columns and not header/footer text.

2.4   Conclusion

We found that there is currently no single algorithm or tool that can extract specific components, in our case Learning Outcomes, from a text based document. The current approaches relies on table of contents or a known structure to identify the headings in the document, this limits their application. The approaches for analysing the layout perform good, but use different techniques for analysis; thus requiring more processing which can be avoided if they use the same technique. Therefore, for specific text extraction to be feasible, there is a need to improve and combine the currently available algorithms.

Chapter 3

Preprocessing PDFs

3.1   Overview

PDF was created to have a fixed layout representation that displays documents irrespective of the application or the platform being used to view them. The format preserves the overall outlook of the original document by including the low-level structural objects in the representation like images, lines, curves and grouped characters [4]. All of the structural objects are shown on the page canvas in a visually oriented way by a PDF parser. While this is an efficient way to store the visual representation of a document, the resulting structure is difficult to work with if the aim is to extract specific parts of the text in a structured manner.

Simple text extraction only gives you text from the PDF with no indication of where that text came from: it could be from a paragraph, heading, or a table. To get information such as font size, character bounding box location and other style attributes, the PDF needs to be converted into mark up languages such as Hypertext Markup Language (HTML) and eXtensible Markup Language (XML). Fig 3.1 shows the conversions that will be performed and their respective outputs.
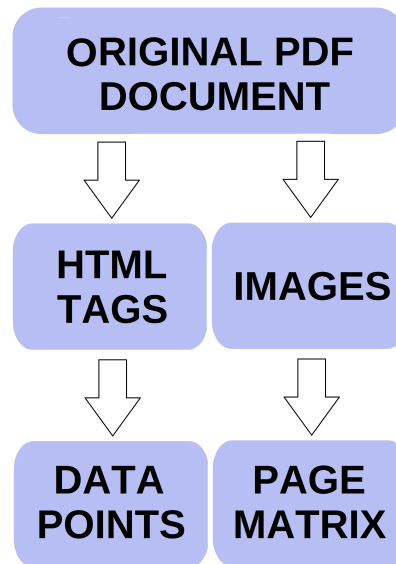
Figure 3.1: Preprocessing PDFs

3.2 Dataset

Our dataset consists of 500 documents[1] downloaded from Google using *Google Custom Search API* [9]. The keywords used for sampling the dadaset were "Course Outlines", "Course Description" and "Syllabus". The downloaded documents were then checked manually to make sure all are course outlines. The documents other than course outlines were discarded.

---

[1]Repository available at: https://github.com/sahib-s/Heading-Detection-PDF-Files

Table 3.1: Text format recovery tools and their supported formats

Tool Name	Formats Supported
pdf2text	HTML, XML
pdftohtml	HTML, XML
pdftohtmlEX	HTML
pdfextract	XML

3.3   Tool Selection

A number of tools are available for recovering the structure of document by converting them into HTML or XML depending on the need. Table 3.1 lists the ones that were suitable for further evaluation for the task at hand. Choosing a tool that outputs an easy to parse format is important because it will reduce the processing time required to extract the formatting information required. After evaluating we found pdf2text, which is a PDFMiner wrapper available in Python as the best suited for the task[48]. The selection of PDFMiner is mainly based on its ability to analyze the page structure and generating an output with elements grouped based on the hierarchy. The fact that it has been widely adopted in many similar studies involving extraction from PDFs and is well maintained are also contributing factors[51, 52]. It also supports XML conversion which provides us finer details about the tags via bounding boxes that will be required to further this research in the future.

3.4   Conversion Techniques

3.4.1   HTML Conversion

HTML conversion provides us with tags which include information such as font size, font type and whether if the text is bold or not. This data helps in training and testing of the supervised classifier used for identifying headings, which is a crucial part of the proposed algorithm.

Pre-Processing HTML Tags

While extracting the text format information the following points need to be addressed:

- Break tags contained in the text are needed to be replaced by new line character to make sure the extracted text does not have any unknown tags left at the end of the process.

- &*amp* entity needs to be replaced with & sign because in HTML they are mentioned as &*amp*, which won't match the & sign in the extracted text.

- Sorting the tags based on their location from top of the page is important because we need to process the document from top to bottom and sometimes the tags are not in this order.

Parsing HTML Tags

The HTML files generated using the PDFminer wrapper only use *span tags* to display text; therefore, the algorithm looks for these tags only. The span tags from the generated file are separated and stored in a list in the sequence in which they appear. Once the list of tags is generated, each tag is processed one by one and this process is done for all the tags. The style attribute in the tags contain the font family and the font size.

To determine if the text in the tag is bold or not, the tag style attribute is extracted using Algorithm 1. This is illustrated in Fig 3.2 which shows some sample text and its corresponding HTML tags generated using the conversion process. Each tag is compared with a *Regular Expression* (REGEX) to extract: the style and the text [15]. A REGEX is a string of characters used to define a search pattern. The REGEX used for parsing the tags are mentioned in Box 3.1.

Box 3.1: Regular Expressions Used for Parsing

To extract Font size and corresponding text:

$r`< \s*?span[^>] * font - size : (\w*)px[^>]* > (.*?) < \/span \ b[^>]* > '$

To check if text is bold we look for the following REGEX for the word bold in

the starting tag:

$r`[Bb]old'$

(a) Sample Text from a Document

Course policies and expectations:

Class attendance. You can miss up to 3 class sessions during the semester without penalty. After that, **each** missed class will result in a one-third letter grade reduction in your **final** course grade.

Policy on laptops, cellphones, iPods, etc.: just as you would not read a newspaper in class, please respect your instructors and your fellow students by refraining from non-course-related use of electronic devices during class. Midterm exams will cover lecture and section material.

Course aims

a. learn lecture notes

(b) HTML Tags from Conversion

```html
<div style="position:absolute; border: textbox 1px solid; writing-mode:lr-tb; left:72px; top:1985px; width:160px; height:15px;">
    <span style="font-family: HOVFKU+Optima-Bold; font-size:15px">Course policies and expectations:<br></span>
</div>
<div style="position:absolute; border: textbox 1px solid; writing-mode:lr-tb; left:72px; top:2007px; width:453px; height:13px;">
    <span style="font-family: HOVFKU+Optima-Bold; font-size:13px">Class attendance.</span>
    <span style="font-family: AALDLR+Optima-Regular; font-size:13px"> You can miss up to 3 class sessions during the semester without penalty. After <br></span>
</div>
<div style="position:absolute; border: textbox 1px solid; writing-mode:lr-tb; left:90px; top:2020px; width:437px; height:39px;">
    <span style="font-family: AALDLR+Optima-Regular; font-size:13px">that, </span>
    <span style="font-family: HOVFKU+Optima-Bold; font-size:13px">each</span>
    <span style="font-family: AALDLR+Optima-Regular; font-size:13px"> missed class will result in a one-third letter grade reduction in your </span>
    <span style="font-family: HOVFKU+Optima-Bold; font-size:13px">final</span>
    <span style="font-family: AALDLR+Optima-Regular; font-size:13px"> course<br>grade.<br></span>
</div>
<div style="position:absolute; border: textbox 1px solid; writing-mode:lr-tb; left:72px; top:2212px; width:469px; height:156px;">
    <span style="font-family: HOVFKU+Optima-Bold; font-size:13px">Policy on laptops, cellphones, iPods, etc.: </span>
    <span style="font-family: AALDLR+Optima-Regular; font-size:13px">just as you would not read a newspaper in class,
    <span style="font-family: HOVFKU+Optima-Bold; font-size:13px"> </span>
    <span style="font-family: AALDLR+Optima-Regular; font-size:13px">please<br>respect your instructors and your fellow students by refraining from non-course-related use of <br> electronic devices during class. Midterm exams will cover lecture and section material. </span>

    <span style="font-family: HOVFKU+Optima-Bold; font-size:15px"> Course aims </span>
    <span style="font-family: HOVFKU+Optima-Bold; font-size:13px"> a. learn lecture notes </span>
</div>
```

(c) Corresponding Data Points

TEXT	LABEL	FONT SIZE	BOLD or NOT
Course policies and expectations:	1	15	1
Class attendance.	0	13	1
You can miss up to 3 class sessions during the semester withoutpenalty. After	0	13	0
that,	0	13	0
each	0	13	1
missed class will result in a one-third letter grade reduction in your	0	13	0
final	0	13	1
course grade.	0	13	0
Policy on laptops, cellphones, iPods, etc.:	0	13	1
just as you would not read a newspaper in class,	0	13	0
please respect your instructors and your fellow students by refraining from non-course-related use of electronic devices during class. Midterm exams will cover lecture and section material.	0	13	0
Course aims	1	15	0
a. learn lecture notes	0	13	0

(a) Sample text from PDF file, (b) Corresponding HTML code for the text and (c) Corresponding data points from parsing HTML tags

NOTE: The 'LABEL' column in (c) is manually tagged and not generated via parsing.

Figure 3.2: Extraction of Data from Documents

$$\boxed{\textbf{abcdefghijklm}}$$

Figure 3.3: CMBX10 Font without making it bold

Extracted Data Points

The final data points are also shown in the Fig 3.2, which was generated by parsing the HTML tags using REGEX. *BoldList* referred to in Algorithm 1, is a list of fonts that are already bold due to their properties and do not require to be made bold such as *CMBX10*, *CMBX12* and *BlairMdITCTT-Medium*. For example, the font style *CMBX10* as shown in Fig 3.3 already looks bold even without style attribute having any mention of it being bold. In such cases the font family attribute has no mention of it being bold, so the algorithm compares the font family to a list of such fonts to check if the text contained in the tag is bold or not. *Document Data* (DOC_DATA) is the output of Algorithm 1. Each element of the output list represents a data point, which contains some text, its boldness flag, and its font size. The whole process yielded 83,194 data points, which was then exported into an Excel file.

It is not enough to only test the heading detection approach from chapter 4 on course outlines. To illustrate the generalizability of this approach, we validated it using 12,919 data points extracted using the same process as discussed above from 25 documents randomly sampled from the internet using search keywords like "report" and "article". This dataset would be referred to as *general set* throughout this research.

3.4.2 XML Conversion

XML Conversion provides finer details about the tags. Each tag represents a single character and contains its font size, font family, location and tells if the character is bold. The location of each character is stored in an attribute called bbox (bounding box) which mentions its location and size using 4 numbers. An example of how this looks in a converted document is given in Box 3.2.

Algorithm 1 Extracting Text Format Information

Input: The document
Output: A list *DOC_DATA* containing text with their font size and boldness flag

BoldList = List of all fonts that are already bold   ▷ *Initializations*
Regex = r "$< \backslash s*?span[^>] * font - size : (\backslash w*)px[^>]* > (.*?) < \backslash /span \backslash b[^>]* >$"
StartIndices = [ ]
EndIndices = [ ]
BoldnessFlag = 0

Convert PDF to HTML using *pdf2txt*   ▷ *Document Conversion*
Set HTMLCode as the extracted code from the generated HTML file

for *m in FindIndices('<span',)* **do**
  | Append m in list StartIndices   ▷ *Look for the starting of span tags*
end

for *n in FindIndices('', HTMLCode)* **do**
  | Append n in list EndIndices   ▷ *Look for the ending of span tags*
end

for *s,e in StartIndices, EndIndices* **do**
  | Append HTMLCode[s:e] in a list TagText   ▷ *Get the content of each span tag*
  | *using the locations found above.*
end

for *tag in TagText* **do**
      ▷ *Extraction of style information from the tags.*
  StartIndex = FindLocation('style=', tag)
  EndIndex = FindLocation('px"', tag)
  if *If tag[StartIndex:EndIndex] contains the word bold or any fonts from BoldList*
  then
    | Set BoldnessFlag for tag element to 1.
  end
end

for *tag in TagText* **do**
  for *For Match in LocateMatches(Regex, HTMLCode)* **do**
    | Append BoldnessFlag, match in the list DOC_DATA
    | ▷ *Consolidation of the final output list using the extracted information. The*
    | *variable Match contains the font size and text from tag*
  end
end

Box 3.2: Regular Expressions Used for Parsing

For example the word *Sings* appears as a sequence of the following tags:

<text   font="Georgia-Bold"   bbox="36.000,742.332,43.788,755.880"
size="13.548">S</text>
<text   font="Georgia-Bold"   bbox="43.801,742.332,51.697,755.880"
size="13.548">i</text>
<text   font="Georgia-Bold"   bbox="51.725,742.332,57.965,755.880"
size="13.548">n</text>
<text   font="Georgia-Bold"   bbox="57.965,742.332,62.213,755.880"
size="13.548">g</text>
<text   font="Georgia-Bold"   bbox="62.231,742.332,70.511,755.880"
size="13.548">s</text>

3.5 PDF To Image Conversion For White Space Analysis

The process begins with the conversion of the PDF document into a set of images
for white space analysis, using *BufferedImage* class which is available in a basic Java
library. A PDF with N pages will generate N images. Once the images have been
generated, their resolution is reduced to make the processing faster due to the fact
that it significantly reduces the number of pixels to be analysed. Deciding how much
to reduce the resolution is important, because the purpose of the process is to rec-
ognize the white spaces only. The next section examines how to reduce the resolution.

3.5.1 Choosing A Pixel Size

To decide how much to reduce the resolution we use the distance between two lines
as a measure. Fig 3.4 shows the distance between the lines; denoted by X. When the
pixel size is equivalent to a square of size X/2 or less as shown in Fig 3.4(a), we will
be able to detect either the pixels from upper line or lower line. But if the size of the
pixel is chosen to be larger than that, as shown in Fig 3.4(b), there is a possibility
of missing the white line. The reason is that the text might be present in upper and

lower pixel, which eliminates the possibility of both rows being white rows(lines only consisting of white spaces). The size of the pixel should be smaller than half of the minimum distance between any two lines.
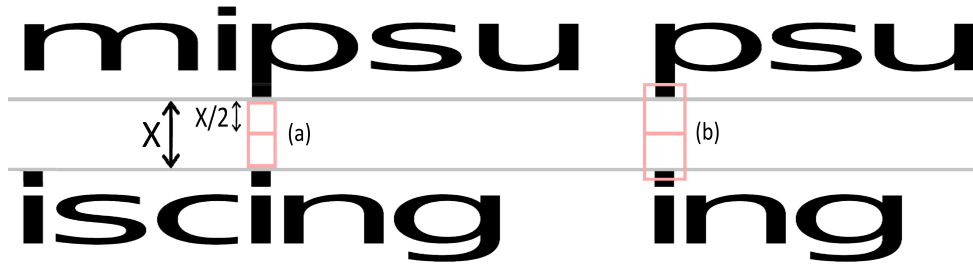
Figure 3.4: Pixel Size Example

Once the images are ready to be analysed, each of the image is converted into a Page Matrix (P) of zeros and ones using the Equation 3.1.

$$PAGE\_MATRIX_{ij} = \begin{cases} 0; & \text{if pixel at (i, j) is white/background color} \\ 1; & \text{otherwise} \end{cases} \tag{3.1}$$

Therefore, pixels containing text is marked as 1 and the ones which are of white/background color are marked as 0. Background color is selected as the most commonly used pixel color in the document.

Note: The algorithm uses the most common pixel color as the background color for the page.

3.6   Conclusion & Future Work

This chapter discussed all the conversions used to extract data from documents used for further analysis. HTML conversion provides us with all the information we need for detecting headings as discussed in chapter 4 because it provides us with font size,

type and if the text is bold or not. This process successfully converted and extracted data from 498 out of 500 documents. *Pdf2Text* failed to convert 2 documents into HTML which resulted in no extracted data from these files. Further testing revealed that some PDF files, generated by conversion from HTML webpages, also fail when they are converted back to HTML for data extraction. Exploring other tools to use as alternatives to *Pdf2Text* that enable HTML conversion for the identified documents would be beneficial in future research.

The steps for converting PDF to image using Java's *BufferedImage* class were also discussed. This process generates a matrix that is used for detecting header/footer text and multiple text columns by performing white space analysis as discussed in chapter 5. Although not required for this application, XML conversion could be considered for future research as it provides us with more details such as the exact location of each character and the bounding box location that could be useful for further extending the classifier and also improving the document layout analysis process by reducing false positives.

Chapter 4

Supervised Learning Approach For Heading Detection

4.1   Introduction

As the amount of information stored using PDF documents increases worldwide, large scale text based analysis requires increasingly automated processes, as the amount of document processing is time consuming and labour intensive for human professionals. Categorizing data into separate sections is quite easy for humans, as they rely on visual cues such as headings to process textual information. Machines, despite being able to process large amounts of information at high speeds, require effort to classify and interpret text based data. This chapter explores the application of classifiers to operationalize a system that would aid in the identification of headings.

A classifier, by being trained on labelled data enables categorization of PDF text into headings and non-headings based on the training data. This research involved comparing and systematically testing a variety of classifiers for the purpose of selecting the ones best suited for detecting headings in a PDF document. Identifying headings is the next step in looking for the relevant text that needs to be extracted.

Section 4.2 discusses the methodology followed, including preprocessing of the data points and training of classifiers. Section 4.3 contains details of the evaluation metrics used. Section 4.4 contains all the test results, which are discussed further in Section 4.5 to make a selection for the best suited classifier for detecting headings. Section 4.6 concludes the chapter by discussing everything that has been done.

4.2   Methodology

The whole process is divided into four sections. The detailed flow structure is shown in Fig 4.1.

4.2.1   Data Collection

The data collection process is explained in the previous chapter. 83,194 data points collected from 500 documents are used to train and evaluate the classifiers using 10
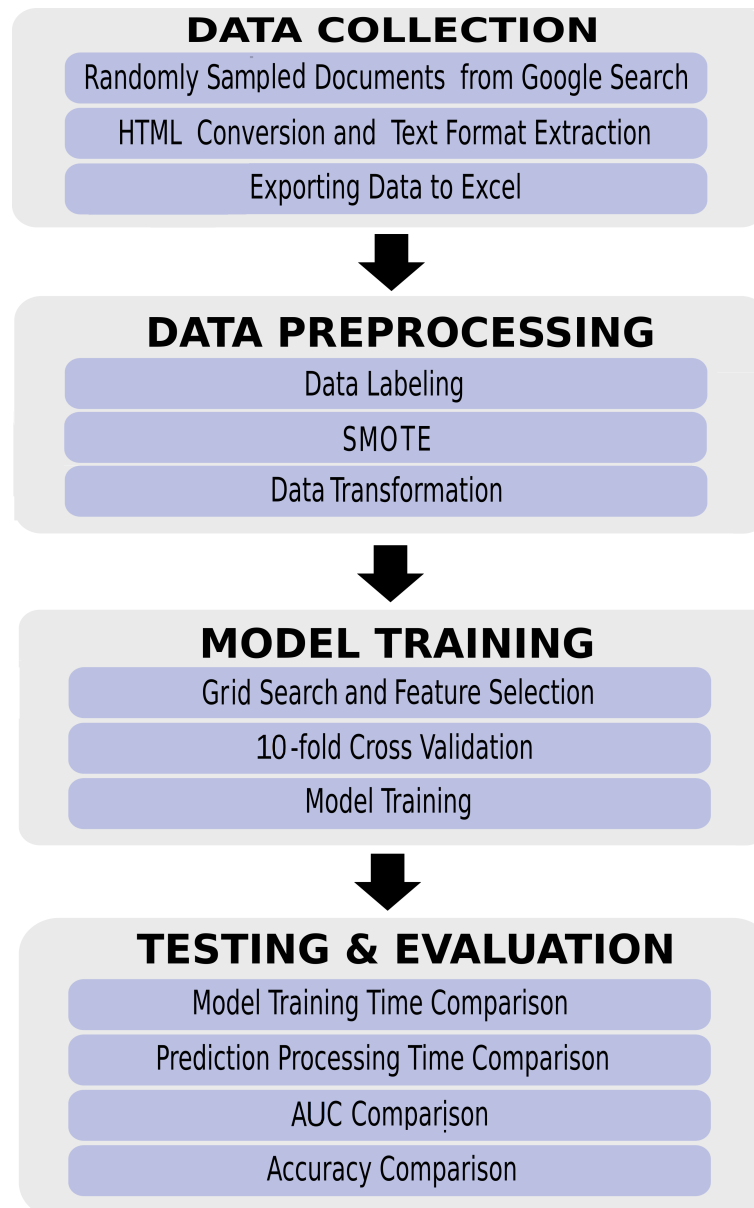
Figure 4.1: The Research Methodology

fold cross validation.

4.2.2 Data Preprocessing

The process of transforming raw data into usable training data is referred to as *data preprocessing*. The steps of data preprocessing for this research are as follows:

Data Labelling:

Data labelling refers to the process of assigning data points labels, this makes the data suitable for training supervised machine learning models. I manually labelled all 83,914 data points. If the text in the data point is a heading the label is set to 1 and for all other elements the label is set as 0. This defines two possibles classes for the classifier to make predictions on. Labelling data is one of the most important steps of preprocessing because the performance of the trained model depends on how well the data is labelled. Example of labelled data points is provided in Fig 3.2(c).

Balancing The Dataset:

The dataset is considered imbalanced if the prevalence of one class is more than the other. The number of headings in our dataset is very less as compared to non-headings, this is because of the fact that the number of headings in a document is far less than the number of other text elements. Sklearn's implimentation for Synthetic Minority Over-sampling Technique (SMOTE) is used to balanced the dataset, which does so by creating synthetic data points for the minority class to make it even [8, 39].

Data Transformation:

The process of transforming data into a form that has more predictive value is known as data transformation. The purpose of data transformation is to convert raw data points into 'features' that contribute to more predictive value in training and decision making related to heading identification. For example, font size and text are two features from the raw data which, in their base form, do not have much value but can be transformed into useful features for training an efficient model. The list of transformed data fields are as follows:

Course Description ⟶ **Font Size: 12** ⟶ **Font Size: 9**

This course will provide accounting students with basic knowledge of information systems that will enable them to examine business processes. Emphasis will be placed on information and document flows; internal control; business processes, the analysis design and development of accounting systems; and using databases. Instructional strategies include assignments, quizzes, group presentations, a midterm and a final comprehensive examination.

Program Outcomes ⟶ **Font Size: 12** ⟶ **Font Size: 9**

Successful completion of this and other courses in the program culminates in the achievement of the Vocational Learning Outcomes (program outcomes) set by the Ministry of Training, Colleges and Universities in the Program Standard. The VLOs express the learning a student must reliably demonstrate before graduation. To ensure a meaningful learning experience and to better understand how this course and program prepare graduates for success, students are encouraged to review the Program Standard by visiting http://www.tcu.gov.on.ca/pepg/audiences/colleges/progstan/. For apprenticeship-based programs, visit http://www.collegeoftrades.ca/training-standards.

Course Learning Outcomes ⟶ **Font Size: 12** ⟶ **Font Size: 9**

The student will reliably demonstrate the ability to:

1. Describe key issues associated with the application of Information Systems (IS) for the accounting function in business organizations.
2. Apply systems techniques to analyze, design and document systems and subsystems relationships
3. Explain the use of the Internet, intranets and e-commerce technology in business practices, and the related systems control requirements
4. Classify the key aspects of information security systems and how they are used to thwart the risks and threats associated with business processes
5. Analyze core business practices and processing systems and understand how they are implemented and controlled in Accounting Information Systems (AIS
6. Explain the systems planning and analysis life cycle – the development, implementation, operation and management of AIS

Figure 4.2: Font Size Threshold Assumption Example

- Font Flag

  Headings tend to be larger in terms of font size as compared to the paragraph text that follows. Therefore, a higher font size increases the probability that the text is a heading. However, since each document is unique, there can not be a single threshold applied across all instances.

  Thresholds are calculated for each document by measuring the frequency of each font size where each character with a particular font size is counted as one instance. The font size which has the maximum frequency is used as the threshold. This approach relies on the assumption that the most frequently used font size is the one that is being used for the paragraph text, so having any font size above that increases the probability of that text being a heading. Fig 4.2 shows that the most frequently used font size is for the paragraph text with size 9 and all other text above it has more chances of being a heading. *Font Flag* can take two possible values 0 and 1. If the font size for that data point is less than the corresponding threshold then the value is set to 0, otherwise it is set as 1.

- Text

  The text is transformed into the following feature variables, which also are listed in Table 4.1.

  - Number of Words: The number of words in the text can be used for training, as headings tend to have less words when compared to regular sentences and paragraphs.

  - Text Case: Headings mostly use title case, while sometimes they are in upper case as well. This variable tells whether the text is in upper case (all letters in upper case), lower case (all letters in lower case), title case (first letter of all words in uppercase) or sentence case (only the first letter of the text in uppercase).

  - Features From Parts of speech(POS) Tagging: POS Tagging is the process of assigning parts of speech (verb, adverb, adjective, noun) to each word, which are referred to as tokens. The text from each data point is first tokenized and then each token is assigned a POS label [5].

  The POS frequencies provides the model with information on the grammatical aspect of the text and can be used to exploit the frequency of these labels in

a text to identify headings and contribute to the accuracy of the model. For example, headings tend to have no verbs in them, though some might have them but absence of verbs increases the probability of the text being an heading. All frequency data collected from POS tagging is analysed in the feature selection process to differentiate between useful and irrelevant features collected through it. The frequency for each POS label is calculated and used to calculate the frequency of each POS tag in the text for each data point. These frequencies serve as potential features for the model.

All these brings the count of total number of features generated using the text to 11, 9 from POS tagging the text and 2 using its physical properties.

Table 4.1: List of all features

All features are integers, except for *Bold or Not* and *Font Threshold Flag* which are binary.

Feature Name	Description
Characters	Number of characters in the text.
Words	Number of words in the text.
Text Case	Assumes the value 0,1,2 or 3 depending on the text being being in lower case, upper case, title case or none of the three respectively.
Bold or Not	Assumes the value 1 or 0 depending on the text being bold or not.
Font Threshold Flag	Assumes the value 1 or 0 depending on the font size of the text being greater than the threshold or not.
Verbs	Number of verbs in the text.
Nouns	Number of nouns in the text.
Adjectives	Number of adjectives in the text.
Adverbs	Number of adverbs in the text.
Pronouns	Number of pronouns in the text.
Cardinal Numbers	Number of cardinal numbers in the text.
Coordinating Conjunctions	Number of coordinating conjunctions in the text.
Predeterminers	Number of predeterminer in the text.
Interjections	Number of Interjections in the text.

4.2.3   Feature Selection

After pre-processing, 14 training features are established. There is a need to select the top features for building each individual model with maximum accuracy. Table 4.1 lists all the features we are choosing from. To achieve this we used *Recursive feature elimination* with Cross-Validation (RFECV), which recursively removes weak attributes/features and uses the model accuracy to identify features that are contributing towards increasing the predictive power of the model [17]. The selection process is performed using the machine learning library, "scikit-learn".

Cross validation is done by making 10 folds in the training set where one feature is removed per iteration. As per this analysis the accuracy does not increase on choosing to train the Decision Tree classifier with more than the following seven features:

- Bold or Not

- Font Threshold Flag

- Number of words

- Text Case

- Verbs

- Nouns

- Cardinal Numbers

The same process is repeated for all the classifiers and their individual set of chosen features are listed in Table 4.2.

Table 4.2: Selected features for each classifier

Classifier Name	Selected Features
Decision Tree	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Cardinal Numbers
SVM	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Adjectives, Adverbs
k-Nearest Neaighbors	Bold or Not, Font Threshold Flag, Words, Verbs, Nouns, Adjectives, Cardinal Numbers, Coordinating Conjunctions
Random Forest	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Adverbs, Cardinal Numbers, Coordinating Conjunctions
Gaussian Naive Bayes	Bold or Not, Font Threshold Flag, Words, Verbs, Nouns, Adjectives, Cardinal Numbers, Coordinating Conjunctions
Quadratic Discriminant Analysis	Bold or Not, Font Threshold Flag, Words, Verbs, Nouns, Adjectives, Coordinating Conjunctions
Logistic Regression	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Adverbs, Coordinating Conjunctions
Gradient Boosting	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Cardinal Numbers
Neural Net	Bold or Not, Font Threshold Flag, Words, Text Case, Verbs, Nouns, Cardinal Numbers

4.2.4 Grid Search

Tuning each classifiers parameters for optimal performance is performed using accuracy from cross validation as a measure. We use various combinations of classifier parameters and choose the combination with the best cross validation accuracy. This process is performed on various classifiers to choose their corresponding parameters. A code snippet for performing this process for Gradient Boosting classifier is given in Box 4.1.

Box 4.1: Grid Search Code Snippet for Gradient Boosting Classifier

```
GDB_params = {
    'n_estimators': [100,150,200,250,300,350,450,500],
    'learning_rate': [0.05,0.1,0.15,0.2,0.25],
    'criterion': ['friedman_mse', 'mse', 'mae']
    'loss': ['deviance', 'exponential']
    'min_samples_split': [2,3,4,5],
    'min_samples_leaf': [1,2,3,4,5]
}
for estim_num in GDB_params['n_estimators']:
  for lrn_rate in GDB_params['learning_rate']:
    for critrn in GDB_params['criterion']:
      for ls in GDB_params['loss']:
        for sp in GDB_params['min_samples_split']:
          for lf in GDB_params['min_samples_leaf']:
            classifier=GradientBoostingClassifier(n_estimators=estim_num,
            learning_rate=lrn_rate,criterion=critrn,loss=ls,
            min_samples_split=sp,min_samples_leaf=lf)
            classifier.fit(training_data, training_labels)
            scores = cross_val_score(classifier, training_data, training_labels
            , cv=10)
            print "Estimator: ",estim_num, "Learning Rate: ", lrn_rate,
              "Criterion: ", critrn[:3],"loss: ",ls[:3],"min_samples_split: ",sp,
              "min_samples_leaf: ", lf,"Cross Validation Score: ",scores
```

The description along with the final selected tuning parameters for each classifier used in this research are discussed in the next section.

4.2.5 Training

After the most suitable features and parameters for each classifier have been selected, we can proceed with training the classfiers using scikit-learn [39].

Decision Tree

Decision trees are the most widely used amongst classifiers as they have a simple flow-chart like structure starting from a root node. It branches off to further nodes and terminating at a leaf node. At each non-leaf node a decision is made, which selects the branch to follow. The process continues to the point where a leaf node is reached, which contains the corresponding decison[26]. The classifier configuration used is listed in Table C.1 (Appendix C). Gini impurity is used as a measure for quality of a split, which tells if the split made the dataset more pure. Gini makes it computationally less expensive as compared to entropy which involves computation of logarithmic functions. The "best" option for strategy chooses the best split at each node. The minimum number of samples required to split an internal node is set to 2 and the minimum number of samples needed to be at a leaf node is set to 3. The code snippet for training this classifier with the chosen parameters is given in Box 4.2.

Box 4.2: Code Snippet for Training Decision Tree Classifier

```
treeclf  =  DecisionTreeClassifier(criterion  =  'gini',  splitter  =  'best',
min_samples_split = 2, min_samples_leaf = 3)
treeclf = treeclf.fit(traindata, truelabels)
```

Support Vector Machine (SVM)

It is a classifier that uses multi-dimensional hyperplanes to make classification. SVM also uses kernel functions to transform the data in sucha way that it is feasible for the hyperplane to effectively partition classes[2]. Table C.2 (Appendix C) lists the classifier configuration parameters used for training. The kernel used is radial basis function(rbf), degree of the polynomial kernel function is set to 3 and gama is set to "auto". The shrinking heuristics were enabled as they speed up the optimization. Tolerance for stop criteria is set to $2e - 3$ and 'ovr'(one vs rest) decision function is chosen for decision function shape. The code snippet for training this classifier with the chosen parameters is given in Box 4.3.

Box 4.3: Code Snippet for Training Support Vector Machine Classifier

```
svmclf = SVC(kernel='rbf', degree=3, gamma='auto', shrinking=True,
tol=0.002, decision_function_shape='ovr')
svmclf = svmclf.fit(traindata, truelabels)
```

k-Nearest Neighbors

The main idea behind k-Nearest Neighbors is that it takes into account the class of its neighbors to decide how to classify the data point under consideration. Each neighbors class is considered as their vote towards that class and the class with the most votes is assigned to that data point[22]. As listed in Table C.3 (Appendix C), the number of neighbours used to classify a point is set to 10. Each neighbours are weighed equally as weights is set to 'distance'. Minkowsky distance function used as the distance metric. The code snippet for training this classifier with the chosen parameters is given in Box 4.4.

Box 4.4: Code Snippet for Training k-Nearest Neighbors Classifier

```
neighclf = KNeighborsClassifier(n_neighbors = 10, weights = 'distance', metric
= 'minkowski')
neighclf = neighclf.fit(traindata, truelabels)
```

Random Forest

This classifier works by choosing random data points from the training set and creating a set of decision tress. The final decision regarding the class is made by aggreggation of the outputs from all the trees [27]. As listed in Table C.4 (Appendix C), the number of trees in the forest is set to 2 and 'gini' is used as a measure for quality of a split. The maximum depth of trees is set to 5 and the maximum number of features to be considered while searching for the best split is se to 'auto'. The

minimum number of samples required to split an internal node is set to 2 and the minimum number of samples needed to be at a leaf node is set to 3. The number of parallel jobs to running for both fit and predict is set to 1. The code snippet for training this classifier with the chosen parameters is given in Box 4.5.

Box 4.5: Code Snippet for Training Random Forest Classifier

```
RandomForestClassifier(n_estimators = 2, criterion = 'gini', max_depth = 5,
max_features='auto', min_samples_split=2, min_samples_leaf=3, n_jobs=1)
rndForstclf = rndForstclf.fit(traindata, truelabels)
```

Gaussian Naive Bayes

This classifier works by using Bayesian theorem with assumption of strong independence between the predictors(features). It is very useful for large data sets as it is quite simple to build and has no complicated iterative parameters[44]. Table C.5 (Appendix C) lists the classifier configuration parameters used for training. This classifier does not have much to set when it comes to configuring parameters. Prior probabilities of the classes is set to [0.5, 0.5] as the number of headings is less as compared to other text. The code snippet for training this classifier with the chosen parameters is given in Box 4.6.

Box 4.6: Code Snippet for Training Gaussian Naive Bayes Classifier

```
gaussianclf = GaussianNB(priors = [0.5, 0.5])
gaussianclf = gaussianclf.fit(traindata, truelabels)
```

Quadratic Discriminant Analysis

It works under the assumption that the measurements for each class are normally distributed while not assuming the covariance to be identical for all the classes. Discriminant analysis is used to choose the best predictor variable(s) and is more flexible than linear models making it better for a variety of problems [49]. Table C.6 (Appendix C) lists the classifier configuration parameters used for training. Prior probabilities of the classes is set to [0.5, 0.5] as the number of headings is far less as compared to other text. The threshold used for rank estimation is set to $1e - 4$. The code snippet for training this classifier with the chosen parameters is given in Box 4.7.

Box 4.7: Code Snippet for Training Quadratic Discriminant Analysis Classifier

```
quadclf = QuadraticDiscriminantAnalysis(priors = [0.5, 0.5], tol = 0.0001)
quadclf = quadclf.fit(traindata, truelabels)
```

Logistic Regression

It is a discriminative classifier, therefore it works by discriminating amongst the different possible values of the classes [40]. Table C.7 (Appendix C) lists the classifier configuration parameters used for training. Penalization method is set to l2. The tolerance for stopping criteria is set to $2e - 4$. The parameter 'fit_intercept' is set to true adding a constant to the decision function. The optimization solver used is 'liblinear' and the maximum number of iterations taken for the solvers is set to 50. Multiclass is set to 'ovr' fitting a binary problem for each label. The number of CPU cores used for parallelizing over classes is set to 1. The code snippet for training this classifier with the chosen parameters is given in Box 4.8.

Box 4.8: Code Snippet for Training Logistic Regression Classifier

```
logisticRegr = LogisticRegression(penalty=l2, tol=0.0002, fit_intercept =
True, solver='liblinear', max_iter=50, multi_class = 'ovr', n_jobs=1)
logisticRegr = logisticRegr.fit(traindata, truelabels)
```

Gradient Boosting

This classification method uses an ensemble of weak prediction models in a stage wise manner. In each stage, a weak model is introduced to make up for the limitations of the existing weak models[43]. Table C.8 (Appendix C) lists the classifier configuration parameters used for training. The loss function to be optimized is set as 'deviance' and learning rate is set to 0.1. The minimum number of samples required to split an internal node is set to 2, the minimum number of samples needed to be at a leaf node is set to 1 and maximum depth of the individual regression estimators set to 3. The number of boosting stages is set to 150 and the measure of quality of a split is set to 'friedman_mse'. The code snippet for training this classifier with the chosen parameters is given in Box 4.9.

Box 4.9: Code Snippet for Training Gradient Boosting Classifier

```
grdbstcf = GradientBoostingClassifier(loss = 'deviance', learning_rate = 0.1,
min_samples_split = 2, min_samples_leaf = 1, max_depth = 3, n_estimators =
150, subsample = 1.0, criterion = 'friedman_mse')
grdbstcf = grdbstcf.fit(traindata, labels)
```

Neural Net

This classifier works by imitating the neural structure of the brain. One data point is processed at a time and the actual classification is compared to the classification made by the classifier. Any errors recorded in the classification process are looped back into algorithm to improve classification performance in future iterations[16, 28].

As listed in Table C.9 (Appendix C), the classifier is configured to have one hidden layer with 100 units. The activation function used for the hidden layer is 'tanh'. The solver used for weight optimization is 'lbfgs'. The batch rate is set to 'auto' and the initial learning rate is set to 0.001. The parameter 'max_iter' is set to 300, which for 'adam' solver defines the number of epochs. Sample shuffle is set to true, which enables sample shuffling in each iteration. The exponential decay rates for estimates of the first and second moment vector is set to 0.9 and 0.999 respectively. The code snippet for training this classifier with the chosen parameters is given in Box 4.10.

Box 4.10: Code Snippet for Training Neural Net Classifier

```
nurlntclf = MLPRegressor(hidden_layer_sizes = (100, ), activation = 'tanh',
solver = 'lbfgs', learning_rate = 'invscaling', batch_size = 'auto', learn-
ing_rate_init = 0.001, max_iter = 300, shuffle = True, beta_1 = 0.9, beta_2 =
0.999)
nurlntclf = nurlntclf.fit(traindata, labels)
```

4.3   Evaluation

Evalution is going to be based on the following measures:

4.3.1   Training and Prediction Time

When dealing with a large number of documents, the time required to train a model and make predictions is important and is dependant on the type of classifier used, the number of features and the amount of data points. In this research all classifiers are trained using the same number of features and data points, therefore 'time taken' provides a good measure of variations in training and prediction speed associated with each different classifier being used. Of note, the training time for a classifier should be considered in context, as training only needs to be performed once and can be saved for later use. Therefore, a model that takes a long time to train can still be practical so long as it does not take a lot of time to make predictions.

4.3.2   Confusion matrix

Confusion matrix is used to represent a classifier's performance with respect to a data set having known true values, an example representation of such matrix is shown in Fig 4.3. A confusion matrix is generated for each classifier, serving as a summary of prediction results for it and further used to calculate the evaluation parameters. These calculated evaluation parameters are used to compare these classifiers to each other and are mentioned below [33]. The code snippet for generating a confusion matrix using sklearn is given in Box 4.11.

Box 4.11: Code Snippet for Training Decision Tree Classifier

```
sklearn.metrics.confusion_matrix(truelabels, predictedlabels)
```

n = 200	Predicted NO	Predicted YES	
Actual No	600 (TN)	121 (FP)	721
Actual Yes	79 (FN)	1200 (TP)	1279
	679	1321	

Figure 4.3: Confusion Matrix Example

- Sensitivity: It is also known as recall or true positive rate and is the odds of getting a positive test outcome given a positive case.

- Specificity: It is also known as true negative rate and is the odds of getting a negative test outcome given a negative case.

- Precision: It gives the odds of that the outcome classified as positive is actually positive.

- F1 Score: It provides a score for the balance between precision and recall.

- Total Accuracy: It is the measure of how many predictions in total are correct or the measure of trueness of the results.

These parameters are calculated as follows:

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F1Score = \frac{2TP}{2TP + FP + FN}$$

$$TotalAccuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where,
 TP means true positives,
 TN means true negatives,
 FP means false positives, and
 FN means false negatives

4.3.3 AUC

A receiver operating characteristics(ROC) curve is used to visualize the trade-offs between sensitivity and specifity. These graphs are used for performance based selection
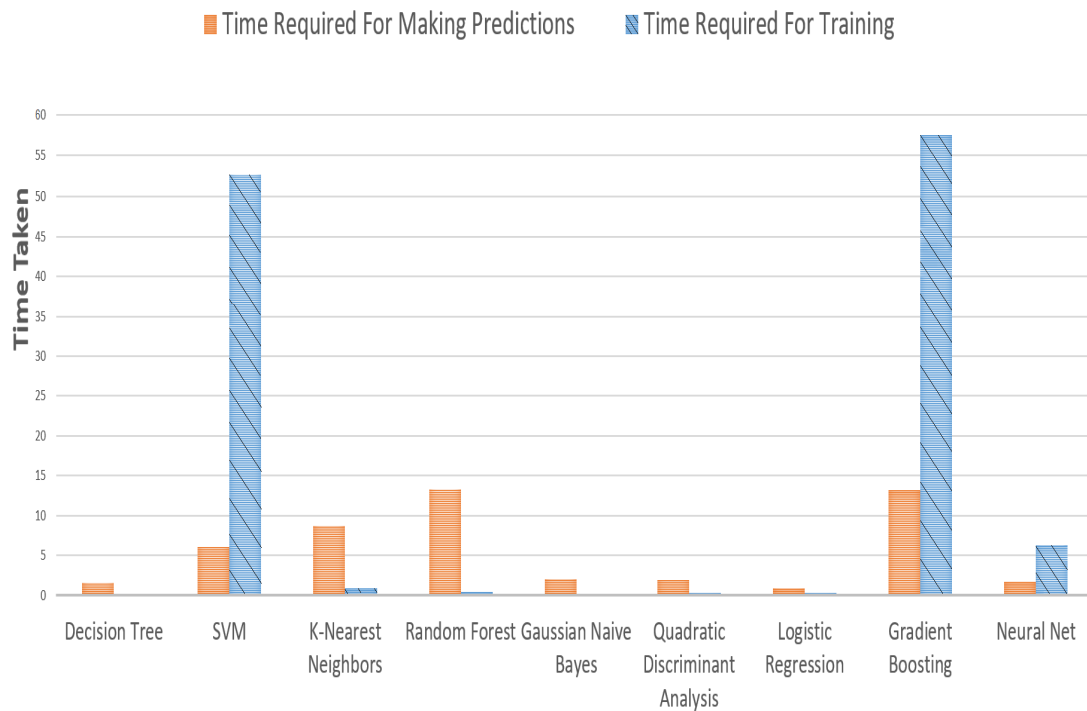
Figure 4.4: Time (in seconds) required to train classifiers and run predictions on test data

of classifiers. The graph can be reduced to a numerical measure, AUC(or AUROC) which is the area under the ROC graph with values ranging from 0 to 1[12].

4.4   Test Results

4.4.1   Training and Prediction Time

Fig 4.4 shows time required for training and making predictions using these classifiers. Time shown is average of 10 observations, which is done to reduce the effect of programs running in the background on the comparison.

4.4.2 Confusion Matrix Based Evaluation

Table 4.3 shows the results of this evaluation. Gradient Boosting classifier shows the best specificity, precision and net accuracy. Gaussian Naive Bayes shows the best sensitivity and SVM shows the best F1 score.

4.4.3 AUC

Table 4.4 shows the AUC scores for the classifiers used in this research. The discussion section provides more information on how we used AUC score to select the best classifier.

Table 4.3: Classifier Accuracy
Highest Value For Each Measure is Bold

Classifier	Sensitivity	Specificity	Precision	F1 Score	Accuracy
Decision Tree	0.972	0.944	0.897	0.933	95.37 %
SVM	0.970	0.933	0.881	0.930	95.09 %
K-Nearest Neighbors	0.940	0.932	**0.907**	0.923	94.82 %
Random Forest	0.971	0.935	0.882	0.926	94.85 %
Gaussian Naive Bayes	0.960	0.911	0.847	0.909	93.47 %
Quadratic Discriminant Analysis	0.963	0.912	0.848	0.910	93.52 %
Logistic Regression	0.956	0.919	0.855	0.903	93.17 %
Gradient Boosting	**0.981**	**0.946**	0.902	**0.940**	**95.83** %
Neural Net	0.980	**0.946**	0.901	0.939	95.80 %

Table 4.4: AUC Values for all Classifiers

Classifier	AUC
Decision Tree	**0.98**
SVM	0.97
K-Nearest Neighbors	0.96
Random Forest	0.97
Gaussian Naive Bayes	0.96
Quadratic Discriminant Analysis	0.96
Logistic Regression	0.95
Gradient Boosting	**0.98**
Neural Net	**0.98**

4.5 Discussion & Future Work

4.5.1 Overall Results

We recorded the time (in seconds) required for training each classifier and also time for making predictions as shown in Fig 4.4. Time taken by a classifier to make predictions is important when processing documents in bulk as it can increase the processing time. Time taken to train a classifier only has to be done once therefore it is not given that much importance. The Decision Tree Classifier took the least time for training while Gradient Boosting took the most. On comparing the prediction time Logistic Regression takes the least time and Random Forest takes the most. While prediction time is not the most important factor while choosing a classifier we take it into consideration when two classifiers are performing approximately the same.

The top three classifiers based on net accuracy are Decision Tree, Gradient Boosting, and Neural Network, however classifier selection can not solely rely on accuracy [18, 25]. Therefore, we also weigh the metrics like AUC, F1 score, sensitivity, and specificity to choose the best suited classifier for detecting headings.

The top three classifiers in terms of F1 score, precision, sensitivity and specificity are Decision Tree, Gradient Boosting, and Neural Network and the top 3 in terms of AUC as shown in Table 4.4 are again Decision Tree, Gradient Boosting, and Neural

Network. The system is going to be dealing with documents in bulk and the prediction time for Decision Tree is better when compared to both Gradient Boosting and Neural Network. Therefore, we would be choosing our configuration of the Decision Tree for making the classifications.

4.5.2 Testing The Generalizability

Testing the chosen classifier on a general set of documents is important to show that it performs well on documents other than course outlines. We tested the chosen Decision Tree classifier on 12,919 data points collected from documents like reports and articles[1]. These data points were manually tagged using a survey. All the participants were graduate students from computer science department and were asked to point out headings and subheadings in the documents. Table 4.5 shows the results which are equivalent as compared to when tested on course outlines.

Table 4.5: Test Results For General Set

Category	Value
Total Data points	12919
Sensitivity	0.928
Specificity	0.966
Precision	0.964
F1 SCORE	0.946
Accuracy	94.73 %
AUC	0.97

[1]Repository available at: https://github.com/sahib-s/Generalizability/

Table 4.6: Pearson Correlation Coefficient Between Each Feature Used in the Selected Classifier and Final Decision Labels

Feature Name	Pearson Correlation Coefficient
Bold or Not	0.7022
Font Threshold Flag	0.2385
Words	0.1389
Verbs	0.1229
Nouns	0.1207
Cardinal Numbers	0.1201
Text Case	0.0660

4.5.3   Analysing The Results

The discussed configuration of Decision Tree is best suited to detect heading as discussed in Section 4.5.1. Analyzing the contribution of each feature towards the final decision made by the classifier is also important to understand the implications of the results. Table 4.6 shows the pearson correlation coefficient for all the features used in the selected classifier and final decision label. The list is in descending order of pearson correlation coefficient, therefore the top feature in the table contribute the most towards the final decision. Each feature was removed from the classifier one at a time and drop in evaluation metrics also verify the order of contribution presented by using the pearson correlation coefficient. Therefore, the top three contributing features are the ones that rely on the physical attributes of the text.

4.5.4   Extending The Classifier

The extension of this work includes tagging of multiple labels like heading, paragraph text, header/footer text and table text. While classifying paragraph text is possible using the existing features, for properly classifying table text and header/footer text more data features are necessary. We are currently looking features from our white space detection approach discussed in chapter 5 and bounding box data from PDF to XML conversion to provide the model with what it needs to make this classification.

4.6   Conclusion

This chapter has provided a structured methodology and systematic evaluation of a heading detection system for PDF documents. The detected headings provide information on how the text is structured in a document. This structural information is used for extracting specific text from these documents based on the requirements of the field of application.

Chapter 5

Document Layout Analysis & Text Extraction

5.1   Overview & The Framework

Extracting learning outcomes requires identifying which text to extract from the outline of a post-secondary course descriptive documents. The approach proposed here
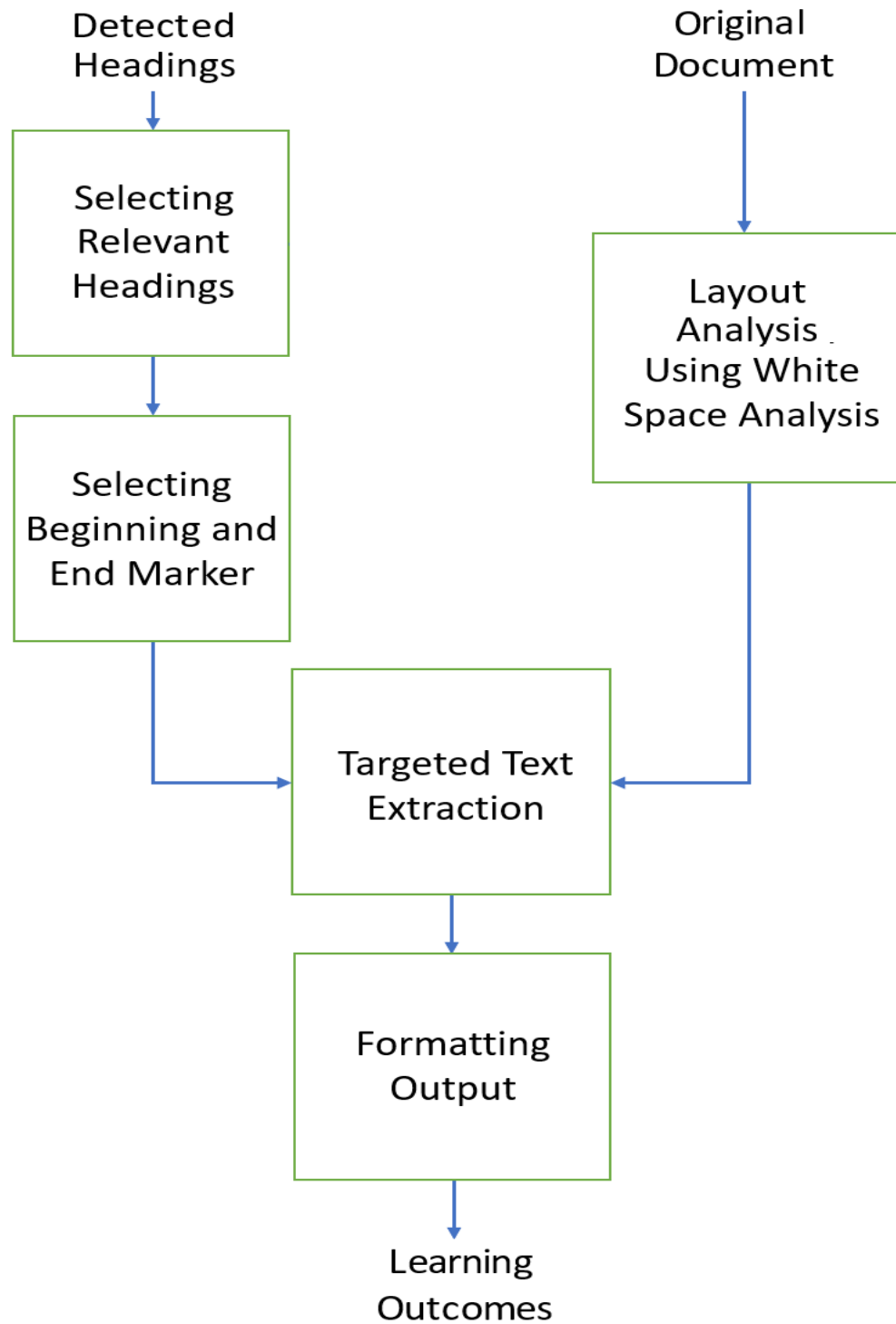
Figure 5.1: A Framework to extract learning outcomes from PDF documents

is similar to the steps a human follows which is to use headings to judge the context of the text that follows. The assumption made here is that all the content within the course outline PDF is appropriately divided using descriptive headings.

This chapter provides a methodology to extract the learning outcomes from course outlines as illustrated in Fig 5.1. The classifier discussed in the previous chapter uses the output of Algorithm 1 to generate a list of headings which is consequently used to identify learning objectives. Once the text is identified for extraction, analyzing the layout of the document is imperative to ensure that the extraction proceeds according to the flow of the text and the header/footer text does not get extracted with the required text as it will disrupt its flow. This is accomplished by performing White Space Analysis, in which the system looks for patterns in the empty spaces around the text. The original document acts as the input for the white space analysis. This process enables the script to extract the learning outcomes while taking the layout of the document into consideration. The extracted text is then formatted to get the final learning outcomes.

5.2   Selecting Relevant Headings - Supervised Approach

This section discusses a supervised keyword based approach similar to *bag-of-words model* to select relevant headings from a list of headings generated using the classifier discussed in last chapter. Bag of words is a representation used to model textual data by describing the occurrences of words within a text [23]. The approach proposed here uses relevant headings to select keywords for identifying relevant headings based on their occurrence in unseen documents.

5.2.1   Data Collection, Labelling & Keyword Selection

Dataset used in this chapter is the same as chapter 4 consisting of 500 documents. Headings from these 500 documents are manually tagged as relevant (1) or not (0) based on the text that is required to be extracted. All relevant headings are tokenized and the frequency of each token (word) is calculated. Most frequent words that are

relevant to what needs to be extracted are selected manually. The selected keywords are given in Box 5.1.

Box 5.1: Selected Keywords

> The list of keywords for locating learning outcomes is as follows:
>
> Keywords = ['Learning', 'Academic', 'Objectives', 'Description', 'Aims', 'Class', 'Course', 'Goals', 'Outcomes']

5.2.2 Data Transformation

The text is transformed into features which are as follows:

- Number of Words
  The number of words in the heading.

- Relevance Score
  Each heading is assigned a *relevance score*, calculated using Equation 5.1.

$$Relevance\ Score = \frac{Number\ of\ relevant\ keywords\ in\ the\ heading}{Total\ number\ of\ words\ in\ the\ heading} \qquad (5.1)$$

- Keyword Flag
  The value is 1 if that keyword is present in the heading else 0. The total number of such features depends on the number of keywords selected. For this research there are nine such features one for each keyword.

Three instances of the data points generated are given in Fig **??**.

LABEL	HEADING TEXT	WORDS	RELEVANCE SCORE	LEARNING	ACADEMIC	OBJECTIVE	DESCRIPTION	AIMS	CLASS	COURSE	GOALS	OUTCOMES
1	COURSE LEARNING OUTCOMES	3	1.00	1	0	0	0	0	0	1	0	1
0	REQUIRED COURSE MATERIALS	3	0.33	0	0	0	0	0	0	1	0	0
0	EVALUATION	1	0.00	0	0	0	0	0	0	0	0	0

Figure 5.2: Sample of Data Points for Relevant Heading Selection

5.2.3 Training

Grid Search is used to select the optimal combination of hyper-parameters, using the same process as discussed in section 4.2.4. Consider the fact that there are only 3 types of features there is no need to perform feature selection. Decision Tree Classifier is trained using the optimized hyper parameters and the code snippet for training is given in Box 5.2.

Box 5.2.: Code Snippet For Decision Tree Classifier

```
treeclf  =  DecisionTreeClassifier(criterion  =  'gini',  splitter  =  'best',
min_samples_split = 2, min_samples_leaf = 1)
treeclf = treeclf.fit(traindata, truelabels)
```

5.3   Selecting Beginning and End Markers

Once the relevant headings have been selected, the next step is to only extract the text that follows the heading, as opposed to extracting all of the text. To extract this specific text we need a beginning point i.e., *beginning marker* and an ending point i.e. *end marker* in the text. The text that makes up the relevant heading is used as the beginning marker and the heading that follows the relevant heading is used as the end marker. These two markers serve as landmarks in the text and encapsulate the part of the text required to be extracted.

RELEVANT HEADING ⟶ Learning Outcomes:

Method of Evaluation:

First test: worth 20% (in-class exam)

Second test: worth 20% (in-class exam)

Assignment: 25%

Cumulative Final Exam: worth 35% (during final exam period)

Learning Outcomes: ——————————————————— **Beginning Marker**

The successful student will be able to:

1. Understand and explain all of the theories related to crime. — **Extracted Text**
2. Critically discuss and apply the theories to cases of crime.
3. Identify and explain criminological concepts and apply them to a given situation.

How to Contact Me: ——————————————————— **End Marker**

If you have a question or would like to discuss course content, please see me during my office hours. Outside of class and/or office hours, email is my preferred method of contact. Please note that I do not discuss matters of grades, course content (e.g., definitions of concepts, differences between theoretical perspectives, etc.,) or matters of special accommodation by email. I require that email correspondence be saved to such items as arranging a meeting during my office hours and/or for discussing information that is **not** posted on the course outline. I also do not answer questions regarding information that is posted on the course outline (e.g., quiz/exam dates, required readings, exam information, etc.,) and/or any material or announcements that are discussed in class.

Email should be professionally prepared, spell- and grammar-checked, and not written in "text message" format. All communications must be written from your UWO account. The subject heading should read the course title. Sign the email with your full name and student number. You can expect a response within 24-48 hours, excluding weekends.

How to get important information:

Your first sources of information are the course syllabus and OWL. OWL will be used to post course information, content, reminders, and important instructions regarding deadlines, expectations, requirements, etc. It is expected that you will check OWL regularly to ensure that you are kept up to date on new and revised course content. It is student's responsibility to ensure that you are kept up-to-date on course content and announcements.

Grades (except the final exam and year-end mark) will be posted to OWL as they become available. Due to privacy regulations, I do not provide grades or discuss issues regarding grades via email. If you wish to discuss your grade(s) please see me during my office hours.

Figure 5.3: Defining the beginning and end markers

Algorithm 2 Locating Beginning and End Markers

Input: A list *HEADING_LIST* which contains all the headings in the document and a list *RELEVANT* which contains all the relevant headings.

Output: Two lists containing the beginning and end marker(s) for all relevant heading(s).

Function *LocateMarkers (RELEVANT, HEADING_LIST)*

Set r = 0   ▷ *Loop Counter Initializations*

while *r is less than length of list(RELEVANT)* **do**

Set n = 0

while *HEADING_LIST[n] is not equal to RELEVANT[r]* **do**

▷ *Finds the index of the relevant heading in HEADING_LIST*

Increment n by 1   ▷ *Increments loop counter*

end

▷ *Appends both marks to respective lists.*

APPEND HEADING_LIST[n] to the list BeginningMarkers

if *HEADING_LIST[n+1] has same font size as BeginningMarkers* **then**

▷ *To ensure text extraction does not stop at sub-headings*

APPEND HEADING_LIST[n+1] to the list EndMarkers

else

Increment n   **while** *Font Size(HEADING_LIST[n+1] is not equal to Font size (BeginningMarkers)* **do**

Increment n

end

APPEND HEADING_LIST[n+1] to the list EndMarkers

end

Increment r by 1   ▷ *Increments loop counter*

end

Return BeginningMarkers, EndMarkers   ▷ *Returns two lists containing the markers.*

Algorithm 3 explains the process of selecting these markers in detail. The algorithm uses *DOC_DATA* (output from Algorithm 1 discussed in chapter 3) and *RELEVANT* (output from Algorithm 2) as input. The text between these two markers is extracted,

and each relevant heading generates its pair of beginning and end markers. In case of multiple relevant headings, there would be multiple pairs of beginning and end markers to enable extraction of multiple relevant paragraphs. As shown in Fig 5.3, the relevant heading "Learning Outcomes:" is chosen as the beginning marker and the heading that follows "How to Contact Me:" is the ending marker. The algorithm uses their location in the text to extract the text that falls between these two markers.

5.4   Layout Analysis

Analysing the layout of the document is one the most important steps of this process as it addresses the following two issues:

- Avoiding text from header and footer section in the final extracted text.

- Maintaining the integrity of text when dealing with multiple columns of text.

To accomplish these tasks, our approach analyses the white spaces in the document to detect headers/footers and text column locations. The PDF document is converted to set of images and pixel color is used to generate a 3D matrix($PAGE\_MATRIX$), which consisting of 2D matrices, one for each page. These matrices are generated using the process described in chapter 3 and are used for further analysis. The 2D matrices represent each page as a grid of pixels, where each box in the grid represents a pixel.

ROW_SUM	NET_SUM	NET_SUM	ROW_SUM
0	0	0	0
6	6	6	6
6	12	12	6
6	18	18	6
0	18	18	0
14	32	36	18
17			14
16			18
17			11
8			18
18			18
12			0
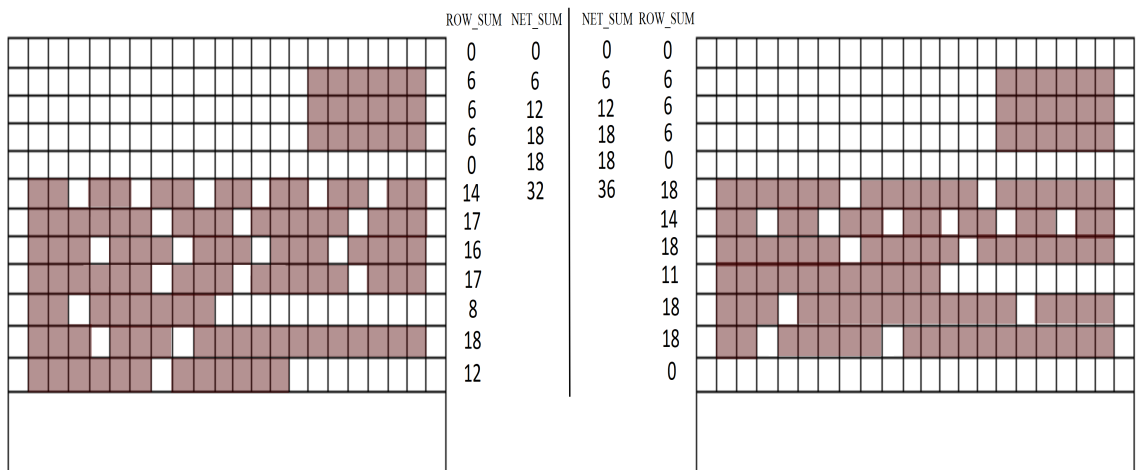
Figure 5.4: Header Detection Example

Algorithm 3 Locating Header Region

Input: A 3-D matrix *PAGE_MATRIX*, which represents the whole page as a grid of pixels.

Output: A floating point number telling what percentage of the top of the page is the header region.

Function *DetectHeader (PAGE_MATRIX)*

    Set p = 0   ▷ *Loop Counter Initializations*

    Set r = 0

    while *p is less number of pages in the document* **do**

        while *r is less than half the number of rows in PAGE_MATRIX[p]* **do**

            ROW_SUM[p][r] is equal to the sum of all values in that row

            if *r is not equal to 0* **then**

                NET_SUM[p][r] = NET_SUM[p][r-1] + ROW_SUM[p][r]

            else

                NET_SUM[p][r] = ROW_SUM[p][r]

            end

            Increment r   ▷ *Increments loop counter*

        end

        Increment p   ▷ *Increments loop counter*

    end

    SET HEADER_CUTOFF as the row till NET_SUM is same for all matrices.

    SET HEADER_REGION as HEADER_CUTOFF divided by number of rows in matrix

    Return HEADER_REGION   ▷ *Returns the header region.*

5.4.1 Detecting Headers and Footers

Using *PAGE_MATRIX*, Algorithm 4 generates a *ROW_SUM* Matrix with each element representing the sum of a row in *PAGE_MATRIX*. The variable *NET_SUM* is the sum of all rows till that row. To search for headers, the algorithm checks the *NET_SUM* of all pages and the row till which *NET_SUM* is the same is marked as *HEADER_CUTOFF*. It is assumed that the maximum height of the header or footer area is half of the page height. Therefore, the algorithm checks the header and footer length until half the page height, as their height can not be greater than half of the page length. The example in Fig 5.4 shows how this works by showing two pages side by side. The coloured pixels depict any text and blank ones depict the white or

background color pixels. The *ROW_SUM* is calculated for all rows and once that is done the *NET_SUM* is calculated which remains same until row 5 in this example. So according Algorithm 4, the header area is until row 5. All text in the header region is considered as header text. The same process is repeated to search for footers, the only difference is that the sum is checked from bottom to top.

5.4.2   Locating Text Columns and Images

To detect multiple text columns, we use two variables *White Rows* and *White Columns*. The rows and columns that are completely blank and have no text in them. The *Row-Sum* gives us the sum of all pixels in that row and if that sum is 0 , then that row is considered as a white row. So, mathematically Row_i is a white row if

$$Row_i = \begin{cases} 0; & \text{if row sum is greater than 0} \\ 1; & \text{otherwise} \end{cases}$$

These white rows act as boundaries to detect columns of text in between them. Same way the algorithm checks for the sum of column pixels using white rows as boundaries and if the sum is zero that column is considered as a white column. In Fig 5.5, the discussed approach first detects the white rows by looking for complete rows with white or background color pixels and find R1, R2, R3 and R4. Once white rows have been located, it starts to choose white rows in pairs as boundaries to detect any columns in between those rows. First it chooses R1 and R2 and does not find any column in between them. Then it chooses R2 and R3 and finds a column C1 between those two white rows. At the end it checks between R3 and R4 and does not find a column in between them. Using these rows and columns the algorithm figures out the layout of the document for text extraction purposes. For the region with images, the text extraction process discussed in the next section does not generate any text, thus not effecting the final output.

5.5   Text Extraction

5.5.1   Targeted Text Extraction

The width of the footer and header region tells the algorithm not to extract the text from that region. The detected columns and rows help the algorithm to divide them

R1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse et dolor urna. Pellentesque vel nisl vehicula, tincidunt justo vitae, pulvinar sem. Phasellus gravida consectetur libero, at pulvinar ante aliquam vitae. Suspendisse consectetur nisl sed leo venenatis varius. Integer facilisis dapibus elit sed sagittis.

R2

C1

Suspendisse et magna tincidunt, rutrum ipsum cursus, dignissim arcu. Duis et efficitur lorem, at egestas nisi. Vestibulum consequat, massa eget feugiat faucibus, turpis augue aliquam velit, non sodales sapien eros eu quam. Integer convallis tellus faucibus ipsum convallis accumsan.

R3

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Cras ultricies metus vel erat facilisis faucibus. Pellentesque luctus sapien pharetra, accumsan ipsum quis, consequat arcu. Suspendisse ex urna, molestie non egestas et, vestibulum ut libero.

R4

Figure 5.5: Layout Example

into boxes of text and extract text from them individually. Now, for this information to be useful the algorithm will also need to be able to extract text from specific regions of a PDF rather than the whole document. To accomplish this the algorithm uses PDFTextStripperByArea() method of PDFBox library available in Java [38]. This method takes a rectangle object as input for the area from where the text needs to be extracted. Once the text extraction is done, the algorithm finds the indices at which these markers are in the text. Now the text in between these indices is extracted and stored in a variable *EXT_TEXT*. In case of multiple relevant headings, this process will use their individual markers to get the relevant text from each of the relevant headings.

5.5.2 Formatting Output

Once we have the text from each of the relevant headings, the text is still in the form of a continuous string that needs to be divided into points in accordance to how the author has formatted the document. Algorithm 5 explains the whole process in detail. This step takes the text extracted from the previous step as input and outputs a list (OUTPUT). At the end, each element of this list will store a sentence or a paragraph depending on the text. To achieve this the algorithm looks for elements such as new lines, periods and bullet points. The bullet points currently being used by the algorithm are shown in Box 5.3.

Box 5.3: Currently Used Bullet Point Separators

```
[
'\ xef \ x82 \ xa7' , '\ xe2 \ x80 \ xa2' , '\ xef \ x82 \ xb7' ,
'\ xef \ x81 \ xb1' , '\ xe2 \ x97 \ x8f' , '\ xef \ x82 \ xa1' ,
'\ xef \ x82 \ xbe' , '\ xef \ x83 \ x98' , '\ xef \ x81 \ xb6' ,
r'\ s[\ d]{1,2}'     , r'\ s[\ d]{1,2}'     , r'\ s[\ d]{1,2}\ .\)',
r'\ ([\ d]{1,2}\ )'  , r'\ s [A-Za-z] \. \ s'  ,   r'\ s[A-Za-z]\)',
r'\ ([A-Za-z]\)'       , r'\ s[-]'       ,r'\ s[*]'        , r'\ s[>]'
]
```

Using these elements the algorithm decides whether to segment the string at that point or not. The indices at which the text needs to be segmented are stored in the

variable *Dlist*. If the author intended that information as separate sentences then this step will output text segmented into separate sentences and if the text was intended as a series of paragraphs then the output will be in the same format. This maintains the integrity of what the author wanted to convey using the text. The system outputs a list of *Learning Outcomes* or a paragraph if the instructor formatted it in that manner. This step is aimed towards retaining as much formatting information as possible to make further analysis of the extracted text more accurate.

Algorithm 4 Formatting Output

Input: A string *EXT_TEXT* which is discussed in the last subsection.

Output: A list *OUTPUT* which contains all the headings in the document.

Function *FormatOutput (EXT_TEXT)*

    Set FirstIndex = 0   ▷ *Index Counter Initializations*

    if *EXT_TEXT contains bullet points* **then**

                           ▷ *Look for bullet points as a point of seperation*

        for *i = 0 to length(EXT_TEXT)* **do**

            if *EXT_TEXT[i] has bullet point* **then**

                | Append i to Dlist

            end

        end

    else

        for *i = 0 to length(EXT_TEXT)* **do**

             ▷ *Look for period followed by new line character as a point of seperation*

            if *EXT_TEXT[i] has a period* **then**

                if *EXT_TEXT[i+1] has new line character* **then**

                    | Append i to Dlist

                end

            end

        end

    end

    for *Index in Dlist* **do**

        Append EXT_TEXT[FirstIndex:Index] to OUTPUT

        Set FirstIndex = Index

    end

    Return OUTPUT   ▷ *Output of the algorithm*

5.6 Test Results & Discussion

Testing was performed on all the 500 documents for layout analysis and selecting beginning/end markers, as there are no files required for training, which implies all files are unseen. For testing everything else 10 fold cross validation was used. Table 5.1 shows the performance of the relevant heading selection classifier. Other classifiers were not compared for this task as the first classifier tested gave near perfect results.

Table 5.1: Test Results For Relevant Heading Selection

Category	Value
Sensitivity	0.999
Specificity	0.100
Precision	0.100
F1 SCORE	0.999
Accuracy	99.96 %
AUC	0.99

Table 5.2 shows the individual performance of the sub processes that use the whole 500 documents for testing. The text markers are correctly detected for 478 of 500 documents. *Layout Analysis* is divided into two parts: Header/Footer Detection and Multiple Text Column Detection. Header/Footer detection worked for 459 out of 500 documents. For *Multiple Text Column and/or Images* 491 out of 500 documents performed correctly.

Table 5.2: Individual Test results

MODULE	Total documents For This Category	Properly Working
Selecting Beginning & End Markers	500	447
Layout Analysis - Header Footer Detection	500	459
Layout Analysis - Multiple Text Column and/or Images	500	491

Table 5.3 shows the overall performance of the algorithm which uses 10 cross validation, Learning outcomes are correctly extracted from 81.8% documents. 9.6% of

the documents are partially working because the current algorithm does not have the mechanisms to address them completely. The documents referred to as partially working are the ones in which one or two of the sub processes fail leading to partially correct output. The reason behind this is the current limitations of the system which are discussed in Section 6.3 of the next chapter in detail.

Table 5.3: Overall Test Results

CATEGORY	Accuracy
Working	81.8%
Partially Working	9.6 %
Not Supported	8.6 %

5.7   Conclusion

In this chapter, we propose a robust approach consisting of multiple sub tasks to extract specific text from a document based on the application area, which in this case is extraction of learning outcomes from course outlines. The keywords decide what is getting extracted from the documents, so they need to be chosen as per the application area. We implemented this algorithm to extract learning outcomes out of course outlines, it can be used to extract other information from all kinds of supported documents. Text under those relevant headings is extracted using the information gathered from layout analysis for the header/ footer and text column location. The extracted text is then segmented into small sentences and paragraphs according to how the author of the document intended for the information to be conveyed.The test results revealed that the accuracy of the system is currently 81.8% with scope of improvement in a few areas.

Chapter 6

Discussion, Future Work & Conclusion

6.1   Overview

The approaches proposed in this thesis has shown good overall accuracy with room for improvement. The extracted learning outcomes are being used for further analysis[35, 36, 37] and, before proceeding with the analysis, the extracted learning outcomes are shown to the user for review. While this step helps in making sure that the extracted learning outcomes are correct, improving the accuracy even further could help reduce human input even further. The areas that can be further improved are discussed in

detail below.

6.2   Main Contributions

The current literature and its shortcomings in supporting the task of extracting specific text from documents were discussed in chapter 2. An exploratory analysis of extracting formatting information from documents is presented in chapter 3, which also discusses all the conversions needed to extract this data. All the extracted data is used for further analysis, which includes heading detection and white space analysis. Detect headings in documents using the classifier discussed in chapter 4 enables our approach to work on documents which do not have a given format, which is a limitation in the approaches presented by El-Haj et al. and Ramakrishnan et al. [10, 42].

We also present an approach for analyzing the white spaces in documents to detect header/footer text and multiple text columns using the same method. This reduces the amount of computation required therefore reduces the time required to perform the analysis. Thus, overcoming all the limitations, we discussed in the literature review.

6.3   Scope For Improvement & Current Exceptions

6.3.1   Extending The Heading Detection Classifier

The classifier currently makes a binary decision where it classifies a candidate text as heading or not. While this serves the purpose, extending its functionality by tagging multiple labels like heading, paragraph text, header/footer text, and table text can make this system more accurate and help support more types of documents. While classifying paragraph text is possible using the existing features, for properly classifying table text[11, 19], text columns, and header/footer text more data features are necessary. Future work should look into white space detection and bounding box data from PDF to XML conversion to provide the model with what it needs to make this classification[48].

COURSE OBJECTIVES ———— Beginning Marker

1. Broaden student's thinking of the concepts of health, illness & care. Topics include:
    a. Key paradigms and methodological approaches to health
    b. Personal and public conceptualizations of health, illness and quality healthcare
    c. The social determinants of health
    d. Current issues in maintaining Canadians' health and how these relate to policy development.
    e. Strategies for improving the health care system from the major perspectives introduced in this course.
2. To provide the students with a skill set to more critically analyze health related data and information. In class discussion and assignments will provide opportunities to:
    a. Learn how to synthesize research and information pertaining to a specific national or global health issue

Page Break

2

Detected End
Marker

Health Studies Program ————

HST209H – Introduction to Health

COURSE SYLLABUS

Fall 2016

UNIVERSITY COLLEGE
UNIVERSITY OF TORONTO
TORONTO, CANADA M5S 3H7

Header
Area

b. Explore, interpret and compare existing data and health information from multiple sources
c. Analyze current health issues
d. Make brief presentations based on the analyses

REQUIRED READINGS ————Intended End Marker

Required Course Text (Available from University of Toronto bookstore):

Raphael, D., Bryant, T. and Rioux, M. (2010). *Staying Alive: Critical perspectives on health, illness and health care* (2nd ed.). Toronto: Canadian Scholars' Press Inc.

Figure 6.1: Heading False Positive

This will enable the classifier to perform both heading detection and layout analysis; thus eliminating the need to perform layout analysis as a separate step. Such classifier will also help avoid text in header/footer areas to be detected as headings. As illustrated in Fig 6.1, the bold text in the header region is classified as heading due to its physical characteristics. The classifier is not able to tell that it is a part of the header region. Therefore, having inputs on the physical layout in the classification process will solve this problem discussed above. Also, if the classifier is able to classify elements as header/footer text, table text and text columns, there won't be a need to perform separate layout analysis thus saving processing and time.

6.3.2   Scanned Documents

Scanned documents are currently not supported because our approach for text extraction and relevant heading detection would not work on them. The two things that are needed in the current algorithm to make it compatible with such documents are:

- For scanned documents, it is important to address the skew angle, as not all documents get scanned perfectly straight. This leads to text skewed in a slight upwards or downwards direction as shown in Fig 6.2. A method to correct or account for skew is required. Future work could extend our approach by combining it with published methods to address skewness in text documents[32, 45, 47].

- A Scanned document is a combination of scanned images of the original document. Hence they require OCR to extract text and its format characteristics like font size. There are many OCR approaches available for extracting text[7, 34]. The proposed approach also needs to extract the text format from scanned documents for which we are still researching the current literature.

Once these points have been addressed, our proposed approach will be able to support scanned documents because our approach for layout analysis works on images and would work fine with scanned documents once the skew is corrected.

Quizzes: There will be 4 - 5 short quizzes during lectures. The purpose of a quiz is for understanding lecture materials. Dates will be announced prior to the quizzes.

Examinations: Students will not be allowed to bring class notes or supplementary material during the exams.

Team project: Detailed information of team project will be distributed in separate sheets and discussed in class.

Course grades: To determine the final grade of the course, the following weights will be applied:

- Team project: 20%

- Assignments: 20%

- Quizzes: 10%

- Mid-term tests: 20%

- Final exam: 30%

Table 1: Important dates

	Description	Date
	Assignment 1 (released)	Thursday, Jan. 19, 2017
	Midterm 1	Tuesday, Feb. 7, 2017
	Assignment 2 (released)	Thursday, Feb. 16, 2017
Tests	Midterm 2	Tuesday, Feb. 28, 2017
	Assignment 3 (released)	Thursday, Mar. 16, 2017
	Final project presentation	Tuesday, Mar. 28, 2017
	Final exam	To be posted by scheduling office

Other important dates:

First Day of Classes	Monday, January 9, 2017
Final Day of Classes	Friday, April 7, 2017
February Break	Monday, Feb 20 - Friday Feb 24, 2017

Figure 6.2: Example of Scanned PDF With Skew

Page 2

Course Outline: BUS502   Principles of Economics for Accountants

3.   How does this course contribute to my learning?

Specific Learning Outcomes	Assessment Tasks	Graduate Qualities
On successful completion of this course you should be able to:	You will be assessed on the learning outcome in task/s:	Completing these tasks successfully will contribute to you becoming:
Demonstrate knowledge of advanced microeconomic principles and how they apply in real world situations.	1 and 3	Knowledgeable Empowered
Demonstrate knowledge of macroeconomic principles and their usefulness in analysing economic policy debates and options.	1 and 3	Knowledgeable Empowered
Demonstrate an ability to structure an economic argument and support it with relevant theory and evidence.	2	Creative and critical thinkers

4.   Am I eligible to enrol in this course?

*Refer to the **Coursework Programs and Awards - Academic Policy** for definitions of "pre-requisites, co-requisites and anti-requisites"*

4.1   Enrolment restrictions
Must be enrolled in a postgraduate program

4.2   Pre-requisites
Nil

4.3   Co-requisites
Nil

4.4   Anti-requisites
Nil

Figure 6.4: Relevant Text in Table Example

6.3.3 Text in Tables

The current algorithm does not detect tables. Therefore, the integrity of the text is not maintained when extracted from tables. An example of such a case is shown in Fig 6.4. To enable proper extraction, detecting tables and their underlying structure is important. Future work should look into detecting tables using visual seperators and table detection for scanned documents[11, 19].

6.3.4 Extending The Keyword Based Approach

The system currently looks for relevant headings using the classifier discussed in chapter 5. While this approach has near perfect accuracy, it can be further improved by using synonyms of these keywords in the scoring mechanism. The purpose of this is to link the keywords with their synonyms while performing the analysis. Future work should look into *WordNet* which is a lexical database that forms sets of synonyms by grouping words [30].

6.3.5 Documents Without Headings

The current approach looks for headings and extracts the text under it to get the relevant text. While this approach works well in majority of the documents, there are a few documents which do not divide the text into paragraphs and are formatted without headings as shown in Fig 6.5. For these kinds of documents a different approach is require. Future work should look into relevant synonym frequency as solution to look for relevant text. This approach looks for synonyms of relevant keywords in the text and uses their frequency to make the decision regarding its relevance. This idea is currently in its rudimentary state therefore needs more research and testing.

Computer Science 4310/5413
Web Health Informatics
Winter 2017

Instructor: Dr. Vijay Mago (AT 5023, Ext. 8310, vmago@lakeheadu.ca)

Textbook : Robert E. Hoyt and Ann K. Yoshihashi: *Health Informatics: Practical Guide for Healthcare and Information Technology Professionals*, Lulu.com, 2014

Time and location: TTh 10:00 a.m. - 11:30 a.m., RC 1003

Office hours: Monday 10:00 a.m. - 12:00 p.m., AT 5023

Course web page: To be on desire2learn

Computer Science 3413

An introduction to the organization and operation of the health systems of Canada as well as to various associated concepts including a variety of Electronic Healthcare Record systems, healthcare data and knowledge sharing, evidence and proximity based medicine, interoperability between diverse and distributed systems, medical terminology standards and clinical web applications.

Table 1: Important dates

	Description	Date
	Assignment 1 (released)	Thursday, Jan. 19, 2017
	Midterm 1	Tuesday, Feb. 7, 2017
	Assignment 2 (released)	Thursday, Feb. 16, 2017
Tests	Midterm 2	Tuesday, Feb. 28, 2017
	Assignment 3 (released)	Thursday, Mar. 16, 2017
	Final project presentation	Tuesday, Mar. 28, 2017
	Final exam	To be posted by scheduling office

First Day of Classes   Monday, January 9, 2017
Final Day of Classes   Friday, April 7, 2017
February Break   Monday, Feb 20 - Friday Feb 24, 2017

Figure 6.5: Example Of Documents Without Headings

6.4   Conclusion

In this thesis, an approach to extract specific text from documents is presented. This approach is applied to extract learning outcomes from course outlines which are being used for further analysis to support the automation of university/college transfer credit agreements in Ontario using semantic similarity algorithms to assess similarities in learning outcomes between related post-secondary credentials. To start with the analysis, we convert the document into HTML to extract text formatting information and convert it into a set of images to perform layout analysis.

The whole process is broken down into multiple sub-tasks, the two main being detecting headings and document layout analysis. The purpose of detecting headings is to know what all information is presented in the document, which is one of the limitations in the current literature as they work on documents with a given structure. Layout analysis is performed using white space detection and is used to detect both header/footer text and multiple text columns.This saves time by reducing the amount of processing required which is a big factor when dealing with documents in bulk. The system has shown an accuracy of 81.8% in extracting learning outcomes from course outlines and can be used for many other applications involving extraction of specific text from documents.

Appendix A

Application API

A.1 Overview

The API focuses on providing access to all the modules discussed in this thesis.

A.1.1 Run & Configure

The file *LO-API.py* should be running as it will be handling all the requests sent to the URLs given. The port on which it will run can be changed from the same file. The system has the following basic dependencies:

- Python 2.7
  External Libraries Required:

    - PIL

    - wand

    - pdf2txt

    - PyPDF2

- Java 6
  External Libraries Required:

    - PDFBox

    - PDF2IMG

A.2 Methods Available

A.2.1 Extracting Learning Outcomes

Extracting learning outcomes from a course Outline

URL

/extract_lo

Method

POST

URL Params

None

Data Params

Key: upfile (Passed using form data in body) < The PDF document >.

Success Response

HTML Code: 200

```
[ [
      "Our primary objective for First Year Chemistry is to offer you a
      comprehensive and relevant course on the fundamental concepts.",
      "We offer a number of resources to support your studies."
   ],[
      "Describe the subatomic composition of atoms, ions and isotopes.",
      "Calculate spectroscopic quantities in relation to electronic transitions.",
      "Predict the relative strengths of acids and bases."
] ]
```

Content Description: The output is a list containing list of learning outcomes. Each list inside the primary list represents the learning outcomes extracted from different relevant paragraphs. This can be used to display the learning outcomes extracted from different paragraphs separately.

Error Response(s)

- Passing a file other than PDF

  HTML Code: 202

  ```
  [ [
        "FILE TYPE NOT SUPPORTED"
  ] ]
  ```

- Calling a method other than post for this URL

  HTML Code: 405

  ```
  {
        "message": "The method is not allowed for the requested URL."
  }
  ```

A.2.2 Extracting Headings

Extracting headings from a document

URL

```
/extract_headings
```

Method

POST

URL Params

None

Data Params

Key: upfile (Passed using form data in body) < The PDF document >.

Success Response

HTML Code: 200

```
[ [
        "Introduction",
         "Learning Outcomes",
         "Grade Breakdown",

         .

         .

         .

        "Attendance",
        "Expectations",
] ]
```

Content Description: The output is a list containing list of headings generated using the supervised learning classifier discussed in chapter 4.

Error Response(s)

- Passing a file other than PDF

HTML Code: 202

```
[ [
    "FILE TYPE NOT SUPPORTED"
] ]
```

- Calling a method other than post for this URL

HTML Code: 405

```
{
    "message": "The method is not allowed for the requested URL."
}
```

A.2.3   Extracting Text Format From the Document

Extracts the text formatting information by converting PDF to HTML.

URL

```
/extract_file_format
```

Method

```
POST
```

URL Params

```
None
```

Data Params

Key: upfile (Passed using form data in body) < The PDF document >.

Success Response

HTML Code: 200

```
[[
      "11",
      "Description",
      0
  ],[
      "10",
      "Conversion of numerical grades to Final Letter Grades ",
      1
]]
```

Content Description: Each element in the list represents some text from the document. The order of text depends on the order it appears in the document. Each list element has the font size first, then the text and then the boldness flag (1 means bold and 0 means normal or not bold).

Error Response(s)

- Passing a file other than PDF

HTML Code: 202

```
[[
      "FILE TYPE NOT SUPPORTED"
]]
```

- Calling a method other than post for this URL

HTML Code: 405

```
{
    "message": "The method is not allowed for the requested URL."
}
```

A.2.4   Locating Header & Footer Area

URL

/locate_header_footer

Method

POST

URL Params

None

Data Params

Key: upfile (Passed using form data in body) < The PDF document >.

Success Response

HTML Code: 200

```
[
    8.522727272727273,
    10.795454545454547
]
```

Content Description: The response consists of two numbers, which tells what percent of the total page height are the header and footer regions.

So, for the above example content the header regions starts from the top of the page and extends till (8.522727272727273 x Page Height) pixels in the downward direction and footer regions starts from the bottom of the page and extends till (10.795454545454547 x Page Height) pixels in the upward direction.

Error Response(s)

- Passing a file other than PDF

  HTML Code: 202

  ```
  [ [
      "FILE TYPE NOT SUPPORTED"
  ] ]
  ```

- Calling a method other than post for this URL

  HTML Code: 405

  ```
  {
      "message": "The method is not allowed for the requested URL."
  }
  ```

A.2.5   Formatting The Extracted Text

Formats the extracted text according to how the author formatter in the document.

URL

> /format_text

Method

> POST

URL Params

> None

Data Params

> Key: text (Passed using form data in body) < Text that needs formatting >.

Success Response

HTML Code: 200

> [[
>
>   "This course is designed to provide an opportunity for students to satisfy
>   ACE Learning Outcome.",
>   "Exhibit global awareness or knowledge of human diversity through
>   analysis of an issue."
>]]

Content Description: The output is a list containing sentences and paragraphs depending on how the author formatted that document.

Error Response(s)

- Passing a file other than PDF

  HTML Code: 202

  ```
  [ [
        "FILE TYPE NOT SUPPORTED"
  ] ]
  ```

- Calling a method other than post for this URL

  HTML Code: 405

  ```
  {
      "message": "The method is not allowed for the requested URL."
  }
  ```

Appendix B

List of Abbreviations

- **API**   Application Program Interface
- **AUC**   Area Under Curve
- **CPU**   Central Processing Unit
- **DOM**   Document Object Model
- **FN**   False Negative
- **FP**   False Positive
- **GPL**   General Public License
- **HTML**   Hypertext Markup Language
- **OCR**   Optical Character Recognition
- **PDF**   Portable Document Format
- **POS**   Parts Of Speech
- **REGEX**   Regular Expression
- **ROC**   Receiver Operating Characteristic
- **SVM**   Support Vector Machine
- **TN**   True Negative
- **TP**   True Positive

- **URL**   Uniform Resource Locator

- **WSP**   White Space Polygons

- **WSR**   White Space Rectangles

- **XML**   eXtensible Markup Language

Appendix C

Values For All Classifier Parameters

Table C.1: Decision Tree Parameters

Parameters	Value
criteria	gini
spliter	best
min_samples_split	2
min_samples_leaf	3

Table C.2: Support Vector Machine Parameters

Parameters	Value
kernel	rbf
gamma	auto
degree	3
shrinking	True
tol	2e-3
decision_function_shape	ovr

Table C.3: K-Nearest Neighbors Parameters

Parameters	Value
n_neighbors	10
weights	distance
metric	minkowski

Table C.4: Random Forest Parameters

Parameters	Value
n_estimators	2
criterion	gini
max_depth	5
max_features	auto
min_samples_split	2
min_samples_leaf	3
n_jobs	1

Table C.5: Gaussian Naive Bayes Parameters

Parameters	Value
priors	[0.5, 0.5]

Table C.6: Quadratic Discriminant Parameters

Parameters	Value
priors	[0.5, 0.5]
tol	1.0e-4

Table C.7: Logistic Regression Parameters

Parameters	Value
penalty	l2
tol	2e-4
fit_intercept	True
solver	liblinear
max_iter	50
multi_class	ovr
n_jobs	1

Table C.8: Gradient Boosting Parameters

Parameters	Value
loss	deviance
learning_rate	0.1
n_estimators	100
max_depth	3
criterion	friedman_mse
min_samples_split	2
min_samples_leaf	1
presort	auto

Table C.9: Neural Net Parameters

Parameters	Value
hidden_layer_sizes	(100,)
activation	tanh
solver	lbfgs
batch_size	auto
learning_rate_init	0.001
max_iter	200
shuffle	True
beta_1	0.9
beta_2	0.999

Bibliography

[1] Ethem Alpaydin. *Introduction to Machine Learning. [Sl].* The MIT Press, 2010.

[2] Asa Ben-Hur and Jason Weston. A users guide to support vector machines. In *Data mining techniques for the life sciences*, pages 223–239. Springer, 2010.

[3] Øyvind Raddum Berg, Stephan Oepen, and Jonathon Read. Towards high-quality text stream extraction from pdf: technical background to the acl 2012 contributed task. In *Proceedings of the ACL-2012 Special Workshop on Rediscovering 50 Years of Discoveries*, pages 98–103. Association for Computational Linguistics, 2012.

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

[5] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[6] Thomas M Breuel. Two geometric algorithms for layout analysis. In *International workshop on document analysis systems*, pages 188–199. Springer, 2002.

[7] Jagruti Chandarana and Mayank Kapadia. Optical character recognition. *International Journal of Emerging Technology and Advanced Engineering*, 4(5):219–223, 2014.

[8] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelli-*

gence research, 16:321–357, 2002.

[9] Google Developers. Custom search json api, 2018. `https://developers.` `google.com/custom-search/json-api/v1/overview`,.

[10] Mahmoud El-Haj, Paul Rayson, Steven Young, and Martin Walker. Detecting document structure in a very large corpus of uk financial reports. *In: Proceedings of the ninth international conference on language resources and evaluation*, pages 1335–1338, 2014.

[11] Jing Fang, Liangcai Gao, Kun Bai, Ruiheng Qiu, Xin Tao, and Zhi Tang. A table detection method for multipage pdf documents via visual seperators and tabular structures. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 779–783. IEEE, 2011.

[12] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[13] David A Freedman. *Statistical models: theory and practice.* cambridge university press, 2009.

[14] Liangcai Gao, Zhi Tang, Xiaofan Lin, Ying Liu, Ruiheng Qiu, and Yongtao Wang. Structure extraction from pdf-based book documents. In *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*, pages 11–20. ACM, 2011.

[15] Jan Goyvaerts and Steven Levithan. *Regular expressions cookbook.* O'reilly, 2012.

[16] Daniel Graupe. *Principles of artificial neural networks*, volume 7. World Scientific, 2013.

[17] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[18] Jin Huang and Charles X Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering*, 17(3):299–310, 2005.

[19] MAC Akmal Jahan and Roshan G Ragel. Locating tables in scanned documents for reconstructing and republishing. In *Information and Automation for Sustainability (ICIAfS), 2014 7th International Conference on*, pages 1–6. IEEE, 2014.

[20] Deliang Jiang and Xiaohu Yang. Converting pdf to html approach based on text detection. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, pages 982–985. ACM, 2009.

[21] Declan Kennedy. *Writing and using learning outcomes: a practical guide*. University College Cork, 2006.

[22] Daniel T Larose. k-nearest neighbor algorithm. *Discovering knowledge in data: An introduction to data mining*, pages 90–106, 2005.

[23] Svetlana Lazebnik, A Torralba, L Fei-Fei, D Lowe, and C Szurka. Bag of words models. *Dostopno na: http://cs. nyu. edu/~ fergus/teaching/vision_2012/9_BoW. pdf*, 3, 2011.

[24] Xiaofan Lin. Header and footer extraction by page association. In *Proceedings of SPIE 5010*, pages 164–171, 2002.

[25] Charles X Ling, Jin Huang, and Harry Zhang. Auc: a better measure than accuracy in comparing learning algorithms. In *Conference of the canadian society for computational studies of intelligence*, pages 329–341. Springer, 2003.

[26] Rokach Lior et al. *Data mining with decision trees: theory and applications*, volume 81. World scientific, 2014.

[27] Gilles Louppe. Understanding random forests: From theory to practice. *arXiv*

preprint arXiv:1407.7502, 2014.

[28] Vijay Kumar Mago. *Cross-Disciplinary Applications of Artificial Intelligence and Pattern Recognition: Advancing Technologies: Advancing Technologies.* IGI Global, 2011.

[29] Tomohiro Manabe and Keishi Tajima. Extracting logical hierarchical structure of html documents based on headings. *Proceedings of the VLDB Endowment*, 8(12):1606–1617, 2015.

[30] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[31] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning.* MIT press, 2012.

[32] A Papandreou, Basilios Gatos, Stavros J Perantonis, and I Gerardis. Efficient skew detection of printed document images based on novel combination of enhanced profiles. *International Journal on Document Analysis and Recognition (IJDAR)*, 17(4):433–454, 2014.

[33] Rajul Parikh, Annie Mathai, Shefali Parikh, G Chandra Sekhar, and Ravi Thomas. Understanding and using sensitivity, specificity and predictive values. *Indian journal of ophthalmology*, 56(1):45, 2008.

[34] Chirag Patel, Atul Patel, and Dharmendra Patel. Optical character recognition by open source ocr tool tesseract: A case study. *International Journal of Computer Applications*, 55(10), 2012.

[35] Atish Pawar, Sahib Budhiraja, Daniel Kivi, and Vijay Mago. Are we on the same learning curve: Visualization of semantic similarity of course objectives. *arXiv preprint arXiv:1804.06339*, 2018.

[36] Atish Pawar and Vijay Mago. Calculating the similarity between words and sentences using a lexical database and corpus statistics. *arXiv preprint*

arXiv:1802.05667, 2018.

[37] Atish Pawar and Vijay Mago. Similarity between learning outcomes from course objectives using semantic analysis, blooms taxonomy and corpus statistics. *arXiv preprint arXiv:1804.06333*, 2018.

[38] Apache PDFBox. Apache pdfbox, 2014. `https://pdfbox.apache.org/`,.

[39] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[40] Maja Pohar, Mateja Blas, and Sandra Turk. Comparison of logistic regression and linear discriminant analysis: a simulation study. *Metodoloski zvezki*, 1(1):143, 2004.

[41] Fuad Rahman and Hassan Alam. Conversion of pdf documents into html: a case study of document image analysis. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, volume 1, pages 87–91. IEEE, 2003.

[42] 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):7, 2012.

[43] Greg Ridgeway. The state of boosting. *Computing Science and Statistics*, pages 172–181, 1999.

[44] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM, 2001.

[45] Daniel Rosner, Costin-Anton Boiangiu, Mihai Zaharescu, and Ion Bucur. Image skew detection: A comprehensive study. In *Proceedings of IWoCPS-3, The Third*

International Workshop On Cyber Physical Systems, Bucharest, Romania, 2014.

[46] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,, 2016.

[47] Mahnaz Shafii and Maher Sid-Ahmed. Skew detection and correction based on an axes-parallel bounding box. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(1):59–71, 2015.

[48] Y. Shinyama. Pdfminer: Python pdf parser and analyzer, 2010. `http://www.unixuser.org/~euske/python/pdfminer/`,.

[49] Toshiyuki Sueyoshi. Dea-discriminant analysis: methodological comparison among eight discriminant analysis approaches. *European Journal of Operational Research*, 169(1):247–272, 2006.

[50] Linda Suskie. *Assessing student learning: A common sense guide.* 2010.

[51] Matthew C Swain and Jacqueline M Cole. Chemdataextractor: a toolkit for automated extraction of chemical information from the scientific literature. *Journal of chemical information and modeling*, 56(10):1894–1904, 2016.

[52] E Umamaheswari Vasanthakumar and Francis Bond. A semantic multi-field clinical search for patient medical records. 2018.

[53] Lakehead University. Learning objective automated gap analysis, 2017. `http://www.loaga.science/`,.