

CS5604: Information Storage and Retrieval

Collection Management of Electronic Theses and Dissertations

Authors

Kulendra Kumar Kaushal
Rutwik Kulkarni
Aarohi Sumant
Chaoran Wang
Chenhan Yuan
Liling Yuan

Instructor

Dr. Edward A. Fox



Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
December 15, 2019

CS5604: Information Storage and Retrieval

Team CME

This research was done under the supervision of Dr. Edward A. Fox as part of the course CS5604.

4th edition, December 7, 2019

3rd edition, October 31, 2019

2nd edition, October 10, 2019

1st edition, September 19, 2019

Contents

List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Overview	1
1.2 VTechWorks ETD Dataset	2
1.3 Problem Definition	3
2 Literature Review	4
2.1 PDF Processing	4
2.1.1 Overview	4
2.1.2 Evaluation of Open-Source Bibliographic Reference and Citation Parsers	4
2.1.3 Big Data Text Summarization	5
2.1.4 GROBID	5
2.1.5 Science Parse	7
2.1.6 Apache Tika	7
2.1.7 PDFMiner	7
2.1.8 PyPDF2	8
3 Requirements	9
3.1 Extract Metadata and Text for ETD corpus	9
3.2 Preprocess the ETD corpus	10
3.3 User support	10
4 Approach, Design, Implementation	11
4.1 Experiment Design	11

4.2	Implementation	11
4.2.1	Chapter Level Text Extraction	11
4.2.2	TF-IDF Calculation	14
4.2.3	Transforming metadata for ingestion in Elasticsearch	18
4.2.4	Development of an Automated System	19
4.2.5	List of Visualizations to be provided in the Front End	22
4.2.6	Text Preprocessing	22
5	Evaluation	24
5.1	Manual Testing	24
5.1.1	Testing of Chapter Level Text Extraction	24
5.1.2	Testing of Extracted Text Preprocessing	26
5.1.3	Metadata Extraction Testing	28
5.1.4	Automated Testing	28
6	User Manual	30
6.1	Where to Get Data	30
6.1.1	VTechWorks ETD collection	30
6.1.2	GitLab repository	31
6.1.3	Metadata extraction and ingestion in ceph	32
7	Developer's Manual	36
7.1	Timeline	36
7.2	Slack	37
7.3	GROBID	37
7.3.1	Install in Ubuntu	37
7.4	PDFMiner	39
7.5	TF-IDF	40
8	Challenges and Limitations	41
9	Future Scope	42
9.1	Improving Chapter Level Text Extraction	42
9.2	Batch Processing of the Documents	42
9.3	Improving Automation Suite	42
10	Acknowledgements	43

Abstract

The class “CS 5604: Information Storage and Retrieval” in the fall of 2019 is divided into six teams to enhance the usability of the corpus of electronic theses and dissertations maintained by Virginia Tech University Libraries. The ETD corpus consists of 14,055 doctoral dissertations and 19,246 masters theses from Virginia Tech University Libraries’ VTechWorks system. Our study explored document collection and processing, application of Elasticsearch to the collection to facilitate searching, testing a custom front-end, Kibana, integration, implementation, text analytics, and machine learning. The result of our work would help future researchers study the natural language processed data using deep learning technologies, address the challenges of extracting information from ETDs, etc.

The Collection Management of Electronic Theses and Dissertations (CME) team was responsible for processing all PDF files from the ETD corpus and extracting the well-formatted text files from them. We also used advanced deep learning and other tools like GROBID to process metadata, obtain text documents, and generate chapter-wise data. In this project, the CME team completed the following steps: comparing different parsers; doing document segmentation; preprocessing the data; and specifying, extracting, and preparing metadata and auxiliary information for indexing. We finally developed a system that automates all the above-mentioned tasks. The system also validates the output metadata, thereby ensuring the correctness of the data that flows through the entire system developed by the class. This system, in turn, helps to ingest new documents into Elasticsearch.

List of Figures

1.1	Position in entire system	2
2.1	The architecture of PDF Miner	8
4.1	Folder structure of a ETD after chapter level text extraction	13
4.2	Sample ETD Introduction chapter	15
4.3	Parsed text of the same document (highlighted text indicates end of page shown in Figure 4.2)	16
4.4	Part of TF-IDF of one document	17
4.5	Part of BOW of one document	18
4.6	Part of doc-index dictionary	18
4.7	Flow Diagram of the Automated System	21
4.8	Folder Structure of an ETD	22
4.9	GROBID Unit test	22
5.1	Chapter level text extraction by XPath vs. manual extraction by Diff Checker	25
5.2	Original text generated by PDFMiner.six	26
5.3	Processed text	28
6.1	GitLab file structure	31
6.2	GROBID Container	32
6.3	Python client to access GROBID	32
7.1	Timeline	36
7.2	Slack	37
7.3	Files in the Gradle folder	38
7.4	Files in the GROBID folder	38

List of Tables

2.1	Human assessment of GROBID and Science Parse outputs	6
5.1	Chapter level text extraction by XPath and manual extraction	25
5.2	Differences between chapter level text extraction by XPath and manually extraction	27
5.3	Different test case scenarios.	29

Chapter 1

Introduction

1.1 Overview

As a leading global research university, on January 1, 1997, Virginia Tech was the first university which required graduate students to submit electronic theses and dissertations (ETDs) [2]. As of 2019, the local ETD dataset covers over 33,000 doctoral dissertations or masters theses.

ETDs are valuable information sources, but due to the lack of discoverability, they are still underutilized. Hence, retrieving ETDs is important for researchers and universities. Retrieving specific information from academic materials has many important applications, such as citation analysis [12]. It could aid those working to prepare award-winning theses [11]. One of the most important problems in ETD information retrieval is how to extract text and metadata properly from PDF files. In this report, we will address that problem, and also tackle problems related to the identification and extraction of sections and chapters. We hope our work would help future researchers be able to discover and reuse the potential useful resources from the ETDs.

The position of our team in the whole system is shown in Fig. 6.2. Many different PDF parsers [5, 7, 18] are implemented to convert PDF files to a structured format, e.g., XML or JSON. To extract metadata or elements – like affiliation, tables, and images – from ETDs successfully, we also propose a new approach to avoid errors during conversion. Moreover, the issue of automatic segmentation to identify sections and chapters is also addressed in this project.

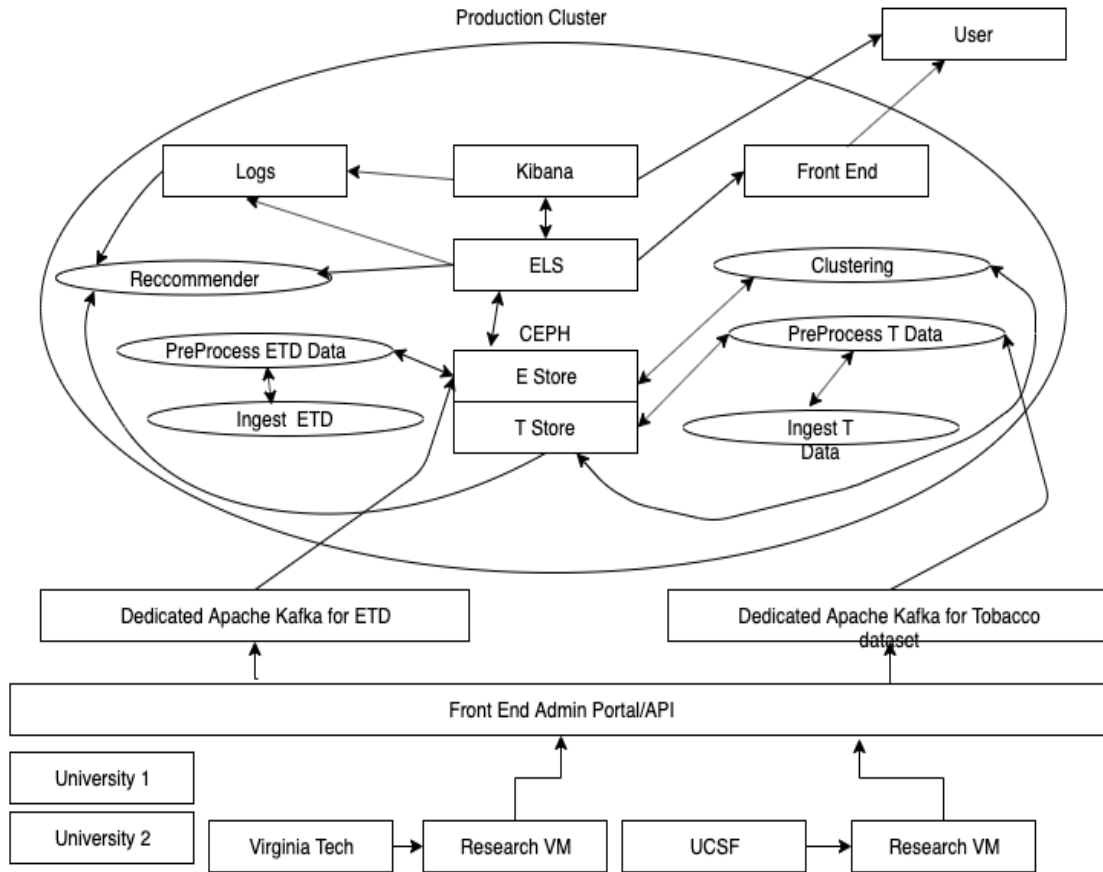


Figure 1.1: Position in entire system

1.2 VTechWorks ETD Dataset

The ETD corpus is downloaded from the Virginia Tech institutional repository, VTechWorks, and consists of over 33,000 documents: 14,055 doctoral dissertations, 19,246 masters theses, and some award-winning and undergraduate theses. The repository is maintained by the university library, and includes ETDs about all disciplines from all departments of Virginia Tech. For each ETD, there is one PDF document which is generally the main part, a metadata record, and some supporting documents. For older ETDs, the PDF

files resulted from scanned paper documents. In such cases, full-text files were extracted using optical character recognition.

1.3 Problem Definition

This project works on managing ETDs by answering the following research questions.

RQ1: *Can we extract metadata from an ETD document, and transform it into a format that can be ingested into Elasticsearch?*

Elasticsearch is a search server based on the Lucene library. Lucene is a open-source search engine software library. Elasticsearch provides a distributed, multitenant-capable full-text search engine with a RESTful web interface and schema-free JSON documents [1]. Enhancing our output to generate according to a suitable format that can be ingested by Elasticsearch should extend the applicability of our work.

RQ2: *Can we extract text files from PDF files and have content suitable for subsequent indexing and searching?*

A suitable structure, properly populated with text that is used in the subsequent indexing, would help future researchers to discover and retrieve the specific information they need.

RQ3: *Can we expand the extracted data by including a file for each chapter?*

Sometimes the researchers might be just interested in some specific sections. This action might be helpful to increase search specificity and save time for users.

RQ4: *Can we develop an automated system that can extract the metadata from new documents, process it and ingest it to Elasticsearch?*

New ETDs need to be added to our system as and when they are added to VTechWorks. So, in order to make our system more robust and up to date, an automated system to process and add the new ETDs to our system is necessary.

Chapter 2

Literature Review

2.1 PDF Processing

2.1.1 Overview

All of our electronic theses and dissertations are available as PDF files. It is difficult to extract the key data from such a file. Additionally, the formatting of different sections, as well as of the bibliography, changes from document to document. Thus, parsing a PDF file becomes a big challenge.

Preprocessing and extraction of metadata from the ETDs are important steps in related works that have been carried out in this domain. The rest of this chapter includes descriptions of some of the work done by researchers related to the extraction of metadata, text parsing, and providing support for big data text summarization. We include descriptions of popular tools and parsers, and highlight the comparison between them on different parameters, as discussed in various works.

2.1.2 Evaluation of Open-Source Bibliographic Reference and Citation Parsers

The growth in the volume of available scientific literature has resulted in a scientific information overload problem, which refers to the end user being overwhelmed by the abundance of information. To leverage the information available in that literature, there is a need for intelligent information retrieval systems to provide desired information in an organised manner.

One such type of information is machine-readable rich bibliographic metadata. As a consequence, there is demand for tools which can parse scientific documents and extract the bibliographic content. Researchers have devised interesting solutions. Regular expressions, template matching, knowledge bases, and supervised machine learning all relate to solutions proposed. Software tools have been proposed, such as Biblio (regular expression based), Bibpro (template matching based), Citation Parser (knowledge based or rule based), and GROBID (ML or machine learning based) [21]. The quality, measured using precision, of machine learning (ML) based tools, is similar to that of tools employing rules, regular expressions, or template matching (0.77 for ML-based tools vs. 0.76 for non-ML-based tools). However, ML based tools are popular and are often preferred because of also achieving higher recall (0.66 vs. 0.22) [21]. Only a few tools like GROBID (F1=0.89), Cermin (F1=0.83), and ParsCit (F1=0.75) have performed reasonably well. Re-training with task-specific data definitely increases the performance of almost all of the tools. Thus, F1 measure of GROBID increased by 3% (0.89 to 0.92), Cermin achieved F1 increases of 11% (0.83 to 0.92), and ParsCit had an increase of F1 by 16% (0.75 to 0.87) [21].

2.1.3 Big Data Text Summarization

For summarizing Electronic Theses and Dissertations (ETDs), three Fall 2018 student teams in Virginia Tech CS4984/5984 (Big Data Text Summarization) [8, 9, 10] used Science Parse and GROBID to extract information from PDFs. Both GROBID and Science Parse have their respective pros and cons. Table 2.1 summarises how GROBID outperforms Science Parse in many situations [2].

2.1.4 GROBID

GROBID (GeneRation Of Bibliographic Data) is a parser which is used to extract metadata from a PDF document into XML format. GROBID takes the PDF of each scientific document as input and makes use of machine learning models (cascading of linear-chain CRF) for extracting the metadata from the document in XML format. It uses the lexical (POS), layout (font, font size), and position information (start/end) of a line in a document in order to train the models and obtain the metadata in the required format. It does not

Table 2.1: Human assessment of GROBID and Science Parse outputs

	GROBID	Science Parse
Output File Format	XML	JSON
Table of Contents	Adds table of contents and list of figures at the end.	Maintains order of table of contents and list of figures.
Abstract	Occasionally misses the abstract.	Often detects the abstract correctly.
Chapters	Occasionally skips chapters especially in case of ETDs of disciplines such as Architecture where there are a large number of images present along with the text.	Often skips chapters and merges some chapters together.
Figures	Adds a <ref type="figure"> tag to indicate the existence of a figure.	Does not indicate the existence of a figure; often appends the figure title as part of the text.
Tables	Adds a <ref type="table"> tag to indicate the existence of a table.	Does not indicate the existence of a table.
References	Parses the reference string into author – first and last name, publication, volume, issue, published.	Parses the reference string into title, author, venue, year. Does not further split these values. Skips some references while extracting.

provide an explicit <chapter> tag. Therefore, chapter-level text and metadata extraction from the ETD documents is a challenging task using GROBID [5] [15].

2.1.5 Science Parse

Science Parse parses the scientific documents from PDF into structured JSON format. It is a combination of Java and Scala and can be used as a library in any JVM-based language. Science Parse can be used in three different ways:

- Server: It functions as a wrapper and makes Science Parse available as a web service. It uses heap memory (about 2GB).
- CLI: Science Parse has a command line interface known as RunSP. It uses heap memory (about 6GB). RunSP can also be used to parse multiple files at a time.
- Core: It provides flexibility in Science Parse but is also quite complex to use as a library. Four model files – general CRF model for extracting title and authors; and a CRF model for each of bibliographies, gazetteer, and word vectors – are available in this service.

Science Parse is difficult to set up and sometimes skips or merges some of the content [20][7].

2.1.6 Apache Tika

Apache Tika is a file extraction framework which is written in Java. The big advantage of Tika is that “it can extract over thousands of different types of files to metadata and text” [4]. In addition, another powerful capability that Tika has is that this library can extract the image metadata from Portable Document Format (PDF) files. However, it is hard to get the image itself compared to getting the metadata of this image. At the same time, since Apache Tika is written in Java, it is complicated to set it up if users are using other programming languages. Another disadvantage is that Tika can only extract PDF to text, which means chapter-wise extraction is difficult.

2.1.7 PDFMiner

PDF Miner.six (or PDFMiner) is a Python-compatible parser that can convert PDF files into text, HTML, or XML. The architecture of PDFMiner is shown as Figure 2.1. As a

rule-based parser, PDFMiner runs efficiently. Tested with an ETD document, PDFMiner converts PDF to text or other formats using around 18s. Moreover, it supports various font types and CJK language extraction [18]. Practically, it can extract specific pages and tables (output without structure) from a PDF file. However, because PDFMiner is used to extract text data, the ability to process images and tables in PDF files is still unstable according to its document.

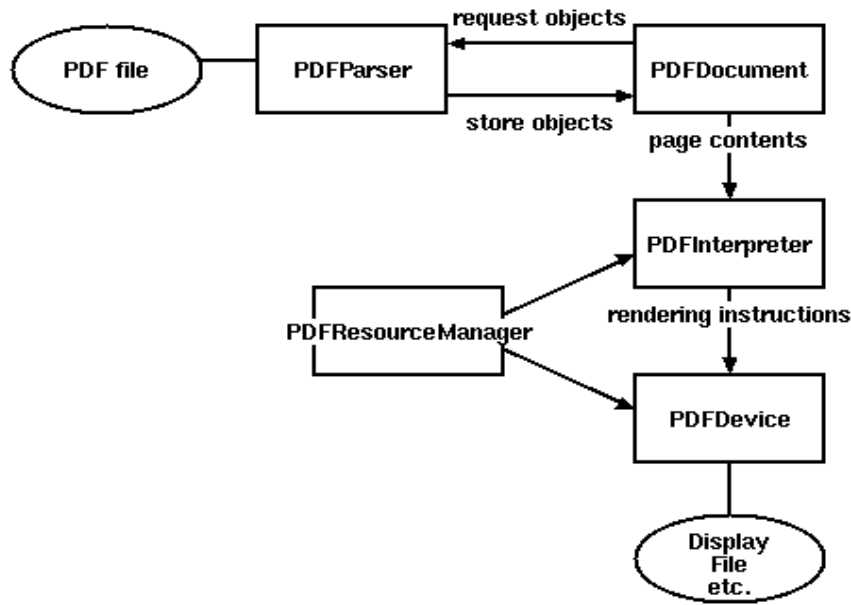


Figure 2.1: The architecture of PDF Miner

2.1.8 PyPDF2

PyPDF2 is a Python based tool for extraction of metadata and text from a PDF file. It also allows splitting, merging, and extraction of data from the file. Predominantly it is used for the extraction of text from a PDF file. It works on StringIO objects as opposed to file streams and so allows for PDF manipulation in memory [6].

Chapter 3

Requirements

In this project, the CME team is responsible for extracting metadata and text from the ETD documents. By the end of this project, we intend to finish the jobs listed below.

- Convert ETD documents from PDF to text format to enable full text search.
- Extract metadata for each ETD document.
- Extract chapter-level text from ETDs.
- Preprocess the ETD corpus, i.e., tokenize, lemmatize, and remove stopwords.
- Develop a pipeline to enable ingestion of new ETDs into Elasticsearch.

3.1 Extract Metadata and Text for ETD corpus

Metadata containing fields like names of author, date of publication, author email, contributor department, etc. has been extracted and put into ceph (mnt/ceph/cme). It contains the data of small subset (2017 ETDs) which include 691 PDF documents and larger dataset (all 30K ETDs). Each folder contains PDF as well as text files of the theses/dissertations.

3.2 Preprocess the ETD corpus

We have performed tokenization, and stopword removal on the ETD corpus. This should help the Text Analysis and Machine Learning team to cluster the documents efficiently.

3.3 User support

Currently, the IP address of the GROBID server is static. Other users are allowed to extract metadata from PDF files in any environment by using the URL we provided. An automated system is also provided through which user can run a driver script to implement all the tasks, from extraction of metadata from PDF to its ingestion into Elasticsearch. Details regarding the same are provided in Section 6.1.3.

Chapter 4

Approach, Design, Implementation

4.1 Experiment Design

This project addresses problems related to management of ETDs by answering the research questions that were listed in the problem definition of Section 1.3.

ETDs in our database are mostly in the form of PDF documents. The main objective is to parse and extract metadata from the ETDs. However, it is difficult to perform this action on the PDF files since they do not contain tags to delimit their elements. The structures of PDF files are often different, and vary according to the domain. To overcome these limitations, suitable machine learning tools need to be used which can extract metadata and represent all the ETDs in the same format.

After exploring and evaluating all the mentioned parsers, as discussed in Section 2.1, we decided to use GROBID for extracting metadata.

4.2 Implementation

4.2.1 Chapter Level Text Extraction

XPath-based Chapter Level Text Extraction

Projects like [8, 9, 10] have successfully used GROBID [5] for capturing the structure of ETD documents. Therefore, due to previous successful usage and ease of installation, we decided to use GROBID for chapter level text extraction. GROBID extracts the in-

formation from the PDF document of an ETD and converts it into a TEI (Text Encoding Initiative) [3] document. The structure of the TEI document is as shown in Listing 1.

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <!-- ... -->
  </teiHeader>
  <text>
    <div xmlns="http://www.tei-c.org/ns/1.0">

      <head><!--Chapter Name --> </head>
      <p> <!-- Chapter Content--></p>
    </div>
    <front>
      <!-- front matter of copy text, if any, goes here -->
    </front>
    <body>
      <!-- body of copy text goes here -->
    </body>
    <back>
      <!-- back matter of copy text, if any, goes here -->
    </back>
  </text>
</TEI>
```

Listing 1: Overall structure of a typical TEI document [3]

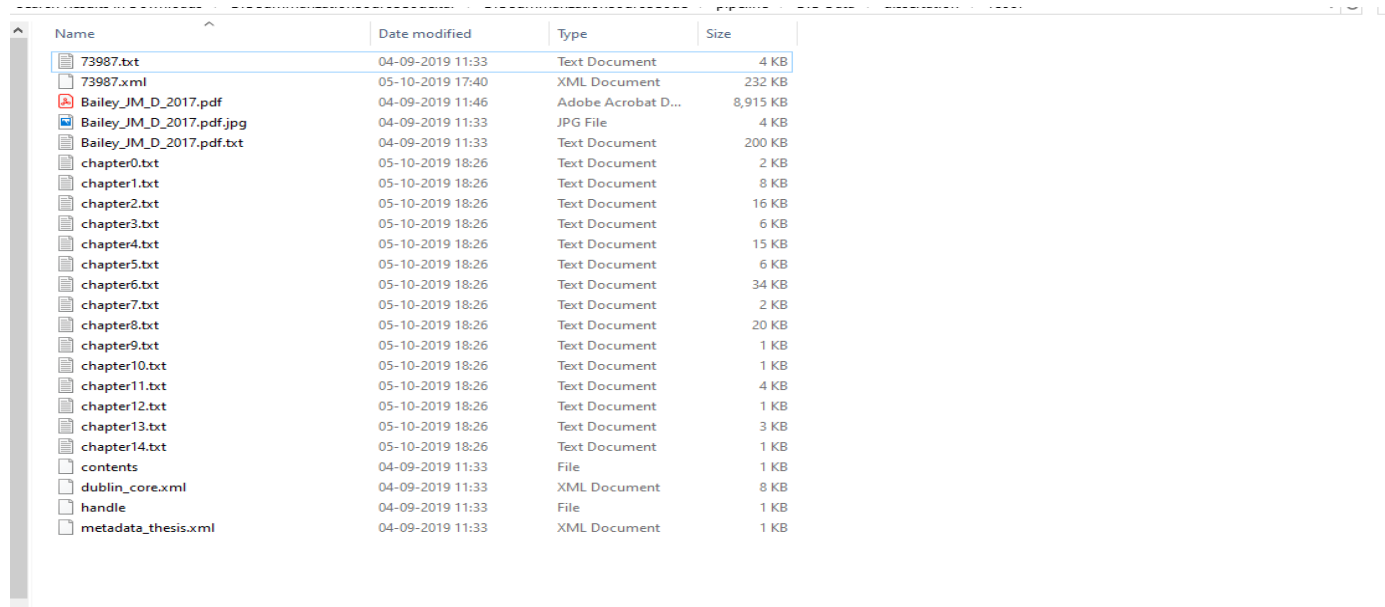
TEI Guidelines for Electronic Text Encoding and Interchange [3, 19] use XML as a markup language for representing the structure and semantic features of texts. The comprehensive tags offered by XML provide a way for incorporating the entire semantic structure of the ETD document. The TEI output format does not explicitly define a chapter tag (<chapter>), neither does it provide @type=chapter attribute for the <div> element. Therefore, due to the lack of explicit tags for the indication of the start or end of a chapter, chapter level extraction from ETD documents is a difficult task.

We use XPath expressions for extracting the chapters from the ETD documents. We can see in Listing 1 that “chapter name” is generally present in the <head> tag which is wrapped inside the <div> tag. Therefore, in order to locate the start of a chapter and the end of the preceding chapter, we need to capture such a pattern of tags from the TEI XML metadata extracted by GROBID. The detailed evaluation of this method is explained in the evaluation Section 5.

The steps involved in chapter level text extraction are:

- Convert the ETD document in PDF into TEI XML format by using a web service provided by GROBID: /api/processFulltextDocument.
- Use the XPath expression `/tei:TEI/tei:text/tei:body/tei:div[tei:head]` for the extraction of chapters, and store each chapter in the text format.[8]

The folder structure after chapter level text extraction is as shown in Figure 4.1.



Name	Date modified	Type	Size
73987.txt	04-09-2019 11:33	Text Document	4 KB
73987.xml	05-10-2019 17:40	XML Document	232 KB
Bailey_JM_D_2017.pdf	04-09-2019 11:46	Adobe Acrobat D...	8,915 KB
Bailey_JM_D_2017.pdf.jpg	04-09-2019 11:33	JPG File	4 KB
Bailey_JM_D_2017.pdf.txt	04-09-2019 11:33	Text Document	200 KB
chapter0.txt	05-10-2019 18:26	Text Document	2 KB
chapter1.txt	05-10-2019 18:26	Text Document	8 KB
chapter2.txt	05-10-2019 18:26	Text Document	16 KB
chapter3.txt	05-10-2019 18:26	Text Document	6 KB
chapter4.txt	05-10-2019 18:26	Text Document	15 KB
chapter5.txt	05-10-2019 18:26	Text Document	6 KB
chapter6.txt	05-10-2019 18:26	Text Document	34 KB
chapter7.txt	05-10-2019 18:26	Text Document	2 KB
chapter8.txt	05-10-2019 18:26	Text Document	20 KB
chapter9.txt	05-10-2019 18:26	Text Document	1 KB
chapter10.txt	05-10-2019 18:26	Text Document	1 KB
chapter11.txt	05-10-2019 18:26	Text Document	4 KB
chapter12.txt	05-10-2019 18:26	Text Document	1 KB
chapter13.txt	05-10-2019 18:26	Text Document	3 KB
chapter14.txt	05-10-2019 18:26	Text Document	1 KB
contents	04-09-2019 11:33	File	1 KB
dublin_core.xml	04-09-2019 11:33	XML Document	8 KB
handle	04-09-2019 11:33	File	1 KB
metadata_thesis.xml	04-09-2019 11:33	XML Document	1 KB

Figure 4.1: Folder structure of a ETD after chapter level text extraction

Chapter Level Text extraction based on Table of Contents

XPath based text extraction sometimes recognizes each subsection of the document as a chapter. In order to overcome this drawback, we tried to explore other methods of chapter

level text extraction. The Table of Contents provides information about all the sections and subsections that are present in an ETD document. Along with this information, it also provides the page numbers on which a user can find these sections and subsections. We decided to use the page numbers from the table of contents to track the start and end of each chapter. This method has a limitation, as most of the ETD documents do not contain the keyword 'Chapter' to distinguish between chapters and their subsections. PDF parsers do not maintain the inherent formatting of a PDF document (for example, they skip spacing between paragraphs), and convert it into a single text file. An example of text output from the parser and the content in the original PDF document is shown in Figures 4.2 and 4.3. As we can see from figure 4.3, there is no delimiter in the parsed text file which can indicate the end of a page. Additionally, the parser does not capture text from the header or the footer of a document, so the page numbers present in the header or footer could not be used as an indicator for the start or end of the page in the parsed text document.

Therefore, when the text is extracted from a PDF document, the mapping of page numbers to the chapters is lost.

Manual Chapter Level Extraction

Apart from exploring various other techniques like OCR on the basis of font size, we did a manual chapter level extraction from 21 ETD documents. This method gives us a gold standard result. The detailed evaluation of XPath based method 4.2.1 with the Manual Level Text Extraction on various parameters is discussed in the evaluation section 5. These documents are submitted to the Text Analysis and Machine Learning Team for solving the big data summarization problem.

4.2.2 TF-IDF Calculation

Term frequency-inverse document frequency (TF-IDF) is calculated to help Text Analysis and Machine Learning team to perform related analysis and calculation. As a weighting technique commonly used in text mining [17], TF-IDF characterizes the importance of a term in a document by calculating the term frequency and the number of documents in which the term appears. TF-IDF value can be calculated by using the following equation

Chapter 1 Introduction

In order to meet stringent requirements for next generation commercial aviation, aircraft will need to employ more aerodynamic designs to 1) reduce noise, 2) improve landing, takeoff, and cruise emissions, and 3) improve fuel consumption [1,2]. These designs are expected to take the aviation industry away from traditional ‘tube-and-wing’ architectures into a more hybrid or blended wing body (HWB, BWB) design that will satisfy aircraft operation improvement goals. A comparison between modern and proposed airframe architectures is shown in Figure 1.1.



Figure 1.1. Modern aircraft ‘tube-and-wing’ design (Left) and future blended wing design (Right).

A common feature of these HWB airframes is the integration of the engines with the aircraft fuselage. This relocation changes the inlet flow environment as the engines are no longer mounted below the wings where uniform inlet flow is ingested for the majority of the flight envelope. Mounting the engines in close proximity to the airframe, or embedding them within the airframe, results in non-uniform (distorted) inlet flow caused by the airframe body or inlet ducts. In the interest of performance and efficiency, modern commercial transport engines are designed to tolerate only small levels of distortion, such as during takeoff or strong crosswind conditions. Integrated propulsion systems will experience distorted inflow over the entire flight envelope and their interaction with such conditions will need to be studied to determine how engine performance is affected, and to support the design of distortion-tolerant engines.

To determine how well engines can withstand such an environment, three criteria will need to be evaluated in great detail; aeromechanics, stability, and performance. Although studies have shown that engine operation penalties are found in each one of these, the overall benefit of highly integrated airframe and propulsion systems can justify the operational challenges introduced to the engines [4].

Figure 4.2: Sample ETD Introduction chapter

Chapter 1 - Introduction 1 Chapter 1 Introduction In order to meet stringent requirements for next generation commercial aviation, aircraft will need to employ more aerodynamic designs to 1) reduce noise, 2) improve landing, takeoff, and cruise emissions, and 3) improve fuel consumption [1,2]. These designs are expected to take the a(cid:89)iation industr(cid:92) a(cid:90)a(cid:92) from traditional (cid:181)tube-and-(cid:90)ing(cid:182) architectures into a more hybrid or blended wing body (HWB, BWB) design that will satisfy aircraft operation improvement goals. A comparison between modern and proposed airframe architectures is shown in Figure 1.1. Figure 1.1. Modern aircraft (cid:181)tube-and-(cid:90)ing(cid:182) design (Left) and future blended wing design (Right). A common feature of these HWB airframes is the integration of the engines with the aircraft fuselage. This relocation changes the inlet flow environment as the engines are no longer mounted below the wings where uniform inlet flow is ingested for the majority of the flight envelope. Mounting the engines in close proximity to the airframe, or embedding them within the airframe, results in non-uniform (distorted) inlet flow caused by the airframe body or inlet ducts. In the interest of performance and efficiency, modern commercial transport engines are designed to tolerate only small levels of distortion, such as during takeoff or strong crosswind conditions. Integrated propulsion systems will experience distorted inflow over the entire flight envelope and their interaction with such conditions will need to be studied to determine how engine performance is affected, and to support the design of distortion-tolerant engines. To determine how well engines can withstand such an environment, three criteria will need to be evaluated in great detail: aeromechanics, stability, and performance. Although studies have shown that engine operation penalties are found in each one of these, the overall benefit of highly integrated airframe and propulsion systems can justify the operational challenges introduced to the engines [4]. Courtesy Boeing. Courtesy NASA Chapter 1 - Introduction 2 1.1 Literature Review Studies have been performed on the evaluation of engines subject to non-uniform inlet flows produced by highly integrated engine-airframe systems. The three metrics that must be most considered are; aeromechanics, stability, and performance. Researchers can find background into aeromechanical fatigue caused by distorted inflow through earlier publications [5, 6]. As this work focuses on the impact of stability and performance, the literature review will investigate previous work on these subjects in detail. An important component of the integrated engine-airframe problem is the issue of fan-distortion interaction, in which a coupling effect between the distortion or distortion producing device and the fan exists. As will be shown later, these components act together as a single system that must be considered when designing distortion-producing devices within the context of distortion tolerant fan research. 1.1.1 Stability Stability is a qualitative measure of how well an engine can handle off-design

Figure 4.3: Parsed text of the same document (highlighted text indicates end of page shown in Figure 4.2)

4.1.

$$\begin{aligned} TfIdf_{i,j} &= Tf_{i,j} \times Idf_i \\ &= \frac{n_{i,j}}{\sum_k n_{k,j}} \times \log \frac{|D|}{|\{j : t_i \in d_j\}|} \end{aligned} \quad (4.1)$$

$n_{i,j}$ is the number of occurrences of $term_i$ in the document d_j and D is the total number of documents in the corpus.

Initially, we convert all ETD PDF documents to text format. Then the Python script reads these documents to calculate TF-IDF according to Equation 4.1. The TF-IDF representation is implemented using gensim [16], a Python library, which indexes the documents and saves the indexes and TF-IDF vectors as key-value pairs. So users need to provide the index of one document to obtain the corresponding TF-IDF vector. To avoid this complicated process, we provide an optional toolkit in which the user needs to enter the path to the saved TF-IDF file and the name of the document in order to obtain its corresponding TF-IDF vector.

As shown in Figure 4.4, the TF-IDF output of each document saved in gensim format is a list of tuples. The first element of each tuple is the index of one term and the second element is its corresponding TF-IDF value. The gensim TF-IDF method takes the bag-of-words (BOW) of each document as input. As shown in Figure 4.5, the format for BOW is similar to that of TF-IDF module. However, the second element of each tuple is the frequency of the term in the document. In addition, the BOW of whole ETD documents is indexed. A dictionary, which gives the corresponding index of the documents, is also provided. Part of this dictionary is shown in Figure 4.6.

[(0, 0.08242093740152368), (1, 0.08242093740152368), (2, 0.16484187480304735), (3, 0.16484187480304735), (4, 0.152368), (9, 0.3296837496060947), (10, 0.08242093740152368), (11, 0.08242093740152368), (12, 0.08242093740152368), (17, 0.08242093740152368), (18, 0.16484187480304735), (19, 0.08242093740152368), (20, 0.16484187480304735), (21, 0.08242093740152368), (28, 0.034207779747346505), (29, 0.06841555949469301), (30, 0.08242093740152368), (31, 0.08242093740152368), (38, 0.08242093740152368), (39, 0.08242093740152368), (40, 0.08242093740152368), (41, 0.16484187480304735), (49, 0.16484187480304735), (50, 0.16484187480304735), (52, 0.08242093740152368), (53, 0.16484187480304735), (59, 0.16484187480304735), (60, 0.08242093740152368), (62, 0.08242093740152368), (63, 0.08242093740152368),

Figure 4.4: Part of TF-IDF of one document

```

(6, 8), (7, 64), (22, 1), (30, 1), (31, 2), (33, 2), (34, 1), (35,
(90, 1), (113, 3), (140, 5), (154, 9), (155, 1), (157, 2), (168, 3),
1, 23), (229, 7), (230, 3), (234, 5), (236, 3), (237, 3), (246, 10),
2), (799, 3), (809, 3), (820, 2), (822, 1), (836, 7), (851, 7), (86
), (945, 5), (947, 1), (948, 4), (950, 11), (953, 7), (955, 22), (95
2), (1062, 2), (1063, 1), (1069, 8), (1076, 5), (1089, 3), (1102, 1
), (1263, 1), (1276, 19), (1277, 1), (1282, 1), (1302, 4), (1315, 2),
1482, 2), (1494, 1), (1508, 3), (1519, 6), (1530, 2), (1543, 2), (15
, 2), (1723, 4), (1734, 3), (1747, 3), (1758, 2), (1759, 2), (1770,
), (1905, 2), (1916, 1), (1928, 1), (1940, 5), (1943, 1), (1954, 1),
, (2074, 2), (2076, 2), (2080, 1), (2085, 11), (2086, 2), (2094, 1),
(2186, 2), (2203, 8), (2204, 5), (2215, 4), (2216, 1), (2225, 6), (
2324, 1), (2325, 3), (2338, 1), (2339, 1), (2348, 2), (2365, 12), (2
2456, 1), (2468, 40), (2472, 1), (2476, 9), (2479, 1), (2484, 6), (2
557, 3), (2565, 4), (2570, 4), (2574, 31), (2594, 14), (2595, 1), (2
13, 25), (2714, 2), (2715, 23), (2717, 4), (2719, 1), (2720, 1), (27
304, 1), (2817, 2), (2820, 177), (2821, 10), (2822, 4), (2823, 2), (

```

Figure 4.5: Part of BOW of one document

```

"LD5655.V856.1995.X54.pdf.txt": 46095, "Wang_T_D_2013.pdf.txt": 46096, "LD5655.V856.199
F.txt": 46101, "25-F15c.pdf.txt": 46102, "08-Ref.pdf.txt": 46103, "12-F2.pdf.txt": 46104
b.pdf.txt": 46110, "02-Ch2a.pdf.txt": 46111, "13-F3c.pdf.txt": 46112, "21-F11c.pdf.txt"
0c.pdf.txt": 46119, "32-Vita.pdf.txt": 46120, "19-F9.pdf.txt": 46121, "24-F14.pdf.txt"
F12c.pdf.txt": 46128, "28-F18.pdf.txt": 46129, "30-App-b.pdf.txt": 46130, "31-App-c.pdf
6.pdf.txt": 46135, "Jia_X_D_2013.pdf.txt": 46136, "LD5655.V856.1995.S344.pdf.txt": 4613
lf.txt": 46141, "LD5655.V856.1977.G737.pdf.txt": 46142, "LD5655.V856.1991.D385.pdf.txt"
1981.B933.pdf.txt": 46150, "Dissertation.whf.pdf.txt": 46151, "Song_M_D_2013.pdf.txt":
46156, "LD5655.V856.1988.C644.pdf.txt": 46157, "Sudano_L_D_2015.pdf.txt": 46158, "Sudan
D384.pdf.txt": 46163, "early31407.pdf.txt": 46164, "Sleep_MD_D_2011_Copyright.pdf.txt"
sertation_Shen.pdf.txt": 46172, "LD5655.V856.1992.G68.pdf.txt": 46173, "LD5655.V856.199
6.1978.W45.pdf.txt": 46178, "Martin_defense_R1_0.pdf.txt": 46179, "Martin_dissertation
": 46187, "RESUME98.PDF.txt": 46188, "DISCH4-F.PDF.txt": 46189, "DISCH1-F.PDF.txt": 461

```

Figure 4.6: Part of doc-index dictionary

4.2.3 Transforming metadata for ingestion in Elasticsearch

Elasticsearch ingests data in bulk as well as one by one. The bulk API is far more complex in terms of the required data format and hence, we decided to ingest each document one by one. Elasticsearch ingests data only if it is in a particular format. Elasticsearch can consume a JSON array only if all the entries of the array are of the same data type, i.e., either string or object. By default, GROBID output contains arrays having entries of mixed data types. For example, in Listing 4.1, description provenance has one entry of string type and two entries of object type. We have written a Python script that iterates through the metadata file and converts each entry to the same data type. If there is a mismatch, all entries are converted to object data type having the key as the immediate

parent-key.

Listing 4.1: Raw Metadata extracted from ETD using GROBID

```
1 "description-provenance": [  
2     "Made available in DSpace on 2017-01-06T13:34:0  
3     6Z (GMT). No. of bitstreams1 Bailey_JM_D_201  
4     7.pdf9128042 bytes, checksum7438e886322739e1  
5     7247ed2c907658b0 (MD5) Previous issue date  
6     2017-01-05",  
7     {  
8         "Author Email": [  
9             "jmb@vt.edu"  
10        ]  
11    },  
12    {  
13        "Advisor Email": []  
14    }  
15 ]
```

4.2.4 Development of an Automated System

Automated System is a system that performs all the tasks, from the extraction of metadata from an ETD document to its ingestion into Elasticsearch automatically for any new document that has been fed to the system developed by the CS5604 fall 2019 class.

The features of this system are:

- Automated unit testing to ensure that all the development scripts are error-free.
- Tests to check whether all the dependent services are running. (e.g. Figure 4.9 shows the output of unit test that checks whether GROBID is running)
- Validation of generated metadata to ensure that it is in the format that can be ingested into Elasticsearch.

- Automatic extraction and preprocessing of the text from the document
- Automatic merging of metadata of new documents to the existing metadata.

The limitations of this system are:

- The system cannot scrape the new data from VTechWorks (The new data should be added in a folder called “temp” on ceph)
- The folder structure of an ETD document should be in the format shown in Figure 4.9.

Such an automation ensures the proper functionality of the system developed by the class and also the correctness of data that has been passed to Elasticsearch (ELS), Front End and Kibana (FEK) and Text Analysis and Machine Learning (TML) teams for further processing and analysis. The detailed description of unit tests is mentioned in the section 5. The figure 4.7 shows the workflow of the automated system.

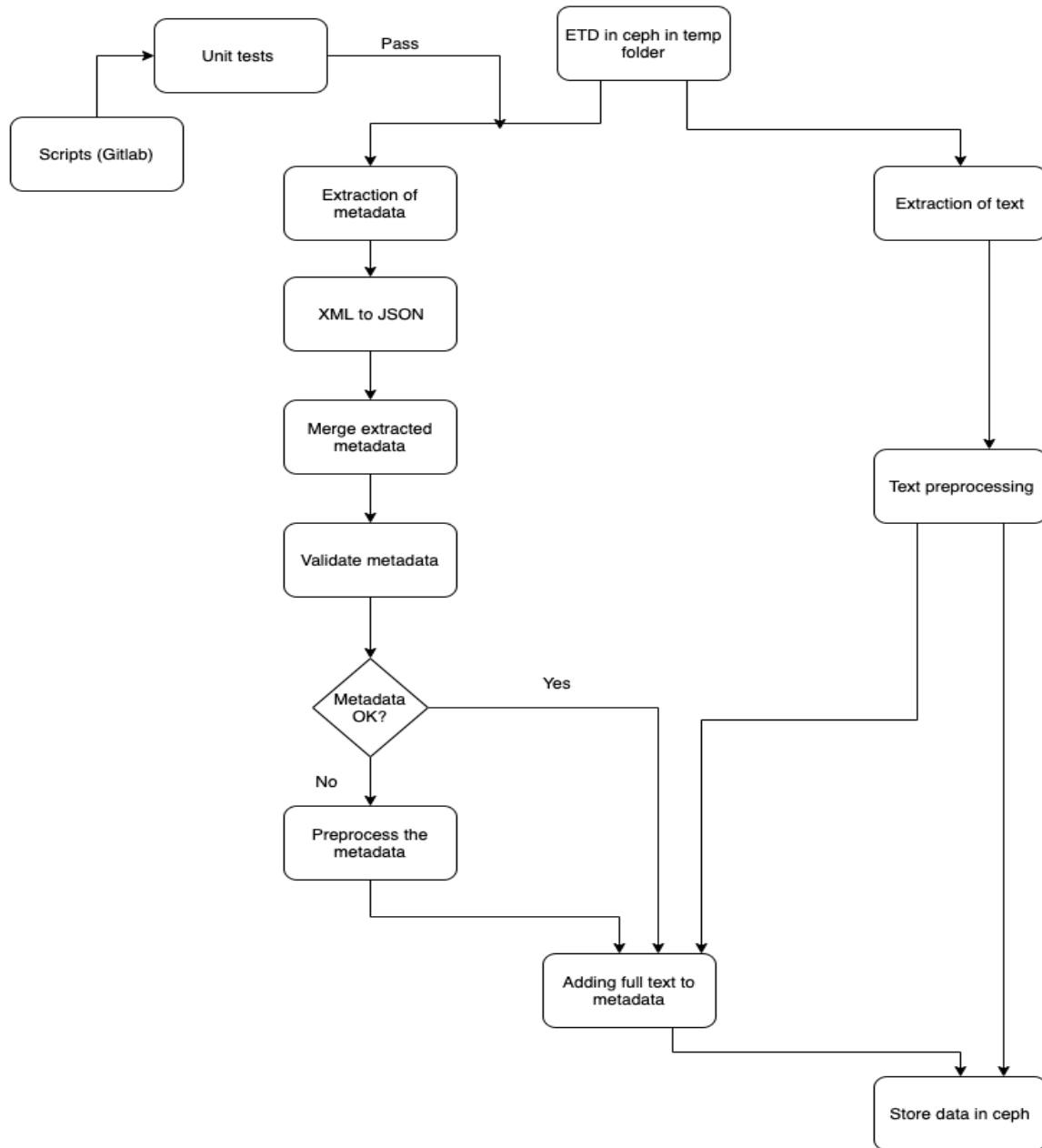


Figure 4.7: Flow Diagram of the Automated System

```
[root@centos-5596447975-72x4d thesis_subset]# cd 73988/
[root@centos-5596447975-72x4d 73988]# ls
73988.txt Muthirevula_N_T_2017.pdf Muthirevula_N_T_2017.pdf.jpg Muthirevula_N_T_2017.pdf.txt contents dublin_core.xml handle metadata_thesis.xml
```

Figure 4.8: Folder Structure of an ETD

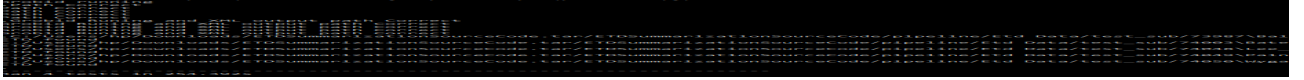


Figure 4.9: GROBID Unit test

4.2.5 List of Visualizations to be provided in the Front End

Visualization Type:

- Type-none: "Dissertation" (Pie Chart)
- Degree-level: "doctoral" (Bar Chart)
- Contributor-department: "Mechanical Engineering" (Pie Chart)
- Year: "2017" (get it from "date-issue") (Bar Chart)

4.2.6 Text Preprocessing

ASCII does not correctly encode all the characters in the PDF files; the text files converted from these PDF files contain many meaningless and wrong characters. These characters may have a negative impact on the query process. To address this problem, the stop words are removed using the "corpus" package in NLTK [14].

The other issue is about numbers and garbage data characters that appear in the text files. In general, the numbers shown in ETD files are related to reference numbers and other numeric values. The reference number are not useful for query search, therefore, we use regular expressions to remove these numbers. The following regular expressions were used to clean the data:

- "[\d{1,20}]" to remove words with length greater than 20

- `replace("...", "")` to remove "..."
- `re.sub("[\\(\\[\\.*?\\[\\]\\]", "")` to remove braces
- `replace('b \' ', '')` to remove byte literal
- `encode('ascii', 'ignore')` to remove non ascii characters

Note that this is an optional process; we provide two different versions, one that contains raw data and another one that contains the processed data which are required by the Elasticsearch and Test Analysis and Machine Learning teams, respectively.

Chapter 5

Evaluation

5.1 Manual Testing

5.1.1 Testing of Chapter Level Text Extraction

In Section 4.2.1, we explained how we use XPath to extract text based on the chapter level. We noticed some problems after comparing the results to the chapter-wise results extracted from ETDs manually. We use Justin Mark Bailey’s dissertation “Full Scale Experimental Transonic Fan Interaction with a Boundary Layer Ingesting Total Pressure Distortion” as an example to show the differences; see Table 5.1. For XPath based extraction, we counted the first file for each chapter, as some chapters were divided into numbers of files. This is why the completeness of XPath based chapter level extraction technique is low.

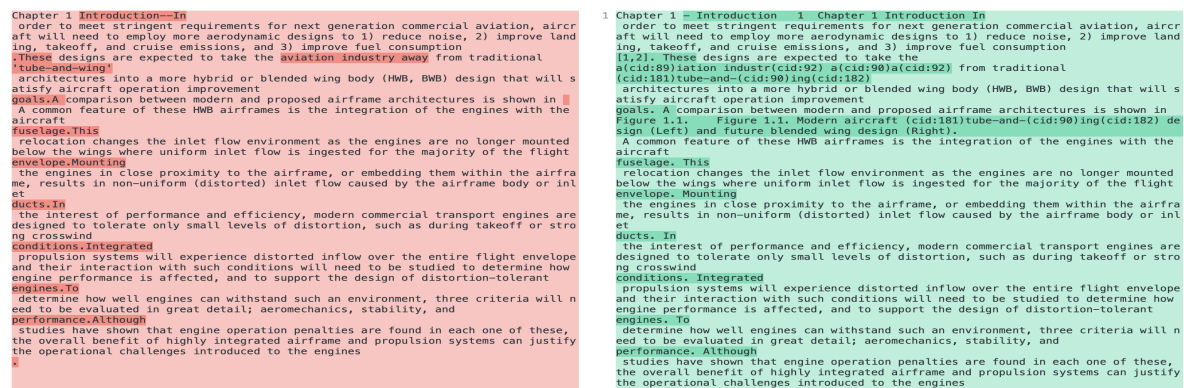


Figure 5.1: Chapter level text extraction by XPath vs. manual extraction by Diff Checker

Table 5.1: Chapter level text extraction by XPath and manual extraction

	XPath	Manual
Appendix	Just one section	Yes
Captions	No	Yes
Chapter completeness on average (calculated by counting the number of words)	43.90%	90.88%
Formulas	No	Yes but lots of illegal characters
Headers	No	Repeated each page
Illegal characters	No	Some letters are converted to {cid:}
References in-text	No	Yes
References	No	Yes
Space between sentence	No	Yes
Texts in figures	No	Yes but many illegal characters

From Table 5.1 we can see the performance of chapter level text extraction by XPath is not as good as manual chapter level extraction.

XPath based technique ignored captions, texts in figures, and formulas which might include useful information. The percentage of the chapter completeness on average is a good indicator to show the performance of extractions. Manual extraction has 90.88% completeness instead of 100% since there are many special characters, figure captions, and formulas that could not be parsed correctly by the PDF to text parser [6]. However, it still performs much better than the chapter level text extraction by XPath which has 43.90% for completeness on average. The differences in number of chapters generated for 21 ETD documents by two types of extraction methods mentioned in sections 4.2.1 4.2.1 are shown in Table 5.3. We can see XPath does not perform well as only one of the 21 documents has the correct number of chapters.

5.1.2 Testing of Extracted Text Preprocessing

The ETD text files extracted by PDFMiner.six [18] include many incorrect characters. As shown in Figure 5.2, these illegal characters are usually from non-English words. To remove these garbage characters, we use NLTK to detect and remove them.

```
incrementally increase gripping forces every time slip sensed [40]. Kavi e
n adjust force limits based deformability 15 degree calculated object. Gra
ian relation, transformed tip forces. Solution detecting deformation based
. For sake simplicity, cylindrical grasps considered common grasp type dai
nation motion sensor optimization method provided following chapters. 19 C
eriments previous glove mechanism part previous work, results show previou
hs, 1, 2 3 joint angles 1, 2, 3, 4 angles constraint joints respect global
22112121221222211112111111112sincos2coscos 2cossin2sinsin(2sincos 2cossin)
```

Figure 5.2: Original text generated by PDFMiner.six

Table 5.2: Differences between chapter level text extraction by XPath and manually extraction

Document	XPath	Manual	Match
73987	15	5	No
73988	9	7	Close
74003	52	5	No
74047	3	1	
74048	36	5	No
74049	46	5	No
74050	75	5	No
74233	5	5	Yes
74234	40	7	No
74235	12	5	No
74236	31	6	No
74237	23	5	No
74238	2	5	No
74239	154	7	No
74275	13	ETD in slides format	
74302	50	7	No
74383	85	5	No
74395	21	5	No
74396	3	1	
74398	0	1	
74423	31	6	No

```

sensor ( -axis ) adjust forces based orientatio
ides reaction force based upon force applied fo
igned manufactured assembled exoskeleton glove
t faster linear Elastic Actuators ( ) This prot
verview chapters presented thesis presented sec
main contributions thesis Chapter : Provides c
developed followed literature review research g
tatement motivating application discussed chapt
rasp stability Chapter : Describes mechanical d
inkage mechanism This section also discusses se

```

Figure 5.3: Processed text

In general, the reference numbers of equations and citations are not useful during processing of search queries. We use regular expressions to remove these characters. The processed text is shown in Figure 5.3. Long string of characters in the last line of Figure 5.2 have been removed in Figure 5.3, and the numbers in parentheses have also been removed.

5.1.3 Metadata Extraction Testing

We prepare a JSON file manually for any given ETD using the list of keys and then run the tool to extract metadata from the same ETD. We inspect and compare both JSON files; if all the key-value pairs match, it means that our script to extract metadata using GROBID is working properly.

5.1.4 Automated Testing

Unit Test

Unit Testing is the first level of software testing where the smallest testable parts of a software are tested. This is used to validate that each unit of the software performs as designed.

A test case is a set of conditions which is used to determine whether a system under test works correctly.

Test suite is a collection of test cases that are used to test a software program to show that it has some specified set of behaviours by executing the aggregated tests together.

Stub

A stub is an object that holds predefined data and uses it to answer calls during tests. It is used when you can't or don't want to involve objects that would answer with real data or have undesirable side effects.

An example can be an object that needs to grab some data from the database to respond to a method call. Instead of the real object, we introduced a stub and defined what data should be returned[13].

Unit test cases and their details

Table 5.3: Different test case scenarios.

Unit Test Name	Description	Expected Behaviour
testGrobid	It hits the GROBID service status API	If service is up, test case passes else fails.
testInputPDFPath	Checks whether files are present or not at the expected file path.	If files are present, test case passes else fails.
testGrobidAndInputPath	Tests both the scenarios where GROBID is up and PDF files are present or not at expected location.	If files are present and GROBID is running, test case passes.
testMetaDataFormat	Test whether the extracted metadata is in elastic search acceptable format or not.	If metadata is present in suitable format, test case passes else it fails.

Chapter 6

User Manual

6.1 Where to Get Data

6.1.1 VTechWorks ETD collection

The Electronic Theses and Dissertations used for the project are available in VTechWorks, the Virginia Tech institutional repository maintained by University Libraries. These ETDs are open access and can be viewed and downloaded free of charge.

The following are the links through which the documents can be accessed:

- ETDs: Virginia Tech Electronic Theses and Dissertations:
<http://hdl.handle.net/10919/5534>
- Masters Theses:
<http://hdl.handle.net/10919/9291>
- Doctoral Dissertations:
<http://hdl.handle.net/10919/11041>

For the initial phase, a subset of these documents, documents from the year 2017, was considered. Metadata extraction, chapter-wise segregation, and full-text extraction were performed on this subset using GROBID. Metadata which includes fields such as author name, title, date of publication, and department have been extracted and stored in MongoDB.

6.1.2 GitLab repository

All files required to run the system are present in the Gitlab repository.

<https://code.vt.edu/cs5604/cme>

Name	Last commit
PyPDF2	Cutom PyPDF2
AddTextToMetadata.py	Initial commit
DataPreProcessing.py	Initial commit
DriverScript.py	Initial commit
MergeMetaData.py	Initial commit
MetadataExtractor.py	Initial commit
TextExtractor.py	Initial commit
XML2JSONCoverter.py	Initial commit
__init__.PY	Initial commit
config.py	Initial commit
util.py	Initial commit

Figure 6.1: GitLab file structure

6.1.3 Metadata extraction and ingestion in ceph

The general steps to extract metadata from the ETDs and ingest it onto ceph are given below.

1. GROBID is used to process the ETD PDF and extract the metadata in XML format. The container for running GROBID is available at the following IP:
`http://2001.0468.0c80.6102.0001.7015.d574.516b.ip6.name:8070/`
Full text as well as header processing of ETDs can be performed using the TEI option.



Figure 6.2: GROBID Container

The GROBID server can also be accessed using a Python client. Figure 6.3 shows a sample code snippet used to access GROBID through a Python client.

```
18 def get_xml(pdf,dir):
19
20     GROBID_URL = "http://2001.0468.0c80.6102.0001.7015.d574.516b.ip6.name:8070/"
21     url = '%s/api/processFulltextDocument' % GROBID_URL
22
23     xml = requests.post(url, files = {'input' : open(pdf, 'rb')}).text
24     f = open(base_path+dir+"/"+dir+".xml", "w",encoding='utf-8')
25     f.write(xml)
26     f.close()
```

Figure 6.3: Python client to access GROBID

2. Elasticsearch requires the data to be in JSON format, but the default output generated using GROBID is in XML format. Moreover, the JSON file needs to have a key value for each object and in NDJSON (newline delimited JSON) format, as mentioned in Section 4.2.3. A Python script (XML2JSONConverter.py) will convert the XML file generated using GROBID to JSON format compatible for Elasticsearch.

The Sample Metadata Format is as shown in Listing 6.1:

Listing 6.1: Raw Metadata extracted from ETD using GROBID

```
1 {
2     "format-medium": "ETD",
3     "description-abstract": "Future commercial
4         transport aircraft will feature more
5         aerodynamic architectures to accommodate
6         stringent design goals for higher fuel
7         efficiency, reduced cruise and taxi NOx
8         emissions, and reduced noise.",
9     "date-issued": "2017-01-05",
10    "publisher-none": "Virginia Tech",
11    "title-none": "Full Scale Experimental
12        Transonic Fan Interaction with a Boundary
13        Layer Ingesting Total Pressure Distortion",
14    "contributor-author": "Bailey, Justin Mark",
15    "contributor-committeemember": [
16        "Dancey, Clinton L",
17        "Lowe, Kevin T",
18        "Wicks, Alfred L",
19        "Ng, Wing Fai"
20    ],
21    "type-none": "Dissertation",
22    "description-degree": "PHD",
23    "degree-discipline": "Mechanical Engineering",
24    "subject-none": [
25        "Experimental Engine Testing",
26        "Distortion",
```

```

20         "Interaction",
21         "Total Pressure",
22         "Boundary Layer Ingesting"
23     ],
24     "contributor-department": "Mechanical
        Engineering",
25     "degree-level": "doctoral",
26     "identifier-uri": "http://hdl.handle.net/10919/
        73987",
27     "date-available": "2017-01-06T13:34:06Z",
28     "handle": "73987",
29     "description-provenance": [
30         {
31             "description-provenance-summary": "Made
                available in DSpace on 2017-01-06T1
                3:34:06Z (GMT). No. of bitstreams1
                Bailey_JM_D_2017.pdf9128042 bytes,
                checksum7438e886322739e17247ed2c9076
                58b0 (MD5) Previous issue date2017
                -01-05"
32         },
33         {
34             "Author Email": [
35                 "jmb@vt.edu"
36             ]
37         },
38         {
39             "Advisor Email": []
40         }
41     ],
42     "identifier-other": "vt_gsexam:9274",
43     "rights-none": "This item is protected by
        copyright and/or related rights. Some uses
        of this item may be deemed fair and
        permitted by law even without permission

```

```

    from the rights holder(s), or the rights
    holder(s) may have licensed the work for use
    under certain conditions. For other uses
    you need to obtain permission from the
    rights holder(s).",
44     "degree-grantor": "Virginia Polytechnic
        Institute and State University",
45     "date-accessioned": "2017-01-06T13:34:06Z",
46     "contributor-committeechair": "O'Brien, Walter
        F",
47     "degree-name": "PHD"
48 }

```

A similar output is generated for all the ETDs and a JSON file containing the meta-data for all the ETDs is created.

3. Another script, AddTextToMetadata.py will convert the ETD to text and add it as a field to the extracted JSON metadata. This will allow for full text search on all ETD documents.
4. A Python script to ingest the data into ceph has been written by the ELS team. The data is available at (mnt/ceph/cme/metadata_subset.json).
5. DriverScript to run all the above scripts to enable all tasks from metadata extraction to its ingestion in Elasticsearch is also present.

Chapter 7

Developer's Manual

In this chapter, we provide details about our timeline of this project, applications we have used to communicate in the team, and what we have done. Therefore, we will focus more on how the project can be used to get the metadata and text extracted.

7.1 Timeline

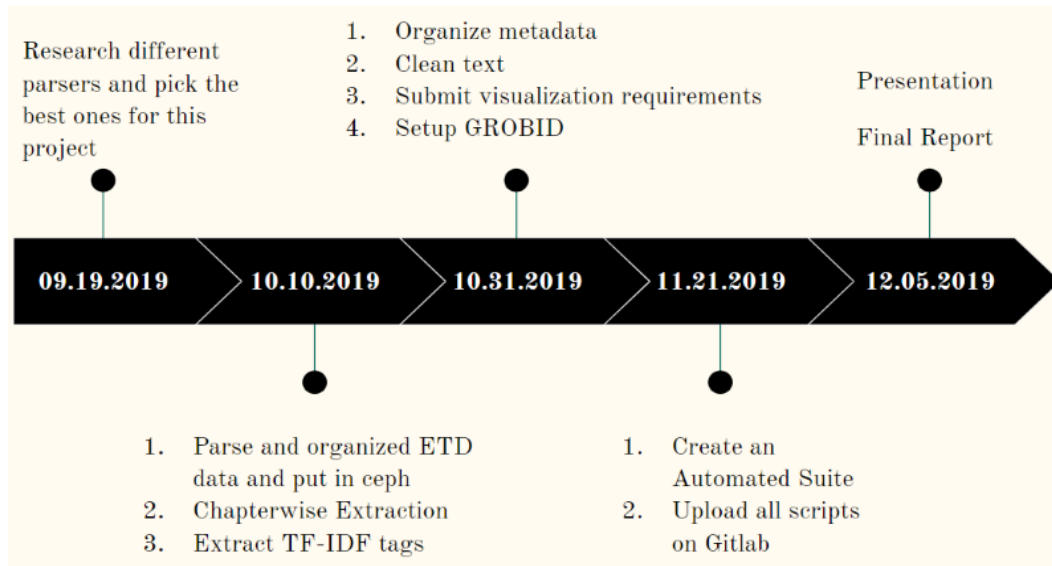


Figure 7.1: Timeline

7.2 Slack

Our group used slack to communicate with all members in the "cme" channel in Slack. At the same time, we use the channel called "general" to communicate with other different groups in this project.

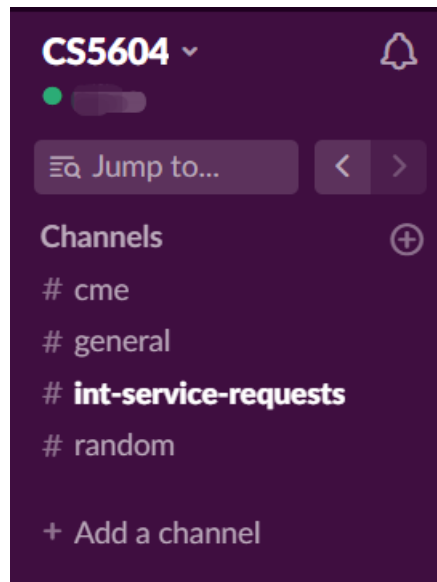


Figure 7.2: Slack

7.3 GROBID

To install GROBID in a local computer, use the following instructions.

7.3.1 Install in Ubuntu

Step 1: Update System

```
apt-get update
```

Step 2: Install JDK

Before installing GROBID on a local computer or empty container, Java JDK Version 1.8 has to be set up already.

```
apt-get -y install openjdk-8-jdk wget unzip
```

Step 3: Download and install GROBID in /opt

```
wget https://github.com/kermitt2/grobid/archive/0.5.5.zip
unzip 0.5.5.zip
```

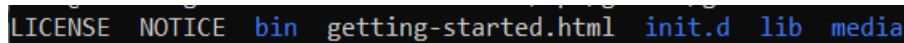
Step 4: Download Gradle Gradle is a dependency required for running GROBID.

```
wget https://services.gradle.org/distributions/gradle-3.4.1-bin.zip
```

Step 5: Install Gradle

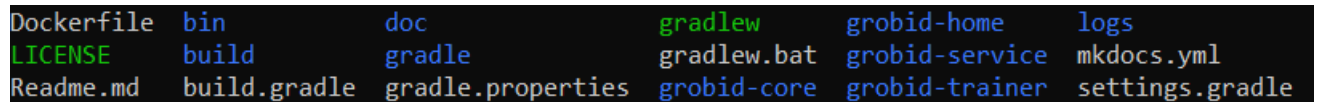
```
mkdir /opt/gradle
unzip -d /opt/gradle gradle-3.4.1-bin.zip
export PATH=$PATH:/opt/gradle/gradle-3.4.1/bin
```

After installing everything, Figures 7.3 and 7.4 show what is available in the directories.



```
LICENSE NOTICE bin getting-started.html init.d lib media
```

Figure 7.3: Files in the Gradle folder



```
Dockerfile bin doc gradlew grobid-home logs
LICENSE build gradle gradlew.bat grobid-service mkdocs.yml
Readme.md build.gradle gradle.properties grobid-core grobid-trainer settings.gradle
```

Figure 7.4: Files in the GROBID folder

Step 6: Run GROBID

First, get into directory `/opt/grobid-0.5.5`, and then run the command below:

```
./gradlew run
```

Step 7: Run GrobidcURL.py

Once GROBID is running, call the command below to run the Python file to get the metadata.

```
python Grobid_cURLpy
```

7.4 PDFMiner

Step 1: Install and Test PDFMiner.six

PDFMiner.six is a fork of PDFMiner for Python3.x.

```
pip install pdfminer.six
pdf2txt.py samples/simple1.pdf
```

Step 2: Run PDFMiner.six

Run PDFMiner.six to extract text:

```
pdf2txt.py -t type -o outputfile pdffile
```

Run PDFMiner.six to extract tables:

```
dumppdf.py -T -o outputfile pdffile
```

Usage: `[-t]` defines the output type, such as txt, html and xml. `[-o]` defines the output path.

7.5 TF-IDF

Step 1: Install Gensim

```
pip install gensim
```

Step 2: Run tf-idf-tool.py

The tf-idf-tool.py is a Python script to read text ETDs and calculate tf-idf values. The saved tf-idf model is in /mnt/ceph/cme/tf-idf.

```
python tf-idf_tool.py
```

Step 3 (optional): Run use-tfidf.py

Run use-tfidf.py to check the result.

```
python use-tfidf.py
```

```
what is document name: Childress_TL_T_2013.pdf.txt
```

```
where is the saved tf-idf model: /mnt/ceph/cme/tf-idf/model.tfidf
```

```
where is the doc to index dictionary: /mnt/ceph/cme/tf-idf/d2i
```

```
where is the BOW corpus model: /mnt/ceph/cme/tf-idf/corpus
```


Chapter 8

Challenges and Limitations

One of our challenges is to extract images, tables, and formulae. from PDF. This includes extraction of both metadata and text. However, we haven't found a reliable library to help us reach this point.

Another issue that limits the ETD output data quality is addressed here. For now, the quality of extracted ETD data relies on the performance of GROBID. However, GROBID does not always process PDF files well and the outputs, such as metadata and content, may be slightly different from the original PDF files.

Chapter 9

Future Scope

9.1 Improving Chapter Level Text Extraction

Chapter level text extraction can be improved by using various techniques based on OCR. Such an extraction can be used for solving the Big Data Summarization problem for obtaining the summary of each chapter.

9.2 Batch Processing of the Documents

In future, one can perform batch processing of the ETD Data. Batch processing will considerably reduce the time required for converting the ETD documents which are in PDF to a TEI XML format.

9.3 Improving Automation Suite

Loggers can be implemented to log the different steps of the automation suite so that it is easier to understand what is going on in the background. Code coverage can be improved significantly. More trigger points can be added to initiate the automation suite to give additional options to the user. This allows users to choose whether they want to execute batch processing or use single-threaded processing.

Chapter 10

Acknowledgements

The project will be implemented during the course of CS5604: Information Storage and Retrieval at Virginia Tech. The data used was the ETDs available on VTechWorks.

We would like to thank Dr. Edward Fox for giving us the opportunity to work on this interesting and challenging project. We are grateful for his advice and guidance. We would also like to thank the GTA, Ziqian Song, for her guidance and support throughout the course project. We thank Bipasha Banerjee for her expertise about the ETD data and also for guiding us in the proper direction. We thank other teams for their help in integration and for sharing their knowledge and insights with us. We also acknowledge the creators of all the open source tools and software packages and libraries we used to implement this project. We also thank IMLS for its support of ETD-related research through grant LG-37-19-0078-19.

Bibliography

- [1] Elasticsearch. <https://xebialabs.com/technology/elasticsearch/> accessed on October 20th, 2019.
- [2] ETDs: Virginia Tech Electronic Theses and Dissertations. <https://vtechworks.lib.vt.edu/handle/10919/5534>.
- [3] The TEI Guidelines. <https://www.tei-c.org/release/doc/tei-p5-doc/en/html/index.html>.
- [4] Apache Tika, 2007 — 2019. <https://tika.apache.org/> accessed on October 12th, 2019.
- [5] Grobid, 2008 — 2019. <https://github.com/kermitt2/grobid> accessed on October 30th, 2019.
- [6] PyPDF2, May 2014 — 2016. <https://pythonhosted.org/PyPDF2/> accessed on October 15th, 2019.
- [7] Science parse, 2015 — 2019. <https://github.com/allenai/science-parse> accessed on October 30th, 2019.
- [8] Big data text summarization: Using deep learning to summarize theses and dissertations, 2018. <http://hdl.handle.net/10919/86406> accessed on October 25th, 2019.
- [9] CS4984/CS5984: Big data text summarization team 10 etds, 2018. <http://hdl.handle.net/10919/86418> accessed on October 25th, 2019.
- [10] CS4984/CS5984: Big data text summarization team 17 etds, 2018. <http://hdl.handle.net/10919/86420> accessed on October 25th, 2019.

- [11] GLATTHORN, A. A., AND JOYNER, R. L. *Writing the winning thesis or dissertation: A step-by-step guide*. Corwin Press, 2005.
- [12] HAYCOCK, L. A. Citation analysis of education dissertations for collection development. *Library Resources & Technical Services* 48, 2 (2013), 102–106.
- [13] LIPSKI, M. Stub. <https://www.softwaretestingmagazine.com/knowledge/unit-testing-fakes-mocks-and-stubs/> accessed on October 25th, 2019.
- [14] LOPER, E., AND BIRD, S. Nltk: the natural language toolkit. *arXiv preprint cs/0205028* (2002).
- [15] LOPEZ, P. Grobid: Combining automatic bibliographic data recognition and term extraction for scholarship publications. In *Research and Advanced Technology for Digital Libraries* (Berlin, Heidelberg, 2009), M. Agosti, J. Borbinha, S. Kapidakis, C. Papatheodorou, and G. Tsakonas, Eds., Springer Berlin Heidelberg, pp. 473–474.
- [16] ŘEHŮŘEK, R., AND SOJKA, P. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (Valletta, Malta, May 2010), ELRA, pp. 45–50. <http://is.muni.cz/publication/884893/en>.
- [17] SALTON, G., AND MCGILL, M. J. *Introduction to modern information retrieval*. mcgraw-hill, 1983.
- [18] SHINYAMA, Y. Pdfminer, Oct. 2007. <https://github.com/euske/pdfminer>.
- [19] SPERBERG-MCQUEEN, C. M., AND BERNARD, L., Eds. *Guidelines for the encoding and interchange of machine-readable texts*, 1.0 ed. Text Encoding Initiative, Chicago, 1990.
- [20] TKACZYK, D., COLLINS, A., SHERIDAN, P., AND BEEL, J. Evaluation and comparison of open source bibliographic reference parsers: A business use case. *CoRR abs/1802.01168* (2018). <http://arxiv.org/abs/1802.01168>.
- [21] TKACZYK, D., COLLINS, A., SHERIDAN, P., AND BEEL, J. Machine learning vs. rules and out-of-the-box vs. retrained: An evaluation of open-source bibliographic reference and citation parsers. *arXiv.org* (2018).