Information Storage and Retrieval

Text Analytics and Machine Learning

Authors

Adheesh Juvekar
Jiaying Gong
Prathamesh Mandke
Rifat Sabbir Mansur
Sandhya Manjunatha Bharadwaj
Sharvari Shirish Chougule

Instructor

Dr. Edward A. Fox

CS 5604
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
December 29, 2019

CS5604: Information Storage and Retrieval, Fall 2019

Authors: Adheesh Juvekar, Jiaying Gong, Prathamesh Mandke, Rifat Sabbir Mansur, Sandhya Manjunatha Bharadwaj, and Sharvari Shirish Chougule.

This research was completed under the supervision of Dr. Edward A. Fox as part of the CS5604: Information Storage and Retrieval course at Virginia Tech in the Fall of 2019.

1st edition, September 19, 2019

2nd edition, October 10, 2019

3rd edition, October 31, 2019

Final edition, December 29, 2019

Contents

Abstract

In order to exploit the burgeoning amount of data for knowledge discovery, it is becoming increasingly important to build efficient and intelligent information retrieval systems. The challenge in informational retrieval lies not only in fetching the documents relevant to a query but also in ranking them in the order of relevance. The large size of the corpora as well as the variety in the content and the format of information pose additional challenges in the retrieval process. This calls for the use of text analytics and machine learning techniques to analyze and extract insights from the data to build an efficient retrieval system that enhances the overall user experience. With this background, the goal of the Text Analytics and Machine Learning team is to suitably augment the document indexing and demonstrate a qualitative improvement in the document retrieval. Further, we also plan to make use of document browsing and viewing logs to provide meaningful user recommendations.

The goal of the class is to build an end-to-end information retrieval system for two document corpora, viz., Electronic Theses & Dissertations (ETDs) and Tobacco Settlement Records (TSRs). The ETDs are a collection of over 33,000 thesis and dissertation documents in VTechWorks at Virginia Tech. The challenge in building a retrieval system around this corpus lies in the distinct nature of ETDs as opposed to other well studied document formats such as conference/journal publications and web-pages. The TSR corpus consists of over 14M records covering formats ranging from letters and memos to image based advertisements. We seek to understand the nature of both these corpora as well as the information need patterns of the users in order to augment the index based search with domain specific information using machine learning based methods.

We use K-Means, Agglomerative, DBSCAN and Birch clustering algorithms for clustering the *Doc2Vec* vectors generated from the abstracts of the ETD corpus. We use the cluster IDs to update the metadata and suggest similar documents to the user. We also explored different pre-trained models of detecting sentiments. We identified a package, *empath*, that shows better results in identifying emotions in the tobacco deposition documents. Besides, we implemented text summarization based on a new proposed model which combines feature-based, graph-based, and topic-based models on 1 million tobacco dataset. We also implemented text summarization on a sample ETD chapter dataset.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

This report covers the possible avenues that we, the Text Analytics and Machine Learning team, intend to explore over the duration of the CS5604 course project. To begin with, we briefly discuss our role in the entire project. Further, we emphasize the requirements of the IR system for each of the two text corpora, i.e., the Electronics Theses & Dissertations (ETDs) and the Tobacco Settlement Records. Finally, we enumerate the models, design strategies, and approaches that we intend to exploit in developing an intelligent IR system.

1.2 What Are We Aiming To Do?

The goal of the project is to build an end-to-end state of the art information retrieval system for the following 2 text corpora.

1. 33K Electronic Theses and Dissertations - VTechWorks
2. 14M Tobacco Settlement Records - UCSF

The project workflow involves building an interactive and user-friendly front-end, a suitable indexing scheme along with the underlying database management (ELS, Kibana), integration with containers that make for a suitable collaboration and deployment environment, and finally an intelligent text analytics component that augments the present

metadata in the ElasticSearch (ELS) for enhanced search and retrieval. The goal of our team (TML) is to understand domain specific and need-based insights from the perspective of the systems users and to materialize them in the form of features in the system using suitable methods drawing primarily from the domains of Natural Language Processing, Machine Learning, and Text Analytics. In order to understand the user-requirements, we treat each of the 2 collections independently to identify the key ideas for enhancing the IR system. The Tobacco Settlements corpus consists of records of cases pertaining to the tobacco companies. With regards to the Tobacco Settlements collection, Dr. David Townsend will be the primary user of our IR system for use in his research. He is an Assistant Professor at the Department of Management in Virginia Tech. His research interest focuses on capital acquisition processes including crowdfunding, angel investments, and venture capital. We discuss (in further sections) his insights and expectations from the IR system that shall best serve the use case. The ETDs are a collection of over 33K Virginia Tech (www.vt.edu) master's and Ph.D. theses and dissertations, accessible at https://vtechworks.lib.vt.edu/handle/10919/5534. The need to develop an IR system for the ETD collection stems from two facts. First, much of the information in ETDs at a plethora of educational institutions remains dormant and inaccessible to the research community. Second, much of the public research in scientific document retrieval systems has been done in the context of conference or journal proceedings (as in Google Scholar: https://scholar.google.com/) that are patently distinct from ETDs in terms of format and content. This motivates the need to approach the problem of designing an IR system targeted towards ETDs that can draw from the basic ideas of retrieval in scientific proceedings and build on them the domain specific enhancements conducive to IR for ETDs.

1.3 Understanding the Data

This section discusses the two datasets that form the basis of our exploration of techniques for efficient retrieval. This section will be iteratively modified throughout the course of our work to include additional insights and nuances as and when they are discovered. We mention here not only the statistics and meta-information pertaining to the data but also the requirements that entail a robust retrieval. Further, we intend to include information from the CME and CMT teams since they are directly dealing with the two datasets. The reason for including a section on the datasets in our report is that under-

standing the data and the needs of the user/system are pertinent to developing Machine Learning methods for relevant document ranking.

1.3.1 Electronic Theses & Dissertations

The Electronic Theses and Dissertations Dataset is a corpus of thesis and dissertation documents from different departments at Virginia Tech. The entire corpus consists of over 33k documents including both theses and dissertations. For each work, different chapters in text format are provided.

The initial subset studied, from 2017, includes 281 different dissertations. For each dissertation, different chapters in text format are present.

1.3.2 Tobacco Settlement Documents

The following information is written with the help of the Collection Management Tobacco Settlement Documents (CMT) team. There are over 14M (14931435) documents. All underwent the procedure of Optical Character Recognition (OCR) and have the .ocr format, which means that the documents are manageable by appropriate text editors.

The metadata representing the descriptive information about the documents, such as number of pages, type, etc., is stored in the MySQL database (5.5.60 - MariaDB). It includes five tables: **idl_collection**, **idl_industry**, **idl_field**, **idl_doc_field** and **idl_doc**. The schema of the tables are in Figure 1.1 (courtesy of the CMT team).

1.4 Challenges Faced

1.4.1 Clustering

We faced quite a few challenges over the course of this work, both in terms of technical difficulties in running our programs on the cloud as well as dealing with the processing of the large scale tobacco corpus. To begin with, the cloud system crashed due to multiple node failures while running the K-Means program with different initializations and computing the Calinski-Harasbasz Index for choosing the optimal cluster count. Further, dealing with TFIDF vectors for the ETD and tobacco corpora proved to be a challenge not only due to their large size but also their sparsity. We discuss about dealing with this

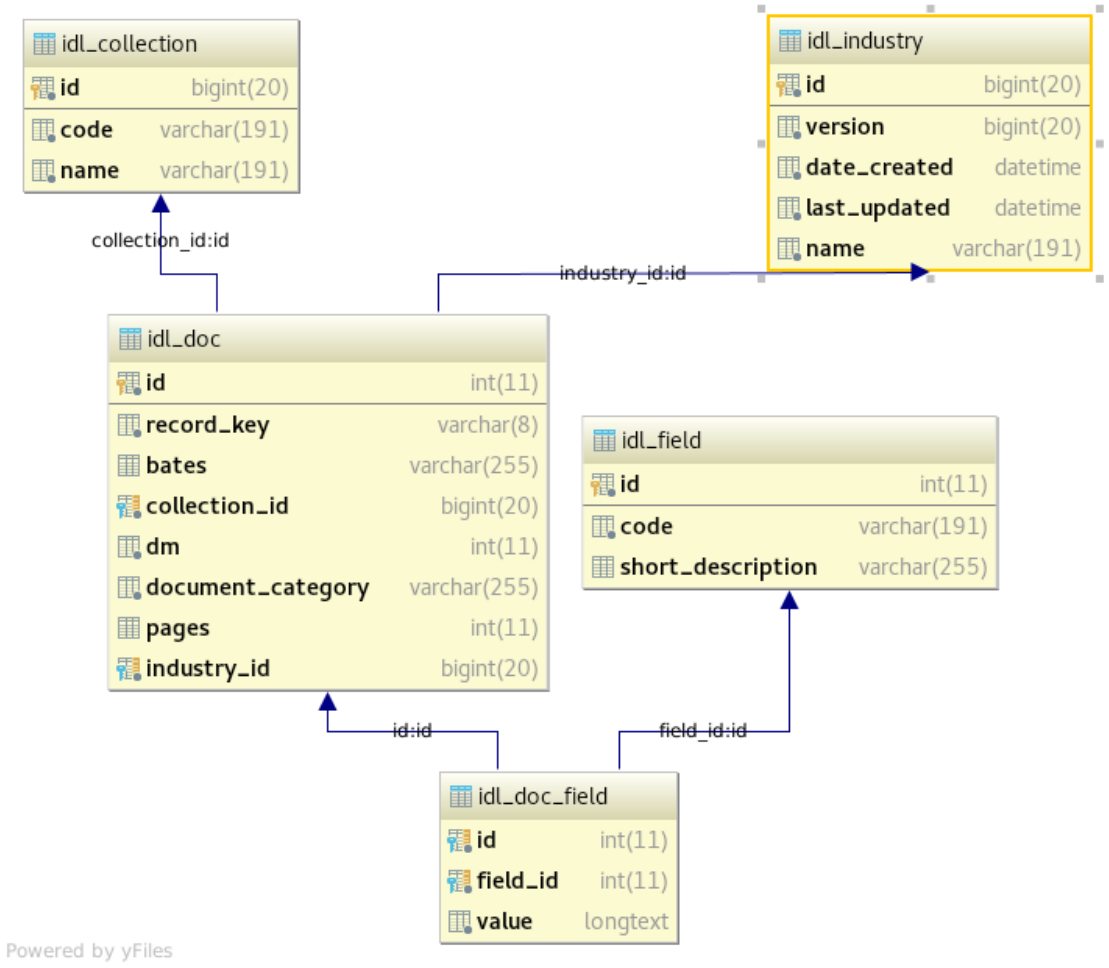

Figure 1.1: Data Structure from the CMT Team

problem in Section 1.5.1. Secondly, choosing an optimal Hierarchical Clustering method that best fits our use case appeared to be a challenging endeavor due to the myriad of such techniques in the literature. While some methods rely on the K-Means algorithm for initial iterations of clustering, others suggest a bottom-up approach by beginning with smaller clusters and combining them to get clusters of the required size.

1.5 Solutions Developed

1.5.1 Clustering

For pre-processing the Tobacco Settlement documents, we first converted all text to lowercase followed by tokenization using a sentence tokenizer which helps demarcate sentences and the *TreeBank* word tokenizer. Finally, we removed from the text a list of common English stopwords provided by *scikit-learn*. More details are given in Section 5.1.1. The reason for the node failures during the K-Means iterations was the unusually large size of the TFIDF document vectors. Moreover, these vectors are extremely sparse with only ~1% of values being non-zero. We strongly believe that this has caused the imbalance in the cluster sizes. An obvious reason for the sparse nature of the vectors is the inexact tokenization of the documents, which in turn is likely to be caused by the presence of invalid characters in the documents. Finally, with regard to the large size of the vectors, we compute *Doc2Vec* based document vectors based on the abstracts of the ETD corpus. We choose 128 dimensional embeddings for the purpose of vectorizing the corpora. These vectors help capture the context of the words in the document and thus encode semantic relationships among documents. We also explored dimensionality reduction of the TFIDF vectors, however, the results were inconclusive primarily due to the extreme sparsity in the vectors. In addition, in order to come up with a robust set of algorithms as well as their hyper-parameters we perform extensive cross-validation experiments across different types of clustering algorithms (both large scale and small scale) as well as number of clusters for each of these algorithms. We present a more detailed coverage of these topics in Chapters 4 and 5.

Chapter 2

Literature Review

The field of information retrieval is large, making it difficult to perform an exhaustive literature review of the myriad approaches involving machine learning and text analytics. Thus, instead of attempting such an exhaustive review, we point to certain review papers/resources that cover the breadth of the ideas we considered in our work. Further, we elaborate on the approaches that are most relevant to our use cases given the size of the dataset, the requirements for retrieval, and the affordable computational budget.

2.1 But What is Information Retrieval?

Textual Information Retrieval is the task of ranking a set of documents from a collection in an order of relevance given the user requirements. From a birds-eye-view, this process involves three primary tasks [39]: generate a representation of the query that specifies the information need, generate a representation of the document to capture the underlying distribution in the corpus, and finally, compute a measure of their mutual relevance. In our case, the user requirement is specified in the form of a textual query. The ranking metric or the measure of relevance is of utmost importance to the IR process. The very first step in generating such a ranking involves the development of a representation for both the query and the document. To this end, there are a myriad of approaches in the literature. Most ad-hoc retrieval systems use exact term counts as simple proxies for semantic understanding. A classic example is the BM25 model [50] that results from different weighting and normalization of tf-idf counts. However, these simple measures often overlook contextual information that arises due to the exact

ordering of the terms. Thus, as mentioned in [39], this motivates a way to perform inexact matches that go beyond term frequency counts. Problems also occur when the query contains terms that are either rare or absent from the document corpus. While exact matching can cover the case of rare term occurrence, the appearance of a new term entails the consideration of a non-fixed size vocabulary.

Another approach to building IR systems is that of Language Modelling [25] [47]. The idea is to build a language model for each document from the corpus and compute a posterior probability of the query given the document aka the query likelihood as a ranking measure. Efforts have also been directed towards a reverse approach where a model for queries is built and the document likelihood is computed. While both these approaches are somewhat similar under Bayesian statistics, the task of building a language model for the queries is somewhat impractical considering the amount of data available. A third idea under Language Models is to build a language model for the query as well as the document and then compute a similarity measure between them [28]. A natural way of comparing probability distributions is to use an information theoretical measure such as the commonly employed KL divergence [18].

A clear lacuna in the two aforementioned approaches is that both are based on a term frequency measure of the corpus and thus fail to capture semantic relationships and contextual nuances. Furthermore, they do not extend to include synonymous terms in the retrieval. With a view to address this, we consider translational models, which allow the inclusion of synonymous terms at the expense of increased computational cost [15].

2.2 Learning Representations for Information Retrieval

Representing the information in a suitable way is a fundamental and important step in building a retrieval system. Often, the smallest unit of representation is the term in a document, which is modeled in a form commonly known as a vector representation. Since the basic requirement is to fetch a relevant group of documents given a set of terms that form the query, the motivation behind learning good vector representations is to develop a notion of similarity among term vectors. The simplest approach to build such a representation is to use a local scheme where each term vector covers all terms in

the vocabulary with Boolean switches indicating their presence or absence. An obvious drawback of this so-called local representation is the size of the vectors as the vocabulary size increases. Moreover, these simplistic representations do not capture any notion of closeness between terms.

A different approach motivated by the distributional hypothesis [24], is to learn vectors that capture the distributional semantics in the term vector space. The idea is to learn vectors for terms based on their context with the underlying assumption that word semantics are distributed in the context in which they appear [23]. While this idea tends to work well compared to sparse local representations, it is plagued by the fact that the notion of semantics between terms is in itself fuzzy and thus different attempts to learn such representations may not lead to the semantic relationships relevant to the task.

A promising idea is to learn a single representation for entities larger than terms such as paragraphs or entire documents. One such noteworthy approach is elucidated by Le et al. [29], where they learn a single embedding for a paragraph/document by training a neural network to predict random words sampled from the paragraph/document. Further, they also augment the input with contextual term representations to make the predictions, thereby simultaneously learning term representations.

Such embeddings can then directly be used for clustering the corpora. This can potentially help increase user experience as elaborated in Section 2.3.1.

2.3 Clustering Basics

Note: This section draws from the coverage of this topic in [37].

Clustering is an unsupervised learning approach to derive meaning from data corpora. As opposed to supervised approaches where data is accompanied with the corresponding class label, the unsupervised way involves dealing with a corpus of points without any pre-determined class associated with them. In the context of information retrieval systems, the documents represent the unlabelled corpora in which we desire to derive meaning for efficient retrieval. The basis on which the clustering approach relies to augment information retrieval systems is the cluster hypothesis which states that:

"Documents in the same cluster behave similarly with respect to relevance to information needs." [37]

Thus, the idea is to exploit the underlying covert semantic association between documents to efficiently retrieve documents relevant to the query. From a perspective of implementation, this is achieved by augmenting the metadata in the ELS (Elastic Search) system. More precisely, our target is to cluster the documents and label the resulting clusters which can be used in the metadata for enhanced searching. Below we discuss common ways of clustering, ranging from the rudimentary ones to the more arcane types. Further, we discuss various ways that the system can be designed so as to take full advantage of the clustering techniques while also enhancing the document retrieval. We also discuss how clustering would be useful on a case-wise basis for both the ETDs and the Tobacco Settlement records.

1. **Flat and Hierarchical clustering**

   Flat clustering treats the entire corpus of data as a single block or level and iteratively splits it into a set of clusters that are not immediately related to each other. On the other hand, the hierarchical approach tends to group clusters together while also maintaining some relationship among two different clusters. Such relationships can be as simple as a one-to-one mapping between similar clusters [37].

2. **Soft and Hard clustering** [37].

   The idea of hard clustering is the most intuitive, in that it suggests to associate each datapoint (here, document) with only a single cluster. On the other hand, soft clustering takes an approach similar to fuzzy logic where each datapoint belongs to every cluster with some finite measure of association. These measures could be modeled as probabilities which would result in a distribution for each datapoint over all clusters. Likewise, these measures of association can be used as ranking metrics for displaying results in an IR system.

2.3.1 Types of clustering

1. **Search result clustering** [37].

   Often, the result of a single search query is a significantly large number of documents, that the user would take a non-trivial amount of time surfing through, to find the one (or multiple) they are looking for. If the results are displayed sequentially as they are retrieved, there is a subtle yet non-trivial problem that might plague the system. Consider the results of a web search for the query *mouse*. It

might happen that the first few results displayed to the user pertain to one of interpretation (here, animal) of the query, while the expected results (here, computer spare part) get placed further down in the list. Thus, the idea of search clustering is to cluster the retrieved documents into multiple classes and display the resulting documents on a per-class basis.

2. **Scatter-Gather** [37].

   The idea of a scatter gather approach to clustering is to replace the conventional notion of query based information retrieval with a multi-step selection based searching interaction which is backed by clustering. Here the user is first presented with a fixed set of categories to choose from. Once one or more of these categories are chosen, the documents relevant to these categories are retrieved and clustered into another set of categories for the user to select from. This process is repeated until the user finds the relevant document/s.

3. **Collection Clustering - A static hierarchical approach** [37].

   Instead of clustering search results, collection clustering builds a hierarchy of clusters around the entire corpus. The user can then be presented with a fixed set of categories to choose from. This would help avoid the need to cluster the documents in real-time as entailed by the scatter gather approach. Another off-shoot of clustering the entire corpus is that it can help improve recall. For example, when a document is returned by ELS in response to a query, (most if not) all the documents belonging to the same cluster can be displayed to the user. This, again, is in accordance with the cluster hypothesis stated above.

4. **Hierarchical Clustering**

   Hierarchical clustering is an algorithm used to cluster unlabeled data. It is divided into two types, i.e., Agglomerative and Divisive. For Agglomerative Clustering, a bottom-up approach is used, i.e., each data point in the dataset is treated as an individual cluster. Then similar clusters are merged iteratively to form a single cluster. Divisive clustering uses a top-down approach. All of the data points are treated as one big cluster which is based on the similarities inside and between the clusters.

2.4 Text Summarization

There are many relevant works on text summarization. Some involve different extractive summarization techniques. For example, a feature-based model [53] is very popular in keyword extraction and text summarization. It can filter out the non-relevant information, and measure the similarity through Term Frequency–Inverse Document Frequency (TF-IDF) to summarize different events. For sentence extraction, a graph-based model such as TextRank [38] builds graphs on texts, identifying connections between various entities in a text, and implements the concept of recommendation. TextRank can extract both keywords and sentences from long documents because it relies on both local context of a text unit and information from the entire text. A topic-based model [45] uses Latent Semantic Analysis based summarization algorithms to extract information such as which words are used together and which common words are seen in different sentences.

2.5 Recommender Systems

With the growing volume of online information, recommender systems have helped to overcome information overload. The field of deep learning is gaining significance in information retrieval and recommender systems [54]. [52] describes different ways of combining predictions from User-based collaborative filtering and Item-based collaborative filtering to minimize overall prediction error using multiple linear regression and support vector regression techniques.

Chapter 3

Requirements

There are two types of requirements for this projects. They are described below.

3.1 Our Requirements

This section describes the requirements that we have from different teams of the class. The requirements relative to the other teams are mentioned below.

CME Team

CME should provide:

1. A tokenized version of the corpus.
2. Results from preprocessing, including removing extraneous characters, tokenization, and lemmatization.
3. Whole documents in text format.
4. Chapter text files from the pre-processed data.

CMT Team

CMT should provide:

1. A tokenized version of the corpus.

2. Results from preprocessing, including removing extraneous characters, tokenization, and lemmatization.

3. Whole documents in text format.

FEK Team

FEK should provide:

1. UI requirements for recommendations.

2. UI requirements for document summaries.

3. UI requirements for clustering.

4. UI requirements for sentiment analysis.

5. User logs along with the required documentation for the various log fields for recommendations.

ELS Team

ELS should provide:

1. User logs along with the required documentation for the various log fields for recommendations.

2. Formats for adding new fields for document summaries, clustering values, and named entity recognition.

INT Team

INT should provide:

1. A stateful container deployment pipeline (e.g., docker commit).

2. A container with Python libraries (NLTK, Gensim, Sumy) installed.

3.2 Class Requirements

The goal of the Text Analytics and Machine Learning (TML) is to improve the performance and effectiveness of the search system. In order to achieve that we will be using different techniques of Machine Learning and state-of-the-art tools. Therefore, the class requirements or expectations towards the TML team can be listed as:

1. Recommendation for similar ETD/tobacco documents.
2. Summarizing the very long documents in the tobacco dataset and summarizing each chapter in the ETD dataset.
3. Clustering index and NER index values.

Chapter 4

Design, Tools, and Conceptual Background

This section describes various techniques used to enhance the information retrieval process.

4.1 Machine Learning for Information Retrieval

Here we discuss Machine Learning based approaches to improve user experience while using the information retrieval system. The system architecture diagram of the TML team can be seen in Figure 4.1.

4.1.1 Clustering

Here we discuss the clustering algorithms that we have used for the ETDs and/or the tobacco document corpora.

Clustering Algorithms

1 **Random Splitting**

To get the system working, data is split randomly into clusters, i.e., each example (a document in this case) is assigned to a random cluster. This is not an actual clustering method since the items in a cluster will not necessarily be similar; also,

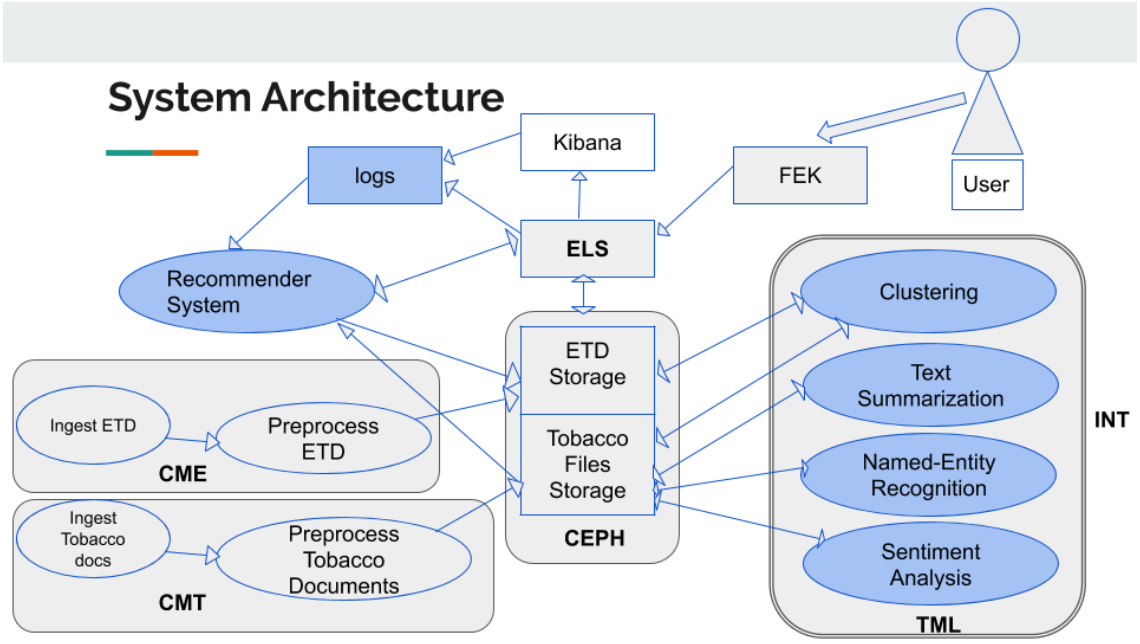

Figure 4.1: System Architecture Diagram of TML Team

two items in different sets could be very much similar to each other. The main purpose of this step is to get the system working by providing metadata for the clustering attribute.

2 **K-means clustering**

The K-means clustering algorithm divides the data into k clusters. The mean of all the points in a cluster is the centroid/representative point for that cluster. Each example/document belongs to the cluster whose centroid is closest to that example. Different distance measures are used with k-means and the most optimal one depends on the data that is to be processed. Hence, training and testing the data with different measures is a good way to finalize the right measures.

3 **DBSCAN**

DBSCAN is a density based clustering algorithm proposed by Ester et al. [20] for discovering clusters in large scale databases with noise. This algorithm addresses the shortcomings that entail the clustering of large scale databases. In particular, DBSCAN requires minimal domain knowledge of the corpus for fine tuning the pa-

rameters of the model. Moreover, it is capable of discovering clusters with arbitrary shape, which is particularly important for large spatial vector spaces wherein the cluster shapes might not be spherical as is assumed by such algorithms as K-Means. Finally, DBSCAN has the added advantage that it scales elegantly to large corpora with more than just a few thousand data points. Thus, DBSCAN suits our use case since both the ETDs and tobacco corpora are large scale spatial databases with potentially non-spherical clusters. Note that the documents occupy a Euclidean space after conversion to an N-dimensional vector space using *Doc2Vec* [29].

4 **Birch**

Birch is an efficient data clustering method for very large databases. It was designed to optimize the memory and I/O cost bottlenecks while dealing with large corpora. Further, Birch efficiently deals with noise (documents that do not belong to any cluster within a similarity threshold). Birch starts by clustering the entire corpus in a single scan and then (optionally) improving the cluster quality in subsequent iterations. In particular, Birch dynamically and incrementally clusters data points within a specified memory and computation constraint [55]. This algorithm is especially crucial in our case while dealing with the ~91k articles from the Tobacco Settlement Records. Using multiple parallel workers with such techniques as K-Means leads to an monumental consumption of memory which calls for such algorithms as Birch that are memory and I/O efficient.

5 **Agglomerative Clustering**

Agglomerative clustering treats every data point as an individual cluster and similar clusters are merged iteratively to form a single cluster containing all the data points [48]. The steps followed in this method are given below.

(a) Let each data point be an individual cluster.

(b) Find the similarity between all the pairs of the clusters and merge two similar clusters.

(c) Repeat until all the clusters are merged into one cluster.

In order to decide which clusters to combine, there are various methods used to calculate the similarity between clusters.

(a) **MIN**

This method is also known as the single linkage algorithm. The similarity of two clusters is equal to the minimum value of all pairwise distances between the elements in cluster 1 and the elements in cluster 2. This method does not work well when much noise is present between clusters.

(b) **MAX**

Also known as the complete linkage algorithm, this can be defined because the similarity between two clusters is the maximum value of all pairwise distances between the elements in cluster 1 and the elements in cluster 2. This method works well when noise is present between the clusters. However, it is biased towards globular clusters.

(c) **Group Average**

The similarity between two clusters is defined as the average distance between elements in cluster 1 and elements in cluster 2.

(d) **Ward's Method**

Ward's method calculates the average of square distances between the elements in cluster 1 and the elements in cluster 2.

Dendrograms are used to visualize the hierarchical relation between clusters. It uses a tree like diagram that records the sequences of merges and splits. Any number of clusters can be obtained by cutting the dendrogram at a desired level. An illustration for this is shown in Figure 4.2.

We plan to use the *Agglomerative* class from *sklearn.cluster* to create the hierarchical clustering model as shown in Figure 4.3. The number of clusters are set using the *n_cluster* parameter. The affinity parameter has the default value of 'euclidean' which is the method we choose to calculate distance between data points. The linkage parameter is set to 'ward' which minimizes the variance between clusters. The *fit_predict* method returns names of the clusters that each data point belongs to.

4.1.2 Text Summarization

Text summarization is a technique to shorten a long text document into a summary that includes the main points of the original document. It is the process of extracting impor-

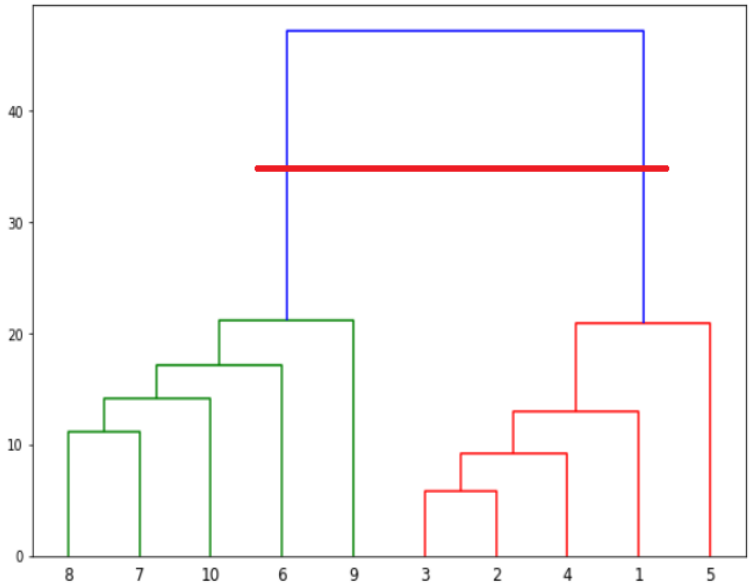

Figure 4.2: Dendrogram
[34]

```
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
cluster.fit_predict(X)
```

Figure 4.3: Sklearn AgglomerativeClustering class snippet
[34]

tant information (e.g., keywords) from a source document to provide a summary. The input is a document and the output is a summary. There are two different datasets (Electronic Theses & Dissertations and Tobacco Settlement Documents) on which we are going to implement automatic text summarization.

Motivations

1. Titles are sometimes quite confusing. When searching papers, the search engine will return a list of papers after we input the keywords. However, the titles may not be sufficiently informative. In other words, the titles sometimes are not very related to the core content of the corresponding paper.

2. Theses and dissertations are too long to review, and their abstracts are too short to include information about each chapter. To address this limitation, we are going to generate the summary of each chapter in order to save readers' time and energy.

3. It is a tedious task for human beings to generate a summary manually because it requires a rigorous analysis of the document. In order to reduce time and help humans better understand the topics, an automatic and accurate summary proves to be very useful.

Approaches

Extractive methods select keywords and sentences from documents and then combine these keywords and sentences into a summary. Extractive summarization is based on the assumption that key information of a document can be found in one sentence or several sentences from the document. It has the advantage that the summary usually follows grammar rules, and uses the wording of the author. The task of keyword extraction is to automatically identify a set of keywords that can best describe the document.

Abstractive summarization methods include keywords based on semantic understanding, where sometimes those keywords don't appear in the source documents. We form the sentences on our own and then combine these sentences to form an abstract [40]. In this method, text is interpreted by natural language processing techniques in order to generate a summary which conveys the most critical information from the original document. However, due to the demand of high quality GPU and immature technologies of abstractive methods, we will focus on several different approaches to extractive summarization.

1. **Feature Base – Term Frequency Inverse Document Frequency (TF-IDF)**
   The feature-based model will extract the features of the sentence, then a summary will be provided based on the evaluated importance. Various features can be used in evaluating the sentence importance such as term frequency, position of the sentence in the document, length of the sentence, name entity recognition, and so on. We will implement the following two feature-based models.

   TF-IDF is a numerical statistic that mainly reflects the importance of the feature item in the document representation. The TF-IDF algorithm estimates the frequency of a term in one document over the maximum in a collection of documents and assesses the importance of a word in one set of documents.

2. **Feature base - Luhn's Algorithm**
   Luhn's algorithm is a naive approach based on TF-IDF and on looking at the window size of non-important words between high importance words [33]. Besides, it also assigns high weights to the sentences near the beginning of the document.

3. **Graph Base - TextRank**
   The graph-based model makes the graph from the document, then summarizes it by considering the relation between the nodes. Graph-based ranking algorithms are essentially a way of deciding the importance of a vertex within a graph, based on global information recursively drawn from the entire graph [38].

   TextRank is an extractive and unsupervised text summarization technique based on PageRank, which is often used in keyword extraction and text summarization. It is a graph-based ranking model for text processing so that the most relevant keywords and sentences in text documents can be found. A graph will be built associated with the text, where the graph vertices are representative for the units to be ranked [38]. There is no need for TextRank to rely on any training data. Besides, TextRank can work with an arbitrary length of text.

   Therefore, we use TextRank for sentence extraction. In this method, we rank the entire sentence instead of each word, and a vertex is added to the graph of each sentence in one document. Here is the process of TextRank:

(a) Input documents (chapters).

(b) Tokenize the text into sentences.

(c) Remove stop words and generate clean sentences.

(d) Find vector representations (word embeddings) for each sentence.

(e) Build similarity matrix (cosine similarity).

(f) Convert the sentences into graphs.

(g) Weight the sentences via PageRank and generate a ranking based on the matrix.

(h) Select key sentences to provide a summary.

The flow of the TextRank Algorithm is shown in figure 4.4 [26].

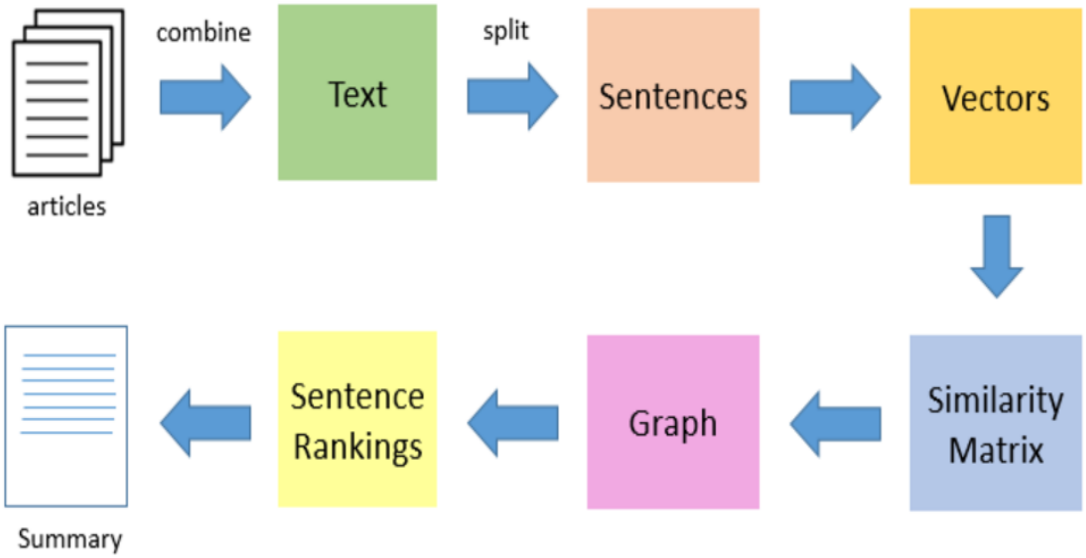

Figure 4.4: Flow of TextRank Algorithm
[26]

4. **Topic Base – Latent Semantic Analysis**
   The topic-based model calculates the topic of the document and evaluates each sentence by the included topics. In other words, the sentence including main topics may get a higher evaluation score.

   Latent Semantic Analysis (LSA) based on Singular Value Decomposition is often used to detect topics. LSA is an algebraic-statistical method that extracts hidden

semantic structures of words and sentences. LSA uses the context of the input document and extracts information such as which words are used together and which common words are seen in different sentences. A high number of common words among sentences indicates that the sentences are semantically related. The meaning of a sentence is decided using the words it contains, and meanings of words are decided using the sentences that contain the words [45]. LSA is also an unsupervised text summarization technique that does not rely on any training data. Here is the process of LSA implementation:

(a) Create an input matrix based on an input document.

(b) Model relationships among words and sentences by Singular Value Decomposition.

$$A = U \sum V^T \tag{4.1}$$

where A is the input matrix (m × n); U is words × extracted concepts (m × n); $\sum$ represents scaling values, diagonal descending matrix (n × n); and V is sentences × extracted concepts (n × n).

(c) Select important sentences to provide a summary.

The flow of the Latent Semantic Analysis is shown in Figure 4.5 [41].

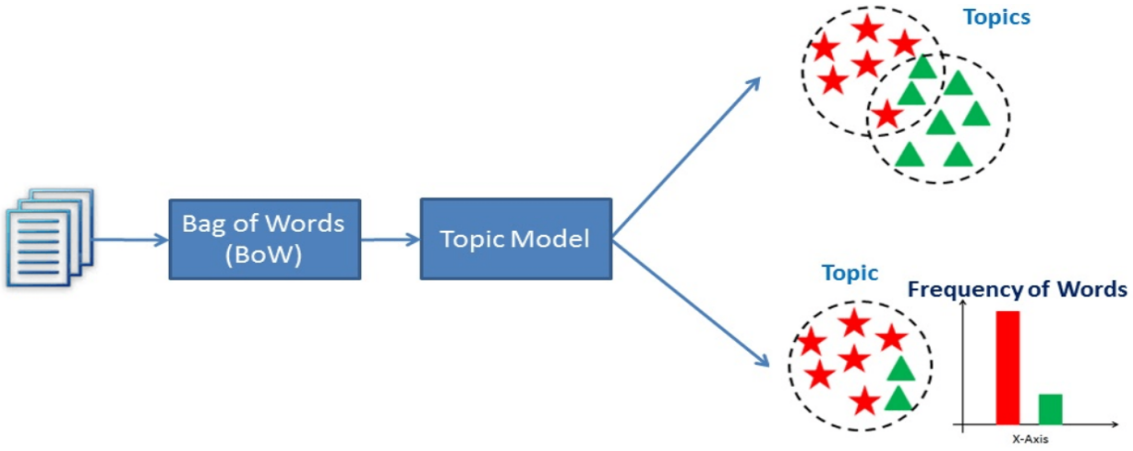

Figure 4.5: Flow of Latent Semantic Analysis [41]

In our project, we implement LSA based on Gensim, a Python library for topic modelling, document indexing, and similarity retrieval with large corpora.

New Model

Top important sentences are extracted from files to form a summary based on three different models (feature-based, graph-based, and topic-based model). However, the results are quite different. In addition, it is hard to evaluate the performance for text summarization. One popular and common evaluation technique is to manually evaluate the summary by human-beings. However, it seems to be impossible to manually evaluate the summaries generated from one million tobacco documents. Besides, these three traditional models mentioned above can all give good performance based on different features proposed. Therefore, we proposed a new model which can provide a summary considering the performance of these three traditional models. An overview of the first version of new model is shown in Figure 4.6.

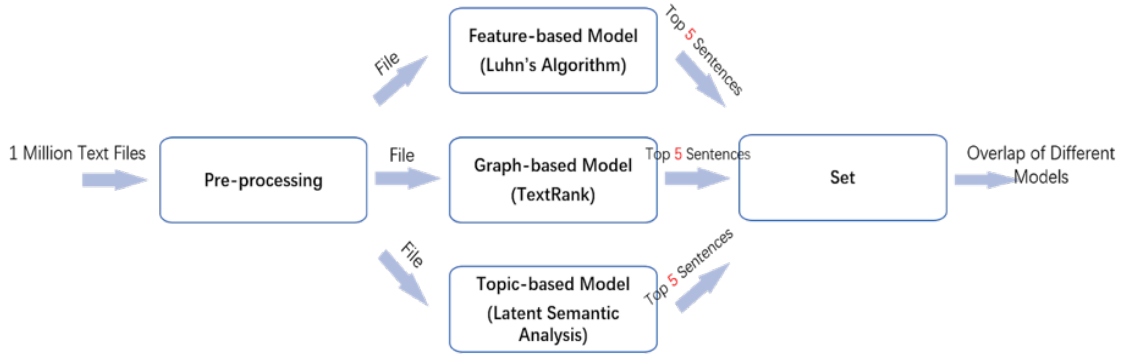

Figure 4.6: First Version of New Model

The inputs are the file documents. Each file is processed using each of three different models, separately. Top important sentences are extracted from different models. The number of extracted sentences can be easily modified by users. Then all of these top-ranked sentences are put in a set in order to find the overlap of the sentences. We assume that a sentence which appears in all three models is one of the most important sentences which can be provided in the final summary.

Problems come when there is no overlap among the three models. This problem can be solved by setting a larger threshold for each model in order to extract more sentences. Nevertheless, the overlap sentences may not be important if the threshold is set to be very large. Therefore, we proposed a second version of the new model as shown in Figure 4.7.

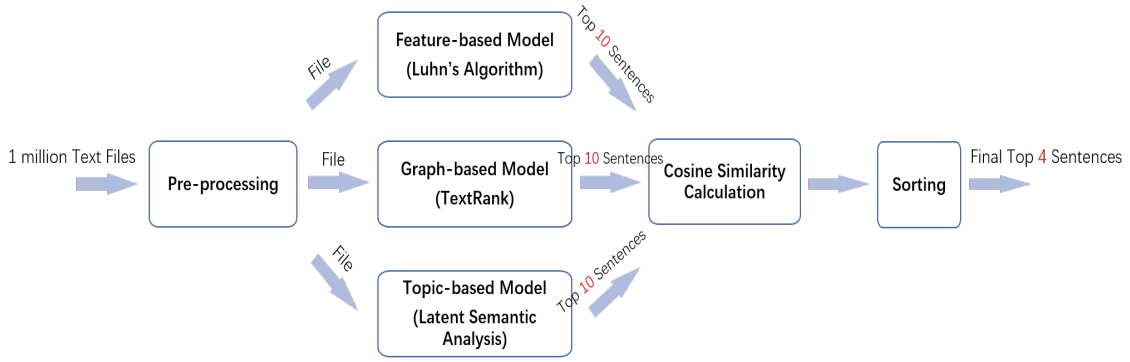

Figure 4.7: Second Version of New Model

In this model, we replace the set union step with cosine similarity calculation and sorting. Even if there's no overlap of the result sentences extracted from the three traditional models, we can still get the most similar sentences which appear in the top ranked sentences of each model. Then we rank these sentences and provide the final summary.

4.1.3 Named-Entity Recognition (NER)

Understanding the context of a document is of very high value, especially for information extraction. In any form of document, there are particular terms representing particular entities which are more informative in a unique context. Real-word objects such as persons, locations, organizations, date-times, and other types of named entities can indicate the essence of the whole document. named entity recognition (NER) is a popular technique used in information extraction to identify and segment the named entities and classify or categorize them under various predefined classes. As a more formal definition, Named-entity recognition (NER) (also known as entity identification, entity chunking, and entity extraction) is a subtask of information extraction that seeks to locate and classify named entity mentions in unstructured text into pre-defined categories such as the person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc. [5].

NER: Applications and Usecases

There are many practical applications and use cases of NER. In our case, we have two different types of datasets. Due to the nature of these two datasets, we have some variety

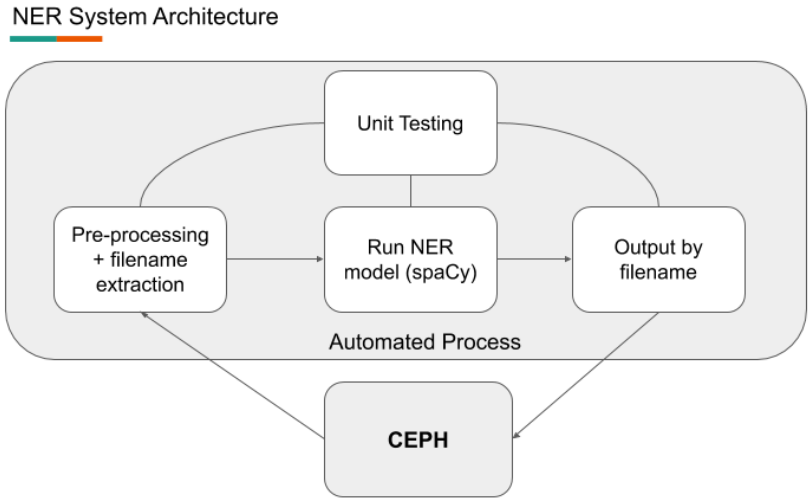

Figure 4.8: System Architecture of NER

in the NER use cases. However, from a broad perspective, we would be using NER to improve the search result in Elasticsearch. Some of the use cases for NER are as follows:

1. **Improve Search Results**
   Our primary goal of using NER is to improve the search results from the Elasticsearch system. We have two different corpora of data, one ETD dataset and another of tobacco settlement documents. The ETD dataset has 33 thousand documents, whereas the tobacco dataset has over 14 million documents. Therefore, if for every search query the algorithm ends up searching all the words in millions of articles, the process will take a long time . Instead, if Named Entity Recognition can be run once on all the articles and the relevant entities (tags) associated with each of those articles are stored separately, this could speed up the search process considerably. With this approach, a search term will be matched with only the small list of entities discussed in each of the articles, leading to faster search execution.

2. **Contributing to Recommender Systems**
   One of the major uses cases of NER involves automating the recommendation process. News media, such as BBC News, use NER to recommend news articles to

their users. These recommendations are mostly content-based recommendations. As mentioned in the article by BBC News Lab [11], they extract named entities by using the Stanford Core NLP Framework [9], which they then match with their DBpedia Lookup Service [2]. Using these matches, the BBC News media recommend similar news articles to their customers.

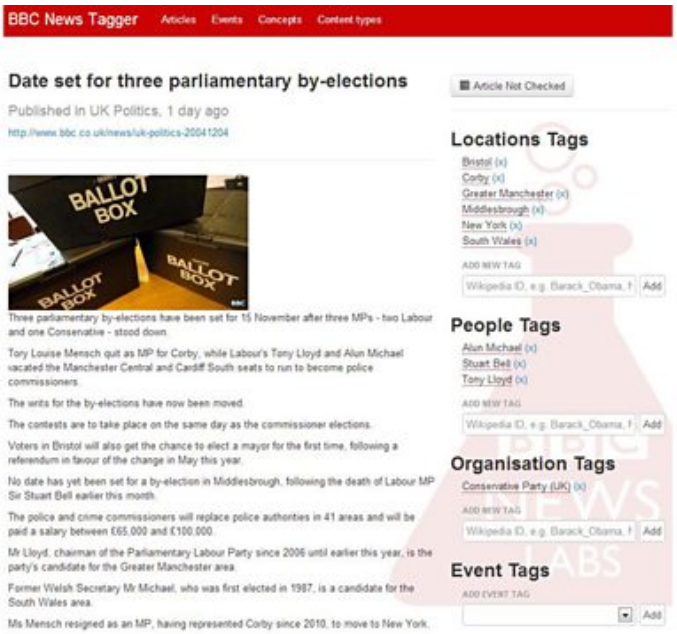

Figure 4.9: BBC News Lab using Name Entity Recognition

3. **Classifying Content from Documents**
   The tobacco deposition corpus is associated with approximately 600 court cases. Named Entity Recognition can automatically scan these documents and reveal which are the major people, organizations, and places discussed in them. Knowing the relevant tags for each document can help in automatically categorizing the documents in defined hierarchies, and enable relevant content discovery.

4. **Classifying Contents in Research Papers**
   Similar to the points mentioned before, organizing the contents of the ETD research papers in a well-structured manner can save a lot of computational overhead. Since the ETD dataset covers a distinctive domain, we need to use scholarly, e.g., scientific

models to identify scientific named entities. Segregating the papers on the basis of the relevant entities can speed up the searching for relevant results.

Popular NER Tools

There are several NER systems that use linguistic grammar-based techniques as well as statistical models such as machine learning. State-of-the-art NER systems for English produce near-human performance. For our research, we selected three high performing, open sourced NER tools. There are as follows:

1. **Stanford NER**
   Stanford NER is a Java implementation of a Named Entity Recognizer (NER). It labels sequences of words in a text which are the names of things, such as person and company names, or gene and protein names. It comes with well-engineered feature extractors for Named Entity Recognition, and many options for defining feature extractors. Included with the download are good named entity recognizers for English, particularly for the 3 classes of PERSON, ORGANIZATION, and LOCATION [10].

2. **NLTK NE_Chunk**
   NLTK provides a classifier that has already been trained to recognize named entities, accessed with the function *nltk.ne_chunk()*. The classifier can also add category labels such as PERSON, ORGANIZATION, DATE, TIME, MONEY, and GPE. Chunkers can be constructed using rule-based systems, such as the "RegexpParser" class provided by NLTK, or using machine learning techniques, such as the "ConsecutiveNPChunker". In either case, part-of-speech tags are often a very important feature when searching for chunks. Although chunkers are specialized to create relatively flat data structures, where no two chunks are allowed to overlap, they can be cascaded together to build nested structures. Relation extraction can be performed using either rule-based systems which typically look for specific patterns in the text that connect entities and the intervening words, or using machine-learning systems which typically attempt to learn such patterns automatically. Research reports describe projects that investigate aspects of computing education using a training corpus [6].

3. **spaCy**
   spaCy is an open-source software library for advanced natural language process-

ing. It is written in the programming languages Python and Cython. spaCy comes with pre-trained statistical models and word vectors, and currently supports tokenization for 50+ languages. It features state-of-the-art speed; convolutional neural network models for tagging, parsing, and named entity recognition; and easy deep learning integration. It is designed with the applied data scientist in mind, meaning it does not weigh the user down with decisions over what esoteric algorithms to use for common tasks and it's incredibly fast [8]. There are several other studies of tailored NER methods for very particular domains, such as ScispaCy for a Biomedical NER model [42], ChemTok for a chemistry NER model [13], etc.

4.1.4 Sentiment Analysis

Since the tobacco documents include deposition documents, there is room for human emotions. Understanding the sentiments in the deposition documents would help the expert distinguish good witnesses from bad. The whole process of identifying a good witness is very complex. The sign of a well versed witness can be found in their use of certain words, size of vocabulary, and consistency in statement. Another sign of a good witness can be in emotions they express in their statements. For identifying witness emotions we have explored some of the state-of-the-art packages detecting sentiments.

Flair

Flair uses state-of-the-art techniques for named entity recognition, text classification, and language models [12]. The tool has standard model training and hyperparameter selection routines. It is capable of using pre-trained models for different applications. There is also room for training new models and using them to classify test datasets.

Twitter Emotion Recognition

The Twitter Emotion Recognition package is a trained recurrent neural network (RNN) that can predict emotions in English tweets [17]. As the name implies, the

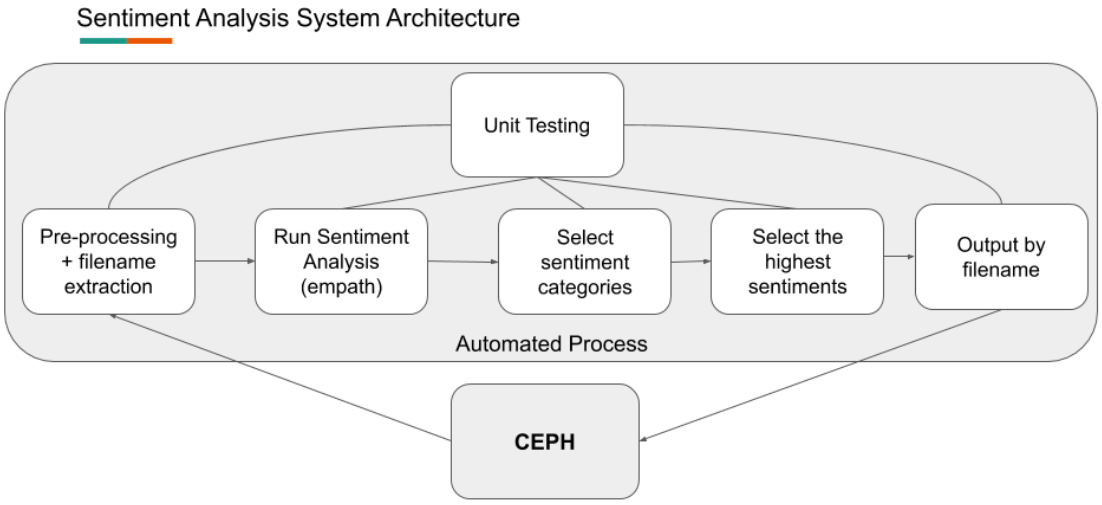

Figure 4.10: System Architecture of Sentiment Analysis

model was trained on English tweets from the popular website Twitter. The package predicts either Ekman's six basic emotions, Plutchik's eight basic emotions, or the Profile of Mood States (POMS) six mood states. Although this package is specified for tweets, we want to see if the performance holds for our tobacco deposition documents.

Empath

Empath is a tool based on neural network word prediction that can generate and validate new lexical categories on demand [21]. The Empath tool uses a small set of seed terms to classify documents into different categories. There are numerous built-in categories (total 200 at the time of this report) pre-validated from common topics in a web dataset. A user can easily create new categories by providing a small set of seed terms. Empath uses a deeply learned neural embedding across over 1.8 billion words. The workflow of the Empath tool can be found in Figure 4.11.

Figure 4.11: Empath Workflow from the Empath Paper [21]

4.1.5 Recommender systems

The abundance of information that is available on the internet makes information seeking a difficult task. Using a search query method might not be very effective as it is mainly dependent on the user's ability to fine-tune the search query. This is when the recommender system comes into the picture. Almost all modern platforms – that we use to obtain information from – use recommendation systems. Companies like Netflix, LinkedIn, Amazon, and Spotify leverage recommendation systems to give better experience to the users by recommending relevant content based on the user's preferences. The two major paradigms of the recommender system are content based filtering and collaborative filtering methods.

Types of Recommendation Systems

1 **Content-based Recommendation System**

Content-based recommendation systems deal with a good amount of items' own attributes rather than users' interaction and feedback. For example, in a music app, Pandora, each music is assigned a certain number of attributes, and based on the attributes of the music chosen by the user, the recommendations are made.

Content based methods are computationally faster and interpretable. They do not face the cold-start problem, making them easily adaptable to new items. Clustering and cosine similarity are some of the methods that are used to implement content-based recommendation systems.

An overview of the content-based recommendation system is shown in Figure 4.12 [51].

2 **Collaborative Filtering Recommendation System**

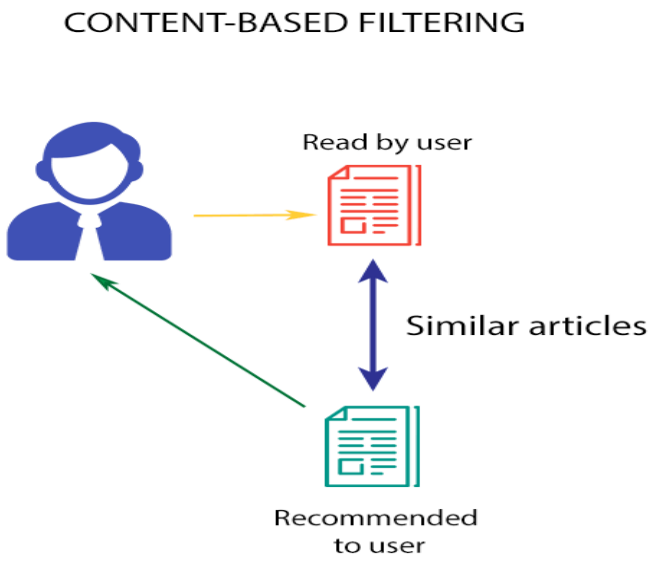

Figure 4.12: Content based recommendation [51]

Collaborative filtering is a method that makes recommendations based on users' historical preferences or by mimicking user-to-user recommendations. User-based filtering and item-based filtering are the two categories of collaborative filtering. The user-based method measures the similarities between the target user and other users while the item-based method measures the similarities between users' interactions with other items.

The applications of collaborative filtering involve large datasets. Thus it is used for many purposes, including with financial, sensing and monitoring, and environmental sensing data. An overview of the collaborative filtering recommendation system for recommending articles to a user based on another user is shown in Figure 4.13 [51].

3 **Hybrid Recommendation Systems**

Both content based and collaborative systems have their own strengths and weaknesses. Hybrid recommendation systems provide recommendations based on the weighted average of both the methods. Hybrid methods are more accurate than the pure approaches and are used to avoid common problems such as cold start, which

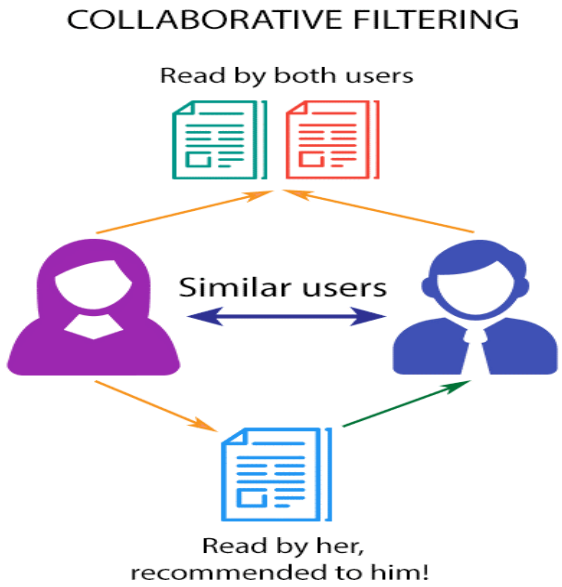

Figure 4.13: Collaborative filtering recommendation [51]

is faced in the collaborative approach, where for a new user there is not enough data to start with, and the sparsity problem, where there are insufficient ratings for certain topics.

We will be implementing some of the collaborative filtering and hybrid recommendation techniques based on user history during the course of our project. A detailed overview of these techniques is given below.

Types of Collaborative Filtering

Collaborative filtering systems make recommendations based on historic data of users' preference for items. The preference is represented as a user-item matrix. It is also called as rating matrix or preference matrix. Figure 4.14 is an example of a matrix describing the preference of 4 users on 5 items, where p_{12} indicates user 1's preference for item 2 [19].

The user-item matrix can have millions of entries since it includes all the users and all the ETDs and tobacco data. This matrix is typically huge, very sparse, and the majority of entries in the matrix will be missing because it is not possible for every

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} & p_{15} \\ p_{21} & p_{22} & p_{23} & p_{24} & p_{25} \\ p_{31} & p_{32} & p_{33} & p_{34} & p_{35} \\ p_{41} & p_{42} & p_{43} & p_{44} & p_{45} \end{bmatrix}$$

Figure 4.14: Preference matrix example

user to have a historical preference for every item. A sample preference matrix with missing values might be as shown in Figure 4.15.

$$\begin{bmatrix} p_{11} & ? & p_{13} & ? & p_{15} \\ ? & ? & ? & p_{24} & ? \\ p_{31} & ? & p_{33} & ? & ? \\ ? & p_{42} & ? & p_{44} & p_{45} \end{bmatrix}$$

Figure 4.15: Preference matrix with missing values as question marks

The goal of recommender systems is to fill these missing entries in order to predict the utility of items to each user.

The different types of Collaborative Filtering recommendation systems are as follows.

1 **User based k-Nearest Neighbors**

The nearest neighbor based method is based on the similarity between pairs of users. This technique works on the notion that there is a high tendency for a user to like the items which have been liked by other users who are alike and have similar preferences. The algorithm is as follows [19]:

(i) Compute similarity between users

The similarity between users is determined using Cosine Similarity:

$$similarity(a, b) = cos(a, b) = \frac{a \cdot b}{||a|| * ||b||}$$

(ii) Find k most similar users to user A

Among all the similar users, the k most similar users are selected.

(iii) Recommend items which have been used or seen by the k similar users but have not been used or seen by user A.

For example, in our project, we recommend to user A the ETDs and tobacco documents which have been searched by similar users but have not been searched by A according to the user search history.

2 **Matrix Factorization**

Matrix Factorization is a Latent Factor Method which creates a new and usually reduced feature space of the original user or item vectors, leading to reduced noise and faster computations in real-time. Matrix factorization attempts to reduce the dimensionality of the preference matrix and approximates it by two or more small matrices with k latent components. One of the methods used to reduce dimensionality is Singular Value Decomposition (SVD). SVD decomposes the preference matrix as follows [19]:

$$P_{m\times n} = U_{m\times m} \sum_{m\times n} V_{n\times n}$$

where U and V are unitary matrices.

A sample equation for 4 users and 5 ETDs is as follows:

$$P_{4\times 5} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ u_{21} & u_{22} & u_{23} & u_{24} \\ u_{31} & u_{32} & u_{33} & u_{34} \\ u_{41} & u_{42} & u_{43} & u_{44} \end{bmatrix} \bullet \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 \\ 0 & 0 & 0 & \sigma_4 & 0 \end{bmatrix} \bullet \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \\ v_{31} & v_{32} & v_{33} & v_{34} & v_{35} \\ v_{41} & v_{42} & v_{43} & v_{44} & v_{45} \\ v_{51} & v_{52} & v_{53} & v_{54} & v_{55} \end{bmatrix}$$

The preference of the first user for the first ETD can be written as follows:

$$p_{11} = \sigma_1 u_{11} v_{11} + \sigma_2 u_{12} v_{21} + \sigma_3 u_{13} v_{31} + \sigma_4 u_{14} v_{41}$$

The reduced matrix which has been obtained from the users' search history is used to provide recommendations to the user.

3 **Neural networks**

This is a Deep Learning based approach which models the sequential information based on the users' item purchase or search history using neural networks. Let us consider a user A who has an item sequence as follows: item1 - item2 - item 3 -

item4. The intuition is to use each item in the sequence to predict its neighboring items, formulated as a classification problem, where each item is one class. The training data include the neighboring K items of each item (the left K and right K items). Figure 4.16 illustrates the pairs of items with K = 1 [19].

Figure 4.16: Deep Learning Approach for Collaborative Filtering [19]

This sequence of items is used to train a neural network which is used to provide recommendations to another similar user.

For example, consider a scenario where there is a user A with a search history sequence as ETD1 - ETD2 - ETD3 and there is another user B who has searched for ETD1, then ETD2 and ETD3 will be among B's recommendations as it is likely that user B will also search for similar content.

Sequential neural network architectures such as LSTM and GRU can be used to model the search history.

Another deep learning approach for recommendation system is the use of autoencoders. In this technique the preference matrix is compressed using a neural network such as an autoencoder. This approach is similar to matrix factorization where we reduce the dimensionality of the rating matrix.

Implementation Methodology

The aim of recommender systems is to provide better recommendations tailored to the needs of the users and optimize the search results.

We will be implementing content based recommendations using techniques such as clustering which take into account the users' personal search history and logs, and the search results for both the ETD and tobacco datasets.

We will be implementing collaborative filtering based on the matrix factorization technique which is based on the historical logs of other similar users for searching of ETDs and tobacco data. The historical user logs will be used for training these models. A part of the logs will be used as test data for validating the model performance. Evaluation metrics such as Precision, Recall, and Root Mean Square Error will be used for analyzing the model performance.

Finally, we will also be implementing a hybrid system by combining the content based and collaborative filtering techniques for searching of ETD and tobacco data, thereby developing a powerful recommendation system which enhances the quality of search engine results.

Chapter 5

Implementation and Preliminary Results

5.1 Clustering

5.1.1 Pre-processing

We run preliminary pre-processing on the text documents in order to tokenize them and generate vector representations for clustering. We make use of the Python programming language in order to ingest the data, extract the TFIDF vectors, and finally run multiple iterations of the K-Means algorithm. Specifically, we use the following packages.

Package/Module	Usage
numpy	Storing TFIDF vectors
scikit-learn	Pre-processing & K-Means
nltk	Tokenization & stopword removal
gensim	Generating Doc2Vec vectors

Table 5.1: Modules and packages used

The *numpy* [44] package in Python is useful for numerical computations involving matrix/vector operations. We use wrapper classes around these numpy arrays to store the TFIDF vectors for each document. The specifics of how the vectors are stored and how they can be accessed can be found in the Developers Manual, Chapter 8.

Scikit-Learn provides efficient Python based implementations of commonly employed pre-processing pipelines along with the implementations of well known machine learning algorithms. In particular, we make use of the *TfIdfVectorizer* pipeline of *scikit-learn* in order to iteratively compute and store the TFIDF values for the Tobacco Settlements corpus.

Initially, we worked with 2 sample subsets of the Tobacco Settlement Records (TSRs) as provided by the CMT team. The first one is a set of 7995 text files (*.ocr* format) that are the output of running Optical Character Recognition (OCR) on the *PDF* versions. These text files have not undergone any pre-processing and contain multiple invalid bytes, that is, bytes that do not have a valid UTF-8 decode value. The second data sample consists of 4553 text files (also in the *.ocr* format) that have been cleaned to remove any invalid bytes by the CMT team. We encourage the reader to peruse the CMT team's report to understand the specifics of this cleaning/pre-processing step. Below we mention the details of the tokenization and TFIDF extraction methods that we used for these experiments. A summary of the two sample datasets is included in Table 5.2.

Corpus	**Number of documents**	**Brief description**
Uncleaned TSRs set	7995	Invalid bytes have not been removed.
Cleaned TSRs sample set	4553	All files have valid UTF-8 bytes.
Cleaned articles from TSRs	916977	Cleaned articles from the Tobacco corpus.
ETDs all	30961 (13071D + 17890T)	Text and metadata for all ETDs

Table 5.2: Sample collections description

Further, we worked with ~91k articles from the Tobacco Settlements corpus. These articles are records of cases against the tobacco companies. The documents have been cleaned by the CMT team to remove any invalid characters. For the record, all the aforementioned corpus subsets are available in ceph. Please see the Developers Manual Chapter 8 for details. The ETD corpus consists of 30961 documents comprising of 13071 dissertations and 17890 theses. Owing to the size of each thesis or dissertation, we only deal with the abstract of each document.

Tokenization

We tokenize the text documents using NLTK's [6] *word_tokenize* utility. *word_tokenize* is based on the *TreebankWordTokenizer* and the *PunktSentenceTokenizer*. The *PunktSentenceTokenizer* uses an unsupervised algorithm to build a model to recognize sentence boundaries and is shown to work well for many European languages [31]. The *TreebankWordTokenizer* works on text that has been segmented into sentences and uses regular expressions to tokenize [32]. We use UTF-8 encoding to decode the bytes of the text files. In the case that a byte without valid UTF-8 character is encountered, we simply ignore and skip it during tokenization. In addition, we use *NLTK's* implementation of the Porter stemmer [49] for stemming the tokens. Finally, we remove common English language stopwords from the tokens. The list of stopwords is not specific to the corpus and is provided by *scikit-learn*. These stopwords are stored and updated, with web-link: [7]. Sample tokens from one of the files in the Tobacco Settlements corpus are shown in Figure 5.1.1.

```
Out[9]: (['access',
          'neither',
          'incorrect',
          'rversi',
          'platinum',
          'conclus',
          'commerc',
          'without',
          'ercent',
          'paragraph',
          '3ppose',
          'someth',
          'justifi',
          'georgia',
          'propos',
          'analyz',
          'doestheindustryilittcip',
          'significantli',
```

Figure 5.1: Sample Tokens from file fmbl0056.ocr from the Tobacco corpus

TFIDF Vectorization

We use *scikit-learn's TfIdfVectorizer* to extract the TFIDF features from each document after tokenization. This utility provides multiple options to tune the TFIDF vectors as desired. We mention here some options that have been used in extracting the vectors. In order to account for the fact that we did not employ any corpus specific stopword removal, we use the *max_df* parameter to ignore those tokens that have a document frequency that is strictly higher than a specified threshold. This has the effect of removing any corpus specific words from the vocabulary. The parameter is a floating point number between 0 and 1 denoting the proportion of documents that constitute the threshold. We use a *max_df* value of 0.7, meaning that we discard those terms from the vocabulary that occur in more than 70% of the documents in the corpus. The *min_df* parameter is the counterpart of *max_df* which discards words that occur in less than a threshold number of documents. We do not make use of this parameter and keep its value to be zero [1]. In addition, we also apply a L2 normalization to each vector to ensure that the squared sum of the vector elements is unity. This helps build a compact vector representation and stabilizes clustering algorithms. Finally, we smooth the *i_df* weights by adding one to each document frequency as if an extra document exists that contains every term once. This primarily helps prevent a division by zero.

Doc2Vec

Doc2Vec [29] is a neural network based technique to obtain distributed vector representations of documents of arbitrary length. The algorithm has two variants. One is the distributed bag-of-words (DBOW) representation and the other is the distributed memory (DM) approach to generate representations. In each case, the algorithm begins by randomly initializing a fixed length vector for each document. This vector is then fed into a vanilla feed-forward multi-layer perceptron to generate a distribution over the vocabulary of the corpus. Depending on which of the two aforementioned variants of the algorithm is used, the specifics of the training task differ. Particularly, in the case of the DM approach, the model is tasked with predicting the next word in a sequence within the document given the document vector as well as the vectors of words lying in the prediction window as input. DBOW is a relatively straightforward and tractable variant

[1] It can be noted that setting this parameter to a small non-zero value can help reduce the number of dimensions of the TFIDF vectors thereby helping simplify clustering. However, we do not explore this direction and switch to Doc2Vec based document vectors instead.

which involves the task of predicting a randomly chosen word from the document given the document's vector representation. Figure 5.1 is a diagrammatic representation of the DM approach.

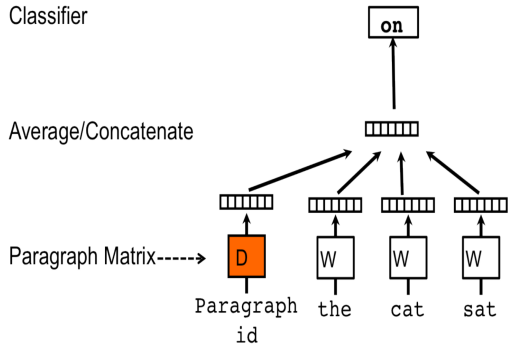

Figure 5.1: Distributed Memory (DM) approach for training document vectors using Doc2Vec. [29]

In all our implementations, we used the DM approach to obtain document vectors. Albeit, the DM approach requires considerably more memory and computation, it has the benefit of capturing semantic relationships within the documents. More precisely, two documents having a different permutation of the same set of words might have very similar DBOW vectors; their DM based vector representation are guaranteed to be distinct given enough training time for convergence. Thus, in order to capture distributed semantics, we chose the DM based training of document vectors.

5.1.2 K-Means Clustering

We implemented K-Means clustering on the initial sample set of Tobacco Settlement documents as well as the full ETD corpus (abstracts) using the full version of Loyd's [30] K-Means algorithm. The tobacco sample dataset consists of 7995 text files (in .ocr format) provided by the CMT team. The documents have not been pre-processed and thus contain bytes that cannot be decoded in UTF-8 format. To tackle this problem, we simply ignore any bytes that do not have a valid UTF-8 conversion. In order to develop a feature vector for each document in the tobacco corpus, we compute the Term Frequency Inverse Document Frequency measure for all the tokens in the corpus.

With regard to the ETD corpus, we compute 128 dimensional Doc2Vec vector embeddings for each document. These vectors are computed by training the Doc2Vec model

using the abstract of each document obtained from the metadata. These embeddings are then pre-processed as explained in Section 5.1.1.

Implementation details

We use *scikit-learn's* implementation of the full Expectation-Maximization version of the Lloyd's algorithm [30]. We present the results of clustering both the cleaned and uncleaned Tobacco corpora with 10 and 20 clusters, respectively. We also present the results of clustering the ETD corpus using K-Means clustering to generate 500 clusters. In all these experiments, we initialize the cluster centroids using *k-means++* [14], a method to initialize the centroids to be generally distant from each other. This has shown to give better results than random initialization and also helps with faster convergence of the algorithm. We use inertia – sum of squared distance of each point to its associated centroid – as the metric for measuring the quality of cluster assignments. We declare convergence either when the inertia is within a specified limit or if a stipulated number of iterations have been completed, whichever happens first. Additionally, we run the algorithm 5 times with different random seeds for initializing the cluster centroid and choose the one that results in the least inertia. Such multiple runs are computed using 4-5 parallel jobs on ECE Guacamole servers and the VT-CS compute cluster. The results of the K-Means iterations have been enumerated in the Evaluation Chapter (9).

5.1.3 Hierarchical Clustering

Aggomerative Clustering

Here, we cover the implementation details of Agglomerative Clustering using *scikit-learn's* API. In an attempt to solve a smaller problem, we first perform the clustering on a set of 200 documents in order to generate 10 clusters. We use the TFIDF values for these documents computed as described in Section 5.1.1. Each TFIDF vector is of size 5417298, which essentially is the vocabulary size of the corpus (here, we refer to the 4553 set of documents cleaned and given to us by the CMT team) [2]. Further, we perform Agglomerative Clustering on the entire ETD corpus with the Doc2Vec embeddings obtained from training a model on document abstracts as described in Section 5.1.1. The algorithm

[2]Here, the vector size does not represent the exact vocabulary size of the corpus. The colossal size of the vectors is a result of erroneous tokenization resulting from the multitude of garbage characters in the corpus.

is made to converge when 500 clusters have been built in a dendrogram fashion. We use Euclidean distance as the affinity measure between clusters along with Ward linkage for all experiments. The results are discussed in Section 5.1.3

5.1.4 BIRCH

We used *scikit-learn's* implementation of the BIRCH clustering algorithm to clusterize the 30961 abstracts of ETDs. As discussed in Section 4.1.1, Birch is designed to be memory and compute efficient. Thus, we apply Birch on the 30961 ETD document vectors obtained from the Doc2Vec algorithm. This algorithm has 2 main hyper-parameters to be tuned. First is the threshold which specifies the maximum radius allowed for a cluster formed by merging two sub-clusters. Intuitively, a low threshold promotes splitting of clusters and vice-versa [46]. We set this value to be 0.5. The other parameter of importance is the branching factor. This represents the number of sub-clusters a node can have. If the number of clusters exceeds the branching factor, a new node is formed. This value has been set to 50. The algorithm is run to generate 500 clusters from the data with an expected 60 documents per cluster.

5.1.5 DBSCAN

DBSCAN is a density based clustering technique designed to deal with large scale databases with noise. It can detect and discard noisy points, thereby keeping the clusters pure. This has an advantage for the ETDs since many OCR'ed documents contain garbage characters that could potentially harm the clustering. We report the results of clustering the ETD corpus using DBSCAN. The hyper-parameters of DBSCAN are *eps* and *min_samples*. *eps* is the maximum distance between two samples for one to be considered as in the neighborhood of the other [46]. *min_samples* is the number of samples (or total weight) in a neighborhood for a point to be considered as a core point. We perform a hyper-parameter selection process as described in the DBSCAN paper [20]. We create plots of k-dist (*eps*) for each document and select the point in the graph near the curvature as shown in Figure 5.2.

We refrain from covering the hyper-parameter selection method because it entails an in-depth coverage of the algorithmic nuances of DBSCAN as pre-requisites which is beyond the scope of this work. The technique is thoroughly explained in Section 4.2 of the DBSCAN paper [20].

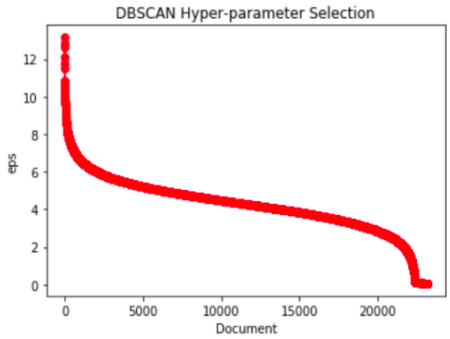

Figure 5.2: Hyper-paramter selection for DBSCAN

5.2 Text Summarization

5.2.1 Keyword Extraction

We implemented TF-IDF on the thesis and dissertation sample dataset. Data in Ceph is in directory: /mnt/ceph/cme/dissertation_subset & /mnt/ceph/cme/thesis_subset. After TF-IDF computation, we extracted the top ranked keywords from the vector so that we can compare the keywords extracted through the TF-IDF method with the keywords listed by the author.

Then we remove stopwords to eliminate some noise. Finally, we choose the top ten keywords of the document to compare with the keywords provided by authors of theses and dissertations. Two examples from the thesis and dissertation dataset are:

Example 1: Liu_L_T_2017.pdf.txt
Title: The effect of hypoxia and 3D culture conditions on heterogeneous ovarian cancer spheroids
Keywords: Ovarian cancer, metabolism, hypoxia, spheroids, stromal vascular fractions, invasiveness

Our findings:
There are 3223 different words.
Top ten keywords (without stopwords): the, and, of, cells, in, to, cancer, mose, cell, spheroids
Top ten keywords (with stopwords removed): cells, cancer, mose, spheroids, ovarian,

hypoxia, tumor, lactate, ten
Same keywords: Cancer, spheroids, ovarian, hypoxia

Example 2: Nabiyouni_M_D_2017.pdf.txt
Title: How Does Interaction Fidelity Influence User Experience in VR Locomotion?
Keywords: User Experience, 3D User Interfaces, Locomotion, Fidelity

Our findings:
There are 3795 different words.
Top ten keywords (without stopwords): the, and, of, to, in, for, fidelity, user, walking, al
Top ten keywords (with stopwords removed): fidelity, user, walking, al, techniques, locomotion, interaction, natural, technique, interfaces
Same keywords: fidelity, user, locomotion, interfaces

Based on these two sample results, we find that TF-IDF, the most basic technique for keyword extraction, is still quite useful for the electronic thesis & dissertation dataset. Stopwords should be eliminated before TF-IDF, because some articles, prepositions, and conjunctions appear more frequently than useful words. But they are meaningless for the document. Compared with keywords the author proposed, most of the keywords can be discovered through TF-IDF methods with stopwords removed.

5.2.2 Summarization on ETD dataset

We implement text summarization on both the Theses and Dissertations sample datasets. Data in Ceph is in directory: /mnt/ceph/cme/dissertation_subset & /mnt/ceph/cme/thesis_subset. The ETD dataset includes text documents of different chapters for each thesis or dissertation. We first get 20 sample documents which are manually cut into different chapters by the CME team.

We implement three kinds of models (feature-based, graph-based, and topic-based model) to extract key sentences from chapters in theses and dissertations. An example thesis is in Liu_L_T_2017.pdf.txt. Below is a comparison between the key sentences extracted based on the three different models, and the abstract, which includes the key sentences the author chose. Because there are no same sentences from the abstract and the content, we calculate the cosine similarity and find similar sentences.

(1) Epithelial ovarian cancer (EOC) is the leading cause of death from gynecological malignancy due to the insufficient accurate screening programs for the early detection of EOC. (2) To improve the accuracy of the early detection, there is a need to deeply understand the mechanism of EOC progression and the interaction between cancer cells with their unique microenvironment. (3) Therefore, this work investigated the metabolic shift in the mouse model for progressive ovarian cancer, and evaluated the effects of hypoxic environment, spheroid formation as well as stromal vascular fractions (SVF) on the metabolic shift, proliferation rate, drug resistance and protein markers in functional categories. (4) The results demonstrated an increasingly glycolytic nature of MOSE cells as they progress from a tumorigenic (MOSE-L) to a highly aggressive phenotype (MOSE-FFL), and also showed changes in metabolism during ovarian cancer spheroid formation with SVF under different oxygen levels. (5) More specifically, the hypoxic environment enhanced glycolytic shift by upregulating the glucose uptake and lactate secretion, and the spheroid formation affected the cellular metabolism by increasing the lactate secretion to acidify local environments, modulating the expression of cell adhesion molecules to enhance cell motility and spheroids disaggregation, and up-regulating invasiveness markers and stemness makers to promote ovarian cancer aggressive potential. (6) Hypoxia and spheroid formation decreased ovarian cancer cells growth but increased the chemoresistance, which leads to the promotion of aggressiveness and metastasis potential of ovarian cancer. (7) SVF co-cultured spheroids further increased the glycolytic shift of the heterogeneous of ovarian cancer spheroids, induced the aggressive phenotype by elevating the corresponding protein markers. (8) Decreasing the glycolytic shift and suppression of the proteins/pathways may be used to inhibit aggressiveness or metastatic potential of ovarian cancer heterogeneous of ovarian cancer spheroids, induced the aggressive phenotype by elevating the corresponding protein markers.

We set the threshold as 10. So, the 10 most important sentences can be extracted based on each model. A comparison of the results based on different models follows.

Feature-based model: Sentence (1), Sentence (5) and Sentence (8) are extracted based on Luhn's Algorithm.
Graph-based model: We first implement TextRank based on TF-IDF; sentence (7) and sentence (3) are extracted from the chapter. Though key sentences are extracted, they are

only ranked by their TF-IDF weight value. For text summarization, it is hard for human beings to understand the summary in different order than they appear in the original text. Therefore, we implement TextRank combined with NLTK to correct the ranking order. In this way, both the original text order and the weight value are considered to provide a summary. Then we implement gensim, a free Python library designed to automatically extract semantic topics from documents, to extract key sentences from text.
Topic-based model: Sentence (1) and sentence (5) are extracted from the chapter. From the results above, we found that sentence (1) and sentence (4) have a much higher possibility to be extracted.

It is hard to evaluate which models perform better than others since they all can extract sentences which are similar with the sentences in the abstract. One method to evaluate the performance of text summarization is for human-beings to manually score the summaries generated by the files. However, it is costly to manually evaluate the performance of text summarization. We calculate the cosine similarity to find similar sentences with the sentences in the abstract because we assume that the abstract is the summary of each file. The results above, based on three different models, are quite different. Therefore, we propose a new model which combines the three different models (feature-based, graph-based, and topic-based model).

New model: Sentence (1), sentence (4), sentence (5), and sentence (7) are extracted based on the new model.

5.2.3 Summarization on Tobacco Dataset

There are nearly 1 million text documents from the tobacco dataset on ceph. These documents include interviews (Questions and Answers), articles, emails and so on, and are provided and pre-processed by the CMT team.

Text summarization on interviews doesn't perform well. Due to the special format in interviews, questions and answers are too short to generate summaries. Further, sometimes there are few relationships between different questions or answers. It is difficult to rank the importance for each sentence since different questions are from different perspectives. Therefore, our extractive methods on the interview dataset do not generate good summaries. The sentences below show an unsuccessful summary as an example from the interview dataset. This summary is generated on document

ttdy0019.ocr.

A. I do not remember to have ever been directly involved in the treatment of Nathan Horton. 08 – from examining those records, tell us, if you know, what was the primary cause of Mr. Horton's death.

From the summary above, it is extremely hard for people to get a general idea of the whole interview. Another example shows that meaningless sentences in interviews may be extracted. The following summary of document *nplp0018.ocr* is generated by a topic-based model (Latent Semantic Analysis).

Q. khat is meant, if you recall, or what did the committee mean by extrapolation of evidence from animals to man can be defined as a subproblem? This is a letter, apparently, that I abstracted from Dr. Hockett, who had visited Dr. Fieser at my request, and then he wrote me about the meeting.

From the summary above, the second sentence includes no useful information about the interview. It seems likely that extractive methods of text summarization, without modification, can't successfully be implemented on interview format documents. Therefore, we try to do sentiment analysis on these interview documents to provide a positive or negative attitude of these questions and answers instead of providing a summary for the interview.

To provide a summary to the CMT team and make the whole system work in a short time, we first provide a fake summary (first ten lines of each document) for the tobacco (interview) dataset. Then we get the article documents of the tobacco dataset from the CMT team. We implement both the feature-based model (Luhn algorithm) and the topic-based model (Latent Semantic Analysis) on 38,038 article documents and provide the first version of summaries for the tobacco dataset. When the CMT team is generating and pre-processing the 1 million tobacco dataset on ceph, we develop a new model which combines the three traditional models and calculates the cosine similarity and then provides the final summary. The summaries for the tobacco dataset are at mntcephtmltext_summary directory on Ceph. The ELS team can successfully ingest and index these summaries. So summaries can be seen through demos provided by the FEK team.

5.3 Named-Entity Recognition

In order to compare different NER tools, we took a popularly used sample text and applied all of the three NER tools to it. The sample that we used is as follows:

The university was founded in 1885 by Leland and Jane Stanford in memory of their only child, Leland Stanford Jr., who had died of typhoid fever at age 15 the previous year. Stanford was a former Governor of California and U.S. Senator; he made his fortune as a railroad tycoon. The school admitted its first students on October 1, 1891,[2][3] as a coeducational and non-denominational institution.

As can be seen, the statement has several proper nouns, such as Leland, Stanford, California, etc. Some of these nouns have the same spelling; however, they appear in different contexts. Therefore, it is very important for an NER tool to identify the correct context of the words. Hence, we applied different NER tools on the sample statement to see their effectiveness. The results are discussed more thoroughly below.

5.3.1 Stanford NER

The Stanford NER tool was applied to the sample set. It identified entities as follows:

Type: PERSON, Value: Leland
Type: PERSON, Value: Jane
Type: PERSON, Value: Stanford
Type: PERSON, Value: Leland
Type: PERSON, Value: Stanford
Type: PERSON, Value: Jr.,
Type: ORGANIZATION, Value: Stanford
Type: LOCATION, Value: California
Type: LOCATION, Value: U.S.

As we can see from the result, the NER successfully identified Person entities and Organization entities even though some of them had the same spelling, such as *Stanford*. As for cons, the NER tool failed or ignored to identify the date entities.

5.3.2 NLTK NE_Chunk

We applied the NE_Chunk tool and found the following results:

(GPE Leland/NNP)
(PERSON Jane/NNP Stanford/NNP)
(GPE Leland/NNP)
Stanford/NNP
Jr./NNP
(PERSON Stanford/NNP)
Governor/NNP
(GPE California/NNP)
(GPE U.S/NNP)
Senator/NNP
October/NNP
]/NNP

The results show that the proper nouns have been parsed in chunks. However, in some cases, it failed to identify the correct context. For example, in our original sample text, the first "Leland" is the name of a person. However, the NER identified it as a GPE (geopolitical entity), which means a geographical area. This is incorrect in the context.

spaCy

Another state-of-the-art, open sourced, and industrially compatible NER tool is spaCy. We applied the spaCy tool on our sample text. The result is as follows:

Type: DATE, Value: 1885
Type: GPE, Value: Leland
Type: PERSON, Value: Jane Stanford
Type: PERSON, Value: Leland Stanford Jr.
Type: DATE, Value: age 15 the previous year
Type: ORG, Value: Stanford
Type: GPE, Value: California
Type: GPE, Value: U.S.
Type: ORDINAL, Value: first

Type: DATE, Value: October 1, 1891,[2][3

The result from the spaCy tool identifies more entities than its counterparts. It successfully identifies person name and organization name. It also groups the names together such as "Jane Stanford" and "Leland Stanford Jr.". It identifies dates as well. However, in one case spaCy identifies reference number as a part of date format, such as "October 1, 1891,[2][3".

Considering all the results, we find **spaCy** tool to be the most thorough, informative, and effective in identifying multiple entities properly.

From the non-processed data from both the ETD and the tobacco datasets, we manually pro-processed some sample data to run the spaCy tool. For example, we considered the following paragraph:
The witness, senior vice-president and controller at R. J. Reynolds Tobacco Holding Inc., was deposed by the plaintiffs. He described the financial status of the holding company and its economic activities. He indicated that industry changes, corporate changes, market changes, structural changes, and some legal developments have all had an adverse effect on the profitability of the company. The witness also noted that advertising and promotion restrictions placed on them in 1998 by the Master Settlement Agreement had caused a drop in sales volume. He said that punitive damage awards would have a devastating effect on the company, although he declined to say whether bankruptcy was being considered.

The spaCy tool correctly extracts the following named entities from the example article above.

Type: ORG, Value: R. J. Reynolds Tobacco Holding Inc. ✓
Type: DATE, Value: 1998 ✓
Type: LAW, Value: the Master Settlement Agreement ✓

Upon closer inspection, we find that the article is indeed about a deposition document where the witness is from the organization R. J. Reynolds Tobacco Holding Inc. The article further discusses about the drop in sales volume caused by the Master

Settlement Agreement law in 1998. Therefore, we can conclude that these named entities can be used to successfully explain the article.

For another example, we considered the following example paragraph.

The witness, Director of Marketing Research at Philip Morris, was deposed by the plaintiffs. He reviewed his previous depositions and trail testimony, as well as the contract work that he has done for Philip Morris. He explained that the contract work consisted of showing advertising or packaging and obtaining information on consumer reactions. He reviewed the organizational structure of the Marketing and Research department of Philip Morris. The witness listed the various companies from which Philip Morris obtained consumer information. He maintained that Philip Morris only conducted studies on people over the age of 18. He explained the importance of having highly reliable information about legal age smokers in order to accurately project future industry sales and brand sales. He described Philip Morris' use of publicly available information and studies on smoking behavior. He commented on surveys in which adults were asked about their age of smoking initiation.; Roper

The paragraph above is from yet another deposition document. This time, the spaCy tool finds more named entities from the example text. The extracted named entities are as follows:

Type: ORG, Value: Marketing Research ✓
Type: ORG, Value: Philip Morris ✓
Type: ORG, Value: Philip Morris ✓
Type: ORG, Value: the Marketing and Research ✓
Type: ORG, Value: Philip Morris ✓
Type: ORG, Value: Philip Morris ✓
Type: ORG, Value: Philip Morris ✓
Type: DATE, Value: the age of 18 ✓
Type: ORG, Value: Philip Morris' ✓
Type: PERSON, Value: Roper ✓

From the result above, we can see some interesting findings. The spaCy tool identifies

Philip Morris multiple times. Since these names all refer to the same organization, having it only once would suffice. Additionally, it also extracts the phrase **the age of 18** as a **value** named entity. This might be very helpful in putting documents in context.

It is observed that the spaCy tool provides better results for named entity recognition on the sample data. Therefore, we will be using spaCy on the entire ETD and tobacco datasets. Currently, we have written automation scripts for NER on a sample dataset consisting of multiple documents on our local machine. Further we will be executing this script on the entire pre-processed ETD and tobacco data on ceph.

5.4 Sentiment Analysis

5.4.1 Flair

We used pre-trained models and the Flair framework on our tobacco sample dataset. The Flair framework classified the documents into two categories, such as "positive" and "negative". The Flair framework also returned a confidence ratio with the category.

5.4.2 Twitter Emotion Recognition

The Twitter Emotion Recognition package is better for identifying single line statements. We used single statements and found good enough results. We categorized the statements into Ekman's six basic emotions. When a bigger paragraph is used the result is more generic.

5.4.3 Empath

The Empath framework has 200 built-in pre-validated categories. We used 6 categories among them that indicate Ekman's six basic emotions, i.e., anger, disgust, fear, happiness, sadness, and surprise. The framework returns classification probability along with confidence ratio. We also automated the package to return only non-zero confidence valued categories. Then we sorted the results, so the highest confidence valued category is listed first. Finally, we automated the script to systematically go through a number of documents. The result for each document is returned with the name of the document. More about our script can be found in the Developer Manual, Section 8.4.

5.5 Recommender System

We collected a sample dataset cosisting of real user logs from a Kaggle challenge [51] and implemented various recommendation techniques which will be described in detail in the following sections. Since the training of machine learning models requires large amounts of user data and we currently have only a few sample log files shared by the ELS and FEK teams, we decided to train and test the recommender models on a sample dataset. Later we will be training these models on the user logs obtained by searching ETD and tobacco documents in our search engine. We implemented content based recommendation and collaborative filter techniques; the results obtained are given below.

5.5.1 Dataset

We collected the user log dataset of Deskdrop, an internal communications platform developed by CI&T, which allows company employees to share relevant articles with their peers, and collaborate around them [51]. This dataset was obtained from a Kaggle competition; it contains a real sample of logs collected over 12 months during March 2016 to February 2017. It contains around 73k logged users' interactions on more than 3k public articles shared in the platform.

The dataset features the following fields:
1. Person ID - Unique user ID of the logged in users.
2. Session ID - Identifier of the user session.
3. Content ID - Unique identifier for articles.
4. Timestamp - Timestamp of the session when the article was accessed by the user.
5. User Agent - Web agent through which the article was accessed.
6. User Region - The state where the article was accessed.
7. User Country - The country where the article was accessed.

It contains 1140 different users and 2926 documents in total. We used 80% of the data for training and 20% for testing the accuracy of the model.

Since our user log data will also contain fields such as user ID, session ID, and document ID similar to this, we have selected this dataset as our sample for preliminary analysis.

We used recall as the performance metric for model evaluation. Recall indicates the ratio of actual positives that were identified correctly by the model.

5.5.2 Content based recommendation

We implemented content based recommendations on the above mentioned dataset. This method uses only information about the description and attributes of the items users have previously consumed, to model user's preferences. Various candidate items are compared with items previously rated by the user and the best-matching items are recommended. The item profiles of all the candidate items are built by computing the TF-IDF values. Then the user profile is built by building the item profiles of all the items the user has interacted with in the past. Cosine similarity is applied between the user profile and TF-IDF matrix to obtain similar items which are recommended to the user. The content based recommendation model is trained on the sample dataset and we obtained a recall of 41.4% for top 5 recommendations and 52.4% for the top 10 recommendations on the test data. The screenshots of the results are shown in Figures 5.3 and 5.4.

```
Evaluating Content-Based Filtering model...
1139 users processed

Global metrics:
{'modelName': 'Content-Based', 'recall@5': 0.41459984658655075, 'recall@10': 0.524162618256200
5}
```

Figure 5.3: Content based method results

5.5.3 Collaborative filtering based recommendation

The next method that we implemented was collaborative filtering. It uses its memory of users' interactions with documents to compute the similarities. We used the matrix factorization algorithm, which decomposes the user-item interaction matrix into lower

	_person_id	hits@10_count	hits@5_count	interacted_count	recall@10	recall@5
76	3609194402293569455	26	16	192	0.135417	0.083333
17	-2626634673110551643	35	21	134	0.261194	0.156716
16	-1032019229384696495	34	22	130	0.261538	0.169231
10	-1443636648652872475	54	34	117	0.461538	0.290598
82	-2979881261169775358	15	8	88	0.170455	0.090909
161	-3596626804281480007	23	14	80	0.287500	0.175000
65	1116121227607581999	15	10	73	0.205479	0.136986
81	692689608292948411	20	11	69	0.289855	0.159420
106	-9016528795238256703	10	5	69	0.144928	0.072464
52	3636910968448833585	11	4	68	0.161765	0.058824

Figure 5.4: Content based method recall result table for 10 selected users

dimensional matrices. This method handles the sparsity of the original matrix better than memory based ones.

We used Singular Value Decomposition (SVD), which is a popular latent factor model to implement collaborative filtering. This model compresses the user-item matrix into three different low dimensional matrices; the original matrix is reconstructed to get the missing scores. Evaluation of the model gave us recall of 33.4% for top 5 recommendations and 46.8% for top 10 recommendations. The screenshots of the results are shown in Figures 5.5 and 5.6.

```
Evaluating Collaborative Filtering (SVD Matrix Factorization) model...
1139 users processed

Global metrics:
{'modelName': 'Collaborative Filtering', 'recall@5': 0.3340577857325492
4, 'recall@10': 0.46816670928151366}
```

Figure 5.5: Matrix factorization results

	_person_id	hits@10_count	hits@5_count	interacted_count	recall@10	recall@5
76	3609194402293569455	45	21	192	0.234375	0.109375
17	-2626634673110551643	56	30	134	0.417910	0.223881
16	-1032019229384696495	34	16	130	0.261538	0.123077
10	-1443636648652872475	51	38	117	0.435897	0.324786
82	-2979881261169775358	48	39	88	0.545455	0.443182
161	-3596626804281480007	34	22	80	0.425000	0.275000
65	1116121227607581999	32	24	73	0.438356	0.328767
81	692689608292948411	21	16	69	0.304348	0.231884
106	-9016528795238256703	28	20	69	0.405797	0.289855
52	3636910968448833585	30	23	68	0.441176	0.338235

Figure 5.6: Matrix factorization recall result table for 10 selected users

5.5.4 Performance Comparison

We compared the output of both the models and observed that the content-based approach gave better performance. This happens because the content based model benefited by rich item attributes for better modeling of users' preferences. We plotted a bar chart for comparing the performance of content-based and collaborative models for recall@5 and recall@10, which can be seen in Figure 5.7.

The content-based recommendation method has already been through clustering. So we will be implementing collaborative filtering on the user logs in our project to produce user specific recommendations.

5.5.5 User Log Format

In order to provide recommendations based on the users' behavior, it is necessary to keep a record of the history of the users' interactions with the items. The web server stores information of all the logged in users such as username, session ID, timestamp, and the

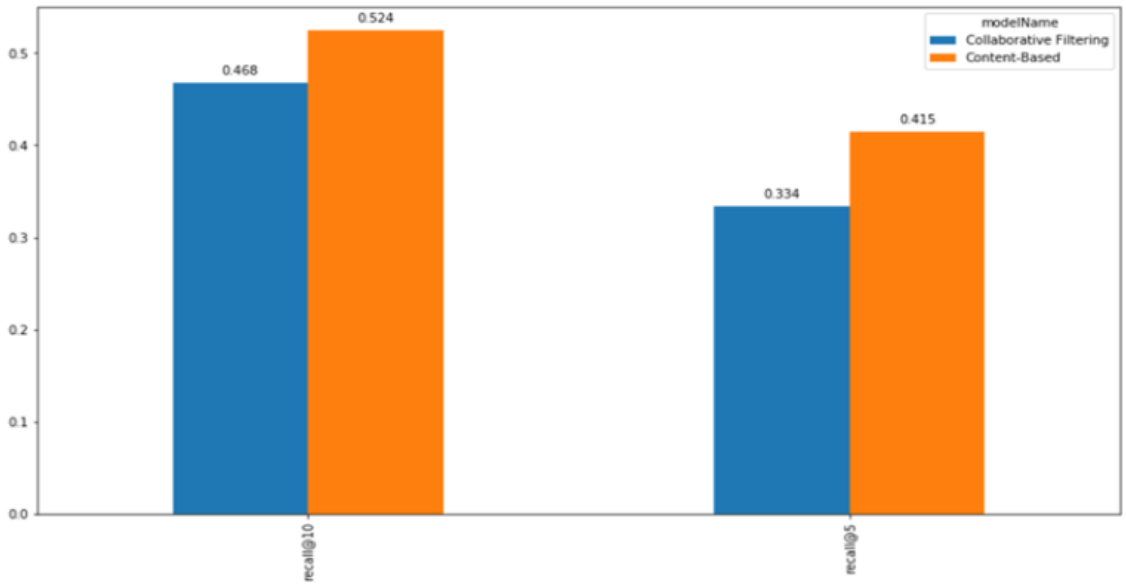

Figure 5.7: Performance comparison of content-based and collaborative filtering

list of documents that were viewed by the user, along with other fields. The combined logs obtained from Kibana and Elasticsearch will contain the data fields required by the recommendation models. Figure 5.8 illustrates the user log formats expected from the front-end and Elasticsearch teams. One way to provide input data to the recommender system is to keep track of the top five search results and check whether or not these documents were viewed by the user. This information will be obtained from front-end logs. Another way would be to keep track of all the search queries that the user has searched for. This can be obtained from the Elasticsearch logs. Once we have the prototype of the complete end-to-end search engine working, we will be able to obtain the required user logs which will enable us to train our collaborative filtering models to provide better recommendations.

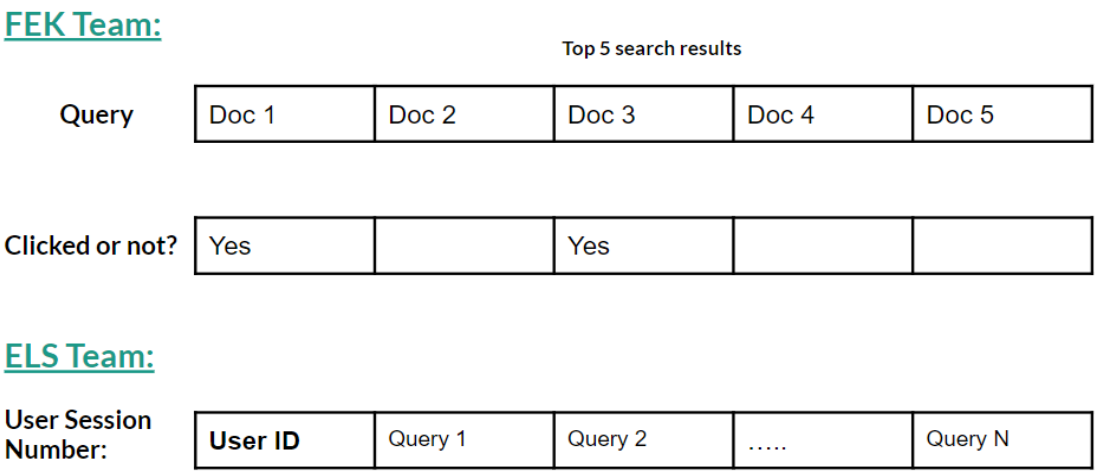

Figure 5.8: Input Data Format

Chapter 6

Project Roadmap

6.1 Team Milestones

Table 6.1 summarizes the TML teams milestones with the date of completion for each task. A more detailed sub-group wise treatment of the tasks is covered in Section 6.2.

No.	Milestone	Completion Date
1.	Understand the requirements of the TML team.	09/03
2.	Perform an initial literature review with a focus on breadth of techniques to explore.	09/08
3.	Discuss potential use of techniques explored with other teams. Narrow down on methods to work on.	09/12
4.	Study and understand both the ETD and tobacco datasets.	09/15
5.	Experiment with NER and Text Summarization with sample data.	09/18
6.	Work on report IR-1.	09/19
7.	Experiment with K-Means clustering, NER, Text Summarization.	10/04
8.	IR-1 review work	10/06
9.	Discuss user log format, obtain sample logs	10/06
10.	Collect sample dataset, do content based and collaborative filtering.	10/07
11.	Submit IR-2 report.	10/10
12.	Work on IR-2 review.	10/20
13.	Submit IR-3 report.	10/30
14.	Work on IR-3 review.	11/20
15.	Implement Large scale clustering algorithms	11/25
16.	Submit Final report.	12/11

Table 6.1: Team Milestones

6.2 Task Timeline

Tables 6.2, 6.3, 6.4, 6.5 and 6.6 describe the timeline of the tasks and goals of the Clustering, Text Summarization, Named Entity Recognition, Sentiment Analysis and the Recommender Systems sub-groups within the TML team along with the team members involved.

Task	Team Members	Date Complete
Review literature on clustering.	Adheesh, Prathamesh	09/14
Preprocess Tobacco data subset.	Adheesh	10/02
Write scripts to extract TFIDF vectors.	Adheesh, Prathamesh	10/06
Run K-Means on unclean/clean subset of tobacco records.	Prathamesh	10/08
Identify and code up the Calinski-Harasbasz Index evaluation.	Prathamesh	10/18
Identify the cause of cloud container problem.	Prathamesh	10/25
Explore Hierarchical Clustering.	Sharvari	10/30
Preliminary experiments with Agglomerative clustering	Sharvari	10/30
Scale hierarchical clustering to entire ETD and Tobacco corpora	Sharvari, Prathamesh	11/15
Compute Doc2Vec vectors for the ETD corpus	Prathamesh	11/20
Implement and evaluate Birch, DBSCAN and K-Means clustering on the ETD corpus	Prathamesh	12/03
Compile and visualize clustering result.	Sharvari, Prathamesh	12/08
Collate results on Ceph	Sharvari, Prathamesh	12/10

Table 6.2: Task Timeline for the Clustering Group

Task	Team Members	Date Complete
Review literature on text summarization.	Jiaying Gong	09/14
Keyword Extraction based on TFIDF	Jiaying Gong	09/16
TextRank combined with TF-IDF for key sentence extraction	Jiaying Gong	09/30
TextRank combined with NLTK including cleaning the sentences	Jiaying Gong	10/04
TextRank based on Gensim	Jiaying Gong	10/06
Latent Semantic Analysis	Jiaying Gong	10/07
Luhn's Algorithm base summarization	Jiaying Gong	10/07
Fake summary (first ten lines) for tobacco (interview) dataset	Jiaying Gong	10/19
Provide summaries based on LSA for tobacco (article) dataset.	Jiaying Gong	10/23
Provide summaries based on Luhn for tobacco (article) dataset.	Jiaying Gong	10/25
Provide summaries for sample ETD dataset.	Jiaying Gong	11/07
Provide summaries based on new model (first version) for sample tobacco dataset.	Jiaying Gong	11/19
Provide summaries based on new model (Second version) for sample tobacco dataset.	Jiaying Gong	11/29
Provide summaries based on new model (Second version) for 1 million tobacco dataset.	Jiaying Gong	12/04
Provide all summaries on Ceph to ELS team.	Jiaying Gong	12/09

Table 6.3: Task Timeline for the Text Summarization Group

Task	Team Members	Date Complete
Review literature on NER.	Rifat Sabbir Mansur	09/14
Meeting with Dr. Townsend and Bipasha	Rifat Sabbir Mansur	09/16
Understanding the data structure for ETD and tobacco datasets	Rifat Sabbir Mansur	09/24
Comparing between state-of-the-art NER methods	Rifat Sabbir Mansur	09/30
Identify the best NER method - spaCy.	Rifat Sabbir Mansur	10/01
Running sample dataset in spaCy	Rifat Sabbir Mansur	10/04
Identifying the problems in spaCy models	Rifat Sabbir Mansur	10/04
Identifying better spaCy models	Rifat Sabbir Mansur	10/07
Using domain specific spaCy models	Rifat Sabbir Mansur	10/22
NER scripts on sample dataset on local machine	Sandhya Bharadwaj	10/29/2019
Explore Blackstone and Graphbrain packages	Sandhya, Rifat Sabbir	11/14/2019
Automating NER to store results in text files	Sandhya	12/05
Executing NER scripts on the entire tobacco data	Rifat	12/10

Table 6.4: Task Timeline for the NER Group

Task	Team Members	Date Complete
Review literature on Sentiment Analysis.	Rifat Sabbir, Adheesh	10/29
Explore state-of-the-art frameworks.	Rifat Sabbir, Adheesh	10/29
Improve script to automate framework.	Rifat Sabbir, Adheesh	10/31
Meeting with Dr. Townsend	Rifat Sabbir, Adheesh	11/04

Table 6.5: Task Timeline for the Sentiment Analysis Group

Task	Team Members	Date Complete
Review literature on recommendation systems.	Sandhya, Sharvari	09/14
Finalize log fields, obtain sample logs from FEK and ELS.	Sharvari, Sandhya	09/24
Analysis of ample logs, discussions about the log format	Sharvari, Sandhya	10/01
Collect sample dataset. Finalize techniques to implement.	Sandhya, Sharvari	10/04
Pre-process sample data, implement content-based method.	Sandhya	10/07
Implement collaborative filtering, result comparison.	Sharvari	10/07
Collect and analyze sample logs from the working system	Sandhya, Sharvari	11/24
Illustrate extension of recommendation system for future scope	Sandhya, Sharvari	12/05

Table 6.6: Task Timeline for the Recommender Systems Group

Chapter 7

User Manual

7.1 Overview

The main purpose of the TML team is to improve the searching and browsing experience of the user. In order to prioritize better and relevant search results at the top, we will be using Recommender Systems. This will provide additional ranking on the result provided by Elasticsearch.

The following are the different fields that are displayed along with the document title in the search engine results. Several keywords from the document, such as relevant person, organization, date, location, etc. are displayed under the document title for the tobacco documents. This will help the user get an idea about the context of the document.

The user can view other documents that are are similar to the document displayed when he/she clicks on a "more like this" link which is present next to the document title for both the tobacco and ETD documents. Once clicked, the user is forwarded to another search result with all the documents that share similar topics and/or attributes.

The sentiment of the document is displayed along with the title for the tobacco documents. This field helps the user understand the context of the document.

Finally, for every search result there is a 2 line summary of the document that is appended along with its title for the ETD documents. This summary provides the user a

basic understanding of the content of the document giving him/her an idea of whether that particular document is relevant.

A sample search result list with the various fields can be seen in Figure 7.1.

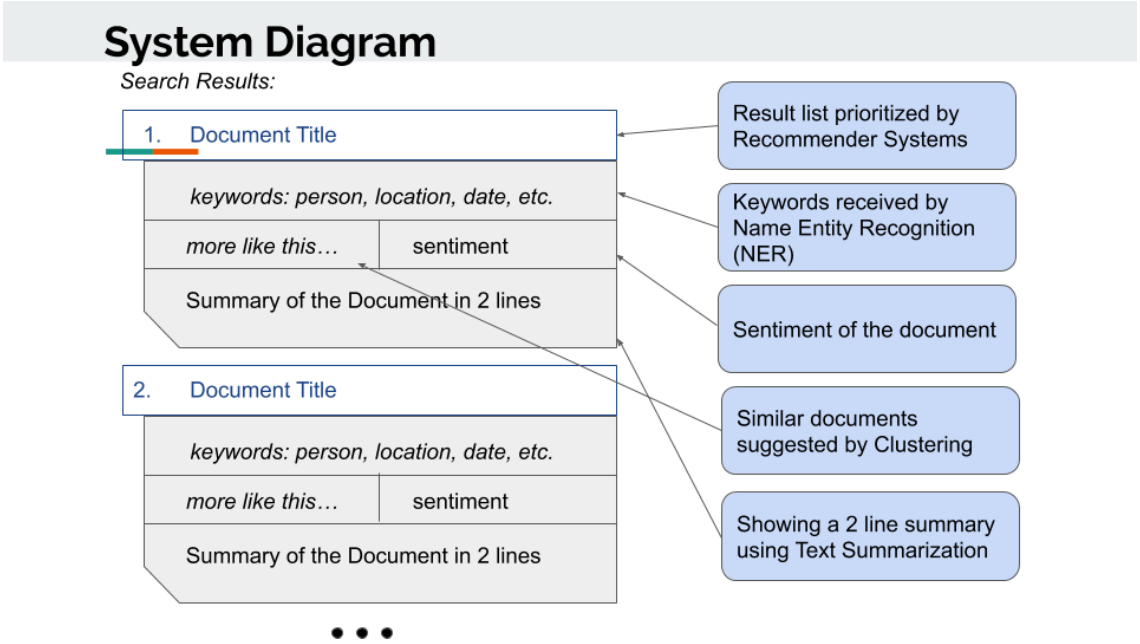

Figure 7.1: Searching and Browsing Experience Improvement done by the TML team

7.2 Text Summarization

The file version of finalmodel.py expects the users to input the path of all the files that they are using to generate summaries. The code automatically generates summaries for each file based on three different models. The output is a folder which contains all the summaries. Each file generates one summary with the same name (identifier) of the original file, which makes it easier for the ELS team to ingest summaries.

Usage:

(1) Change the path in the code to the directory of original files which are needed to generate summaries.

(2) Change the parameters (numbers of sentences to extract from each model) as the user needs.

(3) Set the threshold of summary length.

(4) Input **python finalmodel.py** in the command line.

7.3 Named entity recognition

The code for named recognition on the tobacco documents can be found in the GitHub repository https://github.com/rifatsm/CS5604_NER_on_tobacco_data.

The code for NER determination on the entire tobacco dataset has been automated, and is present in the Python notebook named *NER_Automation_JSON.ipynb*.

The user has to first install spaCy version 2.2.2 and the NER package *en_core_web_sm*.
The user has to place all the tobacco documents which NER needs to process in ceph at location *tobacco/mnt/ceph/shared/tobacco/data/1million_raw*.
The user can then run the Python code. The NER results generated will be stored in text files in a new folder named *NER* present in the same location as the Python script.

The names of the resulting text files containing the NER results are the same as the document name.

7.4 Sentiment Analysis

Github: https://github.com/rifatsm/empath_on_tobacco_documents/ [27]
This repository is used for doing sentiment analysis on deposition data in the tobacco dataset.

The file test_empath.py expects either a sentence or a directory (which contains all the text files) path for running. The code then executes empath on 8 of the attributes – hate, envy, love, joy, fear, surprise, positive emotion and negative emotion – to check which of the aforementioned sentiments are expressed in the text. It returns a list of scores for each of the sentiments.

Usage:

python test_empath.py [options]

The available options are as follows:

-s: Process on given string.
e.g., python test_empath.py -s "example sentence"

-d: Specify data directory path.
e.g., python test_empath.py -d "directory_path"

-t: Number of documents to be processed in the directory (use with option -d)
e.g., python test_empath.py -d "directory_path" -t n, where n is some integer

-h: To see help page.
e.g., python test_empath -h

Chapter 8

Developer's Manual

8.1 Clustering

The clustering codebase is located at the https URL in [35]. The source code files exist in *./src/* and the object files are stored in *./obj/etd/* for the ETDs and *./obj/tobacco/* for the Tobacco Settlement Records. Some of the *.sav* object files have not been bundled with the repository due to their large size. These can be found under the path */mnt/ceph/tml/clustering/cs5604-tml-clustering/*. The data corpora are stored in Ceph which is mounted at the path */mnt/ceph/*. The ETD corpus along with its metadata is located at */mnt/ceph/cme/* while the Tobacco Settlement documents can be found under */mnt/ceph/shared/tobacco/*. We now describe important files in the codebase along with instructions to replicate our experiments.

The source directory within the repository (*./src/*) consists of Python scripts along with IPython notebooks. The Python scripts consist of class definitions for various clustering algorithms with each algorithm bundled in a separate script for ease of usage. The IPython notebooks contain code for visualizing crucial results as well as for hyperparameter search experiments. The scripts *main.py* and *test.py* serve to run iterations of various experiments and perform unit tests on each algorithm before a full run, respectively. Below, some of the important files from the code base have been discussed along with their API. All of these files can be found under *./src/* in the repository [35]. Note that most of the scripts for the various clustering algorithms are self-explanatory and easy to use. As previously mentioned, each algorithm has been bundled into a separate script which makes it easy to run and distribute. Thus, we refrain from providing a verbose

description of each parameter of every script and class in the source directory (*./src/*). Details can be found in the README.md file in the root of the repository.

8.1.1 Python Scripts

kmeans.py

1. class TFIDF:

   A wrapper around *sklearn.feature_extraction.text.TfidfVectorizer* computes TFIDF vectors and stores them as *numpy* arrays in a serialized format (.sav) using *joblib*.

2. class Kmeans:

   A wrapper around *sklearn.cluster.KMeans* runs K-Means clustering as per the parameter instantiation and stores the cluster centroids and the document-to-cluster mappings in a serialized format (.sav) as *numpy* arrays.

birch.py

Class BIRCH is a wrapper around *sklearn.cluster.Birch*. It has two methods, *fit* and *save*, which provide the functionality to train the model initialized in the constructor and save it as a serialized object using *joblib*.

agglo_clus.py

Class Agglo_clus is a wrapper around *sklearn.cluster.AgglomerativeClustering*. Similar to Birch, this class has methods to train the model as well as save it in a serialized format.

dbscan.py

The DBSCAN class in this file is a wrapper around *sklearn.cluster.DBSCAN* with methods to fit the data and save the trained model.

pre_process.py

This script contains a wrapper around *gensim.models.doc2vec.Doc2Vec* for tokenizing and vectorizing the ETD and the tobacco corpora. The class Doc2vec_wrapper provides various methods in the form of Python generators for fetching the data. It also contains code

to tokenize the documents as specified in Section 5.1.1. This script also provides a helper function to extract document to vector mapping from a trained Doc2vec_wrapper object.

utils.py

This script contains common utility functions used at various stages of the training pipeline.

main.py

This script initializes the aforementioned classes from their respective scripts and runs them with pre-defined hyper-parameters as specified in the file. It is essentially the script used to trigger the training of the algorithms.

test.py

Before running *main.py*, this script performs quick tests by running the clustering algorithm for a few epochs with a small percentage of the entire data. It essentially serves as a unit test to the main algorithm.

8.1.2 IPython Notebooks

playground.ipynb

This file contains code for miscellaneous operations such as hyper-parameter tuning and cross-validation. Particulary, this file contains logs and metadata about computing optimal cluster count based on Silhouette Coefficient for Agglomerative Clustering, DBSCAN, and Birch, along with cross-validating to choose optimal *eps* parameter for DBSCAN.

etd_results.ipynb

This notebook visualizes cluster histograms for Agglomerative Clustering, DBSCAN, and Birch, along with computing the Calinski-Harabasz Index, the Silhouette Coefficient, and the Davies-Bouldin score for each of these algorithms.

These are the core scripts that form a part of the main results. We refrain from presenting the details of deprecated scripts and experiments. For these, the remaining

scripts in the *./src/* directory contain sufficient self-explanatory descriptions. The object files representing the trained models for each of the clustering algorithms along with the Doc2Vec vectors for the data corpora are stored in the *./obj/etd/* directory for the ETDs and the *./obj/tobacco/* directory for the tobacco data. Each algorithm has its own directory for each of the 2 corpora along with a sub-directory identifying details of the iteration number bundled with the hyper-parameters for that experiment. Much of the structure of the repository has been designed to be self-explanatory and easy to follow.

8.2 Text Summarization

For text summarization, we need clean chapters from the thesis and dissertation dataset and clean passages or articles from the tobacco dataset. The code for text summarization will be uploaded to ceph after there are useful datasets on it.

Here we describe the usage of different techniques in text summarization. Below is an overview of our file structure with a brief description.

1. **tfidftest.py**

   The script calculates the TF-IDF value of each word in separate chapters. It can calculate both the keywords with stopwords and keywords without stopwords. The top ten keywords will be recorded into a document.

2. **luhn.py**

   It is a feature-based model using the Luhn text summarizer algorithm, which is an automated tool to summarize a long text. The Sumy library, a Python library that can summarize text in several methods, is used in this model. It scores sentences based on frequency of the most important words.

3. **textrank1.py**

   This script aims to extract key sentence from a document based on the TfidfTransformer package. The similarity graph is calculated by normalized matrix multiply of the transpose normalized matrix. The scores are provided by PageRank from the networkx package. The results are the top key sentences extracted from the document without considering the original text order.

4. **textrank2.py**

This script is an improvement of textrank1.py. It uses the NLTK library in Python. The similarity graph is built by cosine distance of two sentences. The results are the key sentences in the same order as the order in the original text.

5. **gensimtest.py**

   It is a graph-based model using the gensim.summarization.summarizer library from Python. This module summarizes documents based on ranks of text sentences using a variant of the TextRank algorithm.

6. **lsa.py**

   It is a topic-based model using the sumy.summarizers.lsa library from Python. In this script, we choose to use a plaintext parser. Besides, the Tokenizer package is also imported in this model.

7. **finalmodel.py**

   It is our new proposed model, which combines the feature-based, graph-based, and topic-based models. After top sentences are extracted from different models, cosine similarity will be calculated to provide the final ranking result. It is a final version of our new model including pre-processing, combination of different models, and implementation of cosine similarity calculation.

8.3 Named Entity Recognition

The spaCy package is used for named entity recognition. The code for NER determination on the entire tobacco dataset has been automated and can be found in the Python notebook named *NER_Automation_JSON.ipynb*.
This Python notebook is present in the GitHub repository at https://github.com/rifatsm/CS5604_NER_on_tobacco_data.

This code requires spaCy version 2.2.2 and the NER package *en_core_web_sm* to be installed. The path where the tobacco documents are present needs to be mentioned in the *mypath* variable. The same script can be used when a new document is added into the system by changing the *mypath* variable to the location where this new document is located.

The key-value pairs of NER results obtained after executing this script are saved in text files inside a directory named *NER*. This directory is present in the same location where the *NER_Automation_JSON.ipynb* notebook is present.

8.4 Sentiment Analysis

Empath [22] is used for the sentiment analysis. Currently, we are still working with empath and the only file we have now is *test_empath.py*. The code now has different options to run such as: process a single sentence, process files in a directory, or process top n files in a directory. Setup and other requirements are mentioned in the Github repository [27]
In the code, we are using Python's empath library, which has the function analyze(). The analyze() function takes parameters such as input text, categories, etc. It returns the list of categories (e.g., joy, fear) with their sentiment score after analyzing the passed input text.

Chapter 9

Evaluation

9.1 Clustering

9.1.1 Metrics

It is not trivial to evaluate clustering algorithms when there is no knowledge of the underlying true labels for the data. Most measures such as mutual information among others rely on the ground truth labels for quantitatively evaluating cluster assignment. A challenge in evaluating cluster assignments is that there are no known cluster assignments to compute extrinsic measures for evaluation. Thus, we rely on intrinsic measures that give an idea of the compactness of a cluster assignment. Note that such evaluations merely give an idea of the structural quality of the clusters by considering such aspects as their overlap, density, and inter-cluster and intra-cluster distance. These aspects, however, cannot be used as proxies to evaluate whether or not the resulting cluster assignment is optimal or even relevant to the task at hand. Nevertheless, we consider the following metrics to evaluate the clustering algorithms. We use *scikit-learn's* [46] implementation of these metrics in our work. Note that we compute these metrics only for clustering entire corpora and do not consider them while applying the algorithms on subsets of data for both the ETD and tobacco corpora.

1. **Calinski-Harabasz Index (CHI)** [16]

   The Calinski-Harabasz Index, a.k.a the Variance Ratio Criterion, is the ratio of the sum of intra-clusters dispersion and of inter-cluster dispersion for all clusters [46]. Thus, a higher value of the index corresponds to better quality of clustering. The

CHI index is generally more relevant in case of convex cluster types as opposed to clusters resulting from such methods as DBSCAN.

2. **Silhouette Coefficient**

   The Silhouette Coefficient is a mean value based on two quantities for every point in the dataset. It is the ratio of the mean distance between a point and all other points in that cluster, and the mean distance between the point and all other points in the next nearest cluster. The value ranges between -1 and 1. A value of -1 represents bad clustering while 1 represents dense clustering. A value close to zero corresponds to overlapping clusters [46]. Among the other metrics discussed here, only the Silhouette Coefficient can help qualitatively evaluate a single clustering iteration. The remaining metrics can only be relatively evaluated since their absolute values do not carry any meaning.

3. **Davies-Bouldin Index**

   The Davies-Bouldin Index signifies the average similarity between clusters, where the similarity is a measure that compares the distance between clusters with the size of the clusters themselves [46]. Smaller values represent better separation between clusters. This metric is easier to compute than the Silhouette Coefficient.

9.1.2 K-Means Clustering

Uncleaned TSR Corpus

The uncleaned TSR corpus as provided by the CMT team consists of 7995 text files in the *.ocr* format. We perform K-Means clustering with 10 centroids. The hyper-parameter settings are given in Table 9.1. The results are tabulated in Table 9.2.

From the results, it can be observed that more than half of the number of documents in the cluster have been allocated to the same cluster (number 3). Also, the number of common tokens in the smallest cluster with 94 documents is a maximum (22386). We are trying to understand and interpret these results in the context of the clustering algorithm. As of now, we do not have a concrete understanding of exactly what is causing this imbalanced cluster assignment despite running the K-Means algorithm with multiple cluster initializations with different random seeds. We conjecture that the nature of the documents being primarily in a question-answer or dialogue based format might require some custom pre-processing before running the K-Means algorithm.

Parameter	Value
Number of centroids	10
Initialization method	*k-means++* [14]
Parallel Jobs	5
Number of random initializations	5
Loss/Metric	Inertia
Maximum Iterations	300

Table 9.1: Hyper-parameters for TSR K-Means

Cluster Number	Number of Documents	Number of tokens occurring in all documents in the cluster
1	94	223866
2	107	7915
3	4806	1754
4	283	4806
5	283	653
6	320	1193
7	340	4768
8	123	2327
9	529	1202
10	259	7238

Table 9.2: K-Means Results for Uncleaned TSRs

Cleaned TSR Corpus

The cleaned corpus provided by Team CMT consists of 4553 text files that contain only valid UTF-8 bytes. In order to cluster this corpus, we use the same set of parameters as given in Table 9.1. The results of the clustering are tabulated in Table 9.3.

As is evident from Table 9.3, the results for the cleaned corpus are not markedly different from the uncleaned ones. Here too, more than half of all documents are assigned to a single cluster (number 4). We are now working towards employing methods to choose the optimal number of clusters for the corpus.

Cluster Number	Number of Documents	Number of tokens occurring in all documents in the cluster
1	130	1651
2	149	317
3	502	2488
4	2809	2265
5	153	2289
6	91	99
7	182	970
8	199	6672
9	170	3984
10	168	10995

Table 9.3: K-Means Results for Cleaned TSRs

The ETD corpus

Here, we evaluate the K-Means clustering algorithm for the ETD corpus. We cluster the 128-d Doc2Vec vectors computed from the abstracts of 30961 ETDs into 500 clusters. The number of clusters was cross-validated over the Silhouette Index. Insignificant variations in the Silhouette score for clusters ranging from 350 through 800 were observed. Thus, 500 was chosen as the final cluster size since this results in a total of approximately 60 documents (in the expected value) in every cluster, which was found reasonable. The hyper-parameters for training the algorithm are the same as in Table 9.1. However, we compute 500 clusters with more parallel workers to speed up the training. We present a pointwise cluster histogram in Figure 9.1.

The average documents per cluster is 46.28, which is close to the uniform document to cluster allocation case with ~60 documents for each cluster. The standard deviation of the cluster sizes is 68.70. Further, the CHI, Silhouette Coefficient, and Davies-Boulden Score are compared and discussed with other clustering metrics in Section 9.1.1.

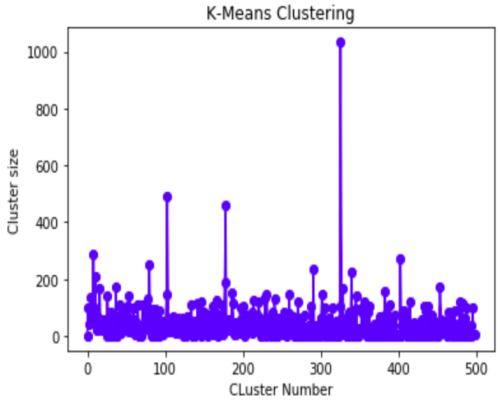

Figure 9.1: K-Means cluster size histogram

9.1.3 Hierarchical Agglomerative Clustering

Sample Tobacco Documents

To begin with, we performed Agglomerative Clustering as described in Section 5.1.3 on a set of 200 documents from the tobacco corpus. The results are enumerated in Table 9.4.

Cluster Number	Number of Documents
1	13
2	5
3	107
4	7
5	3
6	7
7	45
8	3
9	7
10	3

Table 9.4: Agglomerative Clustering on 200 documents

It can be observed that the nature of the clusters is very similar to the case of the K-Means algorithm with the TSRs. Of the 200 documents, 107 of them have been as-

signed to the 3rd cluster while 45 belong to the 7th cluster. In an attempt to probe the reason behind the unbalanced nature of the clusters, we explored the TFIDF vectors for these documents. We found that the vectors for all documents are extremely sparse in nature with ~1% values being non-zero. From this, we conjecture the following. Firstly, we strongly believe that sparsity is the prime reason behind the unbalanced cluster allocations. Thus, we switch over to custom sized (and essentially smaller) embeddings for each document based on *Doc2Vec* (Section 5.1.1). Another solution that we considered but did not find worth implementing was to reduce the dimensionality of the vectors by retaining only the principle components using the PCA technique. Future work based on ours can attempt this endeavor to see if better results can be obtained. Secondly, we consider the possibility that the sparsity could be caused by the inexact and raw pre-processing that we have employed. For this, we work with pre-processed and tokenized versions of the documents for subsequent iterations.

The ETD corpus

Similar to Section 9.1.2, we apply Agglomerative Clustering to the Doc2Vec embeddings obtained from the abstracts from all of the 30961 documents from the ETD corpus. We employ a Ward based linkage along with the Euclidean distance measure for generating 500 clusters. The point-wise cluster histogram is shown in Figure 9.2.

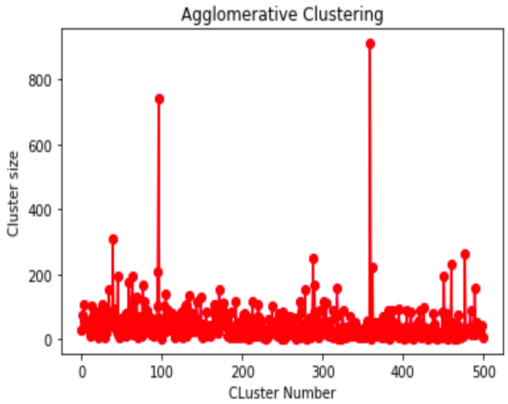

Figure 9.2: Agglomerative Clustering - cluster size histogram

The average number of documents per cluster is 46.28 and the standard deviation is 64.87.

9.1.4 BIRCH

As discussed in Section 4.1.1, Birch is a clustering algorithm designed to be memory and compute efficient when dealing with large scale databases. It works by iterating through the given corpus once and developing an initial cluster assignment and then (optionally) performing further iterations to improve the initial assignment. We apply Birch only to large scale corpora and not to the initial sample data subsets. There are two crucial hyper-parameters involved in Birch, the threshold and the branching factor. The branching factor limits the number of sub-clusters in a node, while the threshold limits the distance between the entering sample and the existing sub-clusters [46]. We set the branching factor and threshold to 50 and 0.5, respectively. We do not perform hyper-parameter selection for Birch as we find that the defaults work reasonably well. The point-wise cluster histogram for Birch with the ETD corpus is shown in Figure 9.3.

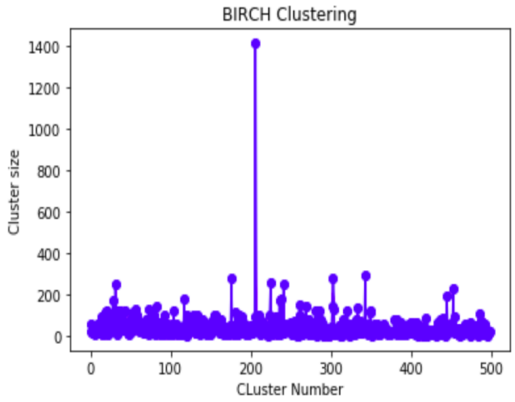

Figure 9.3: Birch Clustering - cluster size histogram

The average cluster size for Birch with the ETD corpus is 46.28 and the standard deviation is 74.61.

9.1.5 DBSCAN

We apply DBSCAN to the ETD corpus on the *Doc2Vec* embeddings obtained from the abstracts of the 30961 ETDs. DBSCAN is a density based clustering algorithm designed for large scale spatial databases with noise. Particularly, it has very few corpus specific hyper-parameters to be tuned, which makes it easy to adopt for a myriad of massive data corpora without extensive mining. Section 4.2 of the DBSCAN paper [20] describes a

graphical technique to choose the hyper-parameters *eps* and *minPts*. We perform a hyper-parameter search for *eps* for values of *minPts* ranging from 3 through 10. In each case, only minor differences are observed with respect to the curvature point in the graph. As a sample, Figure 5.2 from Chapter 5 shows the plot of *eps* for each document. However, we do not obtain satisfactory results with DBSCAN for clustering the ETDs based on abstracts. The clustering is trivially wrong as all documents are allocated to the noisy clustering. It should be noted that DBSCAN can detect noisy documents. This means that it will only allocate documents to a cluster if the resulting cluster dimensions and distances are within a threshold (*eps*). In our case, DBSCAN assigns all clusters to the noise class and fails to perform the clustering successfully.

9.1.6 Results Summary and Discussion

The ETD corpus

Table 9.5 notes the key metrics for the clustering algorithms used to cluster the ETD corpus.

	K-Means	**Agglomerative Clustering**	**Birch**
Calinski-Harabasz Index	26.637	25.153	25.069
Silhouette Coefficient	-0.070	-0.082	-0.074
Davies-Bouldin Score	2.985	3.422	3.447
Standard Deviation of Cluster Size	68.700	64.874	74.611

Table 9.5: ETD Clustering Metrics

Here, we provide a comparative overview between different algorithms based on the metrics in Table 9.5. Before proceeding with the same, we briefly mention our efforts towards hyper-parameter selection by cross-validation. We cross-validated the number of clusters for each of the 3 algorithms mentioned in Table 9.5 against the Silhouette Coefficient. However, on varying the cluster counts from 300 through 800 with an interval of 50 points, we did not observe any significant variation in the Silhouette Coefficient for any of the 3 algorithms. The range of values of the Silhouette Coefficient we obtained for each of these algorithms was between -0.06 to -0.09. As mentioned in Section 9.1.1, a Silhouette Coefficient value near zero indicates dense and overlapping clusters while a

value closer to 1 indicates highly disparate and non-overlapping clusters. Thus, within a significant range of clusters, we do not observe any appreciable variation in either the density or the distribution of the clusters. Thus, we refrain from mentioning the details of these cross-validation experiments here. The interested reader can find these experiments at the HTTPS URL in [36] under the heading 'Compute Optimal cluster count based on Silhouette Coefficient.'

Considering the CH index and the Davies-Boulden (DB) score, insignificant differences are observed across the 3 clustering algorithms in consideration. Thus, we conclude that all 3 algorithms, viz., Birch, K-Means clustering, as well as Agglommerative clustering, albeit following vastly different ways to cluster the data perform similarly on the ETD corpus. The final parameter we consider for evaluation is the standard deviation in the cluster sizes. Here, we observe that Agglomerative clustering has the least standard deviation among the 3 algorithms with a value of 64.874 documents. Further, Birch tends to have a standard deviation value on the higher side. K-Means, on the other hand, has a reasonable cluster size standard deviation value of 68.7 that lies between the other 2 algorithms. Intuitively, a lower standard deviation would mean a uniform distribution of documents across the clusters. Thus, we seek to employ a clustering algorithm that has a reasonable number of documents per cluster (as dictated by the application) while having a mid-range value of standard deviation in cluster sizes. With this consideration, the K-Means algorithm tends to perform reasonably well (in the expected value sense) across the range of algorithms we explored.

9.2 NER

9.2.1 Pre-Trained Models

The Name Entity Recognition tool, spaCy, has a number of pre-trained models in several different languages, such as English, German, French, Multi-language, etc. [4]. Since both of our datasets are in the English language, we used models pre-trained with Convolutional Neural Network (CNN) on OntoNotes [3] as mentioned in Table 9.6.

We found the models in Table 9.6 perform effectively with the tobacco dataset. However, it performed not very poorly with the ETD sample dateset. We evaluated the performance by precision and recall. The models had very low precision and recall. In Figure 9.4, we can see the named entities that are extracted by the models **en**, **en_core_web_sm**, and **en_core_web_md**. As we can see, these named-entities have precision of under 35%.

Number	Model Name	Model Description
1	en_core_web_sm	English multi-task **CNN** trained on OntoNotes
2	en_core_web_md	English multi-task **CNN** trained on OntoNotes with GloVe vectors trained on Common Crawl
3	en_core_web_lg	English multi-task **CNN** trained on OntoNotes with GloVe vectors trained on Common Crawl
4	en_vectors_web_lg	English multi-task **CNN** trained on blogs, news, comments
5	en_trf_bertbaseuncased_lg	pretrained transformer model published by **Google Research** using **HuggingFace's** transformers
6	en_trf_robertabase_lg	pretrained transformer model published by **Facebook** using **HuggingFace's** transformers
7	en_trf_distilbertbaseuncased_lg	pretrained transformer model published by **HuggingFace**
8	en_trf_xlnetbasecased_lg	pretrained transformer model published by **CMU** and **Google Brain** using **HuggingFace's** transformer

Table 9.6: spaCy Pre-Trained English Models

A possible reason behind this might be in the pre-trained models. Since the models that we used were pre-trained on OntoNotes, blogs, news, and/or comments, they are biased away from scientific articles. These models are especially good to extract named entities that are geolocation, organization, date, person, etc. At the same time, they are not very appropriate for extracting entities that are technical terms in the scientific domain. This may explain why they perform somewhat well with the tobacco dataset, as they are mostly identifying organization, person, date, etc. However, it would be more accurate to use a model that is more aware of legal terminology. Therefore, we plan to use pre-trained models with more specific domain knowledge.

The best approach would be to create a custom model from scratch by training on a sam-

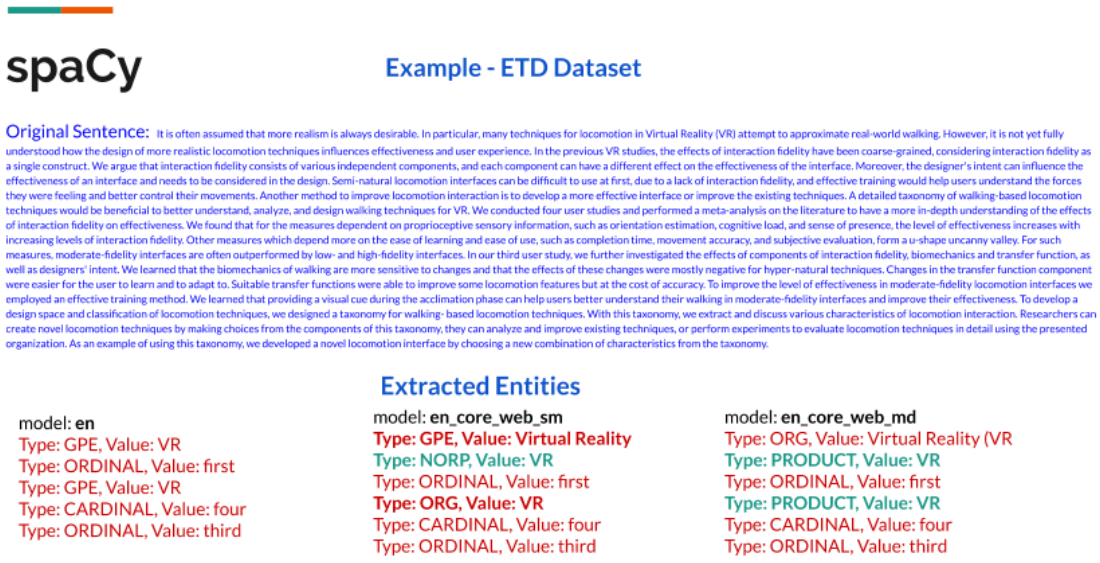

Figure 9.4: Name Entities Extracted by spaCy Models on ETD Sample Dataset

ple dataset. However, we don't have the whole dataset available to us and by the time we will, we might not have enough time to train and test the model. Therefore, we consider this approach to be outside the scope of our project. Another feasible solution would be to use domain-specific pre-trained models that exist. Since we have two different types of dataset, we have considered to use two different spaCy models as found in Table 9.7.

Package Name	Description
ScispaCy [43]	Python package containing spaCy models for processing biomedical, scientific or clinical text
Blackstone [1]	spaCy model and library for NLP on long-form, unstructured legal text

Table 9.7: New spaCy Packages and Models to consider

For the ETD dataset, we will be using the **ScispaCy** package [43]. ScispaCy is a Python based package that has pre-trained models trained on scientific documents, especially on biomedical and clinical texts. This is the closest open sourced spaCy pre-trained model we have at the time of this project. Another feasible open source solution for the tobacco

dataset is the **Blackstone** model [1]. Blackstone is a spaCy model that was pre-trained on UK legal-informatics documents in an experimental research project.

9.2.2 Automation of NER

As described in the previous section, the spaCy package provides the best results for named entity recognition. Therefore, we will be using spaCy for NER determination on the entire tobacco dataset.

We have written a Python script for automating this process of recognizing named entities among the 1 million tobacco articles. The NER results of of each tobacco document which are in the form of key-value pairs are stored in a text file. The name of the text file is the same as the document on which NER is determined. The generated text files are stored in the directory located at */mnt/ceph/tml/ner* on ceph which will be used by the the ELS team.
Initially a sample subset of the cleaned tobacco documents present in ceph are downloaded on a local machine and the automation has been performed. Later the same procedure is performed by executing the same script on a container on ceph for named entity recognition on all the tobacco documents.

An example to illustrate this process is shown below. Consider a sample tobacco document obtained named *jtvf0005* which is obtained from ceph. The content of this document is as follows:

Attendance at PR meeting September 10, 1958; James P. Richards Robert K. He jnann 0. D. Campbell 6ene L. Cooper W. S. Cutchins Margaret Carson H. C. Robinson Jr. Dan Provost James C. Bowling Jokm Scott Fones Rex Lardner John Jones Richard W. Darrow Carl C. Thompson Leonard S. Zatm Kenneth L. Austin D7alco'-m Jo'nnson W. T. Hoyt The Tobacco Institute, Inc, The Amer:can Tobacco Company Braun and Company, Inc. Braun and Company, Inc. Brown and Williamson Tobacco Corp. M. Carson, Inc. Liggett Myers Tobacco Company D:cCann-Erickson, Inc. Philip Morris, Inc. Benjamin Sonnenberg Sidney J. Wain, Inc. Sidney J. Wain, Inc. Hill end Hnowlton, Inc. F.111 and Knowlton, Inc. Hill and Knowlton, Inc. Hil'_ and Knowlton, Inc. Hi1= and Kaowlton, Inc.

Tobacco Industry Research Committee

The results obtained after running the NER script on this document is saved in a text file which has the same name as the document, that is, *jtvf0005.txt*. The *jtvf0005.txt* file contains key-value pairs of NER as follows:

[('DATE', September 10, 1958), ('PERSON', James P. Richards), ('PERSON', Robert K.), ('NORP', D.), ('ORG', Campbell), ('PERSON', James C. Bowling), ('PERSON', John Jones Richard W. Darrow), ('PERSON', Carl C. Thompson), ('ORG', The Tobacco Institute), ('PERSON', Inc), ('ORG', The Amer:can Tobacco Company Braun and Company), ('PERSON', M. Carson), ('ORG', Liggett Myers Tobacco Company D), ('ORG', cCann-Erickson), ('ORG', Philip Morris), ('PERSON', Benjamin Sonnenberg Sidney J. Wain), ('PERSON', Sidney J. Wain), ('ORG', Inc. Hill end), ('GPE', Hnowlton), ('ORG', Inc. F.111), ('GPE', Knowlton), ('ORG', Inc. Hill), ('GPE', Knowlton), ('ORG', Inc. Hil'), ('WORK_OF_ART', _ and Knowlton, Inc. Hi1=), ('GPE', Kaowlton)]

This text file,*jtvf0005.txt*, is saved in the */mnt/ceph/tml/ner* directory on ceph and provided to the ELS team.

The same script can be used for determining NER when a new file is added to the system. The path of the file needs to be changed according to the location where the new file is saved on ceph and the same script can be executed.

9.3 Recommender System - Future Scope

The implementation of content-based and collaborative filtering recommendation techniques have been implemented a sample dataset as described in Section 5.5.

Clustering, a form of content-based recommendation system, has been implemented on both ETD and tobacco dataset as discussed in Section 9.1.

Collaborative filtering based recommendation requires a large number of user search logs in order to produce accurate recommendations to the user. Our complete search

engine with completely integrated front end, Kibana, and Elasticsearch was not available until the last few weeks of the deadline of this course project. We were provided only 1-2 sample user logs by the FEK and ELS team.
Due to time constraints on this course project, we were not able to collect a large number of user logs, which are required in the collaborative filtering technique. Hence, the implementation of collaborative filtering based techniques which provides recommendations based on the users' previous search history has been performed on a sample dataset obtained by Deskdrop [51]. This dataset comes from real users with 73k user interactions and 2926 documents, and has fields similar to our search logs.

This approach can be extended to provide user-specific recommendations to the search engine users during future system enhancements.

The steps required to extend this technique to our system have been described below.

The required fields for recommender systems present in the sample dataset, and the the fields currently present in the logs from the FEK and ELS teams, can be seen in Figure 9.5.

The logs obtained from the FEK and ELS teams are missing some fields that are important in knowing whether or not the document was viewed by the user. Adding a new field called 'Document Viewed' would be helpful in further processing the data and to run the collaborative filtering model and improve the recommendations based on user preferences.

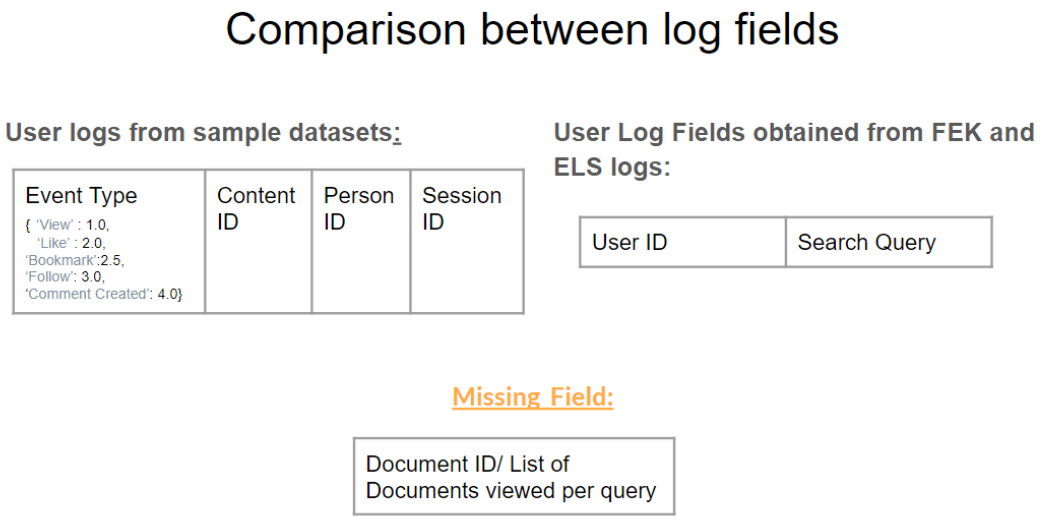

Figure 9.5: Comparison between sample dataset log fields and FEK, ELS log fields

Chapter 10

Acknowledgements

While working on our project, we took the help and guidance of some respected persons, who deserve our deepest gratitude. The completion of this project gives us much pleasure. To begin with, we would like to thank our instructor Dr. Edward A. Fox, for giving us a good road-map and vision for the project through numerous consultations, suggestions, and feedback. We would also like to extend our gratitude to the Teaching Assistant, Ziqian Song, who helped us throughout the semester with her valuable advice as well as with setting up our workflows across various containers and VMs.

In addition, we would like to thank Dr. David Townsend, Assistant Professor at the Pampling School of Business at Virginia Tech, who helped us with understanding the tobacco data set and giving us valuable suggestions towards what could be done as the Text Analytics and Machine Learning team in the project.

Many people, especially our classmates and the other teams (CME, CMT, ELS, FEK and INT), who all contributed towards the completion of this project, gave us valuable suggestions which inspired us to constantly improve our work. Last, but not the least, we are grateful to the Department of Computer Science at Virginia Tech for providing us with the compute resources along with an excellent cloud infrastructure that facilitated a seamless development and deployment workflow throughout the course. Moreover, it helped us get an idea of how software is used and deployed in large scale distributed systems.

We thanks IMLS for its support of the ETD-related work through grant LG-37-19-0078-19.

Finally, we thank anyone who we might have failed to mention above for their direct or indirect help towards the completion of this project.

Bibliography

[1] Blackstone: A spaCy pipeline and model for NLP on unstructured legal text. https://github.com/ICLRandD/Blackstone. (Accessed on 10/10/2019).

[2] DBpedia Lookup | DBpedia. https://wiki.dbpedia.org/lookup. (Accessed on 09/19/2019).

[3] English · spaCy Models Documentation. https://spacy.io/models/en. (Accessed on 10/10/2019).

[4] Models · spaCy Models Documentation. https://spacy.io/models. (Accessed on 10/10/2019).

[5] Named-Entity Recognition - Wikipedia. https://en.wikipedia.org/wiki/Named-entity_recognition. (Accessed on 09/19/2019).

[6] nltk.chunk package - NLTK 3.4.5 documentation. https://www.nltk.org/api/nltk.chunk.html. (Accessed on 09/19/2019).

[7] SKLearn Stopwords. https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/feature_extraction/stop_words.py. (Accessed on 10/09/2019).

[8] spaCy: Industrial-strength Natural Language Processing (NLP) with Python and Cython. https://github.com/explosion/spaCy. (Accessed on 09/19/2019).

[9] Stanford CoreNLP - Natural language software | Stanford CoreNLP. https://stanfordnlp.github.io/CoreNLP/. (Accessed on 09/19/2019).

[10] The Stanford Natural Language Processing Group. https://nlp.stanford.edu/software/CRF-NER.html. (Accessed on 09/19/2019).

[11] BBC Blogs - Technology & Creativity Blog - BBC News Lab: Linked data. https://www.bbc.co.uk/blogs/internet/entries/63841314-c3c6-33d2-a7b8-f58ca040a65b, January 2013. (Accessed on 09/19/2019).

[12] Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., and Vollgraf, R. FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)* (2019), pp. 54–59.

[13] Akkasi, A., Varoğlu, E., and Dimililer, N. Chemtok: a new rule based tokenizer for chemical named entity recognition. *BioMed research international 2016* (2016).

[14] Arthur, D., and Vassilvitskii, S. K-means++: the advantages of careful seeding. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms* (2007).

[15] Berger, A., and Lafferty, J. Information Retrieval As Statistical Translation. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 1999), SIGIR '99, ACM, pp. 222–229.

[16] Caliński, T., and Harabasz, J. A dendrite method for cluster analysis. *Communications in Statistics 3*, 1 (1974), 1–27.

[17] Colneriĉ, N., and Demsar, J. Emotion recognition on Twitter: Comparative study and training a unison model. *IEEE Transactions on Affective Computing* (2018).

[18] Cover, T. M., and Thomas, J. A. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006.

[19] Deng, H. Recommender Systems in Practice. https://towardsdatascience.com/recommender-systems-in-practice-cef9033bb23a, September 2019. (Accessed on 09/19/2019).

[20] Ester, M., peter Kriegel, H., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, pp. 226–231.

[21] Fast, E., Chen, B., and Bernstein, M. S. Empath: Understanding topic signals in large-scale text. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (2016), ACM, pp. 4647–4657.

[22] Fast, E., Chen, B., and Bernstein, M. S. Empath: Understanding Topic Signals in Large-Scale Text. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2016), CHI '16, ACM, pp. 4647–4657.

[23] Firth, J. R. A synopsis of linguistic theory 1930-55. 1–32.

[24] Harris, Z. S. Distributional structure. *Word 10*, 2-3 (1954), 146–162.

[25] Hiemstra, D. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, 1 2001. Imported from HMI.

[26] Joshi, P. An Introduction to Text Summarization using the TextRank Algorithm. https://www.analyticsvidhya.com/blog/2018/11/introduction-text-summarization-textrank-python/, November 2018. (Accessed on 10/19/2019).

[27] Juvekar, A., and Mansur, R. CS5604 empath on tobacco document - Github. https://github.com/rifatsm/empath_on_tobacco_documents., December 2019. (Accessed on December 29, 2019).

[28] Lafferty, J., and Zhai, C. Document Language Models, Query Models, and Risk Minimization for Information Retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2001), SIGIR '01, ACM, pp. 111–119.

[29] Le, Q., and Mikolov, T. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32* (2014), ICML'14, JMLR.org, pp. II–1188–II–1196.

[30] Lloyd, S. P. Least squares quantization in PCM. *IEEE Transactions on Information Theory 28* (1982), 129–137.

[31] Loper, E., and Bird, S. The PunktSentenseTokenizer. https://www.nltk.org/api/nltk.tokenize.html#nltk.tokenize.punkt.PunktSentenceTokenizer. (Accessed on 10/09/2019).

[32] Loper, E., and Bird, S. The TreebankWordTokenizer. https://www.nltk.org/api/nltk.tokenize.html#nltk.tokenize.treebank.TreebankWordTokenizer. (Accessed on 10/09/2019).

[33] Luhn, H. P. A business intelligence system. *IBM Journal of Research and Development 2*, 4 (Oct 1958), 314–319.

[34] Malik, U. Hierarchical Clustering with Python and Scikit-Learn. https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/, July 2018. (Accessed on 10/30/2019).

[35] Mandke, P. CS5604 TML Clustering Codebase - Github. https://github.com/pkmandke/cs5604-tml-clustering, December 2019. (Accessed on December 29, 2019).

[36] Mandke, P. CS5604 TML Clustering Cross-Validation - Github. https://github.com/pkmandke/cs5604-tml-clustering/blob/master/src/playground.ipynb, December 2019. (Accessed on December 29, 2019).

[37] Manning, C. D., Raghavan, P., and Schutze, H. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[38] Mihalcea, R., and Tarau, P. TextRank: Bringing Order into Text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing* (Barcelona, Spain, July 2004), Association for Computational Linguistics, pp. 404–411.

[39] Mitra, B., and Craswell, N. Neural Models for Information Retrieval. *CoRR abs/1705.01509* (2017).

[40] N. R. Kasture, Neha Yargal, N. N. S. N. K., and Mathur, V. A Survey on Methods of Abstractive Text Summarization. In *International Journal for Research in Emerging Science and Technology, Volume-1, Issue-6, November-2014* (2014).

[41] Navlani, A. Latent Semantic Analysis using Python. https://www.datacamp.com/community/tutorials/discovering-hidden-topics-python, October 2018. (Accessed on 10/21/2019).

[42] Neumann, M., King, D., Beltagy, I., and Ammar, W. ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing. *arXiv preprint arXiv:1902.07669* (2019).

[43] Neumann, M., King, D., Beltagy, I., and Ammar, W. ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing.

[44] Oliphant, T. E. *A guide to NumPy, Chapter 16*, vol. 1. Trelgol Publishing USA, 2006.

[45] Ozsoy, M., Alpaslan, F., and Cicekli, I. Text summarization using Latent Semantic Analysis. *J. Information Science 37* (08 2011), 405–417.

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

[47] Ponte, J. M., and Croft, W. B. A Language Modeling Approach to Information Retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 1998), SIGIR '98, ACM, pp. 275–281.

[48] Popat, S. K., and Emmanuel, M. Review and comparative study of clustering techniques. *International journal of computer science and information technologies 5*, 1 (2014), 805–812.

[49] Porter, M. F. Readings in Information Retrieval. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, ch. An Algorithm for Suffix Stripping, pp. 313–316.

[50] Robertson, S., and Zaragoza, H. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr. 3*, 4 (Apr. 2009), 333–389.

[51] Techlabs, M. What are Product Recommendation Engines? And the various versions of them? https://towardsdatascience.com/what-are-product-recommendation-engines-and-the-various-versions-of-them-9dcab4ee26d5, September 2017. (Accessed on 09/19/2019).

[52] Thakkar, P., Varma, K., Ukani, V., Mankad, S., and Tanwar, S. Combining User-Based and Item-Based Collaborative Filtering Using Machine Learning. S. C. Satapathy and A. Joshi, Eds., Springer Singapore.

[53] Wang, Z., and Zhang, Y. A Neural Model for Joint Event Detection and Summarization. In *Proceedings of the Twenty-Sixth International Joint Conference on Articial Intelligence (IJCAI-17)* (2017).

[54] Zhang, S., Yao, L., Sun, A., and Tay, Y. Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Comput. Surv. 52*, 1 (Feb. 2019).

[55] Zhang, T., Ramakrishnan, R., and Livny, M. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *SIGMOD Rec. 25*, 2 (June 1996), 103–114.