Final Report
CS 5604: Information Storage and Retrieval

Elasticsearch Team
Yuan Li
Satvik Chekuri
Tianrui Hu
Soumya Arvind Kumar
Nicholas Gill

January 6, 2020

Instructed by Professor Edward A. Fox
Assisted by Ziqian Song

Virginia Polytechnic Institute and State University
Blacksburg, VA 24061
Abstract

In this project, we are building an Information Storage and Retrieval System that works as a search engine to support searching, ranking, browsing, and recommendations for two large collections of data.

The first collection is part of Virginia Tech’s collection of Electronic Theses and Dissertations (ETDs). The Virginia Tech Library has an extensive collection of ETDs. Currently, Virginia Tech is moving towards digitizing the pre-1997 theses and dissertations and loading them into VTechWorks. This dataset contains thirty thousand (30K) ETDs. The second collection is the tobacco settlement documents. These documents are of several different types, such as depositions, articles, letters, conference proceedings, reports, etc. In total, there are fourteen million (14M) documents in this dataset. Thus, the goal is to build a state-of-the-art information retrieval and analysis system for these two important collections.

To achieve this goal, the class is divided into six teams: Collection Management ETDs, Collection Management Tobacco Settlement Documents, Elasticsearch, Front-end and Kibana, Integration and Implementation, and Text Analytics and Machine Learning. We are using ceph, a distributed storage system, to share data among the teams for storing and retrieving information.

This report addresses the work performed by the Elasticsearch (ELS) team. The ELS team helps to enable searching and browsing by ingesting the 30K ETDs and 14M tobacco settlement documents into Elasticsearch. As the team name suggests, we use Elasticsearch, which is a document-oriented search engine designed to store, retrieve, and manage document-oriented or semi-structured data. Using Elasticsearch, we not only ingest the metadata and full-text data from the data sets, but also incorporate the data from the TML team related to text summarization, clustering information, named-entity recognition, and sentiment analysis. Therefore, the ingestion of the data is supported based on facets associated with information extracted from documents, analysis, classification, clustering, summarization, and other processing. Furthermore, for in-depth search inside each document, we have implemented nested queries. Unit testing also has been incorporated to facilitate work of the Integration team in the future. Lastly, we have developed code for automatically ingesting and updating scripts to monitor a designated directory on ceph for new incoming files.

The report covers goals, overview, and the process of implementation with Elasticsearch. The Elasticsearch team works closely with the other subteams from CS 5604. The data ingested in Elasticsearch is consumed by the FEK team for information visualization and data retrieval on the webpage. The report also describes the connections established with the other groups, as a high-level overview of the course project. The user manuals have been provided for the reference of other groups.
## Contents

- Abstract ........................................... 2
- List of Tables ...................................... 5
- List of Figures ...................................... 6
- 1 Overview ........................................ 7  
  1.1 Challenges Faced ................................. 7
- 2 Literature Review ................................. 9
- 3 Requirements .................................. 10
- 4 Design ........................................... 11  
  4.1 Approach ........................................ 11
  4.2 Tools ........................................... 13  
  4.2.1 Elasticsearch .................................. 14
  4.3 Data Schema .................................... 14
  4.4 Deliverables ..................................... 17
  4.5 Quality Control ................................ 20
  4.6 Configuration .................................. 20
- 5 Implementation .................................. 21  
  5.1 Overview ....................................... 21
  5.2 Tasks and Schedule .............................. 21
  5.3 Search Implementation .......................... 24
  5.4 Expected Outcomes .............................. 26
  5.5 Achievements ................................... 26
- 6 User Manual ..................................... 29  
  6.1 Kibana .......................................... 29
  6.2 Searching an Elasticsearch Index ............... 32  
  6.2.1 Basic Searching .............................. 32
  6.2.2 Searching in Nested Field ................... 37
  6.2.3 Date Range Searching ......................... 38
- 7 Developer Manual ................................ 40  
  7.1 Tutorial ........................................ 40  
  7.1.1 Connecting to Elasticsearch by terminal ... 40
  7.1.2 Importing Data .............................. 41
  7.1.3 Kibana API ................................... 42
  7.1.4 Python API .................................. 42
  7.1.5 Indexing ..................................... 43
  7.2 Indexing/Configuration .......................... 45
  7.2.1 Fields ....................................... 45
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Elasticsearch data types</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>Tobacco settlements data schema</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>ETD data schema</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Data schema of the fields added in ETD and tobacco datasets for the data generated by the TML team</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>ETD dataset fields used for searching and filtering</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>Tobacco settlements dataset fields used for searching and filtering</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>Tasks and Schedule</td>
<td>21</td>
</tr>
<tr>
<td>8</td>
<td>Expected Outcomes</td>
<td>26</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Workflow of the project</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Overall architecture of the search engine and the role of Elasticsearch in it</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Planned search implementation for tobacco settlements dataset</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>Planned search implementation for ETD dataset</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>Searching single document</td>
<td>33</td>
</tr>
<tr>
<td>6</td>
<td>Searching</td>
<td>33</td>
</tr>
<tr>
<td>7</td>
<td>Mapping</td>
<td>34</td>
</tr>
<tr>
<td>8</td>
<td>Search Hamlet</td>
<td>35</td>
</tr>
<tr>
<td>9</td>
<td>Aggregate</td>
<td>36</td>
</tr>
<tr>
<td>10</td>
<td>Term vectors</td>
<td>37</td>
</tr>
<tr>
<td>11</td>
<td>Connecting</td>
<td>40</td>
</tr>
<tr>
<td>12</td>
<td>Listing indices</td>
<td>41</td>
</tr>
<tr>
<td>13</td>
<td>Ceph</td>
<td>41</td>
</tr>
<tr>
<td>14</td>
<td>Kibana indices</td>
<td>42</td>
</tr>
<tr>
<td>15</td>
<td>Python results</td>
<td>43</td>
</tr>
<tr>
<td>16</td>
<td>Create index</td>
<td>44</td>
</tr>
<tr>
<td>17</td>
<td>Reindex</td>
<td>45</td>
</tr>
<tr>
<td>18</td>
<td>Lifecycle policy</td>
<td>48</td>
</tr>
<tr>
<td>19</td>
<td>Python script for recommendation system using clusterID</td>
<td>54</td>
</tr>
<tr>
<td>20</td>
<td>Result of the executed Python script for recommendation system</td>
<td>54</td>
</tr>
</tbody>
</table>
1 Overview

In this project, we are building a state-of-the-art information retrieval and analysis system that can support two important content collections, i.e., (1) at least thirty thousand (30K) Electronic Theses and Dissertations (ETDs), and (2) fourteen million (14M) tobacco settlement documents. The goal of the Elasticsearch (ELS) team in this project is to ingest the document data into Elasticsearch to perform indexing for easy retrieval and full-text search of the ETD and tobacco settlement data.

To achieve this goal, we perform a detailed study of the data and present the appropriate schema designs for the associated datasets. The data received from the Collection Management ETDs (CME) team and Collection Management tobacco settlement documents (CMT) team are analyzed to decide the indexing pattern. We also coordinate with the Front-End Kibana development (FEK) team to understand their requirements for the data input query. The main challenges at this step include collecting the data in the desired JSON format from the CME and CMT teams and then indexing the data as per the requirement of the FEK team. For ingesting data into Elasticsearch, we study the Elasticsearch documentation and related tutorials thoroughly. We test the Shakespeare sample dataset on the cloud to learn Elasticsearch commands and study the indexing mechanism. To establish the connection with Kibana, we changed the Elasticsearch configuration files to support ReactiveSearch, a tool used by the FEK team.

Furthermore, the metadata and full-text data for the tobacco settlement documents and ETDs have been ingested into Elasticsearch. According to the statistics, the ingestion operation has achieved a success rate of 99.8% of the ETDs and 99.9% of the tobacco settlement documents. Apart from the high success rate, we have incorporated the expected data format generated by the text and machine learning (TML) team. We have discussed with the TML team about clustering results and using them to improve the rankings and recommendations in searches. We have also discussed about the interaction of the Elasticsearch, front-end, and TML teams to support recommendation on the dashboard. This will help us to analyze the habits of different users and create a more personalized search for the user. In the future we plan to provide user-specific logs and index logs to support user logging and recommendation.

Finally, we have implemented a working prototype that uses Elasticsearch in conjunction with Kibana to search the metadata and full-text data of the documents given by the CME and CMT teams. Figure 1 shows the entire workflow of the project for a better understanding.

1.1 Challenges Faced

Here we provide a summary of the challenges we faced throughout the project.

- The primary challenge we faced in our implementation is deciding what fields should be made searchable. After reviewing the user demands and preferences, we decided on a set of searchable fields for both the ETD and the tobacco datasets. This information is also relayed to the FEK team to help them in their website structure.

- Determining how to display the data according to the user preferences is not easy. The anticipated user preference options include ordering the results based on the date or relevance search score. Thus, we performed research to understand which fields should be used for filtering and which ones should be used for searching.
To implement the searching and sorting functionalities, we have to choose to declare our fields as either text or keyword. A keyword field can only be used for filtering, whereas a text field can only be used for searching. We had several issues determining field types as, at places, the requirement is to have a field that is both filterable and searchable. To resolve this, we created an inner object that affords both functionalities.

The next challenge we faced is in regards to updating the existing fields in Elasticsearch. We wrote a script to update the fields as per the requirements from the TML, CMT, and CME teams.

Elasticsearch lists results with one of its default rankings, but this may not be the desired setting for every field. To fix this, we use the boost functionality in Elasticsearch to provide custom weightings for different fields. The adjusted search scores of related documents can help present tailored search results that best meet a user’s information needs. Specifically, while searching, boost can add more weights to fields we choose so that Elasticsearch pays more attention to the important fields and pays less attention to those fields which are not as important.

The TML team requested us to update the default score field in Elasticsearch. The major challenge is that Elasticsearch computes scores using TF-IDF weighting by default. We did not know if it was possible to overwrite the default settings until we figured out a method to calculate a custom similarity score.

![Figure 1: Workflow of the project](image-url)
2 Literature Review

Manning et al. [10] introduces us to the foundation of information storage and retrieval systems. In their textbook, Chapters 4 and 5 give us an idea about index construction and compression. Zhai and Massung [14] discuss the implementation of search engines in depth. From the Elasticsearch website [1], we learned from the documentation that Elasticsearch is an open-source analytics engine that can facilitate several use cases. Elasticsearch uses Apache Lucene [12] to create the inverted index. These well-written materials helped us to understand the creation of an inverted index using Elasticsearch, as well as the stop-word list concept and single-pass in-memory indexing.

Shaik [11] describes how Elasticsearch is related to Lucene and how it is imperative to know about the Apache Lucene library to understand how Elasticsearch works. The paper compares Elasticsearch, Solr [13], and Lucene in detail to analyze their features and use cases. The extensive features of Elasticsearch are highlighted such as that it is scalable to megabytes of structured and unstructured data and can be used as an alternative to MongoDB. We concluded that Elasticsearch can power extremely fast searches that will be beneficial for the search engine we are trying to build around the ETDs and tobacco settlement documents.

From the Apache Log4j2 website [3], we learned the use case of Elasticsearch in log analytics to load log data into Apache Log4j2 on the Elastic Stack. This supports full-text searching and distributed document storage, among others, which will help us in our future implementations.

Kılıç et al. [7] and Luburic et al. [9] provided a detailed comparison of Solr and Elasticsearch. They concluded with the following differences:

- Indexing/Searching: Solr is text-oriented whereas Elasticsearch is keyword-centric and is also inverted-indexed which results in better performance.

- Scalability and Clustering: Solr provides Solrcloud (a cluster of Solr servers that combines fault tolerance and high availability [4]) but Elasticsearch has better inherent scalability and is designed for the cloud.

- Query execution: Solr currently has limited capability in this aspect, but Elasticsearch implements faster range queries depending on the context.

- Multi-tenancy: Elasticsearch uses denormalization to improve search performance and handles multi-tenancy very easily when compared to Solr.

Additionally, the project reports from previous years from teams that worked on Solr [8, 6] were also helpful when comparing the two search engines. We not only focused on the differences, but also discovered the similarities, that will help us in our implementation of Elasticsearch. We realize that Solr and Elasticsearch are very similar and there is even an Elasticsearch plugin that allows using Solr clients/tools with Elasticsearch.
3 Requirements

The following list covers our envisioned requirements for the ELS team:

1. Create data schemas for the ETD metadata as well as its entire text to index into Elastic-search with the CME team.

2. Create data schemas for the tobacco settlement document text and its metadata to index into Elasticsearch with the CMT team.

3. Incorporate additional metadata from the TML team, including text summarization, named-entity recognition (NER), sentiment analysis, and clustering information to improve recommendations, by updating indexed ETD and tobacco settlement data.

4. Help establish a connection with Kibana for the visualization of information and aid in creation of a front-end for searching and generating logs with the FEK team.

5. Implement a customized ranking function based on the clustering information from the TML team to improve search results.

6. Keep track of search results from previous user searches and generate the log files for the TML team.

7. Implement a recommendation function which will use data from the TML team such as clustering and topic information.

8. Provide feedback to the CME and CMT teams to improve data schema design. This includes removal of less accessed data fields, changing field data types, etc.

9. Modify configuration values to achieve optimized performance. For example, the logged data index is constantly growing and updating, the ETD dataset will gradually grow to 200K or even over 5M, whereas tobacco settlement data is rather static. Therefore, we should specify different Index Lifecycle Management (ILM) for the different datasets.

10. Explore various data types in Elasticsearch for improved searching and filtering and provide HTTP query examples to the FEK team.

11. Use boosting on search results for the ETD and tobacco settlement data to assign higher weights to more important fields, improving search results for the users.

12. Implement nested queries to enable searching inside documents.

13. Implement automatic ingesting and updating scripts that can monitor a designated directory on ceph for new incoming data files.

14. Implement unit testing scripts for ingesting and updating.
4 Design

4.1 Approach

The schema for Elasticsearch is set into 3 parts: metadata, text data, and the data generated by the TML team. The ETDs (stored in MongoDB) and tobacco settlement documents (stored in MySQL) have both metadata and data, but no TML data. Metadata stores the details of the records, and that describes and gives information about the source data and the TML generated data. The data part stores the text content of the ETD and tobacco settlement datasets (page-wise). Data generated by the TML team consists of clusterID, text summary, sentiment analysis, and NER keywords. The following points describe in detail the approach of our design for the metadata and data.

- The pre-processed text and metadata is delivered in JSON formats. The JSON files shared by the CMT team have been provided with a unique ID field. However, the JSON files shared by the CME team do not have a unique ID field. So, a Python script is written to generate the unique ID on the go while ingesting. For the tobacco dataset, the file names are considered as a unique ID. For the ETD dataset, the field value of identifier-uri is being processed as a unique ID by the Python script. Our team has ingested the data from both teams in Elasticsearch as “tobacco” and “30k” indices for tobacco settlements and ETD data sets respectively. Text and metadata will be stored in Elasticsearch as different schemas, as listed below in Table 2 and Table 3. Dependent on the inputs from the FEK team, we applied different indexing techniques on desired fields to assist searching, filtering, and recommendation.

- Initially, all available data fields will be indexed. Based on the feedback from the TML team, improvements can be made by eliminating less important attributes. The FEK team utilizes the indexed text and metadata to help visualize the information retrieved from the ETD and tobacco settlements documents. Also, facet names, field types, and search queries are provided to the FEK team for searching, filtering, and recommending on the ETD and tobacco datasets. Starting with the default settings, we will explore different configuration values to optimize effectiveness and usability.

- Nested search functionality, which is used in tobacco settlement documents, is used for searching the data in the tobacco index where the data is stored page-wise in a nested object. Since the data in tobacco settlements are large, just by returning the resultant record details when a user searches would not be helpful for the user to locate matching terms in that chunk of data. The nested search functionality will help in this scenario by returning the page number of a search term hit, along with the count of hits.

- The indexing of the ETD and tobacco settlement datasets is done using a Python script that uses the BULK API and INDEX API from Elasticsearch. The script also logs the errors we get while indexing, which helps us to fix them. The BULK API is used to index large chunks of data and we are using it for tobacco settlement datasets primarily due to the size and number of records (14M). The Python script also allows parsing and transforming files into the desired format for ingesting. This gives us the capability to fix a few of the data format
issues. Additionally, we are using this script to assign an ID and the name of the index for ETD datasets when ingesting into Elasticsearch.

- Index Lifecycle Management: The indices created in Elasticsearch should be managed properly over time. Depending on its nature, each index is handled differently. The four stages in the lifecycle are cold, warm, hot, and delete. The tobacco index for tobacco settlement documents and the 30k index for the ETD documents are put in the warm stage as long as possible. As there will be frequent addition of records to both these indexes in the future they are assigned to the warm stage. The index used for logs is put in the hot stage as the frequency of updating or inserting records into the index is almost daily.

- User activity logs will be generated by the FEK team, and Elasticsearch will index the log data along with additional information such as the top 5-10 results of the search query. The indexed log data along with the search results will be forwarded to the TML team to help with the recommendation function.

- The TML team provided the clusterID, text summary, NER keywords, and sentiment analysis result data by making them available on ceph. This data will be appended to the tobacco and 30k index using the UPDATE API in Elasticsearch by invoking a Python script written specifically for this purpose. Table 4 displays the fields which are created for accommodating the data generated by the TML team. The fields with this data are shared with the FEK team to pull them up on their platform for visualization and searching functionality.

- With the clusterID received from the TML team, recommendations are provided when a user searches for a term. A Python script is written to facilitate this. The script gets the clusterID of the top hit from the search result and queries the index for the records with the same clusterID, and these results can be displayed on the front-end. The summary field, which consists of the results of the text summarization, is given a boost using the boosting functionality available in Elasticsearch in addition to the abstract field. This would assign an extra weight to the field when displaying the search results.

- In the future, we will need to ingest new ETD and tobacco settlement documents. While this could be done manually, one of our goals was to create a procedure that could automatically ingest new documents into Elasticsearch. We have created a shell script for that purpose. The shell script uses a tool called inotify to watch a directory for new files either moved into or created in the directory. First, the CME and the CMT teams will parse and format the new documents for us to ingest. The teams will move the files into specific directories for us. The shell script will call our ingesting script for the correct document type. There will be two directories, one for each of the document types. We can also extend this shell script to an update script for the fields used by the TML team.
Figure 2: Overall architecture of the search engine and the role of Elasticsearch in it

Figure 2 displays the architecture of the project with detailed focus on the role of Elasticsearch in this project. It also depicts the flow of data to-and-from Elasticsearch.

4.2 Tools

The tools that we will be incorporating into our project, can be summarised as follows:

- Elastic Stack: A set of open source tools for data ingestion, enrichment, storage, analysis, and visualization:
  - Elasticsearch (v 7.4.0): It is a distributed, open source search and analytics engine.
  - Kibana (v 7.4.0): It lets you visualize data in Elasticsearch and navigate the Elastic Stack to do anything from tracking query load to understanding the way requests flow through your applications.
  - Apache Log4j 2: It is a Java-based logging utility.

- Container Teaching Cluster (cloud.cs.vt.edu): It is a departmental resource that provides the ability to run Linux containers. It uses Rancher, Kubernetes, and Docker.

- Apache Lucene: It is a free and open-source search engine software library.
4.2.1 Elasticsearch

Elasticsearch is one of the most popular search engines available today, which is used for searching and analysis. This search and the analytic engine is built on Apache Lucene and also helps in storing data. The ingested data is indexed by Elasticsearch. It uses an inverted file index which is based on a keyword-centric data structure. That is the reason it is able to retrieve the search results quickly; it searches through the index instead of the text. It uses denormalization to improve search performance and handles multi-tenancy very easily as compared to Apache Solr. Table 1 displays the available data types in Elasticsearch.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Elasticsearch type</th>
<th>SQL</th>
<th>SQL type</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>null</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
<td>BOOLEAN</td>
<td></td>
</tr>
<tr>
<td>byte</td>
<td>byte</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>short</td>
<td>short</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>integer</td>
<td>integer</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>long</td>
<td>long</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>double</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>keyword</td>
<td>keyword</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>text</td>
<td>text</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>binary</td>
<td>binary</td>
<td>VARBINARY</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>datetime</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>ip</td>
<td>ip</td>
<td>VARCHAR</td>
<td></td>
</tr>
</tbody>
</table>

4.3 Data Schema

Tables 2 and 3 display the data schema for the tobacco settlement and ETD datasets, respectively. The first column in both tables represents the name of the fields. The second column represents the data type of those fields in their respective databases, i.e., MongoDB and MySQL. The third column represents the data type of these columns after ingesting into Elasticsearch. For all the text/string variant fields we are using both the ‘Keyword’ data type and the ‘Text’ data type which is clubbed into a nested data type where the Text data type of the field is used for searching and the Keyword data type of the field is used for filtering, sorting, and aggregations. To access them, you can use ‘field_name’ for text data type variant and ‘field_name.keyword’ for keyword data type variant.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Elasticsearch Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>URL/String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Adverseruling</td>
<td>String-(Alphanumeric)</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Area</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>attending</td>
<td>List&lt;String&gt;</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Author</td>
<td>List&lt;String&gt;</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>availability</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Bates</td>
<td>ID-Alphanumeric</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Batesalternate</td>
<td>ID-Alphanumeric</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>batesmaster</td>
<td>List&lt;Id&gt;-Alphanumeric</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Box</td>
<td>Number</td>
<td>Numeric</td>
</tr>
<tr>
<td>Brand</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Case</td>
<td>ID-Alphanumeric</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Cited</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Collection</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Copied</td>
<td>List&lt;String&gt;</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Country</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Dateaddedindustry</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Dateaddeducsf</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Datemodifiedindustry</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Datemodifiededucsf</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Dateprivilegelogged</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Dateproduced</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Dateshipped</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Depositiondate</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Description</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Documentdate</td>
<td>Date</td>
<td>Date</td>
</tr>
<tr>
<td>Exhibitnumber</td>
<td>Number</td>
<td>Numeric</td>
</tr>
<tr>
<td>Expresswaiver</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>File</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Format</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Genre</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Grant</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>ID</td>
<td>ID-Alphanumeric</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Language</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Mentioned</td>
<td>List&lt;String&gt;</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Metadata</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Minnesotarequestnumber</td>
<td>ID-Alphanumeric</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Organization</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Othernumber</td>
<td>ID-Alphanumeric</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Otherrequest</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Pages</td>
<td>Number</td>
<td>Numeric</td>
</tr>
<tr>
<td>Person</td>
<td>List&lt;String&gt;</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Privilegecode</td>
<td>Id-Alphanumeric</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Recipient</td>
<td>List&lt;String&gt;</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Recommend</td>
<td>List&lt;string&gt;</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Referenceddocument</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Elasticsearch Data Type</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>dc.contributor.author</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.date.accessioned</td>
<td>Date/Time</td>
<td>Date</td>
</tr>
<tr>
<td>dc.date.available</td>
<td>Date/Time</td>
<td>Date</td>
</tr>
<tr>
<td>dc.date.issued</td>
<td>Date/Time</td>
<td>Date</td>
</tr>
<tr>
<td>dc.identifier.other</td>
<td>String-(Alphanumeric)</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.identifier.uri</td>
<td>URL</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.description.abstract</td>
<td>String-(Alphanumeric)</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.format.medium</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.publisher</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.rights</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.subject</td>
<td>List&lt;String&gt;</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.title</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.type</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.contributor.department</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.description.degree</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>thesis.degree.name</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>thesis.degree.level</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>thesis.degree.grantor</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>thesis.degree.discipline</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.contributor.committeechair</td>
<td>String</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>dc.contributor.committeemember</td>
<td>List&lt;String&gt;</td>
<td>Text, Keyword</td>
</tr>
</tbody>
</table>

Table 4: Data schema of the fields added in ETD and tobacco datasets for the data generated by the TML team

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>clusterID</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>summary</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>sentiment</td>
<td>Text, Keyword</td>
</tr>
<tr>
<td>NER</td>
<td>Text, Keyword</td>
</tr>
</tbody>
</table>
4.4 Deliverables

The deliverables that we achieved for this project have been listed below:

- **Data schema for ETD and tobacco settlement datasets**: The data schema has been provided to the FEK, CMT, CME, and TML teams and this would help maintain uniformity in the schema across all the teams.

- **Storing ETD (30K records) and tobacco settlement (14M records) data in Elasticsearch by ingesting them from ceph into Elasticsearch**: This would, in turn, be used by other teams, mostly by the FEK team.

- **Indexing data/metadata**: This supports searching and browsing, which would get implemented whenever we ingest new records into Elasticsearch.

- **Indexing logs generated from the front-end on user search activity**: The TML team will implement a recommendation system using these logs. The logs will contain essential information regarding user session activities, such as terms searched and filters applied along with user credentials.

- **Facet names, field types, usage recommendation, and field examples**: All these are provided to the FEK team for supporting them in filtering, searching, and visualizing both the ETD and tobacco datasets.

- **Incorporating the cluster results received from the TML team to help in ranking and recommendations**: The cluster results would be in the form of scores that would be assigned to each record in the tobacco and ETD datasets as a new field. These scores will be constantly updated via the ‘Update’ API of the Elasticsearch platform.

- **Configuring elasticsearch.yml file**: This helps to enable the FEK team to implement the Reactivesearch tool which connects the front end interface to Elasticsearch. Additional permissions are granted to enable the Reactivesearch tool to access Elasticsearch.

- **Providing the details of the fields to the FEK team**: This will contribute to searching, sorting, and filtering purposes on the ETD and tobacco settlement datasets. Tables 5 and 6 display the fields which will be used for searching and filtering, along with examples.

- **Search query format with an example**: Different types of search queries are shared with the FEK team, along with the examples, to help them retrieve better results while searching or filtering the datasets. An ordinary search query is used to directly search for a term in a certain field. It can search in multiple fields and query the documents in a specific date range. Another search query is the nested search query which is used to locate the page that contains the searched term in the text data.

- **Search Preference using Boosting**: Elasticsearch rank searching results are based on a designed score. A certain field can contribute more to the score by adding more boosting weight to it. The boosting weights are added to the field directly when ingesting or can also be determined in queries while searching.
- Automated scripts: These scripts can automatically ingest new documents into Elasticsearch. A shell script is written for monitoring new files with the help of a Linux tool named inotify and a Python script is written for ingestion and updating of the documents.

- Unit Testing: Unit tests are written on the Python scripts that are coded by our team which can be used by the INT team to implement CI/CD. These unit tests are written in such a way that, they will run on an updated code that is pushed to a repository before the code is committed and is run on production servers. If they pass the tests, the code will be deployed to production servers. If they fail in the tests, the developer will be notified and the errors will be fixed. These tests are written in Python using the unittest Python testing framework.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Searching</th>
<th>Filtering</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree-level</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td>Yes</td>
<td>masters</td>
</tr>
<tr>
<td>contributor-department</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td>Yes</td>
<td>Computer science</td>
</tr>
<tr>
<td>contributor-author</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>Tony Stark</td>
</tr>
<tr>
<td>contributor-committeechair</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>John Wick</td>
</tr>
<tr>
<td>contributor-committeechoair</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>Chris Scott</td>
</tr>
<tr>
<td>contributor-committeemember</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>David Knight</td>
</tr>
<tr>
<td>date-available</td>
<td>Date</td>
<td>Yes</td>
<td></td>
<td>1/23/2017</td>
</tr>
<tr>
<td>date-issued</td>
<td>Date</td>
<td>Yes</td>
<td>Yes</td>
<td>2/21/2018</td>
</tr>
<tr>
<td>degree-name</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>MS or PhD</td>
</tr>
<tr>
<td>description-abstract</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>This field conveys the abstract of the thesis in 10-15 lines</td>
</tr>
<tr>
<td>Author Email</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td><a href="mailto:tony_s@stark.com">tony_s@stark.com</a></td>
</tr>
<tr>
<td>subject-none</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>Soils – Aluminum content Cations</td>
</tr>
<tr>
<td>title-none</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>Hydrolysis of aluminum in synthetic cation exchange resins and dioctahedral vermiculite</td>
</tr>
<tr>
<td>type-none</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td>Yes</td>
<td>Dissertation</td>
</tr>
</tbody>
</table>

Table 6: Tobacco settlements dataset fields used for searching and filtering
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Searching</th>
<th>Filtering</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>Minnesota v. Philip Morris Inc.</td>
</tr>
<tr>
<td>Brands</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>Marlboro</td>
</tr>
<tr>
<td>Witness_Name</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>Wyant, Timothy (affiliation: Decipher; expertise: Statistical analysis; job_title: Independent Statistical Consultant; side: Plaintiff; witness_type: Expert)</td>
</tr>
<tr>
<td>Topic</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>advertising; health effects</td>
</tr>
<tr>
<td>Person_Mentioned</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>Burns, David Michael, M.D</td>
</tr>
<tr>
<td>Organization_Mentioned</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>R.J. Reynolds Tobacco Co.</td>
</tr>
<tr>
<td>Description</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>The plaintiffs expert witness, a statistician, was deposed by the defendants.</td>
</tr>
<tr>
<td>Title</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>Deposition of TIMOTHY S. WYANT, Ph.D., August 19, 1997, MINNESOTA v. PHILIP MORRIS INC.</td>
</tr>
<tr>
<td>Date_Added_UCSF</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td>Yes</td>
<td>1/20/2006</td>
</tr>
<tr>
<td>Document_Date</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td>Yes</td>
<td>1/21/2006</td>
</tr>
<tr>
<td>Document_Type</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td>Yes</td>
<td>deposition</td>
</tr>
<tr>
<td>availability</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td>Yes</td>
<td>public</td>
</tr>
<tr>
<td>availabilitystatus</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td>Yes</td>
<td>no restrictions</td>
</tr>
<tr>
<td>Context</td>
<td>Text, Keyword</td>
<td>Yes</td>
<td></td>
<td>Described the methodology he used to analyze major tobacco-related diseases among nursing home entrants and the conclusions he drew</td>
</tr>
<tr>
<td>page-number</td>
<td>numeric</td>
<td>Yes</td>
<td></td>
<td>'3', '12'</td>
</tr>
</tbody>
</table>
4.5 Quality Control

The Elasticsearch team has delivered the finalized indexed metadata and data of the ETD documents and tobacco settlement documents to the FEK team. We play an important part in the data quality control process. Thus, we take the advantage of Python scripts written by our team and the available tools in the Elasticsearch platform to validate and fix our indexed data in various ways as follows:

- Convert and provide timestamps in the format as requested from the FEK team
- Convert necessary data to different data types (e.g., text to date or text to keyword).
- Provide a fix on improper data format (e.g., the date fields in the ETD documents which are not in the prescribed format by Elasticsearch, making it not possible to filter data based on date fields.)
- Drop records in different scenarios (e.g., invalid entries such as date-“0000-00-00”)
- Remove unnecessary fields based on the input from the FEK and the TML teams
- Add/Remove values to certain fields
- Sanitize unknown Elasticsearch fields that could cause errors
- Logging the errors while indexing and ingesting data into Elasticsearch

4.6 Configuration

This section refers to the log4j2.property for logging and elasticsearch.yml for Elasticsearch configuration. We will introduce both of them in Section 7.
5 Implementation

5.1 Overview

The general goal of the project is to build a state-of-the-art search engine for ETD and tobacco settlement data. Based on current understanding and previous work [6, 8], our team hopes to maintain the same high quality work as teams from previous years. Our approach is as follows:

- Become familiar with Elasticsearch by following the tutorials and instructions posted on the Elasticsearch official website [2].
- Examine the ETD and tobacco settlement data to extract useful and essential information.
- Work with other teams to design data schemas for each of the document collections and receive feedback from collaborating teams to refine the schemas.
- Properly index the ETD and tobacco settlement data.
- Incorporate cluster information from the TML team to support ranking and recommendation for the ETD dataset.
- Incorporate text summarization, NER, and sentiment analysis from the TML team for the tobacco settlement documents.
- Work with other teams to exploit the search activity log file for a better recommendation.
- Assist the FEK team by understanding their roles and functionalities thoroughly. As an outcome, provide an easy to understand user guide for them to connect to Elasticsearch and search the indexed data of ETD and tobacco settlement data.
- Optimize search engine performance by enabling boost, supporting, filtering, etc.
- Provide scripts for automatic ingesting and updating upon receiving new data.
- Provide unit testing scripts to validate ingesting and updating methods.
- Assign an appropriate index lifecycle management policy to different indices corresponding to the read/write nature of their respective dataset.

5.2 Tasks and Schedule

Our planned schedule is listed in Table 7. We update the table to keep track of our progress and incorporate new design requirements. The table includes: simple task description, planned work time (in weeks), team member that is assigned to a given task, and whether the task has been completed or not.

Table 7: Tasks and Schedule
<table>
<thead>
<tr>
<th>Task Description</th>
<th>Planned Completion Week</th>
<th>Assignee</th>
<th>Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign up and form the ELS team.</td>
<td>1</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Set up shared Google Drive to exchange documents, Slack to promptly communicate within team and between teams, and Trello to assign and track tasks among team members.</td>
<td>1</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Finish the Elasticsearch tutorial provided by TA on cloud.cs.vt.edu.</td>
<td>1</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Create shared Google Slides to present and update on the ELS team progress during class sessions.</td>
<td>1</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Learn Elasticsearch by completing Elasticsearch Tutorial [1].</td>
<td>2</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Learn Elasticsearch queries and interaction with the FEK team.</td>
<td>2-3</td>
<td>Soumya, Tianrui, Nick</td>
<td>Yes</td>
</tr>
<tr>
<td>Learn Elasticsearch logging functionality.</td>
<td>3</td>
<td>Satvik, Yuan</td>
<td>Yes</td>
</tr>
<tr>
<td>Draft final report using Overleaf and iteratively updating.</td>
<td>3</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Specify initial schema which includes all fields to gather initial data from the CME and the CMT teams, and hear feedback.</td>
<td>3-4</td>
<td>Yuan, Satvik</td>
<td>Yes</td>
</tr>
<tr>
<td>Index ETD and tobacco settlement data on Elasticsearch container, starting with metadata only.</td>
<td>3-4</td>
<td>Soumya, Tianrui, Nick</td>
<td>Yes</td>
</tr>
<tr>
<td>Prepare for team presentation 1.</td>
<td>4</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Draft interim report 1.</td>
<td>4</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Locate and modify the configuration file for Elasticsearch and Log4j2 to provide better searching experience.</td>
<td>5</td>
<td>Satvik, Yuan</td>
<td>Yes</td>
</tr>
<tr>
<td>Write Python script to automatically index JSON data file provided by CME and CMT.</td>
<td>5-6</td>
<td>Nick, Tianrui</td>
<td>Yes</td>
</tr>
<tr>
<td>Write HTTP search queries to guide the FEK team testing the actual searches.</td>
<td>5-6</td>
<td>Soumya</td>
<td>Yes</td>
</tr>
<tr>
<td>Study Elasticsearch and Log4j2 configuration files to change logging levels.</td>
<td>6</td>
<td>Satvik, Yuan</td>
<td>Yes</td>
</tr>
<tr>
<td>Work with the INT team to export logging files.</td>
<td>6</td>
<td>Satvik, Yuan</td>
<td>Yes</td>
</tr>
<tr>
<td>Prepare for team presentation 2.</td>
<td>7</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Draft interim report 2.</td>
<td>7</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Task</td>
<td>Week</td>
<td>Team</td>
<td>Completed</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-------</td>
<td>----------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Provide feedback on recommendation to the CME and the CMT teams to improve ETD and tobacco settlement data.</td>
<td>8-9</td>
<td>Soumya, Tianrui, Nick</td>
<td>Yes</td>
</tr>
<tr>
<td>Provide HTTP query examples to the FEK team for searching and filtering functions.</td>
<td>8</td>
<td>Satvik, Yuan</td>
<td>Yes</td>
</tr>
<tr>
<td>Changed default data type for custom use.</td>
<td>8</td>
<td>Satvik, Yuan</td>
<td>Yes</td>
</tr>
<tr>
<td>Fix wrongly indexed column that were made by default setting.</td>
<td>8</td>
<td>Soumya, Tianrui, Nick</td>
<td>Yes</td>
</tr>
<tr>
<td>Use text and keyword data types simultaneously for improved searching and filtering.</td>
<td>9</td>
<td>Satvik, Yuan</td>
<td>Yes</td>
</tr>
<tr>
<td>Optimize updating existing values in indexed dataset without re-indexing.</td>
<td>9</td>
<td>Satvik, Yuan, Soumya</td>
<td>Yes</td>
</tr>
<tr>
<td>Help the FEK team to implement range searching within a time frame.</td>
<td>9</td>
<td>Soumya, Tianrui, Nick</td>
<td>Yes</td>
</tr>
<tr>
<td>Help the TML team to update the cluster information in Elasticsearch.</td>
<td>9</td>
<td>Soumya, Tianrui, Nick</td>
<td>Yes</td>
</tr>
<tr>
<td>Implement nested query to support detailed search inside documents.</td>
<td>9</td>
<td>Satvik, Yuan</td>
<td>Yes</td>
</tr>
<tr>
<td>Prepare for team presentation 3.</td>
<td>9</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Draft interim report 3.</td>
<td>9</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Collaborate with the TML team to find ways to use cluster information to implement recommendation.</td>
<td>10-14</td>
<td>Soumya, Satvik, Yuan</td>
<td>Yes</td>
</tr>
<tr>
<td>Extract search activity log from Elasticsearch.</td>
<td>11-12</td>
<td>Tianrui</td>
<td>Yes</td>
</tr>
<tr>
<td>Adjust specific data type such as keyword, date to both ETD and tobacco settlement document.</td>
<td>11-12</td>
<td>Tianrui, Yuan</td>
<td>Yes</td>
</tr>
<tr>
<td>Communicate with the FEK team to inform them about the specific data types along with detailed query examples.</td>
<td>11-12</td>
<td>Satvik</td>
<td>Yes</td>
</tr>
<tr>
<td>Based on the text summary samples provided by the TML team, implemented script for adding new field to existing index in Elasticsearch.</td>
<td>11-12</td>
<td>Nick, Tianrui</td>
<td>Yes</td>
</tr>
<tr>
<td>While waiting for the TML team to deliver cluster information, explore recommendation query based on anticipated cluster information.</td>
<td>12-14</td>
<td>Satvik, Yuan, Soumya</td>
<td>Yes</td>
</tr>
<tr>
<td>While waiting for the TML team to deliver text summarization, NER and sentiment analysis information.</td>
<td>12-14</td>
<td>Nick, Yuan, Tianrui</td>
<td>Yes</td>
</tr>
<tr>
<td>Implement shell script to monitor specific directory for new-coming files.</td>
<td>12-13</td>
<td>Yuan, Satvik</td>
<td>Yes</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Integrate python scripts used for indexing and updating with shell script to implement automatic ingesting and updating.</td>
<td>12-13</td>
<td>Yuan, Nick</td>
<td>Yes</td>
</tr>
<tr>
<td>Implement unit testing scripts for indexing and updating.</td>
<td>12-13</td>
<td>Yuan, Nick</td>
<td>Yes</td>
</tr>
<tr>
<td>Ingest 30K ETD data with full text.</td>
<td>11-13</td>
<td>Nick, Satvik, Tianrui</td>
<td>Yes</td>
</tr>
<tr>
<td>Ingest 5.6M tobacco settlement document metadata as Elasticsearch can store for its current setup.</td>
<td>11-13</td>
<td>Nick, Tianrui, Yuan</td>
<td>Yes</td>
</tr>
<tr>
<td>Ingest 100K tobacco settlement document metadata with full text data.</td>
<td>11-13</td>
<td>Nick, Tianrui, Soumya</td>
<td>Yes</td>
</tr>
<tr>
<td>Prepare final presentation.</td>
<td>14</td>
<td>All</td>
<td>Yes</td>
</tr>
<tr>
<td>Finish final report.</td>
<td>14</td>
<td>All</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**5.3 Search Implementation**

Figures 3 and 4 depict our planned search engine implementation for tobacco settlement and ETD datasets with fields that will be utilized for searching and sorting. Our implementation allows the user to search through multiple field options as required for a particular document. For example, for the ETD dataset, the search fields contain department, title, author, committee chair, committee member, and abstract. Similarly, for the tobacco dataset, the search fields include case, topic, author, witness name, etc. In the second part of our implementation, it allows us to sort the data according to the user demand. The sorting is structured to re-arrange the documents according to relevance and date. This is a useful tool on the front-end, to enable the user to make an effective search.
Figure 3: Planned search implementation for tobacco settlements dataset

Figure 4: Planned search implementation for ETD dataset
5.4 Expected Outcomes

We list our expected outcomes in Table 8:

<table>
<thead>
<tr>
<th>Completion Date</th>
<th>Expected Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/05</td>
<td>Test out Elasticsearch.</td>
</tr>
<tr>
<td>09/19</td>
<td>Send out initial data schema to both the CME and the CMT teams.</td>
</tr>
<tr>
<td>10/10</td>
<td>Share initial log file with the TML team.</td>
</tr>
<tr>
<td>10/10</td>
<td>Share sample HTTP queries with the FEK team.</td>
</tr>
<tr>
<td>10/30</td>
<td>Provide a nested query example to the FEK team.</td>
</tr>
<tr>
<td>10/30</td>
<td>Provide data schema for both ETD and tobacco settlement original data.</td>
</tr>
<tr>
<td>11/20</td>
<td>Provide guideline for recommendation functionality to the FEK team.</td>
</tr>
<tr>
<td>11/21</td>
<td>Provide detailed data type along with usage suggestion to the FEK team.</td>
</tr>
<tr>
<td>11/23</td>
<td>Provide unit testing scripts for ingesting and updating operations.</td>
</tr>
<tr>
<td>11/25</td>
<td>Provide shell scripts for automatic ingesting and updating when there is new file in ceph.</td>
</tr>
<tr>
<td>12/1</td>
<td>Index 5.6M tobacco settlement documents metadata into Elasticsearch.</td>
</tr>
<tr>
<td>12/1</td>
<td>Index 100K tobacco settlement documents metadata with page-wise full text into Elasticsearch.</td>
</tr>
<tr>
<td>12/8</td>
<td>Index 30K ETD metadata with full text into Elasticsearch.</td>
</tr>
<tr>
<td>12/10</td>
<td>Incorporate cluster from the TML team for the ETD dataset.</td>
</tr>
<tr>
<td>12/10</td>
<td>Incorporate text summarization, NER, and sentiment analysis from the TML team for the tobacco settlement documents.</td>
</tr>
</tbody>
</table>

5.5 Achievements

Here we list what has been accomplished by the end of the course.

- For ETD dataset: we have successfully ingested 30,925 ETD documents of metadata and full text data.
  - The ingestion operation achieved a success rate of 99.8%.
  - All documents are fully searchable.
  - The dataset can be filtered and sorted by designated fields. We are able to query for documents in a date range.
- We also provide tested scripts that can monitor new data files and automatically ingest and update new data into existing index.

- For tobacco settlement documents: we have successfully ingested 5,595,936 tobacco settlement documents of metadata.

- The ingestion operation achieved a success rate of 99.9% with only 81 records failed to be indexed.

- Within the ingested metadata index, we also include 100,000 records with full text data

- All documents are fully searchable.

- The dataset can be filtered and sorted by designated fields. We are able to query for documents in a date range.

- We also provide tested scripts that can monitor new data files and automatically ingest and update new data into existing index.

- We have tested to sample data provided by the TML team. Yet, due to the fact that the TML data were uploaded on ceph only two days prior to the course deadline, we could not fully check, analyze, update into the indexed dataset.

- After discussion with the TML team, both teams agreed all TML data to be written in plain text files that are named after the document ID for the tobacco settlement documents.

- When generating cluster information for the ETD dataset, the TML team used a different field that we originally set up as document ID. We re-indexed the ETD dataset to accommodate their changed cluster information.

- We receive 97,484 records of text summarization only for the tobacco settlement documents.

- We receive 765,530 records of sentiment analysis only for the tobacco settlement documents. We randomly checked 10 files and all of them were empty.

- We receive 213,883 records of NER only for the tobacco settlement documents.

- We were promised cluster information only for the ETD dataset. Yet, at the end of the course, the cluster information still was not provided to us.

- Based on the initial design of the additional TML data (plain text, named after document ID), we wrote a Python script and shell script that could detect newly added files and update the existing indices accordingly. The scripts are validated with the sample text summarization files.

- We have provided helpful information to the FEK team.

- We provide detailed data schema for both ETD and tobacco settlement documents, including field names, data type, and usage suggestions.
- We provide HTTP search query examples, covering common search, nested search, and recommendation-based search.
6 User Manual

6.1 Kibana

This section is used to help others interface with Elasticsearch. There will be step-by-step guides connecting Kibana to Elasticsearch. We firstly show how to connect Kibana to Elasticsearch indices.

1. Login to cloud.cs.vt.edu

Go to cloud.cs.vt.edu and login with your account username and password.

2. Select Project from Clusters
Select the cs5604 project from the testing cluster

3. Select the Port that is running Kibana

   Click on the 5601/tcp under the kibana-kibana workspace

4. Using Elasticsearch Data
Click on the Connect to your Elasticsearch Index Hyperlink. This link will move you to the webpage used to connect Kibana to Elasticsearch indices.

5. Create an Index Pattern

You can create an index Pattern here. The index pattern can make one or more indices. The * symbol is used as a wildcard. Clicking the Next Step button will bring you to a
confirmation page.

You can review your index pattern here before creating the Index Pattern, connecting Kibana to Elasticsearch. The latest created index pattern is used as default unless overwritten in the index patterns settings.

6.2 Searching an Elasticsearch Index

6.2.1 Basic Searching

Elasticsearch is designed as a powerful search engine. It’s capable of many kinds of searching. In this section, we will introduce several basic searching query. And, we will show how to use Kibana to conduct the searching. Let’s see the most simple searching query first, which is querying for a single document:

```
GET /tobacco/_doc/ffvv0000
```

As shown in Figure 5, the request will get the results that match the query. In this case, the query will match with the document with ID ffvv0000 in the tobacco index. Note that ID can be string or integer in Elasticsearch as long as it is unique for each document.
If you’d like to search for general information, you can write requests with `_search` API:

```
GET /tobacco/_search
```

Figure 6 shows that it returns 10000 documents with the tobacco index. The maximum hits for a query is 10000 by default. The results are shown in the order of ID since no other query information is sent except the index.
The search criteria can be specified as request body parameters. Here is an example for the most simple criterion. It queries for matches with the word TEXT from the FIELD field.

```
GET /tobacco/_search
{
  "query": {
    "match": {
      "FIELD": "TEXT"
    }
  }
}
```

The _mapping command can help you when looking for what fields an index has as well as their datatype. It will list all fields in the tobacco index as shown in Figure 7.

```
GET /tobacco/_mapping/
```

For instance, if we are going to search for documents with “info” in the “Title” field:

```
GET /tobacco/_search
{
  "query": {
    "match": {
      "Title": "info"
    }
  }
}
```
It will return documents contains “info” as shown in Figure 8.

![GET /tobacco/_search](image)

Here is an example of how to use a cURL command to do the searching:

```bash
curl -X GET "10.43.54.87:9200/tobacco/_search?pretty"
-H 'Content-Type: application/JSON' -d'
{
  "query": {
    "match": {
      "Title": "Retrieve"
    }
  }
}
```

If we would like to know how many documents are from the “United States of America”, the following “agg” command will help you aggregate documents by “Country” field in the tobacco index.

```bash
GET /tobacco/_search
```
From Figure 9, we can see that there are 35668 documents in total from the USA.

TF-IDF, term frequency inverse document frequency, is an important weighting method in information retrieval. The _termvectors API can calculate the term frequency and document frequency for each word in documents.

```
GET /tobacco/_search
{
  "size": 0,
  "aggs": {
    "play_name": {
      "terms": {
        "field": "Country.keyword",
        "size": 20
      }
    }
  }
}
```

Figure 9: Aggregate
6.2.2 Searching in Nested Field

**Nested datatype:** The nested datatype allows arrays of objects to be indexed in one field, and they can be queried independently of each other. As mentioned before, we add a nested field for storing text contents for each document. A sample structure of the nested field is shown below. The “user” field is a nested field with two sub-fields “first” and “last”. It contains two documents, “John Smith” and “Alice White”.

```json
PUT my_index/_doc/1
{
  "user" : [  
    {
      "first" : "John",
      "last" : "Smith"
    },  
    {
      "first" : "Alice",
```
Searching query and highlight: The nested query can search a nested field as if the objects in it were indexed as separate documents. If an object matches the search, the nested query returns the root whole document instead of the separate document. However, our goal is to show the page number on which content matches the searching query. So, we use the inner hits parameter, which allows us to highlight the matching nested documents.

```
POST tobacco/_search
{
  "query": {
    "nested": {
      "path": "text_content",
      "query": {
        "match": {"text_content.content" : "SANDRIDGE"}
      },
      "inner_hits": {
        "highlight": {
          "fields": {
            "text_content.page": {}
          }
        }
      }
    }
  }
}
```

6.2.3 Date Range Searching

The range query returns documents that contain terms within a provided range. In our project, we use the range query with date fields. As shown below, we can search for a document within a time span. The format of date datatype is: yyyy-MM-dd

```
GET /tobacco/_search
{
  "query": {
    "range": {
      "Date_Modified": {
        "time_zone": "+04:00",
        "gte": "2000-01-01",
        "lte": "now"
      }
    }
  }
}
```
7 Developer Manual

In this section, we show how to connect and interact with Elasticsearch. Initially, all operations were tested on the sample file. In this section, we don’t show the architecture, module overviews, etc. We focus on showing how to process data in Elasticsearch.

7.1 Tutorial

7.1.1 Connecting to Elasticsearch by terminal

Elasticsearch is a distributed search and analytics engine which provides real-time search and analytics for all types of data. You can interact directly with your Elasticsearch server by submitting HTTP requests to its API.

First, we can use a cURL command to send HTTP requests to our Elasticsearch server. The internal server IP is 10.43.54.87:9200. If you are using the terminal of the Elasticsearch container on the Computer Science cloud, you can send the following empty request to check if you are connecting to Elasticsearch correctly:

```
curl 10.43.54.87:9200
```

If you’d like to access the Elasticsearch server from your own terminal, you can use the one with external IP.

```
curl 2001.0468.0c80.6102.0001.7015.40b4.a1fb.ip6.name:9200
```

![Shell: elasticsearch](image)

Figure 11: Connecting

Figure 11 shows that Elasticsearch works well and then we can start exploring its features. Firstly, we can use this command:
It can list every existing index in Elasticsearch as shown in Figure 12.

### 7.1.2 Importing Data

In our project, all data will be stored in ceph connecting to Elasticsearch. If you log into the ceph container and go to the `/mnt/ceph/` dictionary, you will find the data in JSON format storing there as shown in Figure 13.
Note that Elasticsearch expects the following newline delimited JSON (NDJSON) structure:

```json
{ "index" : { "_index" : "index", "_id" : "0" } }
{ "field1" : "value1" }
```

It must end with a newline character \n. After data is prepared, we can import it into Elasticsearch by this command:

```bash
curl -s -H "Content-Type: application/x-ndJSON" -XPOST \
10.43.54.87:9200/_bulk --data-binary @tobacco.JSON
```

Run the listing indices command again and you will see the new index imported.

### 7.1.3 Kibana API

Kibana, a front-end platform which lets you explore and visualize your data in Elasticsearch, is also installed and running on our server. We can send the requests, access, and interact with Elasticsearch through the Kibana console.

![Figure 14: Kibana indices](image)

Figure 14 shows that we can send the HTTP request to query information from Elasticsearch through the Kibana console. Kibana will package the request and send it to the Elasticsearch server the same as a cURL command. In the following section, we will show our commands and results mainly in Kibana. We will also show how to convert HTTP requests to cURL commands if necessary.

### 7.1.4 Python API

There is an official Python Elasticsearch client, which provides a more convenient way to write and manipulate queries, especially when dealing with multiple queries. And it allows us to save the returned results easily [5].

Here is a simple example using the Python client to connect and interact with Elasticsearch. The returned results will be shown in Figure 15.
import elasticsearch as els

# connect to Elasticsearch:
es = els.Elasticsearch('2001.0468.0c80.6102.0001.7015.40b4.a1fb.ip6.name', port=9200)

# list indices:
print(es.cat.indices())

# print the id=lzlj0158 doc from tobacco index
res = es.get(index='tobacco', id='lzlj0158')
print(res)

# search for docs with thesis in description-abstrac
res = es.search(index='tobacco', body={
    "size": 20,
    "query": {
        "match": {
            "description-abstrac": "thesis"
        }
    }
})
print(res)

Figure 15: Python results

7.1.5 Indexing

Create a new index:
PUT /index
or curl -X PUT "10.43.54.87:9200/tobacco?pretty"

The Reindex command can help us organize documents to proper indices. It will link documents from the source index to the destination index. Note that it won’t create the new index for you. Please create the destination index before running a Reindex command.

```
POST _reindex
{
  "source": {
    "index": "tobacco"
  },
  "dest": {
    "index": "new_tobacco"
  }
}
```

If you only want to link part of documents from the source index, the search query can also be added in the body part of the reindex request. For instance, this command will reindex all docs with Document_Type deposition to the new index:

```
POST _reindex
{
  "source": {
    "index": "tobacco",
    "query": {
      "match": {
        "Document_Type": "deposition"
      }
    }
  },
  "dest": {
    "index": "new_tobacco"
  }
}
```
All 4244 documents from Hamlet are listed in the Hamlet index. But they won’t be deleted from the source index. If you want to delete them from the tobacco index, you can use the following request:

```json
POST /tobacc/_delete_by_query
{
  "query": {
    "match": {
      "Document_Type": "deposition"
    }
  }
}
```

### 7.2 Indexing/Configuration

#### 7.2.1 Fields

Firstly, we figured out which fields are searchable and which fields are not. Some fields can be set as filters, so that we can filter search results with the option fields. The rest of the searchable fields can be used for full text searching and partial searching. Full text searching means searching the given keyword by the user in all the fields and partial searching means searching only in designated fields. We can modify the setting of fields and delete some fields by running the reindex command or `_delete_by_query` as mentioned before.

Then, we went over each of the fields to check their datatype. Most of the fields are a string or integer which will be automatically converted into the text datatype when ingested into Elastic-
search. But the text type is not the best datatype for all features. In order to make our searching more effective and efficient, we are going to modify the datatype of part of the fields.

The way to change the datatype is shown below. For example, if we'd like to change the datatype to keyword for "Country" field in the tobacco index, we need to create a temporary index, with the keyword field "Country". Then, we reindex all documents to the temporary index and delete the original index. After that, we create the tobacco index again and reindex all documents back.

PUT tobacco2
{
    "mappings": {
        "properties": {
            "Country": {
                "type": "keyword"
            }
        }
    }
}

POST _reindex
{
    "source": {
        "index": "tobacco"
    },
    "dest": {
        "index": "tobacco2"
    }
}

DELETE tobacco

PUT tobacco
{
    "mappings": {
        "properties": {
            "Country": {
                "type": "keyword"
            }
        }
    }
}

POST _reindex
{
    "source": {
        "index": "tobacco2"
    },
    "dest": {
        "index": "tobacco"
    }
}
However, though it is feasible to operate as described above, it is inefficient.

Lastly, we add a boosting weight to certain fields. Elasticsearch rank searching results based on a designed score. The scores are calculated by a similarity model based on TF-IDF as well as using the Vector Space Model for multi-term queries. But what if we’d like to change the score calculation? There are two ways. The first is to modify the score directly. We are able to do that, but it will need a careful design. So, we didn’t implement it in our project. Another way is to use boosting. If you want a certain field to contribute more to the score, then you can add more boosting weight to it. The boosting weights can be added to the field directly when ingesting or determined in queries while searching. Here is an example that the score of a field is boosted automatically while searching, with the boosting parameter set. While searching in the tobacco index, contents in the “Title” field will contribute two times that of others which are set by default to the result ranking score.

PUT tobacco
{
  "mappings": {
    "properties": {
      "Title": {
        "type": "text",
        "boost": 2,
        "fields": {
          "keyword": {
            "type": "keyword"
          }
        }
      }
    }
  }
}

7.2.2 Configuration

In order to support Reactivesearch from the FEK team to connect to Elasticsearch, we needed to configure the CORS (Cross-Origin Resource Sharing) in the configuration file in Elasticsearch to enable access. The settings are listed as follows.

http.cors.enabled: true
http.cors.allow-credentials: true

However, it’s not easy to configure files in a container since we need to redeploy the Elasticsearch node to apply the changed settings. We worked with the INT and FEK teams to make it happen. Moreover, we should specify the lifecycle for each index. Index Lifecycle Management (ILM) APIs allow us to configure the lifecycle for indices. An index’s lifecycle changes, based on the use of the index. There are four stages in a lifecycle [1]:

1. 1
• Hot: the index is actively being updated and queried.

• Warm: the index is no longer being updated, but is still being queried.

• Cold: the index is no longer being updated and is seldom queried. The information still needs to be searchable, but it’s okay if those queries are slower.

• Delete: the index is no longer needed and can safely be deleted.

We can control the maximum size or age for an index. If an index reaches the maximum size or age, it can be moved to the next stage or even be moved to other storage hardware.

We can create the index lifecycle policy in Kibana:

![Index Lifecycle Policies](image)

<table>
<thead>
<tr>
<th>Name</th>
<th>Linked indices</th>
<th>Version</th>
<th>Modified date</th>
</tr>
</thead>
<tbody>
<tr>
<td>sim-history-lm-policy</td>
<td>0</td>
<td>1</td>
<td>2019-10-07 23:37:28</td>
</tr>
<tr>
<td>watch-history-lm-policy</td>
<td>0</td>
<td>1</td>
<td>2019-08-29 12:22:11</td>
</tr>
</tbody>
</table>

Figure 18: Lifecycle policy

We can also change the lifecycle policy by running the command below:

```bash
PUT my_index/_settings
{
   "lifecycle.name": "my_policy"
}
```

### 7.3 Updating Elasticsearch Fields

The UPDATE API allows you to script document updates. This script enables us to modify, update, or delete the desired field. This operation fetches the document from the supplied index, then it runs the specified script and indexes the result. The update removes some network round trips and also reduces the chance of a version conflict between the GET and index operation. The “source” field must be enabled in order for the script to update.

```bash
POST tobacco/_update/1
{
   "script" : {
```
We have successfully managed to update sample fields in Elasticsearch. As per the requirement, we need to update the following fields in our database: 1) Cluster ID, 2) Similarity Scores, 3) NER, 4) Text Summary, and 5) Sentiment

### 7.3.1 Updating the Similarity Score in Elasticsearch

A similarity score or a ranking model defines the method by which the matching documents are scored. The similarity is per field; therefore we can define a different similarity for each field. There are multiple built-in similarity scores provided by Elasticsearch. These built-in similarities can be tested when searching along with the custom similarities. Some of the similarities are described as follows:

1. **BM25 Similarity**: This is the default similarity incorporated by Elasticsearch. This is a TF-IDF based similarity that has built-in TF normalization and works better for short fields (like names).

2. **DFR Similarity**: This similarity implements the divergence from the randomness framework.

3. **IB Similarity**: This similarity score is based on the fact that the information contained in any symbolic distribution sequence is primarily determined by the repetitive usage of its basic elements.

4. **LM Dirichlet Similarity**: This algorithm attempts to capture important patterns in the text while leaving out the noise.

5. **Scripted Similarity**: This similarity allows you to use a script to specify how scores should be computed.

To select what type of similarity we want to use, we can use the following configuration:

**Script for default similarity section:**

```json
PUT /tobacco
{
  "settings": {
    "index": {
      "similarity": {
        "default": {
          "type": "boolean"
        }
      }
    }
  }
}
```
Script for configuring a different similarity:

```json
PUT /tobacco
{
   "settings" : {
      "index" : {
         "similarity" : {
            "my_similarity" : {
               "type" : "DFR",
               "basic_model" : "g",
               "after_effect" : "l",
               "normalization" : "h2",
               "normalization.h2.c" : "3.0"
            }
         }
      }
   }
}
```

Here the DFR similarity is configured to be referenced as my_similarity.

```json
PUT /index/_mapping
{
   "properties" : {
      "title" : { "type" : "text", "similarity" : "my_similarity" }
   }
}
```

### 7.3.2 Updating the Similarity Score using Custom Similarity:

Elasticsearch also allows us to configure a custom similarity, which is a high-level feature provided by Elasticsearch. This is enabled by the scripted similarity measure mentioned above. This is an important feature for our project, as it will enable us to use the similarity score computed by the TML team’s algorithm. Thus, we can have our custom similarity measure and ranking for our ETDs and tobacco settlement documents. Another advantage is that we can have multiple custom similarities developed by the TML team and analyze which similarity works best for our dataset. While scripted similarities provide a lot of flexibility, there are a set of rules they need to satisfy. Failing to do so could make Elasticsearch silently return wrong top hits or fail with internal errors at search time. The rules are:

- Returned scores must be positive.
• All other variables equal, scores must not decrease when doc.freq increases.
• All other variables equal, scores must not increase when doc.length increases.

Here we present an example of customizing the existing TF-IDF similarity, making the calculation slightly more efficient. The custom similarity can be configured by running the script below. This will help us to change the existing formula of TF-IDF calculation to our custom formula. As mentioned in the code, we update the variables IDF and TF in the weight_script. Therefore, now the similarity score will be calculated based on our formula. This is a feature that we can exploit to boost the searching process.

```
PUT /tobacco
{
  "settings": {
    "number_of_shards": 1,
    "similarity": {
      "scripted_tfidf": {
        "type": "scripted",
        "weight_script": {
          "source": "double idf = Math.log((field.docCount+1.0)/(term.docFreq+1.0)) + 1.0;
          return query.boost * idf;"
        },
        "script": {
          "source": "double tf = Math.sqrt(doc.freq); double norm = 1/Math.sqrt(doc.length);
          return weight * tf * norm;"
        }
      }
    },
    "mappings": {
      "properties": {
        "field": {
          "type": "text",
          "similarity": "scripted_tfidf"
        }
      }
    }
  }
}
```

In the above example, we can see that the type field is set as scripted.

### 7.4 Logs

#### 7.4.1 Log4j2 Log

Elasticsearch applies a tool called Log4j2 for logging. We configured the logging setting in the logging configuration file “log4j2.properties” and Elasticsearch configuration file “elasticsearch.yml”.

51
7.4.2 Slowlog

In our project, we use a specific kind of logs called slowlog, which is designed to log slow searches like queries into a dedicated log file. However, it can be configured to log all queries. We can use the following command to configure the settings:

```
PUT /tobacco/_settings
{
    "index.search.slowlog.threshold.query.warn": "0s",
    "index.search.slowlog.threshold.query.info": "0s",
    "index.search.slowlog.threshold.query.debug": "0s",
    "index.search.slowlog.threshold.query.trace": "0s",
    "index.search.slowlog.threshold.fetch.warn": "0s",
    "index.search.slowlog.threshold.fetch.info": "0ms",
    "index.search.slowlog.threshold.fetch.debug": "0ms",
    "index.search.slowlog.threshold.fetch.trace": "0ms",
    "index.search.slowlog.level": "info"
}
```

Levels, which include warn, info, debug, and trace, allow us to control under which logging level the log will be logged.

The log file will contain the following fields.

- "timestamp"
- "cluster.name"
- "node.name"
- "cluster.uuid"
- "node.id"
- "message"

The message field contains all query information like: the term we are searching for, if the fields are searchable, and so on.

7.4.3 User Log

A user is required to register before he or she is able to search in our front-end website. Therefore, it enables us to collect user-orient information. The user log can be used for detecting and preventing malicious behaviors or users, characterizing websites, and recommendation. We provide a designed user log sample below. The fields with examples are followed by the datatype in the parenthesis.

- status: 200 (Keyword)
- username: xxx (Keyword)
- email: xx@xx.xx (Keyword)
Indices for these logs including system logs and user logs can be created continuously every day, week, or month. It will depend on the number of users and amount of traffic flow that there is for our website. We can decide how long we would like to keep these logs by index Lifecycle. We are also able to merge several log indices into one, or split a log index into several indices if necessary when the traffic changes rapidly. There are also several fields that are helpful, but we may need more effort to collect them. Examples include:

- number of results retrieved (Numeric),
- time spent searching this query (Numeric),
- cookie (Keyword).

In addition, other features can also be taken into consideration. Another example is the ratio of the numbers of clicks on an URL to the searching times. It can measure the quality of our searching.

Lastly, the close index API could be used to close opened indices for old logs to save memory.

POST /tobacco/_close

7.5 Recommendation System

We will work with the clusterID values generated by the TML team to build the recommendation system. When a user queries for Jeong-Ah in the “contributor-author” field for the tobacco index, we would get the search result with the matching records. Then the Python script would take the clusterID of the top hit search result and run another query to find the records with matching clusterID and these results will be shown on the front-end as recommendations to the user. Figure 19 shows the Python script used to implement the recommendation system.
Figure 19: Python script for recommendation system using clusterID

In Figure 20, you can see the results of this Python script, which includes the number of records matching the query and its clusterID. The second query searches for the records with matching clusterID 1 in this scenario and gives a result of 2 matching records.

Figure 20: Result of the executed Python script for recommendation system

7.6 Automatic Script

The ETD and tobacco settlement documents are in growing collections. There is a need to ingest new documents created in the future into Elasticsearch. We could ingest these new documents manually, but, in collaboration with the other teams, we can create a procedure to automatically ingest new documents in the future. The CME and CMT teams will create and format these new documents into the specified JSON format. They will move these files into a specific directory where a script will be running in the background. This shell script watches for the addition of new files in the directory and runs the ingesting script on these new files.
7.7 Unit Testing

One of the goals of the project is to implement the requirements needed to practice Continuous Integration and Continuous Deployment (CI/CD). In CI/CD, unit tests are run automatically to test code that is pushed to a repository for the project. We have written a majority of our code in Python. So, we have written our unit tests using the unittest Python framework to test our code. These unit tests cover the methods used when ingesting the CME and CMT documents.
8 Future Focus

After successfully building a working search engine for the two large document data sets, namely tobacco settlement documents, and ETDs, we would continue to improve the performance of Elasticsearch.

The first significant step forward is to increase the space limit for ingesting data in the future so that we could continue to work on ingesting the rest of the documents into Elasticsearch.

Another task is to continue working with the TML team to improve the recommendations. Additional TML data is not fully available, including text summaries, sentiment analysis, NER, and clustering.

Furthermore, we need to extend support for user logs and recommendations that include user-specific logs generated by the FEK team and index these logs once they are available on ceph.

Lastly, we can improve the potential text search use-case capability by further working on our indexing formats while we continue to make Elasticsearch more robust to deliver faster responses.
References


