Technology Review: Apache Lucene to Elasticsearch

Due to my career, I have accumulated some experience with Elasticsearch use cases in Cybersecurity which has caused me to thoroughly enjoy the study of Text Retrieval and Text Mining in this course. Setting up our own instance of Elasticsearch is supposed to be outside the scope of our work duties but Lucene queries are something we have to be familiar with to show proficiency in satisfying some of our requirements. Realizing Elasticsearch is built on top of Apache Lucene, I decided to read *Mastering ElasticSearch* (Kuć & Rogoziński 2013) to gain further understanding on similar topics presented in our Text Information Systems course. Next, with the knowledge obtained from this work, I will cover how Apache Lucene works at a basic level. Lastly, using *The Complete Guide to The ELK Stack,* found at the following link: https://www.google.com/amp/s/logz.io/learn/complete-guide-elk-stack/amp/, and some Cybersecurity use cases with Elasticsearch, I will then cover how Elasticsearch has been used to build upon the functionality of Apache Lucene. There is much more that can be discussed from Apache Lucene and Elasticsearch respectively but we will only cover some of the topics to get a general idea of the Text Retrieval and Text Mining topics used to get to the end state of Elasticsearch's use in Cybersecurity.

The similarities from the Text Information Systems course I will discuss in this paper involve querying to find relevant documents. Another topic will be the structure of an inverted index where terms are mapped to documents. To make an inverted index I will also need to discuss the parts used such as a field, term, token, and document. Lastly, I will cover the implementation of scoring algorithms also known as similarity modules applied for ranking documents.

Apache Lucene is a full text search engine library. Natively, it comes with the Lucene query language and the query syntax is made up of terms and operators. All of the data is stored in fields which make up a document. A field is a section of the document which is built of two parts, the name and value. A document has the data and is made up of one or more fields which is used during indexing and searching. Terms are found in a query and represent a unit of search representing a word from text. Lastly, a token is an occurrence of a term from the text of the field. A token consists of the term text, start offset, and end offset, as well as a type. A Lucene query in Elasticsearch can be run against a field with a clause using a colon in between. The following example will show what a query would look like when attempting to find a term like elasticsearch in the field named title: "`title:elasticsearch`". An example of applying an operator in your query to match multiple terms to a single field can be expressed a couple of ways like so: "`title:(+elasticsearch +"mastering book")`" or "`+title:elasticsearch +title:"mastering book"`". Additionally, there are boolean operators, term modifiers, and ways to handle special characters.

So far we have covered querying documents, basic Lucene query syntax, and some definitions to understand Apache Lucene's architecture like documents, fields, terms, and tokens. These definitions are used by Apache Lucene to write all of the information to a structure called an inverted index. As we learned in our course, in an inverted index each term points to a number of documents which contain the term along with a count associated based on the term frequency within the document. An inverted index allows for very efficient and fast search using term-based queries. Apache Lucene actually uses a more complicated and advanced index based on term vectors for single fields but the general idea of how it is organized is the important part to know, not what is stored.

For brevity we will only cover the default Apache Lucene 4.0 scoring mechanism: the TF / IDF (term frequency / inverse document frequency) algorithm. The practical formula Apache Lucene utilizes for assigning a score for ranking the documents based on the terms is below.

$$\boxed{score(q,d) = coord(q,d) * queryNorm(q) * \sum_{t\ in\ q} (tf(t\ in\ d) * idf(t)^2 * boost(t) * norm(t,d))}$$

The sum is calculated by multiplying the term frequency, the inverse document frequency, the norm which in this case is the length of the field not necessarily the length of the document which is calculated during indexing and stored in the index, and the boost which is a value given for a field during both indexing and querying. To complete the formula, the Coord is a factor based on the number of terms the document has and the queryNorm is a query based on the normalization factor that is calculated as the sum of a squared weight of each of the query terms that allows score comparison between queries. According to *Mastering ElasticSearch* (Kuć & Rogoziński 2013), the few rules that come from this formula are, the more rare the term matched is, the higher the score the document will have. The smaller the document field is (contain less terms), the higher the score the document will have. Lastly, the higher the boost, the higher the score the document will have. Additionally, there are three other similarity models that can be used in Apache Lucene to rank documents which are, Okapi BM25, Divergence From Randomness (DFR), and Information Based (IB).

Elasticsearch is incorporated into the ELK Stack to extend Apache Lucene capabilities into Cybersecurity by allowing us to aggregate all of the information in a network environment for analysis. Essentially, instead of using the World Wide Web as a database for documents, we use something called Beats for data collection against many different types of systems. Beats ships the different types of logs to Logstash for aggregation and processing the different types of logs into meaningful data for analysis. The data is then moved from Logstash to Elasticsearch for the indexing and storage as we just discussed. Lastly, Kibana is used to interface with Elasticsearch for analysis and to create visualizations based on the data. This is where Lucene queries can be used to search for anomalies within the network environment. Kibana's capability of creating visualizations can also allow you to easily communicate incidents in obscure environments such as which systems are communicating the most or logins from unexpected geographical regions. While Elasticsearch also satisfies many other concerns such as data provenance and availability, it would be outside the scope of this paper.

Understanding the capabilities underneath Elasticsearch has sparked my interest in creating potential applications and conducting research in additional effects Text Retrieval and Text Mining can have. Covered early in this course, we defined that about 90% of the world's data is unstructured. Using statistical means, I have learned that not all data has to be structured for it to be analyzed and Apache Lucene is a clear indication of implementing several topics we covered.