Vashistha Kumar Singh
(vksingh3@illinois.edu)

Introduction to Elasticsearch - searching using Human Language.

Introduction:

Information retrieval has been of the utmost importance since ages. The information is only useful if that can be retrieved with accuracy and relevancy. In the information retrieval concepts, it is a tradeoff between precision and recall. **Precision** here means returning as few irrelevant documents as possible and **recall** means returning as many relevant documents as possible.

Searching with a few words or sentences which match either in fraction or keeping the sequence as is still easier to solve. However the natural language/human language search is the closest to matching with human needs. For example searching “fastest animal on the planet” vs searching for “animals who are fastest on the planet”. As a human we would expect to have the same answers to both these search requirements.

In this document we will go over some basic techniques to achieve the searching using the human language.

Procedure:

The basic flow in achieving this is to recognize the language, identify the words, tokenization, handle the stopwords, add the synonyms and remove the typos and misspellings. There are many language analyzers available in elasticsearch. The language analyzers perform broadly 4 main roles including tokenize text into words, convert to lowercase, remove the stopwords and stemming the tokens. As long as the search input is only in one language, it could be lucky and easier to perform a search. However at times there could be possibility that the search text can be mixed with many languages. Multilingual documents come in three main varieties, one predominant language per document, one predominant language per field and mixture of languages per field.

At times stemming can be a challenge for mixed language contents. The reason is that the stemming is per language and it is not easy to use for the mixed language. If we apply the multiple stemmers to the same text, it is more likely that the result will be garbage. The reason is the stemmer in the chain may compound the problem.

Words in the English language are relatively easy to spot. Words are separated by some punctuation. Standard analyzer does its job pretty well in identifying the words. Once you get the tokenization done it is like finishing only half the job. The next important step is to normalize. This step is important because it needs to remove insignificant differences between the otherwise identical words. The next step is to reduce the words to their root form. For example most languages are inflected. Meaning that words can change their form to express differences in the many categories like Number,tense,Gender etc. While inflections aid expressivity it interferes with retrievability. Stemming does help in solving this however one needs to be extra careful and ensure that there is no overstemming or understemming. **Overstemming** is the failure to keep two words with distinct meanings separate. For instance general and generate may both be stemmed to **‘gener’**. **Overstemming** will reduce precision. **Understemming** is the failure to reduce words with the same meaning to the same root. For example ‘jumped’ and jumps may be reduced to jump, while jumping may reduce to ‘jumpi’. **Understemming** reduces the recall. There are various stemmers available and should be used based on the need. Algorithmic stemmers are typically 4-5 times faster than Hunspell stemmers. If an algorithmic stemmer is available for the language it makes sense to use it rather than Hunspell stemmer. It will be faster and will consume less memory.

Information retrieval should be discussed with performance as criteria too. The terms can be roughly divided into two groups: Low-frequency and high frequency. Also the stopwords need to be handled too. The biggest disadvantage of keeping stopwords is that of performance. When Elasticsearh performs a full text search it has to calculate the relevance score on all matching documents in order to return the top 10 matches. Leaving stopwords in the index could make the relevance calculation less accurate, especially if the documents are very long.

While stemming helps to broaden the scope of search by simplifying inflected words to their root form, synonyms broaden the scope by relating the concerts and ideas. Last but not the least is removing typos and misspellings.

Natural Language Processing Support:

Elasticsearch in the latest releases started adding NLP support in the product. It can analyze natural language data and even make predictions. Using the NLP operations we can extract information, classification of text and even search and compare the texts. The fundamental step is to build a trained model and deploy the same. The model should be imported along with the vocabulary. Elastic search has the support for many third party NLP models like BERT, BART, DPR bi-encoders, MobileBERT, RoBERTa, MPNet etc.

The Elastic Stack machine learning features are structured around BERT and transformer models. These features support BERT’s tokenization scheme (called WordPiece) and transformer models that conform to the standard BERT model interface.

Conclusion:

While Elasticsearch started with document search with simple text, over time it started adding more and more human language support. Of late it has also started giving support on Natural language processing using the machine learning stack of Elasticsearch. This is very flexible and even extensible.

Vashistha Kumar Singh
(vksingh3@illinois.edu)

References and Further Readings:

- https://www.amazon.com/Elasticsearch-Definitive-Distributed-Real-Time-Analytics/dp/1449358543
- https://www.elastic.co/guide/en/machine-learning/8.5/ml-nlp.html
- https://github.com/stdatalabs/sparkNLP-elasticsearch
- https://buildingvts.com/elasticsearch-architectural-overview-a35d3910e515