|  |  |
| --- | --- |
|  | **PES UNIVERSITY**  **(Established under Karnataka Act No. 16 of 2013)**  **100 Ft. Road, BSK III Stage, Bengaluru – 560 085**  **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  **SESSION: AUG-DEC 2020** |

|  |  |  |
| --- | --- | --- |
| **Course Title: Algorithms for Information Retrieval** | | |
| **Course code: UE17CS412** | | |
| **Semester : VII sem** | **Section: G** | **Team Id: 39** |
| **SRN: PES1201700048** | **Name: S Sriya** | |
| **SRN: PES1201700062** | **Name: Bhavna Arora** | |
| **SRN: PES1201700798** | **Name: Lavanya L** | |
| **SRN: PES1201701563** | **Name: Abhishek M** | |

**ASSIGNMENT REPORT**

**Problem Statement:**

The project aims in building a search engine for indexing and ranking the retrieved documents for a query. The corpus is built from the Environmental News NLP archive dataset from Kaggle. The primary objectives of the project is to search through documents for a given query and rank the pages based on relevance. The query response time and efficiency of the search engine is measured using metrics like precision, recall, F1-score and accuracy. The results of the search engine is compared with elastic search for the same corpus.

**Description**

The dataset is Environmental News NLP archive dataset from Kaggle which includes 418 documents. The code for the project is developed in Python using NLTK and sklearn libraries in Google Colab notebook. The corpus for the dataset is built using the snippet column which is the actual snippet of news published and aired. Pre-processing is done on the corpus for better results which is used to build the search engine. The pre-processing includes uniform casing, stopwords removal, punctuation removal, lemmatization and tokenization.

The inverted index is built as an index data structure storing a mapping from content to its locations in a set of documents. The structure of the inverted index is a dictionary with mapping from term to its respective posting list. The posting list consists of individual postings each of which consists of document id and a payload information about occurrences of term in the document with respect to the rows. The posting list also includes the document frequency i.e the number of documents in which the term occurs. Given a term, retrieval involves fetching posting lists associated with the term ids of the query terms and traversing the postings to compute the result.

The B-tree is an alternate indexing method which is built during the course of the project. Although the lookup in a hash is faster than the tree, the B -tree is always balanced and after any update, remains in sorted condition and supports prefix search. The degree of the B-Tree is set to 3 which means that the number of children for a node is at most 3, as the degree increases the height of the tree will be small and the number of comparisons of the keys in the node increases thereby reducing the efficiency. The key defined in the nodes is stored as a tuple that includes the term id and the posting list.

The Inverted index for phrase query has been extended to biword indexing which includes indexing every consecutive pair of terms in the text as a phrase and storing each of the bi-word as a vocabulary term in the inverted index. This method makes it easier and efficient to retrieve documents of the most frequently occurring bi-words.

Search for a single word query in a dictionary is done by searching for the term id which returns the posting list. Search for a single word query in a B-tree involves the traversal through it using the term id. This returns the node with data in the node and position of the key in the respective node. Each term in the key stores the term id and the posting list, traversing the postings helps in computing the result set. The searched row and the document in which the query is found is displayed.

Document retrieval for boolean query processing which includes AND, OR and NOT is implemented. Optimization strategy for AND includes the processing results of the terms in order of increasing document frequency. This technique of intersection on the documents increases the efficiency of the search. OR is implemented using the union of the documents with respect to the query terms. The implementation of the NOT boolean query retrieval includes finding the complement of the query without NOT i.e, finding the difference between the Corpus size and posting list of the query without NOT.

The phrase queries which the most common bi-words are retrieved directly from the inverted index. Search for the phrase query is implemented using the extension to the implementation of proximity query which is Modified Intersect For Proximity Constraint K. The parameters for this algorithm includes two words from the query, the two words chosen have a high TF-IDF score compared to the other words in the query. The window of size K is decided based on the position of two words with the highest TF-IDF values in the query. Then the intersection algorithm is run for the documents and the rows of each document. This retrieves the documents in which the phrase query occurs.

As part of tolerant retrieval, search for the wild card query that includes prefix and suffix query is implemented using B-tree implementation of the inverted index. The implementation for retrieval of the wild card query is done using K-gram indexing over the permuterm index as the permuterm index bloats up the dictionary as the number of rotated terms for each original term is very large. K-gram indexing is implemented by enumerating all character k-grams occurring in a term in a list. For every term in the standard inverted index, the bigram i.e, k=2 are generated. Inverted index from k-grams to the terms that contain the k-gram are stored in a list. As the intersection algorithm needs to be applied, the posting list of the k-gram index is stored in lexicographically sorted manner. Since there are many false positives, post filtering is done against the query. Surviving terms are then looked up in the term-document inverted index for the corpus. For wildcard query retrieval the two indexes that are being used are: standard inverted index and bigram index.

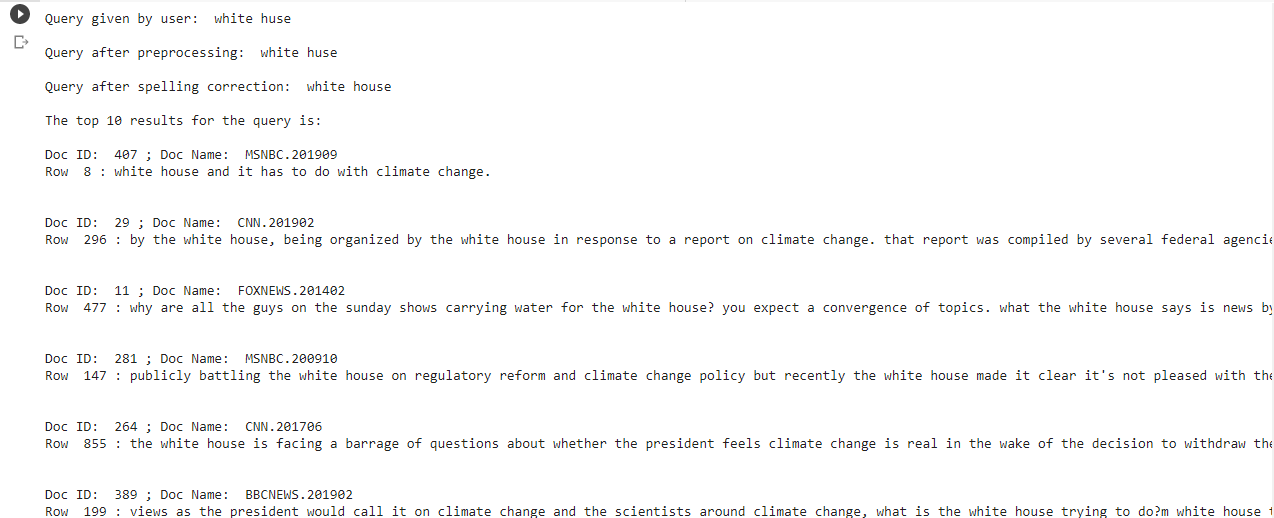
Spelling correction is also done on the query, as a part of tolerant retrieval. This helps in finding the words that are mis-spelled and replaces them with the best possible suggestion based on the terms present in the inverted index. This method of query processing avoids errors while searching for a phrase or a term with respect to the inverted index.

The bag-of-words(TF-IDF weighted) document is used for the vector space model which is the encoding based on a frequency distribution over the top words of the corpus. The vectors are constructed for each row of the documents in the corpus and the pre-processed query. The ranking for the retrieved documents of the search functions are done based on the cosine similarity and the top-k documents are displayed. For tolerant retrieval, when no documents are retrieved from the search functions, ranking is done based on relevance.

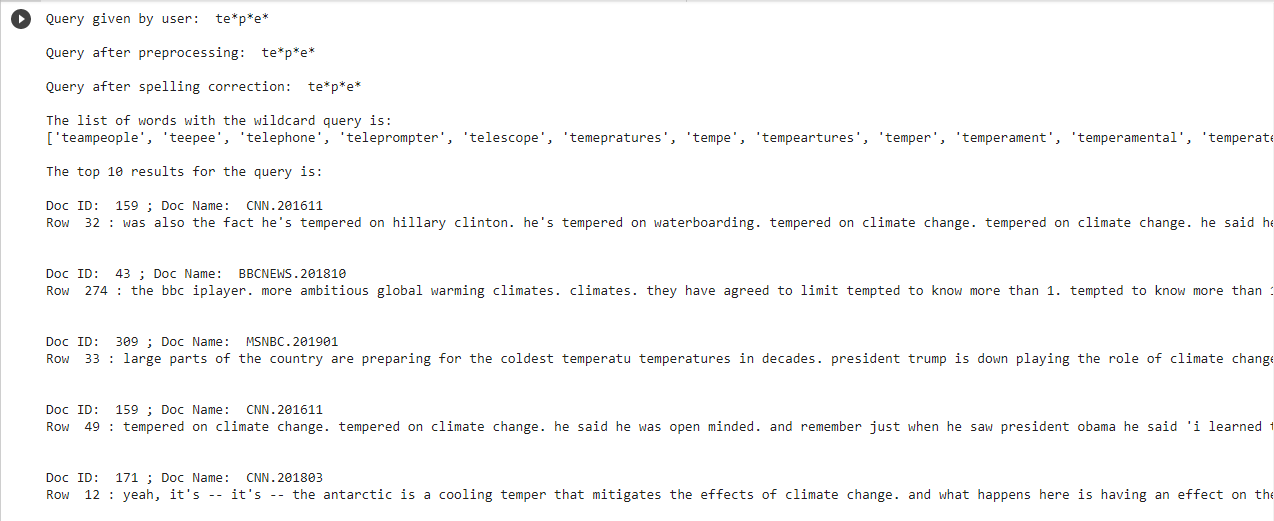
Elastic Search, which is a real-time distributed and open-source full-text search and analytics engine, is also used for evaluation of the search engine built. The elastic search is used to compare the query response time of the results and compute the performance metrics like accuracy, precision, F1-Score and recall of the search engine built.

**Output Screenshots**

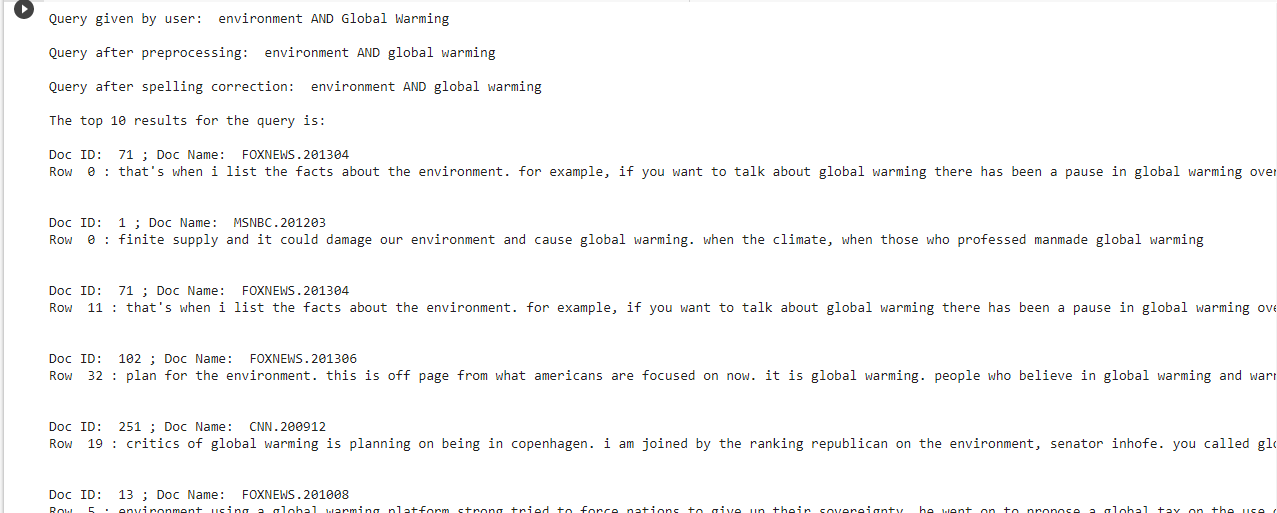
1. The top 10 results for phrase query “white huse” is:



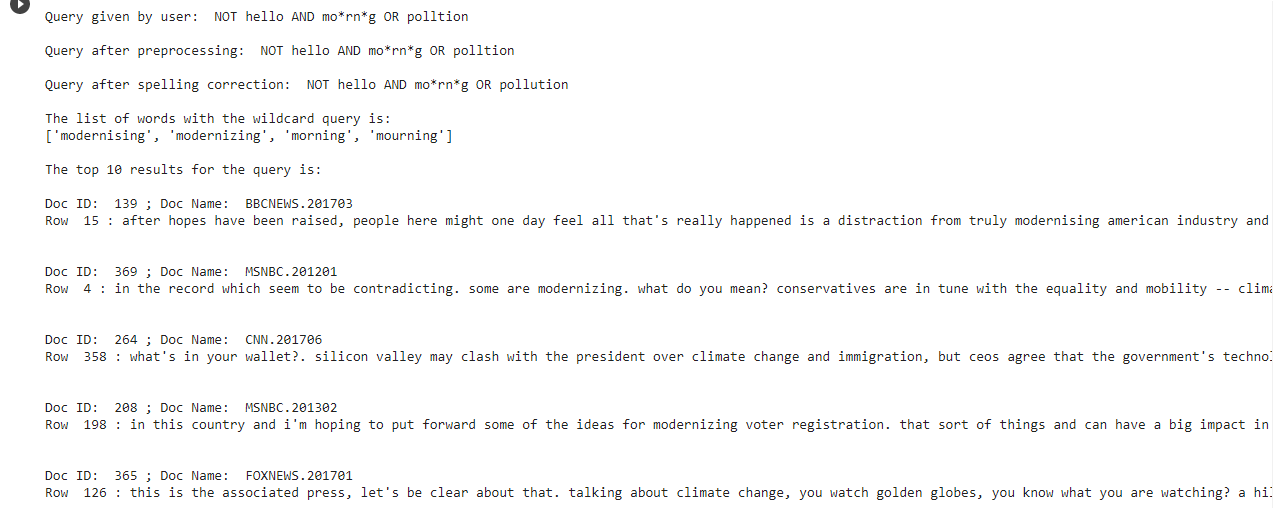
1. The top 10 results for wildcard query “te\*p\*e\*” is:



1. The top 10 results for boolean query “environment AND Global Warming” is:



1. The top 10 results for a combination of the above query “NOT hello AND mo\*rn\*g OR polltion” is:



**Interpretation of efficiency**

Average metrics over 50 queries:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Query | Precision | Accuracy | Recall | F1-Score | Query Response Time(ms) |
| Standard Query | 96.05 | 93.97 | 63.20 | 66.97 | 20032.10 |
| Wild Card Query | 94.77 | 99.92 | 100 | 96.46 | 22056.22 |
| Boolean Query | 100 | 98.088 | 82.96 | 88.88 | 14819.098 |
| Combination Query | 90.21 | 89.52 | 78.62 | 72.29 | 23014.75 |
| Average measures | 95.25 | 95.37 | 81.195 | 92.54 | 19750 |

The query response time of the search engine is measured and the efficiency of the search engine is calculated using the accuracy measure. The documents extracted by elastic search for a query are considered as the relevant documents and the documents from the search engine built are considered as the retrieved documents. With the help of the above measures, values like true positive, false positive, true negative and false negative are calculated for each query. These values are used for the computation of accuracy, precision, recall and F1-score and an average over all the query is used as the final measures.

**Learning Outcome**

We learnt how to apply theoretical aspects of information retrieval into practice for a large realtime dataset. We explored alternate methods of storing the inverted indexes which supports all types of queries given by the user. We familiarized ourselves with different indexes for better and efficient results for retrieval. We also understood how to apply tolerant retrievals strategies on queries to perceive the intended meaning of the user’s query. One of the most important learning aspects in the project was ranking the documents based on the relevance of the retrieved documents with respect to the query. We learnt to present the retrieved data in a user understandable format. We gained an understanding of the working of the elastic search engine for any given dataset and to compare the results given with our search engine.

**Name and Signature of the Faculty**