CS F469: Information Retrieval

Open-Source Search Engine [tf-idf]

1. Ujjwal Raizada 2017A7PS1398H
2. Satyam Mani 2017A7PS0277H
3. Daksh Yashlaha 2017A7PS0218H
4. Pranjal Gupta 2017A7PS0124H

Open-Source Search Engine

Design Document

This search engine indexes all the documentation in open source projects on GitHub and then uses tf-idf (term frequency - inverse document frequency) to rank the documents for a specific query. We have to build the index for every query because tf-idf ranking is based on the number of documents the work occurs.

tf-idf stands for *Term frequency-inverse document frequency*. The tf-idf weight is a weight often used in information retrieval and text mining. Variations of the tf-idf weighting scheme are often used by search engines in scoring and ranking a document's relevance given a query. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus (data-set).

- Python3 is the language of choice because of rich in-built data structures and libraries for data pre-processing.
- NLTK library is used for data pre-processing and tokenization.
- pickle library is used to persistently store the index.

1. **Normalized Term Frequency (tf)**
2. **Inverse Document Frequency (idf)**

Component:

1) **Pre-processing:** Firstly the document is converted to a string stream, then all words are converted to lower-case for consistency, then NLTK library is used to remove stop words then stem all the remaining words. Then frequency ratio of each word is calculated (number of occurence / total number of words).
    - Tokenize
    - Remove stop words
    - stemming

2) **Scraper:** This component scrapes GitHub for README files of repositories which are most starred and most forked. These README files are then processed and added into the corpus.

3) **Query expansion:** Expand the input query words using wordnet corpus dataset. I.e. also include synonyms of the words in the given query for better results.

4) **Dynamic Index:** Index is automatically updated whenever a new document is added in the “corpus/” directory. A list is maintained of the already processed documents and is used to check for any new documents.

Performance:

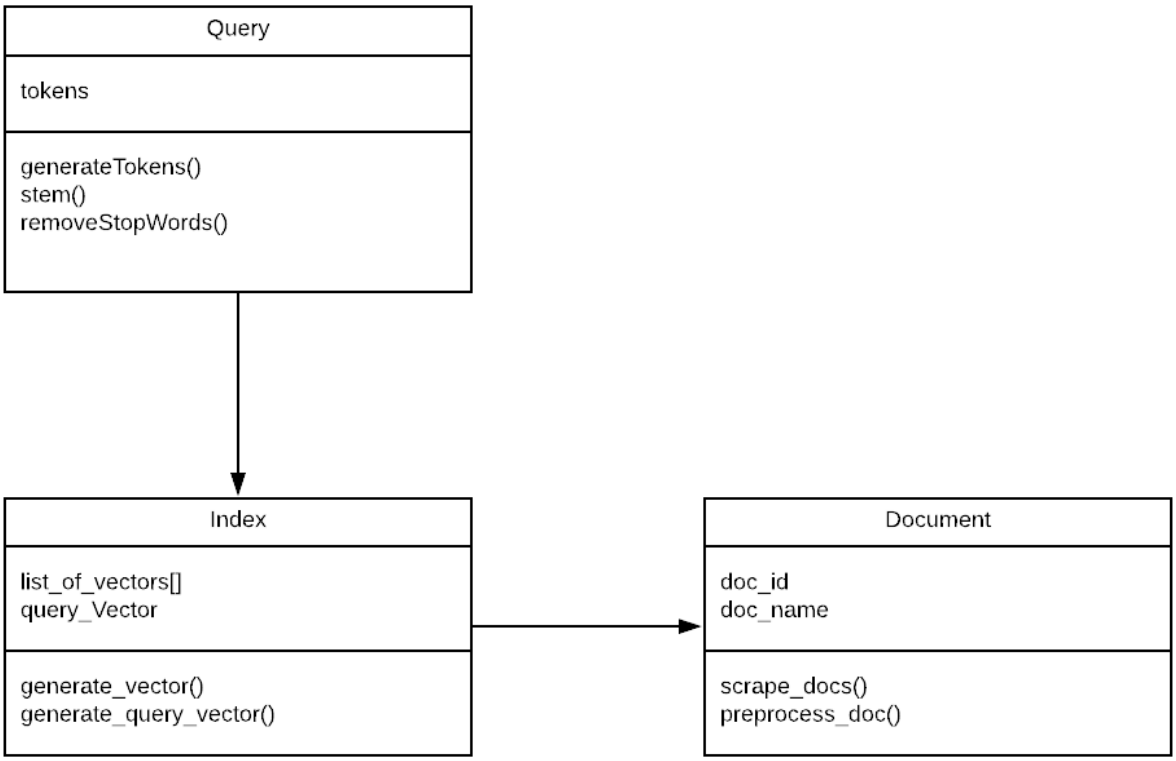
The performance of a model is judged on the basis of precision as well as recall it provides for different queries. We took 10 labeled documents and ran different queries and compared the results with the expected results, then took the average of the result..

- **Precision:** Number of actually relevant documents among the returned result.
- **Recall:** Number of relevant documents returned among the expected documents.

  Precision: 0.84
  Recall: 0.76

Note: Automatic calculation was not possible because our data (github README) is not labelled.

Class Diagram:



Data Structure [tf-idf]:

Every document is represented as a vector with each dimension value containing tf-idf scores with respect to the term. Then finally we take the dot product of query vector with document vector and then finally generate a list of 2-tuple, with the first element being the tf-idf score and the other being the document ID. Score is used to sort the list in reverse order thus ranking the document with highest tf-idf score first.

Final result:

[(score, doc-id), (score, doc id), ……., (score, doc-id)]

Finally the score values are ignored and doc-ids are displayed in the order.

Future Prospects:

1) Better interface for user to search.
2) **User open source projects recommender.**
3) Additional indexes for important compound statements. (eg: “machine learning”, “web development” etc.)

Project is available on github: https://github.com/ujjwal-raizada/open-source-search