Information Retrieval

Assignment 1

**Plagiarism Checker**

Developers:

Rishab Nahar-2018A7PS0173H

Nipun Wahi-2018A7PSH

Samkit Jain-2017BA7H

**1.1 Objective:**

The aim of this Assignment is to develop a small version of a Plagiarism Checker using the algorithms taught in class. The major task is to build a plagiarism checker which will rank documents based on similarity. The Plagiarism Checker developed by us takes the user’s query in form of text file or raw data and fetches the documents from where the data has been copied based on vector space model of information retrieval.

We are working on a dataset of almost 2300 articles which were taken from BBC news website. The objective is to effectively find the copied data percentage from other documents based on the upper-bound mentioned by the user.

**1.2 Definitions and Acronyms**

The terms used in the document and their respective definitions are given as below:

• **Term Frequency** - Defines the number of times a word occurs in a given document.

• **Tokens** – Each word is taken as a token for processing the document.

• **Stop word** – These words are usually the connectors and should not be considered in while processing a document. These will be removed in the pre-processing stage of the project.

• **Inverse Document Frequency** – Given a set of documents, idf of a word gives the number of documents in which the term occurs.

• **Document Frequency** -It basically tells the number of documents in which the word appears.

• **Collection Frequency**-It describes the number of times the word has appeared across all the corpus.

**1.3 System Architecture:**

The project is divided into several modules which take care of pre-processing, tokenizing, removal of stopwords, calculating the term frequency and inverse document frequency for all the terms in the documents.

We have not used Boolean model and explored the use of Vector space model, which is representation of a set of documents as vectors in a common vector space and is a major part to a host of information retrieval operations ranging from scoring documents on a query, document classification and document clustering.

Using the single word queries gives results easily and in a way the output for single word queries are generalized and doesn’t present much important information retrieval across corpus.

In the pre-processing module, all the documents are taken to retrieve the tokens in each of them which are stored in an array and the words which are identified as stopwords by NLTK package in python. Stopwords are removed from the array as they carry negligible importance in this project. The remaining tokens are then sent to remove the punctuations to get the primitive form of the token for effective computation of tf and idf. On each document, tf is calculated for each term and stored which is further used in calculating tf-idf score

Query can be given in the form of a text file or as raw text as input. All the pre-processing steps are applied on query and the tf-idf vector of the query is also computed. The score is calculated by using cosine similarity between the query input and the pre-processed data and while giving output we give output ranked by tf-idf score.

A screenshot of a cell phone

Description automatically generated

Data Structures used:

1) Dictionary

2) List

3) Set

4) Matrix(2-d Array using Numpy)

**Data pre-processing:**

* Dataset consists of approximately 2300 documents.
* The dataset is read using the file i/o stream. The ‘os’ library is used for fetching the current working directory path and iterating through each file.
* The data is then read and pre-processed. Pre-processing involves first lowercasing the article, using nltk library to tokenize the article, removing stop words and punctuations.
* Next, once we have all the unique words present in the corpus, we compute the tf, idf vectors which correspondingly gives us the tf-idf vectors for each document
* Tf score is computed simply by using this formula: - (No of a particular term/No of terms in a document after pre-processing). The idf score is computed using (log (Total number of documents/Document frequency of term)).
* The tf-idf score vector for the documents is stored in the form of a matrix where the indexing of matrix was enumerated by the document names.
* The input query is in the form of a file or raw data is also similarly pre-processed. Along with this, a spell checker function is implemented which automatically corrects a wrong word in the query, if any.
* Similarly, tf-idf score of the query is also computed

**Computations involved**

* Once we have the tf-idf score vectors ready for all the documents and the user query, the next step is to find those documents whose plagiarism percentage is in the limits given by the user
* For this purpose, we will be computing the cosine similarities between tf-idf vector of the query and for every document in the dataset. Cosine Similarity is simply the normalised dot product value between the query and document vector.
* Once the cosine sums are computed, we sort the values in decreasing order and return those documents whose plagiarism percentage is in the limits given by the user
* Running Time for pre-processing the dataset and storing all the unique words of corpus- 8.770602 seconds
* Running Time for computing tf vectors and idf vectors 1.4433964 seconds
* Running Time for computing tf-idf vectors of all documents 0.9638842
* Running Time for computing cosine sums and returning the desired results- 0.826724seconds
* If the user enters same query many times, all these computations will have to be done every time, in order to give the same result each and every time.
* To overcome this, we have implemented a **caching mechanism**, wherein the results of each query are stored in a file using the picklifying concept.
* When the user enters a query, the first step is the check in this text file if the result for the same query exists or not.
* If it is found, the process ends there itself and prevents unnecessary computations.

**Results obtained for same queries**

|  |  |  |  |
| --- | --- | --- | --- |
| Text File | Plagriasm Percentage | Time without caching mechanism (in sec) | Time with caching mechanism (in sec) |
| 2.txt | 0.1% | 0.8267241 | 0.0169629 |
| 4.txt | 12% | 1.3291655 | 0.0389891 |
| 1000.txt | 0% | 0.8942894 | 0.0803612 |
| 1234.txt | 2.9% | 0.7093315 | 0.0210634 |

As we can easily see from the above table that the time taken to give the output to a particular query is reduced to a significant amount with the introduction to the caching mechanism. This mechanism will be more effective when the data set is too big to handle as due to it, we are skipping the calculations for a query which has been already queried.

**Disadvantages of using Tf-IDF Model**

The following cases which won’t be successfully recognized by the above model: -

1. The tf-idf model is weak in handling synonyms, for example if we take 2 text – i) U.S President speech in public and ii) Donald Trump presentation to people will result in a zero value when cosine similarity is applied on them but actually these 2 sentences means the same thing but they are written in different forms.
2. This model is also weak on capturing document topics, for example if we take 2 text- i) The food was very good ii) The pizza was brilliant. As we can see that the similarity score will be zero in these two texts even though these 2 texts are closely related as pizza is a subdomain of food. This model doesn’t get the main gist of any document.
3. This model is only based on terms but actually in real life situation this will fail badly