INFORMATION RETRIEVAL (CS F469) Assignment I

(Domain Specific Information Retrieval System)

Group Members:

2015A7PS0119H	Afroz Ahamad Siddiqui
2015A7PS0121H	Tilak Mundra
2015A7PS0145H	Ankit Anand
2015AAPS0253H	Tanveer Singh Hora

Obtaining Dataset:

The Dataset for this Information Retrieval System is Content Management System(CMS) of BITS Pilani Hyderabad Campus. The dataset contains all files uploaded by instructors.

Functionality Implemented:

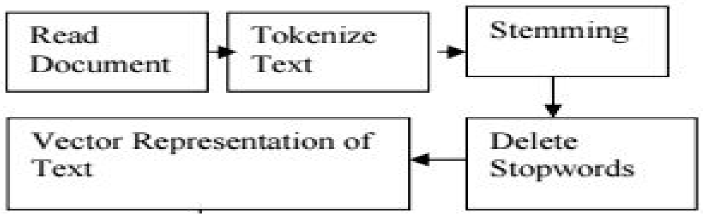
- Searching top files based on query given
- Comparing relevance between two files
- Text extraction from files of various formats – pdf, pptx, docx etc.

Data Structures Used:

- Lists in python – List in python were used to manipulate data.
- Dictionary in python - Dictionary in python is equivalent to Hash tables

Detailed documentation of the data structures can be found in code in specific methods and classes.

The flow of our information retrieval system is depicted in brief by the image shown below:



After we have the file and query vectors, the ranking model can easily be displayed.

Below are the steps involved in detail:

1. Preprocessing

Preprocessing comprises of one or more than one steps of processing on the available data so that it is converted to some usable form. Specifically to our project on Text Classification, a number of preprocessing steps are involved which consists of Extracting text from documents of various formats, Tokenization (N-grams technique), Stopwords Removal, Stemming, Vectorization of features and finally using TfIdf on Vectors as part of feature transformation. A detailed analysis of the above steps will follow next.

1.1 Tokenization

Prior to tokenization, text is extracted from documents and stored in files. In lexical analysis, tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens. The list of tokens becomes input for further processing such as parsing or text mining. Tokenization is useful both in linguistics (where it is a form of text segmentation), and in computer science, where it forms part of lexical analysis.

1.2 Removal of Stopwords

A **stop word** is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. We would not want these words taking up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to be stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages.

1.3 N-Grams Technique

N-grams of texts are extensively used in text mining and natural language processing tasks. They are basically a set of co-occurring words within a given window and when computing the n-grams you typically move one word forward (although you can move X words forward in more advanced scenarios). For example, for the sentence *"The cow jumps over the moon"*. If N=2 (known as bigrams), then the n-grams would be:

- the cow
- cow jumps
- jumps over
- over the
- the moon

So you have 5 n-grams in this case. Notice that we moved from the->cow to cow->jumps to jumps->over, etc, essentially moving one word forward to generate the next bigram.

If N=3, the n-grams would be:

- the cow jumps
- cow jumps over
- jumps over the
- over the moon

So you have 4 n-grams in this case. When **N=1**, this is referred to as **unigrams** and this is essentially the individual words in a sentence. When **N=2**, this is called **bigrams** and when **N=3** this is called **trigrams**. When N>3 this is usually referred to as four grams or five grams and so on.

We have used 3 grams in our program.

1.4 Stemming

Stemming is the process for reducing inflected words to their word stem. Consider an example of a data consisting of two set of words as follows:

Example:
Banks, Banking, Banker,
Investing, Invested, Investment

Clearly, if we treat every word above as different, it would not only
consume a lot of extra memory but also make us compromise on the processing time. So, in the above example, Banks, Banking become bank, while investing, invested and investment become invest. The classifier doesn't understand that the verbs investing and invested are the same, and treats them as different words with different frequencies. By stemming them, it groups the frequencies of different inflection to just one term — in this case, invest.

2. Feature Extraction – Tf Idf Metric

What is tf-idf frequency:

Tf-idf stands for *term frequency-inverse document frequency*, and the tf-idf weight is a weight often used in information retrieval and text mining. 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. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

How to Compute:

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

- **TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

- **IDF: Inverse Document Frequency**, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

- IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Example :-
Consider a document containing 100 words wherein the word *cat* appears 3 times.
The term frequency (i.e., tf) for *cat* is then (3 / 100) = 0.03. Now, assume we have 10 million documents and the word *cat* appears in one thousand of these.
Then, the inverse document frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4.

Thus, the Tf-idf weight is the product of these quantities:
tf*idf = 0.03 * 4 = 0.12.

Code Execution

The details on setting up the code and followed by its usage can be found in the README file attached.