**Project 2**

**Measuring document to collection and query to document similarity with evaluation**

**Abstract:**

The purpose of this project is to find the similarity for a given document to all the documents in collection and retrieving the relevant documents at top. Besides, to find the similar documents in collection for a given query. Firstly, I have accessed the single text file which consists of set of documents delimited by “\\\\\”, and split them into separate text files. Then basic concepts of pre-processing like tokenizing, filtering the stop words, stemming, lemmatization are performed on document collection. Next, tf-idf (term frequency\_inverse document frequency) weights are calculated for all the terms in each of the document for whole collection. Later, the cosine similarity between the documents is measured by taking weights of tf-idf and inputting to cosine functions, through this document to document similarity and query to document similarity are achieved likewise. Finally, by the similarity score these documents are ranked after which precision and recall are calculated for checking effectiveness/performance.

**Introduction and Goals:**

To start with natural language processing, where I am using natural language toolkit (nltk) which is compatible with python as a base platform for this project. One of the basic concepts of information retrieval is to find the tokens in a given document or sentence. Tokenization is a process used for dividing a set of sentences in document or corpus into a bag of words, by considering each word in document separated by space as a token with some exceptions such as excluding “”, ;, etc., In NLTK, for tokenizing, we have few pre-defined techniques which need to be imported to tokenize the text into words or sentences using “word\_tokenizer” or “sent\_tokenizer” respectively.

Then, for finding which word has repeated how many times, we use frequency distribution terminology where the frequency of each token/words in document can be calculated by importing “freqdistribution” from nltk. Stop word filtering is used for excluding the unnecessary words such as the, of, an, and, with, by, etc., which has low importance while calculating the document similarity from given documents or query by importing “stopwords”.

As we know for sure we gonna take each word/term in a document or query and compare with terms in other documents for checking the similarity. The ambiguity here is that regularly we see that each word has many verb forms such as if we consider play, the verb forms will be played, played, playing. So, for better applicability of searching, we either stem or lemmatize the terms in a document or query. By applying stemming, we will just get the root word which is play instead of played or playing in our example. Lemmatization captures the morphological structure of given term. We have pre-defined stemming algorithms such as PorterStemmer, LancasterStemmer and wordnet lemmatizer for lemmatization and can be applied based on our need.

By having the basic concepts in place, we apply the tf-idf concept where the weights of each term in a given query or document is calculated, tf gives the frequency of a term in particular document or query and inverse document frequency is knowing number of documents contain the term and applying logarithm to it for normalizing, so that the importance of particular term in the whole corpus can be calculated by using “TfidfVectorizer”. Later, Cosine similarity needs to be calculated for finding the document to document similarity and to find the relevant document for a given query by using “cosine\_similarity”. Finally, Need to rank the documents based on relevancy for given query or document and retrieve the results with similar documents.

The main goals of this experiment is to find the following measures:

* When a documents is given, which documents would be the similar in the whole collection of documents.
* When a query is given, which documents will be relevant to the query in the collection.
* Then find the precision and recall of the retrieved results(documents) through which we can find how effective our techniques are.

**Experiment setup and Expectations:**

First of all the primary purpose was to divide the text files (mnrs.txt, snrs.txt) which contains documents delimited by “\\\\\” into separate set of documents (1.txt,2.txt,…431.txt) and stored in “text” folder. Then, import all the required corpora like nltk, math, pandas, sklearn etc., and read through all the separated documents and stored into corpus with file\_id’s. Then in the pre-processing task did tokenization for all the text files in corpus. Followed by stop word filtering.

Later, have lemmatized the words so that it will be feasible for searching in the documents with the morphological words rather than facing difficulty in searching for exact words in document collection. As per the prior knowledge I have by reading notes and nltk books, I used lemmatization instead of stemming since if a word “saw” is processed for stemming and lemmatization, we get “s” and “see” as output respectively. From which we can inherit that lemmatization is working better than stemming. Will check final results and come to conclusion on how do they fare.

Next, from the pre-processed text went on to work with tf-idf where all the terms weights for each file are calculated. Then a 2d array has been formed by using dataframe from pandas to visualize the weights of terms with respective to the files (431 files). Then considered queries “Premature children” and “Gynecological cancer” and few others for which I expected to find the similar documents in the whole collection (text files) by using cosine similarity between query and all documents in collection. Similarly for a given a document, similar documents in the collection is expected to be found by using cosine similarity between given document and whole collection. I am expecting that if I consider a document, then the more similar document in the whole corpus will be itself. Finally, retrieve the documents with similarity weights greater than some required value (which would be mostly between 0 and 1 since cosine values range from 0 to 1) and evaluate them by using precision and recall.

**Results and analysis:**

I will start with the same order as it is mentioned in project assignment.

**A.** work with tf-idf

Used “TfidfVectorizer” and found the tf-idf weights for all the terms in a given query or document and used those weights further for calculating the similar documents. As said before, 2d array with tf-idf weights for each document with respect to its terms is calculated and this matrix rows and columns can be known by using command “df.shape”. Tf-idf weights of a particular term in all the documents can be found by using command “df[“term”]”. For example, when I use df[“benefit”], I got the tf-idf weights of term benefit in all 431 documents. In order to calculate the tf-idf weights of all the terms in a particular documents, we can use command tfs[0] for tf-idf weights of terms in document 1 (1.txt) where “0” is document id which can be varied from 0 to 430 as there are total of 431 documents, but the indices start from 0.

**B.** Work with plain words, stop word filtering, and stemming.

In order to perform stemming or lemmatization, we need to split the text in the documents to build plain words by eliminating the special characters such as $, “”, ;, :, \, / by using tokenization. The words with low importance are removed from the documents by using stop words filtering. Found that words such as the, an, is, with, by, on, in, etc,.. were eliminated. Then either stemming or lemmatization can be performed. In my experiment, I found that lemmatization was working effectively compared to stemming since by stemming words essence is going missing. For example, recently is changed to rec, genetic is changed to genet, etc., whereas from lemmatization, recently was unchanged, genetic was changed to gene. Henceforth, I have used lemmatization instead of stemming.

**C.** Queries “Premature children” and “Gynecological cancer”

Once a query is given as input, query also need to be pre-processed which means that splitting (or) tokenization, stop words filtering, lemmatization are applied so that there will be more chance to get good results. When query “Premature children” is given as input to find the relevant documents in collection, it has been changed to “premature child” after pre-processing. The retrieved documents are 46.txt, 55.txt, 79.txt,… which were having cosine similarity score more than 0.15. 46.txt was retrieved since the word child was present many times.

When query “Gynecological cancer” is taken as input which is unchanged after pre-processing and the documents which have similarity score more than 0.15 are 2.txt, 72.txt, 122.txt, 363.txt,…. 363.txt was retrieved because the word “cancer” is repeated many times in that document. When I input “benefit and burden”, it’s been pre-processed to “benefit burden”. The retrieved documents are 1.txt, 309.txt, 401.txt,.. with similarity more than 0.15. Since the 1st document contains the heading benefit and burden, it’s been retrieved along with few other documents. When query “women abuse” was considered, after pre-processing it’s been changed to “woman abuse”. The retrieved documents are 48.txt, 59.txt, 90.txt,… which have similarity more than 0.15. 59.txt was retrieved since this document has word “woman”.

When I input “important behavior” which was unchanged after pre-processing. The documents which have similarity score more than 0.2 are 155.txt and 417.txt. Since these documents have the terms “behavior” many times. When I input “health care center” which hasn’t been changed after pre-processing. The retrieved documents are 71.txt and 174.txt when the similarity score is more than 0.22. Since these documents include health and care words in them they are retrieved. When the similarity score was more than 0.20, many documents were retrieved for the same health care center, which shows that this health care is present in most of the documents.

**D.** Using VSM to find documents similar to given document

When a document is considered, the cosine similarity of the given document with respect to all the documents in collection is calculated. When I consider first document “1.txt” which is tfs[0] and checking the similarity with the documents in collection, then documents 1.txt, 309.txt, 352.txt are retrieved which have cosine similarity score more than 0.18. These documents are retrieved since 1.txt will be obviously similar to tfs[0] which are one and the same. The document 309.txt is retrieved because of words study, prior, describe, etc., being in common. The document 352.txt was retrieved since the words children, family, adult, participations, identify, etc., are common in both documents. Similarly, for checking the similar documents for a given document, we can run the similarDocs line in the program by changing the value from 0 to required document id – 1 since the document id starts from 0 and ends with 430. To list the similar documents we can run the command “(list(zip…” with required similarity score.

**E.** top-k with precision-recall curves

Finally, for a given query, similar documents are sorted in ascending order and the top-10, top-5, top-3 ranked similarity scores are retrieved. When I consider “premature child” query, the top-10 similar documents with cosine similarity score greater than 0.22 are 55.txt, 79.txt, 101.txt, 146.txt, 214.txt, 251.txt, 282.txt, 347.txt, 348.txt, 349.txt. The top-5 similar documents are 55.txt, 101.txt, 214.txt, 348.txt, 349.txt with similarity score more than 0.24. When top-3 documents are considered, 101.txt, 214.txt, 348.txt are retrieved which has similarity score more than 0.30. For top-10, precision was 40% since 4 out of 10 retrieved documents are relevant. For top-5, precision was 40% since 2 out of 5 were partially relevant. For top-3, precision was 66.67% since 2 out of 3 were partially relevant.

When I consider the query “health care center”, The top-10 retrieved documents are 14.txt, 38.txt, 57.txt, 71.txt, 174.txt, 283.txt, 287.txt, 342.txt, 345.txt, 407.txt where similarity score was greater than 0.152. The top-5 retrieved documents are 71.txt, 174.txt, 283.txt, 345.txt, 407.txt and the similarity score was greater than 0.205. The top-3 retrieved documents with similarity score more than 0.215 are 71.txt, 174.txt, 283.txt. In these top-10 documents precision is 70%, top-5 precision was 80% and top-3 precision was 100%.

For these queries, I was not able to calculate recall, since for recall we need to find the relevant documents that are retrieved. As, the not-retrieved documents from the whole collection are more than 420, for each of which it is not possible to check if it’s relevant to particular query or not. If the provided document collection was trained and pre-classified as either relevant or non-relevant then it will be easier to calculate the recall and further can draw precision-recall curves.

Similarly for a given document, similar documents in collection with top-5, top-3, top-1 ranked cosine similarity scores are retrieved. When I consider the 1st text document, The top-5 similar documents that are retrieved with cosine similarity score greater than 0.173 are 1.txt, 235.txt, 309.txt, 352.txt, 386.txt. The top-3 similar documents with score greater than 0.18 are 1.txt, 309.txt, 352.txt. As expected, the top-1 document is the document itself (1.txt) which has more similarity score than all others. The top-5 precision is 40% since 2 out of 5 are relevant. The top-3 precision is 66.67% as 2 out of 3 are relevant. Top-1 precision is 100% since the same document is retrieved which is part of the collection. As said above recall isn’t easier to calculate in this case as well.

**Conclusion:**

Finally, after a lot of efforts in understanding and getting adjusted to python and tools like nltk, pandas, numpy, accomplished the main goals of this project. I understood how good and efficient the tools like nltk, numpy, pandas, sklearn are for working on natural language processing and information retrieval concepts. By having the inbuilt features like TfidfVectorizer, cosine\_similarity, it was easier to work on vector space model (VSM) model. After completing the project, I have complete idea on each of the basic concepts of information retrieval and about applying VSM for finding similar documents.

The precision was not that great when I consider top-10 or top-20 similar documents, but it was fine when top-5, top-3, top-1 are considered. Need to find a better way in order to get higher precision which have utmost importance while checking the performance of the VSM model or any machine learning algorithm in general. However, recall wasn’t been calculated because of the un-trained data provided. When I consider an inbuilt corpora from nltk like movie\_reviews, Gutenberg, etc., it is further more easier to work with pre-processing and similarity finding. I am intrigued by the idea/concept of support vector machines and its kernel tricks. Will probably work on it and know in detail about how does it works by doing a mini project in the rest of the summer so that I can get more accustomed to python as well.