Documentation Q1

Problem Statement

In the given problem statement, we had to implement 2 word scoring or term weighting methods; jaccardian and tf-df. The given dataset consisted of files having various conversations and dialogues which needed to be imported and preprocessed before applying any computation. This required thorough reading and discarding the file which were not encoded in utf-8.

After the successful import, we then would convert this data into a data structure (given unigram inverted index) which will be used to solve the following queries. The queries will be input in the manner given in the doc.

The function **importDocument** imports all the files from the given dataset and discards the file which does not follow the utf-8 encoding. This results in **5** discarded files and **1128** total processed files. All the files are stored in the dictionary **documents** and the name of the files in **files**. Each value in the dictionary is a 2d object with ith iteration containing the lines and the jth words in each line.

Preprocessing

Preprocessing steps have been carried out exactly in the previous assignment manner where we first make the data uniform by removing all the unwanted symbols. This leaves us with only alphanumeric values and then we lowercase them, remove stop words, lemmatize and finally remove any other unwanted underscores.

Jaccardian

Jaccardian coefficient is given by:

Jaccard Coefficient = Intersection of (doc,query) / Union of (doc,query)

Our data structure **documents** is a dictionary type object having a 1D matrix of all the words in the specific doc. This variable will be used for union and intersection with the given query. We first preprocess the query using the same preprocessing steps used for the previous assignment and then take union and intersection of the query list and **documents**. This gives the jaccardian score for all the documents which is then sorted in descending order. We finally report the top 5 documents for the query

Sample Output

```
Enter query : American dream
Top 5 documents
p-law.hum       0.012658227848101266
carowner.txt    0.012345679012345678
```

vonthomp 0.011363636363636364
psalm.reagan 0.010309278350515464
psalm_nixon 0.010309278350515464

TF-IDF

First we find out the number of unique words in all the documents and frequency of each word in every document (bag of words for each document). This way, our computation will reduce for calculating the tf-idf score. According to the question we had to implement 5 types of ways of calculating the term frequency value for each word in the document. A function is built which uses nested else if to calculate the score on the basis of type of tf score. Inverse document frequency is independent of the unique words or bag of words. It is somewhat similar to the assignment 1 task of calculating each occurrence of a word in a document. After finding both the values, a nested dictionary is used to calculate the tf-idf score. Formula used:

Term Frequency :-

Binary : 0,1
Raw count : f(t,d)
Term frequency : f(t,d)/Pf(t‘, d)
Log normalization : log(1+f(t,d))
Double normalization : 0.5+0.5*(f(t,d)/ max(f(t‘,d))

Inverse Document Frequency :-

IDF(word)=log(total no. of documents/document frequency(word)+1)

TF-IDF score :-

TF-IDF(doc, word) = TF(doc, word)*IDF(word)

Sample Output

Enter query : sherlock holmes
Top 5 documents
Binary:
acronyms.txt 3.900498216638722
collected_quotes.txt 3.900498216638722
consp.txt 3.900498216638722
crzycred.lst 3.900498216638722
insult.lst 3.900498216638722

Raw count:
collected_quotes.txt 9.649185550268843
consp.txt 9.649185550268843
blake7.lis 7.392756467965595
hackmorality.txt 5.544567350974196

acronyms.txt 3.900498216638722

Term frequency:
livnware.hum 0.008209236398589293
blake7.lis 0.004475034181577237
mov_rail.txt 0.002762392504701645
subb_lis.txt 0.0025934163674459585
oxymoron.jok 0.0022294199239944494

Log normalization:
collected_quotes.txt 2.0919210164445876
consp.txt 2.0919210164445876
blake7.lis 1.2918287551172607
acronyms.txt 1.1741669612421208
crzycred.lst 1.1741669612421208

Double normalization:
collected_quotes.txt 2.052900018428604
consp.txt 2.0320218672199446
crzycred.lst 1.999005336027345
subb_lis.txt 1.9766038259993524
mov_rail.txt 1.9691835656816852