IR Assignment 2

Methodology Followed for Solving Question 1 (Part A) Jaccard's Coefficient:

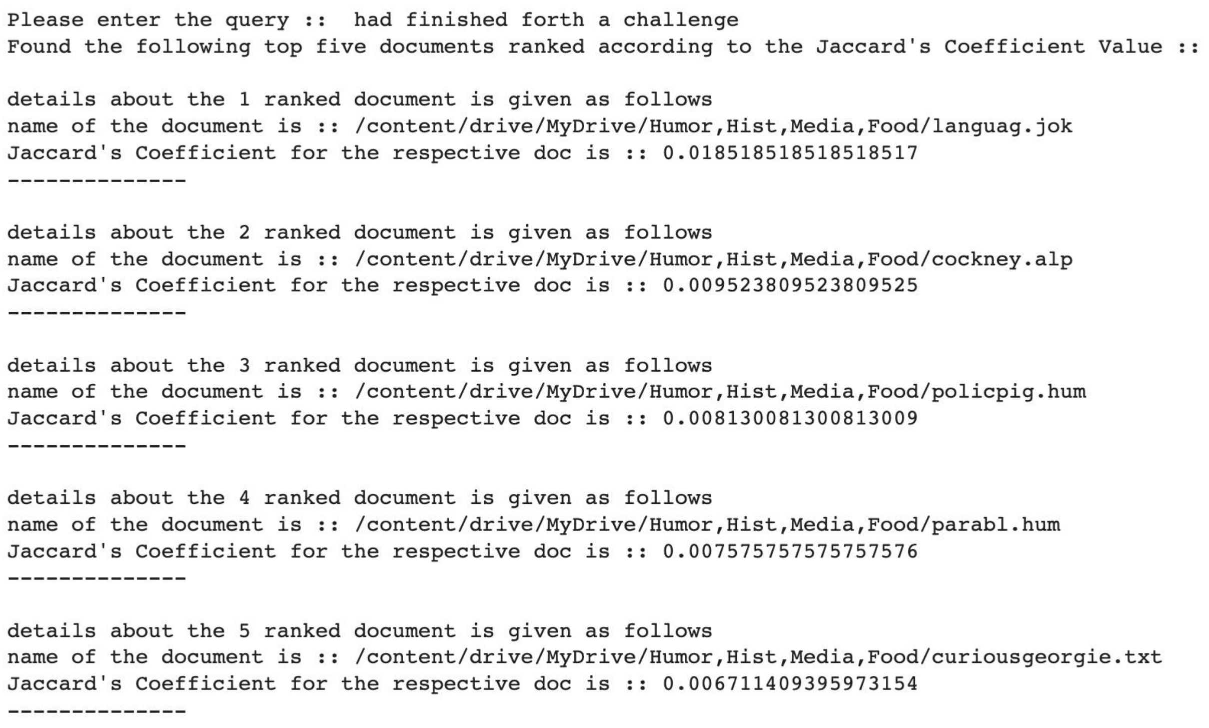
Methodology followed for getting the Top 5 relevant documents based on Jaccard's Coefficient is written as follows:

1. Pre-processed all the files while iterating over each line of all the files, steps for which are given in the next section.
2. Created Dictionary for storing docID as the key and list of words in that document.
3. As we need to calculate Jaccard's Coefficient from the list of words hence, removed duplicates from the list of words from all the docID's list of words.
4. While iterating over each value in the dictionary, calculated jaccard's coefficient for each document using below formula and stored the value as key-value pair in a dictionary where, docID is the key and it's jaccard's coefficient as the value.

   jaccards_coeff_dict[key]= (intersection of (doc, query))/ (union of (doc, query))

5. Finally, using the jaccard's value we sorted the values and found the top five jaccard's value and their corresponding keys, which are their docID.
6. Using the doc-docID mapping got the top 5 relevant documents based on the value of the Jaccard coefficient.

Like the below sample screenshot we got the ranked document according to their Jaccard's Value:



```
Please enter the query ::  had finished forth a challenge
Found the following top five documents ranked according to the Jaccard's Coefficient Value ::

details about the 1 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/languag.jok
Jaccard's Coefficient for the respective doc is :: 0.018518518518518517
--------------

details about the 2 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/cockney.alp
Jaccard's Coefficient for the respective doc is :: 0.009523809523809525
--------------

details about the 3 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/policpig.hum
Jaccard's Coefficient for the respective doc is :: 0.008130081300813009
--------------

details about the 4 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/parabl.hum
Jaccard's Coefficient for the respective doc is :: 0.007575757575757576
--------------

details about the 5 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/curiousgeorgie.txt
Jaccard's Coefficient for the respective doc is :: 0.006711409395973154
--------------
```

Methodology Followed for Solving Question 1 (Part B) TF-IDF Matrix:

Methodology followed for getting the Top 5 relevant documents based on different weighting scheme is as follows:

1. Pre-processed all the files while iterating over each line of all the files, steps for which are given in the next section.
2. Created Dictionary for storing docID as the key and list of words in that document. Here, we did not removed the repetitions from the list of words as repetitions are counted here.
3. Combined all the words from all list of words from each document and then, selected unique words, as these would act as the columns each of the weight matrix.
4. We created 5 different matrices each of size (number of documents)X(corpus of unique words from all documents).
5. Filled each of matrix while iterating over each cell at the respective matrix and using the appropriate formula as per the matrix, filled using below mentioned formula's:

Weighting Scheme	TF Weight
Binary	0,1
Raw count	f(t,d)
Term frequency	$f(t,d)/\sum f(t', d)$
Log normalization	$\log(1+f(t,d))$
Double normalization	$0.5+0.5*(f(t,d)/\max(f(t',d))$

Using the respective formula we found five asked frequency matrices.

6. Then, we created one matrix for the inverted document frequency with the size of (1 X corpus of unique words from all documents) and filled each of the value of matrix using the formula

$$\text{IDF(word)}=\log(\text{total no. of documents/document frequency(word)}+1)$$

7. Then, multiplied each row of frequency matrix with the IDF matrix and got the respective 5 multiplied matrix corresponding to each weighting scheme.
8. Calculated TF-IDF Score for each of the document using row wise sum for each of the document for each of the weighting scheme.
9. Then, for each of the weighting scheme got the top ranked document in it by sorting the values and getting their corresponding docID's.
10. Then, mapped those docID with the docName and displayed with the score.

Output for which is shown as follows:

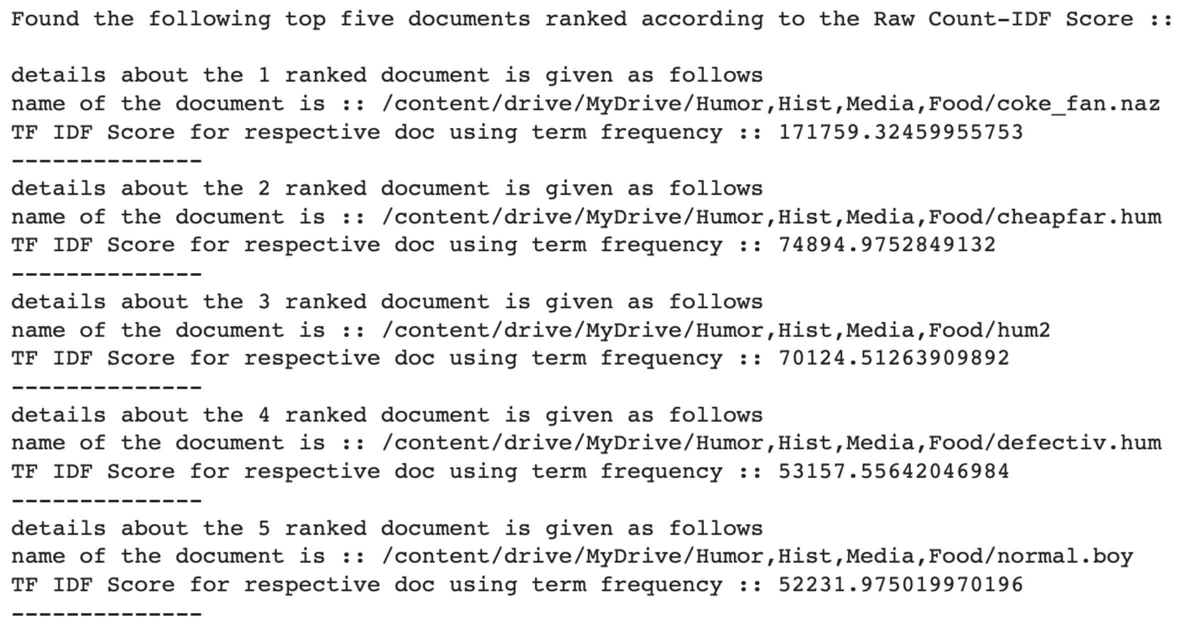
Term Frequency Output:

```
Found the following top five documents ranked according to the TF IDF Score ::

details about the 1 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/japantv.txt
TF IDF Score for respective doc using term frequency :: 8.465813701323276
--------------
details about the 2 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/lp-assoc.txt
TF IDF Score for respective doc using term frequency :: 7.9762603790843
--------------
details about the 3 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/prover_w.iso
TF IDF Score for respective doc using term frequency :: 7.233163552199315
--------------
details about the 4 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/bb
TF IDF Score for respective doc using term frequency :: 7.037122375818296
--------------
details about the 5 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/some_hu.mor
TF IDF Score for respective doc using term frequency :: 6.932742616335451
--------------
```

Binary Term Frequency Output

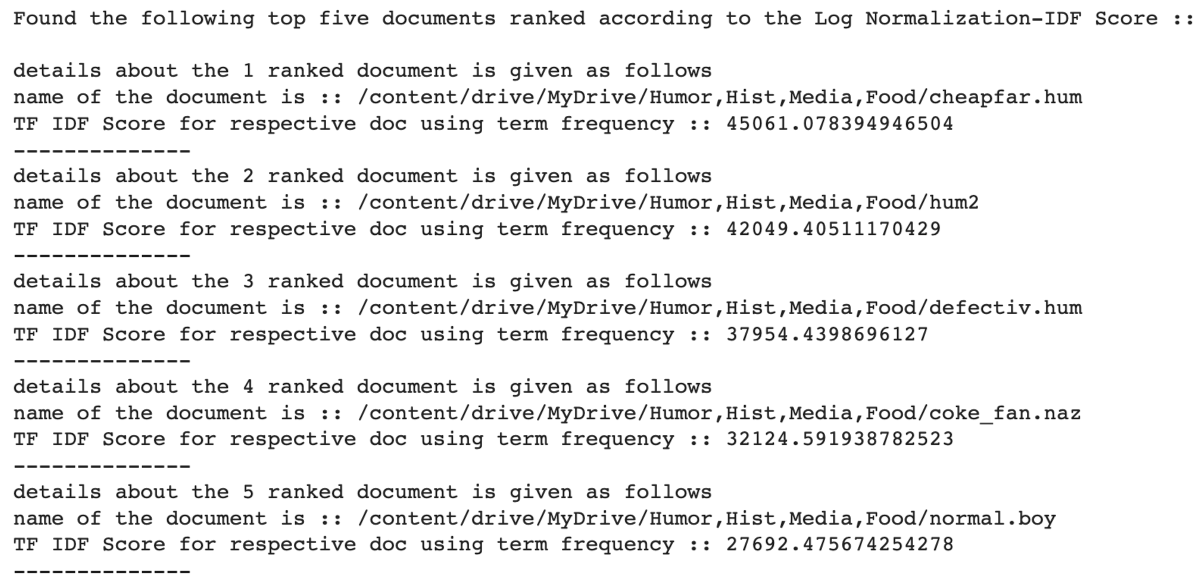
```
Found the following top five documents ranked according to the Binary Term IDF Score ::

details about the 1 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/mlverb.hum
TF IDF Score for respective doc using term frequency :: 31109.159692918016
--------------
details about the 2 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/acronyms.txt
TF IDF Score for respective doc using term frequency :: 29787.334611576873
--------------
details about the 3 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/humor9.txt
TF IDF Score for respective doc using term frequency :: 28758.770734606267
--------------
details about the 4 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/practica.txt
TF IDF Score for respective doc using term frequency :: 18388.77878042205
--------------
details about the 5 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/candy.txt
TF IDF Score for respective doc using term frequency :: 17247.334026895012
--------------
```

Raw Frequency Output:



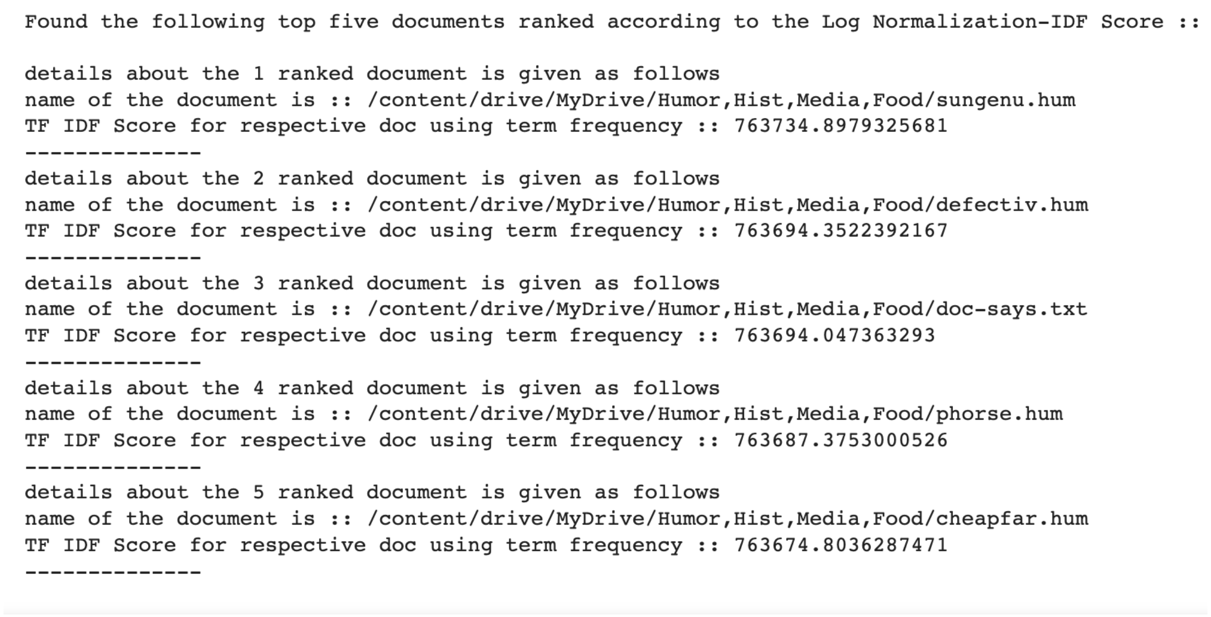
```
Found the following top five documents ranked according to the Raw Count-IDF Score ::

details about the 1 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/coke_fan.naz
TF IDF Score for respective doc using term frequency :: 171759.32459955753
--------------
details about the 2 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/cheapfar.hum
TF IDF Score for respective doc using term frequency :: 74894.9752849132
--------------
details about the 3 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/hum2
TF IDF Score for respective doc using term frequency :: 70124.51263909892
--------------
details about the 4 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/defectiv.hum
TF IDF Score for respective doc using term frequency :: 53157.55642046984
--------------
details about the 5 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/normal.boy
TF IDF Score for respective doc using term frequency :: 52231.975019970196
--------------
```

Log Normalization Output:



```
Found the following top five documents ranked according to the Log Normalization-IDF Score ::

details about the 1 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/cheapfar.hum
TF IDF Score for respective doc using term frequency :: 45061.078394946504
--------------
details about the 2 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/hum2
TF IDF Score for respective doc using term frequency :: 42049.40511170429
--------------
details about the 3 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/defectiv.hum
TF IDF Score for respective doc using term frequency :: 37954.4398696127
--------------
details about the 4 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/coke_fan.naz
TF IDF Score for respective doc using term frequency :: 32124.591938782523
--------------
details about the 5 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/normal.boy
TF IDF Score for respective doc using term frequency :: 27692.475674254278
--------------
```

Double Log Normalization Output:

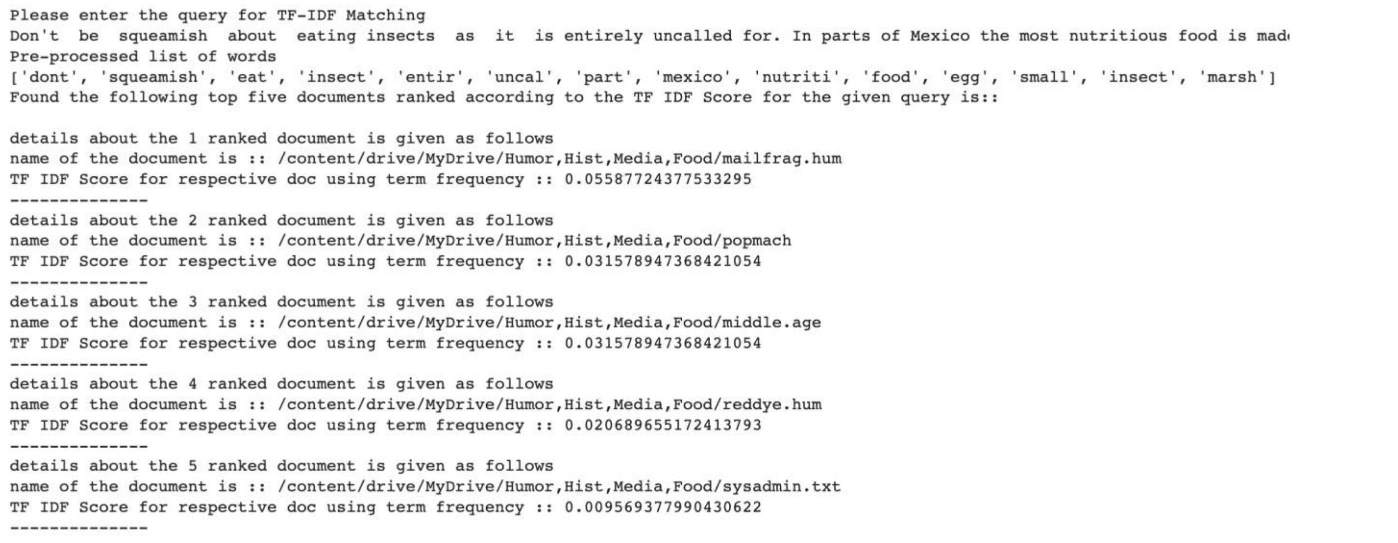


```
Found the following top five documents ranked according to the Log Normalization-IDF Score ::

details about the 1 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/sungenu.hum
TF IDF Score for respective doc using term frequency :: 763734.8979325681
--------------
details about the 2 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/defectiv.hum
TF IDF Score for respective doc using term frequency :: 763694.3522392167
--------------
details about the 3 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/doc-says.txt
TF IDF Score for respective doc using term frequency :: 763694.047363293
--------------
details about the 4 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/phorse.hum
TF IDF Score for respective doc using term frequency :: 763687.3753000526
--------------
details about the 5 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/cheapfar.hum
TF IDF Score for respective doc using term frequency :: 763674.8036287471
--------------
```

Methodology followed for ranking of documents using TF-IDF Score for user input query using the TF-IDF matrix:

1. Pre-processed the user input query and stored in form of a list.
2. On basis of the formed list and earlier created TF-IDF matrix created a dictionary with docID as the key and score for the respective document as it's value, score is calculated using the summation of TF-IDF matrix for the selected document and all the words in the pre-processed query.
3. For the created dictionary, sorted the values and ranked them and got their respective keys, then using the docID-docName mapping, mapped those docID's with their names and displayed relevant documents to the user.

Sample output for which is shown as follows:



```
Please enter the query for TF-IDF Matching
Don't  be  squeamish  about  eating insects  as  it  is entirely uncalled for. In parts of Mexico the most nutritious food is mad
Pre-processed list of words
['dont', 'squeamish', 'eat', 'insect', 'entir', 'uncal', 'part', 'mexico', 'nutriti', 'food', 'egg', 'small', 'insect', 'marsh']
Found the following top five documents ranked according to the TF IDF Score for the given query is::

details about the 1 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/mailfrag.hum
TF IDF Score for respective doc using term frequency :: 0.05587724377533295
--------------
details about the 2 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/popmach
TF IDF Score for respective doc using term frequency :: 0.031578947368421054
--------------
details about the 3 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/middle.age
TF IDF Score for respective doc using term frequency :: 0.031578947368421054
--------------
details about the 4 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/reddye.hum
TF IDF Score for respective doc using term frequency :: 0.020689655172413793
--------------
details about the 5 ranked document is given as follows
name of the document is :: /content/drive/MyDrive/Humor,Hist,Media,Food/sysadmin.txt
TF IDF Score for respective doc using term frequency :: 0.009569377990430622
--------------
```

Pros and Cons of using each scoring scheme to find the relevance of documents

1. Binary Weighting Scheme:

   Cons – Here, only presence and absence of document is considered to be there, no weightage is given to the frequency of token, even if a token is a present 100 times and one is present single time, then too both are given equal weightage.
   Pros – Easy to populate the table and useful for documents where each token is present at max single time.

2. Raw Count Weighting Scheme:

   Cons – No normalization applied to the frequency here, hence at times when the document size keeps on increasing then, to store large values in the database could become overhead over the memory.
   Pros – Frequency of the token is considered here, hence it overcame the con of binary weighting scheme as, token with more number of repetitions is given more weightage.

3. Log Normalization Weighting Scheme:

   Cons – No normalization inside the log performed only frequency taken into consideration.
   Pros – Best known weighting scheme in information retrieval increases with the number of occurrences within a document Increases with the rarity of the term in the collection.

4. Double Normalization Weighting Scheme:

   Cons – Unstable weighting scheme because change in stop word changes the weighting and ranking for the complete dataset.
   Pros – Normalization of frequency is taken into account hence, not much of a overhead from memory point of view because logarithm as well as normalization taken into consideration.

5. Term Frequency Weighting Scheme:

   Cons – Does not helps with the semantic meaning. Only considers importance based on the weighting scheme and not the semantic context.
   Pros – Computation quite cheap here because of the normalization done with the the help of sum of frequency of the all the tokens in the particular document. Also, this is weighting scheme which is mostly preferred for calculating similarity in query and pool of documents.

Methodology Followed for Solving Question 2 (Ranked-Information Retrieval and Evaluation):

Methodology followed for ranked information retrieval and evaluation is written as follows:

1. Read the file and stored the result in a dictionary after extracting relevance score, query id(with id as 4 as asked in the question) and feature-75 from it.
2. Populated dataframe with columns as relevance score, query-id and feataure-75 using the created dictionary.
3. While rearranging the query-url pairs using the sort function on the relevance score, got the new dataframe and stored in form of a CSV.
4. For counting the number of files which could be used we, first counted number of entries with each relevance score.
5. Stored factorial for each of the count in a separate list, then multiplied them and got the number of files which could be made while rearranging the query - url pairs in order of max DCG. Count for which is :

   19893497375938370599826047614905329896936840170566570588205180312704
   857992695193482412686565431050240000000000000000000000

6. Then, we found out DCG, IDCG and NDCG first for 50 documents and then, for the complete dataset, for this we iterated over the documents and calculated DCG and IDCG with the help of below mentioned formula:

   $$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

   Found the IDCG on the sorted dataframe and calculated it in the similar fashion. Finally, calculated NDCG using:

   NDCG=DCG/IDCG

   Got the value of NDCG at k=50 as – 0.3521042740324887
   Value of NDCG at complete dataset as – 0.5979226516897831

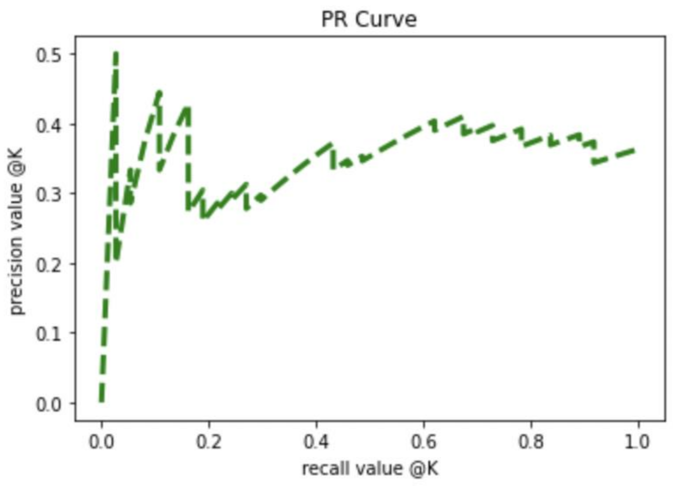
7. For plotting the precision-recall curve, for classifying whether the document is relevant or not, we set the threshold as average for all of the feature-75 values, below which all document is considered to be irrelevant and otherwise relevant.
8. Then, using the relevant and irrelevant documents, we calculated precision and recall at each level and stored both of them in list.
   For precision, used the below formula:

   $$P(\text{relevant}|\text{retrieved})$$

   For recall, used the below formula:

   $$P(\text{retrieved}|\text{relevant})$$

9. Using the matplotlib.pyplot plotted the graph between the precision and recall, and obtained the graph as follows:



Methodology Followed for Solving Question 3 (Implement Naïve Bayes Algorithm for text classification using TF-ICF):

The following functions were implemented:

- **preprocess(data):** takes the document data as a parameter and performs pre-processing of the data and returns the pre-processed tokenized data as output. The following steps are performed in the pre-processing:
  - ⇨ **Conversion to lowercase**
  - ⇨ **Removal of tags**
  - ⇨ **Punctuations and digits removal**
  - ⇨ **URL removal**
  - ⇨ **Removal of alphanumeric characters**
  - ⇨ **Lemmatization**
  - ⇨ **Stopwords removal**
  - ⇨ **Removal of unwanted spaces**

- **split(dataset,split_size):** takes the document dataset and the split size (in float, between 0 and 1) and randomly splits the dataset into train and test data. Returns the list of train and test documents.
- **Get_dataframe(path):** takes the path of the documents folder as input and creates a dataframe out of it and returns that dataframe.
- **findTF(path,train_set):** takes the path of the documents folder and the training document set as input and finds the Term Frequency of each unique word in the documents belonging to the training set. Returns the dictionary of unique words with their TF values.

- **calculate_TFICF(dictionary,vocab2,vocab3,vocab4,vocab5):** takes the vocabulary dictionary containing words with their TF values of one class and similar vocabulary dictionary of the remaining 4 classes and computes the ICF and the TF-ICF value for each word in the dictionary. Returns a dictionary containing the input dictionary words and their corresponding TF-ICF values.
- **effective_Vocab(dict1,dict2,dict3,dict4,dict5,k):** takes the dictionaries containing words with their TF-ICF values for each of the 5 classes along with 'k' as input and finds the top 'k' features of each class and creates a list of those features and finally returns it.
- **get_wordOccurance(path,train_set,effective_features):** takes the path of the documents folder, along with the training document set and the effective_features (containing the top 'k' features of all 5 classes) as input and creates a list of 0,1 values for each feature indicating the non-occurrence/occurrence of that feature in the training documents. Returns the list as output.
- **find_accuracy(y_true,y_pred):** takes the true class labels of the test set along with the predicted labels as input and computes the accuracy score and returns that value.
- **predict_NB(train_data,test_data,effective_features,train_spilt_ratio):** takes the training data, testing data, effective features (top k features) and the train split ratio as input and **implements the Naïve Bayes Algorithm.** It performs the following computations:
    - ⇨ **Computes prior probabilities of each class in the training dataset**
    - ⇨ **Divides the whole training dataset class wise**
    - ⇨ **Computes the conditional probabilities of each class of the training dataset**
    - ⇨ **Generates prediction labels for each test instance using the prior probabilities and the conditional probabilities, obtained earlier**

  Finally, it returns the predicted labels of the test set.

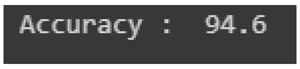
- **naive_bayes(k,train_split_ratio):** takes the train split ratio (between 0 and 1) and 'k' as input and performs the following computations:
    - ⇨ **Creates dataframe of the documents of each class using the get_dataframe() method.**
    - ⇨ **Splits the dataframe into train-test using the split() method.**
    - ⇨ **Obtain dictionary of unique words with their TF values for each class in the train data using the findTF() method.**
    - ⇨ **Obtain the dictionary consisting of TF-ICF values for each class using the calculate_TFICF() method and sort the dictionary using the TF-ICF values (descending order).**
    - ⇨ **Obtain top k features of each class using the sorted TF-ICF dictionary obtained earlier and combine all the features using effective_Vocab().**
    - ⇨ **Create list of word occurrences of the top k features for each document of each class for both the training and the testing set.**
    - ⇨ **Create the training and testing dataframe for all the classes using the word occurrences and the top k features with the documents as rows and top k features as columns of the dataframe.**
    - ⇨ **Obtain the predictions on the test set using the predict_NB() method.**

⇨ **Compute Accuracy score using the find_accuracy() method and create the confusion matrix and classification report.**

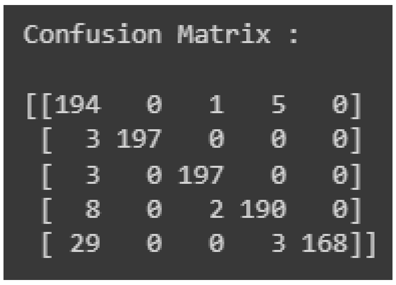
<u>NOTE</u>: - naïve_bayes() IS MAIN FUCNTION TO BE CALLED. PASS THE 'k' VALUE ALONG WITH THE TRAIN SPLIT RATIO (E.g.: - 0.8 for 80:20 split, 0.7 for 70:30 split) INTO THE FUNCTION AND OBTAIN THE OVERALL ACCURACY AND CONFUSION MATRIX.

<u>PERFORMANCE ON DIFFERENT TRAIN-TEST SPLITS</u>: -

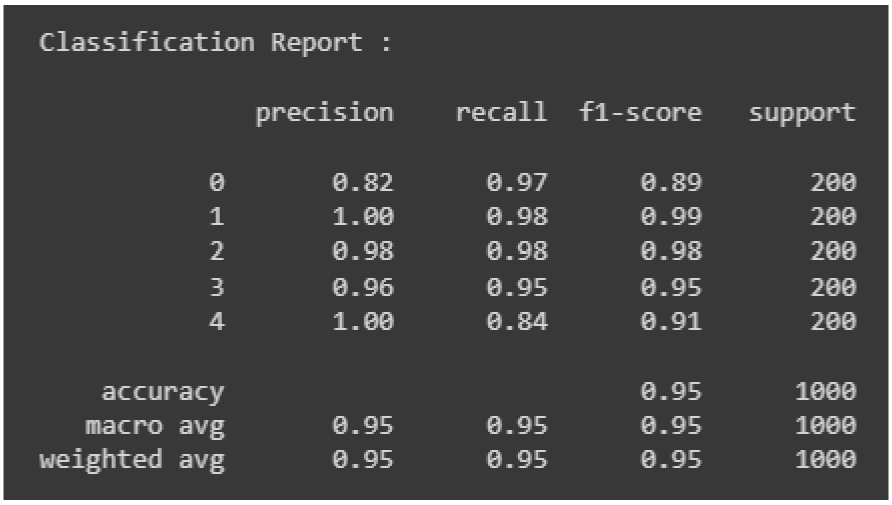
1. **For 80:20 Split and k=35:**



```
Accuracy :  94.6
```

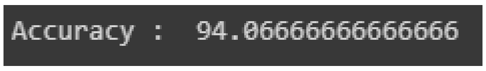


```
Confusion Matrix :

[[194   0   1   5   0]
 [  3 197   0   0   0]
 [  3   0 197   0   0]
 [  8   0   2 190   0]
 [ 29   0   0   3 168]]
```

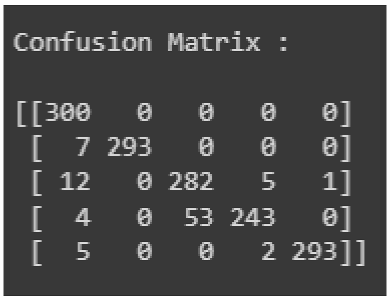


```
Classification Report :

              precision    recall  f1-score   support

           0       0.82      0.97      0.89       200
           1       1.00      0.98      0.99       200
           2       0.98      0.98      0.98       200
           3       0.96      0.95      0.95       200
           4       1.00      0.84      0.91       200

    accuracy                           0.95      1000
   macro avg       0.95      0.95      0.95      1000
weighted avg       0.95      0.95      0.95      1000
```

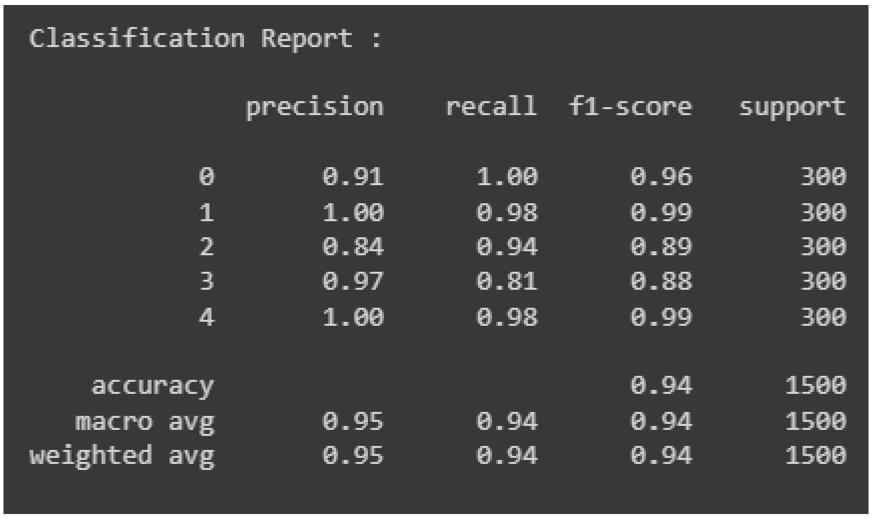
2. **For 70:30 Split and k=35:**



```
Accuracy :  94.06666666666666
```

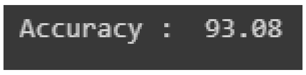


```
Confusion Matrix :

[[300   0   0   0   0]
 [  7 293   0   0   0]
 [ 12   0 282   5   1]
 [  4   0  53 243   0]
 [  5   0   0   2 293]]
```

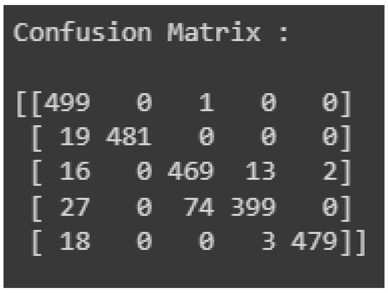


```
Classification Report :

              precision    recall  f1-score   support

           0       0.91      1.00      0.96       300
           1       1.00      0.98      0.99       300
           2       0.84      0.94      0.89       300
           3       0.97      0.81      0.88       300
           4       1.00      0.98      0.99       300

    accuracy                           0.94      1500
   macro avg       0.95      0.94      0.94      1500
weighted avg       0.95      0.94      0.94      1500
```

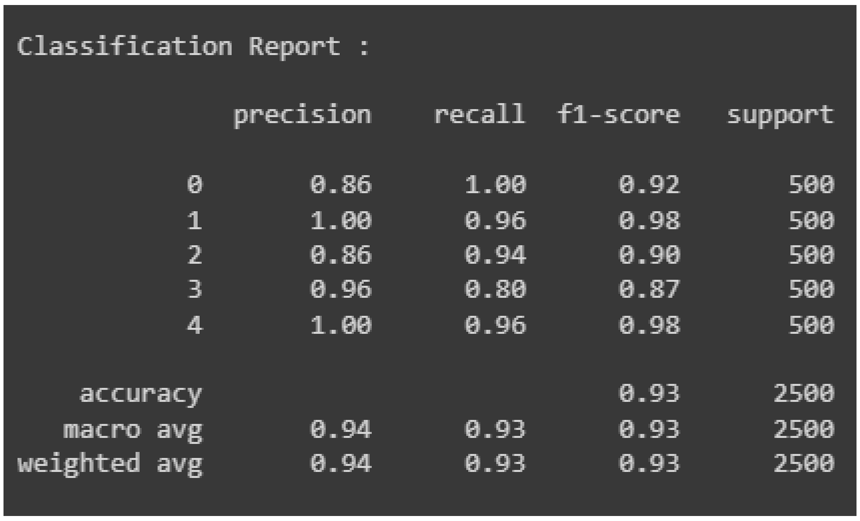
3. **For 50:50 split and k=35:**



```
Accuracy :  93.08
```



```
Confusion Matrix :

[[499   0   1   0   0]
 [ 19 481   0   0   0]
 [ 16   0 469  13   2]
 [ 27   0  74 399   0]
 [ 18   0   0   3 479]]
```



```
Classification Report :

              precision    recall  f1-score   support

           0       0.86      1.00      0.92       500
           1       1.00      0.96      0.98       500
           2       0.86      0.94      0.90       500
           3       0.96      0.80      0.87       500
           4       1.00      0.96      0.98       500

    accuracy                           0.93      2500
   macro avg       0.94      0.93      0.93      2500
weighted avg       0.94      0.93      0.93      2500
```

<u>ANALYSIS:</u> -

We got the highest accuracy on the **80:20 split with k = 35**, having **an accuracy score of 95%,** followed by the **70:30 split with accuracy score of 94%** and, lowest accuracy on the **50:50 split, with an accuracy score of 93%.**