INFORMATION RETRIEVAL
Assignment 2

Group Members :
Rohit Kesarwani (MT22059)
Shambhavi Sharma (MT22066)
Sushil Kumar(MT22077)

LIBRARIES USED :

● os : os is used for importing dataset folders from directory.
● NLTK : The Natural Language Toolkit (nltk) is used for performing tasks such as: Tokenization, Part-of-speech tagging, Sentiment Analysis,Stemming and Lemmatization etc.
● re : Provides operations of detecting patterns with the help of regular expressions.
● json : Using json we can easily convert between Python objects and JSON data, making it easier to exchange data between applications written in different programming languages. The json module provides a method for encoding Python objects into JSON format: json.dumps() used to return a string representation of a JSON-encoded object.
● string : For working with text data present in dataset files.
● numpy : It offers functions for mathematical operations like linear algebra, Fourier transforms, and random number generation, as well as efficient numerical operations on multidimensional arrays and matrices.
● joblib : provides a channel for lightening.
● pandas : It offers capabilities for cleaning, combining, and reshaping data as well as data structures for effectively storing and handling huge datasets.
● BeautifulSoup : It is used for web scraping and parsing HTML and XML documents, providing a convenient and flexible way to extract and manipulate data from web pages.
● operator : It is used for providing mathematical operations.

PREPROCESSING

DATA UPLOADING :
Import files from the device.

DATA UNDERSTANDING:
1. Read data using os.listdir("file location").
2. Checked the total count of files in the folder. The number of files in the folder is 1400.
3. A dictionary is created named "docId". The key contains the document number and the value contains the file name.
4. The key and value of the dictionary are taken and a dataframe is created for storing the filenames

PRE-PROCESSING STEPS ON THE DATASET:
1. Converted the complete text in all the files to lowercase using the .lower() function.
2. Then performed tokenization using the .word_tokenize() method.
3. Removing stopwords from the tokenized text file using stopwords module.
4. Used re.sub() to remove the punctuation marks from the text files.
5. Then used re.sub() to remove the blank spaces from all the files in the folder

Ans 1.1 : TF-IDF Matrix :
METHODOLOGY

The TF-IDF matrix quantifies the relevance of each word in a group of texts by converting them into a set of numerical values. The matrix is frequently used in tasks involving information retrieval, NLP etc.

1. After Preprocessing and Tokenization do Term Frequency (TF) Calculation.
2. Using findUniqueWords(fileData) to find the unique words and storing them into a "uniqueWordsDict" dictionary .
3. Calculating Term Frequency (TF) using calculateTF() function against each document and inverse document frequency.

The Term Frequency is calculated using 5 different weighting schemes,

Weighting Scheme	TF Weight
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)

1. Binary

```
experimental        1.8769576090935038
investigation       2.1106841210051632
aerodynamics        4.16877715700487
awing               5.860786223465865
slipstream          4.948759890378168
study               2.458479072456532
wing                2.223032461292092
propeller           4.1250283821607026
wasmade             6.147755791434351
order               2.1972245773362196
determine           2.917770732084279
spanwise            4.214593690373678
distribution        2.129113151477074
liftincrease        7.244941546337007
due                 2.5162529975227446
different           2.90602365959536
angles              2.849495929024197
attack              2.7365074059162455
wingand             5.860786223465865
free                2.088241095631685
stream              2.106143700286832
velocity            1.8870696490323797
ratios              2.90602365959536
theresults          3.9674927514725074
intended            5.303304908059076
...
forthe              3.100638140420171
specific            3.4271001249214255
configuration       3.9674927514725074
experiment          3.311093750018161
```

2. Raw count

```
Word                    Raw TF-IDF Value
experimental            3.7539152181870077
investigation           2.1106841210051632
aerodynamics            4.16877715700487
awing                   5.860786223465865
slipstream              24.743799451890844
study                   2.458479072456532
wing                    2.223032461292092
propeller               4.1250283821607026
wasmade                 6.147755791434351
order                   2.1972245773362196
determine               2.917770732084279
spanwise                4.214593690373678
distribution            2.129113151477074
liftincrease            7.244941546337007
due                     5.032505995045489
different               5.81204731919072
angles                  2.849495929024197
attack                  2.7365074059162455
wingand                 5.860786223465865
free                    2.088241095631685
stream                  2.106143700286832
velocity                1.8870696490323797
ratios                  2.90602365959536
theresults              3.9674927514725074
...
forthe                  3.100638140420171
specific                3.4271001249214255
configuration           3.9674927514725074
experiment         3.311093750018161
```

3. Term frequency

```
experimental        3.7539152181870077
investigation       2.1106841210051632
aerodynamics        4.16877715700487
awing               5.860786223465865
slipstream          24.743799451890844
study               2.458479072456532
wing                2.223032461292092
propeller           4.1250283821607026
wasmade             6.147755791434351
order               2.1972245773362196
determine           2.917770732084279
spanwise            4.214593690373678
distribution        2.129113151477074
liftincrease        7.244941546337007
due                 5.032505995045489
different           5.81204731919072
angles              2.849495929024197
attack              2.7365074059162455
wingand             5.860786223465865
free                2.088241095631685
stream              2.106143700286832
velocity            1.8870696490323797
ratios              2.90602365959536
theresults          3.9674927514725074
intended            5.303304908059076
...
forthe              3.100638140420171
specific            3.4271001249214255
configuration       3.9674927514725074
experiment          3.311093750018161
```

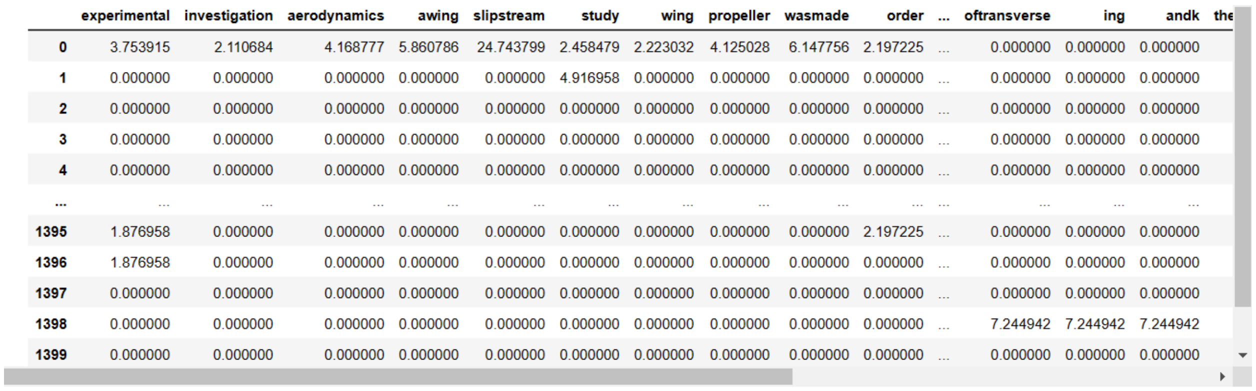
4. Log normalization log

```
Word                    Log TF-IDF Value
experimental            2.0620486946592376
investigation           1.4630147475273754
aerodynamics            2.88957613276063
awing                   4.062387446659934
slipstream              8.866987394521075
study                   1.7040878375388746
wing                    1.5408886828378492
propeller               2.8592517928244434
wasmade                 4.261299593603796
order                   1.523000020837618
determine               2.0224445564645452
spanwise                2.92133373368825
distribution            1.4757887780394334
liftincrease            5.0218108061651066
due                     2.764386464476454
different               3.1925933035917344
angles                  1.975120069220164
attack                  1.8968023929922553
wingand                 4.062387446659934
free                    1.4474584277665135
stream                  1.459867567707908
velocity                1.3080170067470396
ratios                  2.014302106289018
...
forthe                  2.1491985849688735
specific                2.3754847890859225
configuration           2.7500564145751882
experiment              2.2950752973947446
```

5. Double normalization

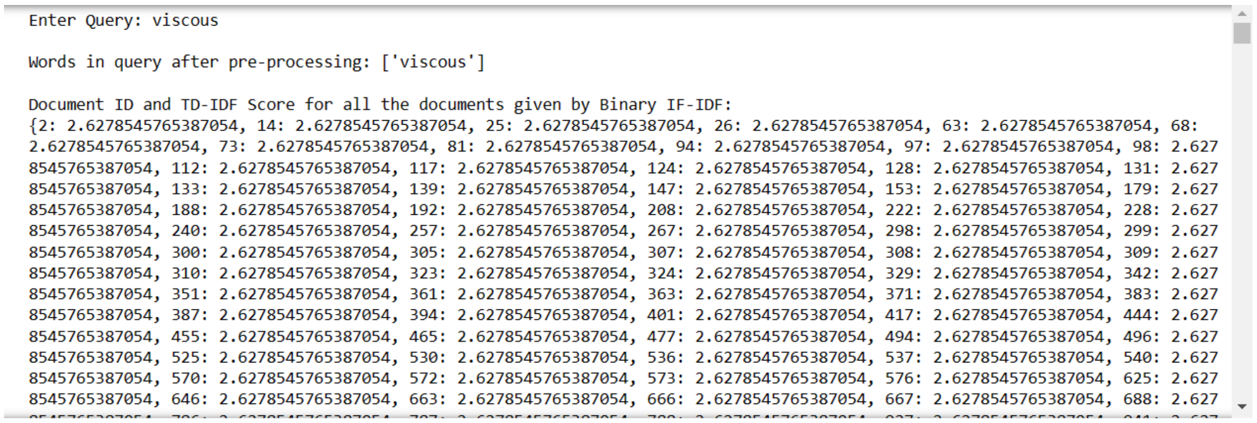
```
Word                    Double Log TF-IDF Value
experimental            1.3138703263654525
investigation           1.2664104726030978
aerodynamics            2.501266294202922
awing                   3.516471734079519
slipstream              4.948759890378168
study                   1.4750874434739192
wing                    1.3338194767752551
propeller               2.4750170292964215
wasmade                 3.6886534748606103
order                   1.3183347464017316
determine               1.7506624392505674
spanwise                2.528756214224207
distribution            1.2774678908862442
liftincrease            4.346964927802204
due                     1.7613770982659211
different               2.034216561716752
angles                  1.7096975574145181
attack                  1.6419044435497472
wingand                 3.516471734079519
free                    1.2529446573790108
stream                  1.263686220172099
velocity                1.1322417894194279
ratios                  1.743614195757216
...
forthe                  1.8603828842521026
specific                2.056260074952855
configuration           2.3804956508835042
experiment              1.9866562500108964
```

Matrix :



	experimental	investigation	aerodynamics	awing	slipstream	study	wing	propeller	wasmade	order	...	oftransverse	ing	andk	the
0	3.753915	2.110684	4.168777	5.860786	24.743799	2.458479	2.223032	4.125028	6.147756	2.197225	...	0.000000	0.000000	0.000000	
1	0.000000	0.000000	0.000000	0.000000	0.000000	4.916958	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1395	1.876958	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.197225	...	0.000000	0.000000	0.000000	
1396	1.876958	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	
1397	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	
1398	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	7.244942	7.244942	7.244942	
1399	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	

query vector :



```
Enter Query: viscous

Words in query after pre-processing: ['viscous']

Document ID and TD-IDF Score for all the documents given by Binary IF-IDF:
{2: 2.6278545765387054, 14: 2.6278545765387054, 25: 2.6278545765387054, 26: 2.6278545765387054, 63: 2.6278545765387054, 68:
2.6278545765387054, 73: 2.6278545765387054, 81: 2.6278545765387054, 94: 2.6278545765387054, 97: 2.6278545765387054, 98: 2.627
8545765387054, 112: 2.6278545765387054, 117: 2.6278545765387054, 124: 2.6278545765387054, 128: 2.6278545765387054, 131: 2.627
8545765387054, 133: 2.6278545765387054, 139: 2.6278545765387054, 147: 2.6278545765387054, 153: 2.6278545765387054, 179: 2.627
8545765387054, 188: 2.6278545765387054, 192: 2.6278545765387054, 208: 2.6278545765387054, 222: 2.6278545765387054, 228: 2.627
8545765387054, 240: 2.6278545765387054, 257: 2.6278545765387054, 267: 2.6278545765387054, 298: 2.6278545765387054, 299: 2.627
8545765387054, 300: 2.6278545765387054, 305: 2.6278545765387054, 307: 2.6278545765387054, 308: 2.6278545765387054, 309: 2.627
8545765387054, 310: 2.6278545765387054, 323: 2.6278545765387054, 324: 2.6278545765387054, 329: 2.6278545765387054, 342: 2.627
8545765387054, 351: 2.6278545765387054, 361: 2.6278545765387054, 363: 2.6278545765387054, 371: 2.6278545765387054, 383: 2.627
8545765387054, 387: 2.6278545765387054, 394: 2.6278545765387054, 401: 2.6278545765387054, 417: 2.6278545765387054, 444: 2.627
8545765387054, 455: 2.6278545765387054, 465: 2.6278545765387054, 477: 2.6278545765387054, 494: 2.6278545765387054, 496: 2.627
8545765387054, 525: 2.6278545765387054, 530: 2.6278545765387054, 536: 2.6278545765387054, 537: 2.6278545765387054, 540: 2.627
8545765387054, 570: 2.6278545765387054, 572: 2.6278545765387054, 573: 2.6278545765387054, 576: 2.6278545765387054, 625: 2.627
8545765387054, 646: 2.6278545765387054, 663: 2.6278545765387054, 666: 2.6278545765387054, 667: 2.6278545765387054, 688: 2.627
```

Result of each Schema :

Top 5 Documents according to Binary TF-IDF:
cranfield0002
cranfield0014
cranfield0025
cranfield0026
cranfield0063

Top 5 Documents according to Raw Count TF-IDF:
cranfield0329
cranfield0310
cranfield0063
cranfield0305
cranfield0536

Top 5 Documents according to TermFreq TF-IDF:
cranfield0329
cranfield0310
cranfield0063
cranfield0305
cranfield0536

Top 5 Documents according to Log TF-IDF:
cranfield0329
cranfield0310
cranfield0063
cranfield0305
cranfield0536

Top 5 Documents according to Double Log TF-IDF:
cranfield0063
cranfield0305
cranfield0537
cranfield1253
cranfield0329

pros and cons of using each weighting scheme:

1. Binary Weighting Scheme :
   Pros :
   - It is simple and easy to implement.
   - It is useful for that type of applications where only presence or absence of terms matters, for example text classification.

   Cons :
   - It does not take into account how often the term appears in the document, which can lead to the loss of important data.
   - It ignores the term's frequency in the corpus, which could result in an excess of rare terms.

2. Raw Count Weighting Scheme :
   Pros :
   - It is simple / easy to implement and understand.
   - It is more informative than the binary weighting method since it takes the frequency of the term in the document into account.

   Cons :
   - The length of the paper, which could influence its ranking, is not taken into consideration.
   - It ignores the term's frequency in the corpus, which could result in an excess of common terms.

3. Term Frequency Weighting Scheme :
   Pros :
   - It is more informative than the binary and raw count weighting systems since it takes the frequency of the term in the document into account.
   - It is widely used in information retrieval systems.

Cons :
- The length of the paper, which could influence its ranking, is not taken into consideration.
- It ignores the term's frequency in the corpus, which could result in an over-representation of common terms.

4. Log Normalization Weighting Scheme :
   Pros :
   - It minimizes the influence of high frequency terms, which may impact the document's ranking.
   - It is widely used in information retrieval systems.

   Cons :
   - Log Normalization Weighting Scheme not suitable for short/small documents.
   - It ignores the term's frequency in the corpus, which could result in an over-representation of common terms.

5. Double normalization :
   Pros :
   - It is more informative than the previous weighting systems since it takes into account the frequency of the word in the corpus and the document.
   - It is widely used in information retrieval systems.

   Cons :
   - More complex than other weighting schemes .
   - Harder to implement.

1.2 : Jaccard Coefficient :
METHODOLOGY

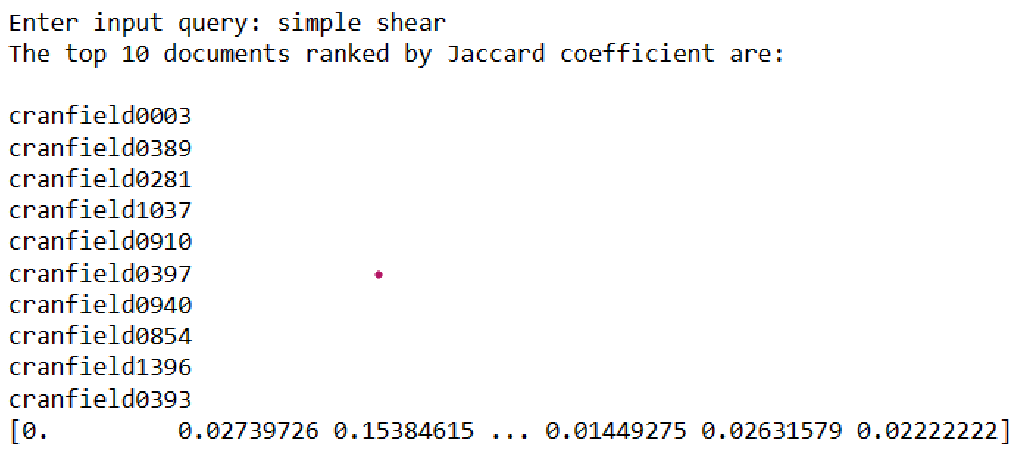
Basically the jaccard coefficient is used to measure the similarity between 2 sets.
It is calculated by dividing the size of the intersection of the 2 sets with the size of the union of the 2 same sets.
1. First we are performing an intersection on two sets using the function " def intersection(l1,l2)" passing sets as a list.
2. After performing the intersection function, return intersectl1l2 ,using this we can get the size of the intersection of the two sets.
3. Similarly we are performing an union on two sets using the function " def union(l1,l2)" passing sets as a list.

4. After performing the union function, return unionl1l2 ,using this we can get the size of the union of the two sets.
5. Dividing the size of the intersection of 2 sets by the size of the union of 2 sets. Which we obtained from above steps to get the Jaccard coefficient.
6. Formula for calculating the Jaccard coefficient is:

$$\boxed{J(l1, l2) = |l1 \cap l2| / |l1 \cup l2|}$$

l1 and l2 are (doc and query)

7. For each document we perform the above steps to calculate jaccard's coefficient.
8. Returning those document which have the top 10 documents ranked by Jaccard coefficient are:



```
Enter input query: simple shear
The top 10 documents ranked by Jaccard coefficient are:

cranfield0003
cranfield0389
cranfield0281
cranfield1037
cranfield0910
cranfield0397
cranfield0940
cranfield0854
cranfield1396
cranfield0393
[0.         0.02739726 0.15384615 ... 0.01449275 0.02631579 0.02222222]
```

9. The documents most relevant to the question are those with a high Jaccard value.

Ans 2 :

PRE-PROCESSING STEPS ON THE DATASET:
1. Reading the csv format dataset
2. Removing unnecessary columns
3. Text cleaning by :
    - Converted the complete text in all the files to lowercase using the .lower() function.
    - Then performed tokenization using the .word_tokenize() method.
    - Removing stopwords from the tokenized text file using stopwords module.

4. performing stemming and lemmatization
5. Apply the clean_tokenize_text function to the 'Text' column.
TF-ICF weighting scheme :

1. Converting list of words back to a string for TfidfVectorizer
2. Generating the TF-IDF matrix.Using:

“Tfidf_matrix = tfidf_transformer.fit_transform(term_doc_matrix).toarray()”

3. Computing ICF values using :

“icf_values = np.log(num_docs / np.count_nonzero(tfidf_matrix, axis=0))”

4. Now Converting tfidf_matrix and icf_values to sparse matrices.

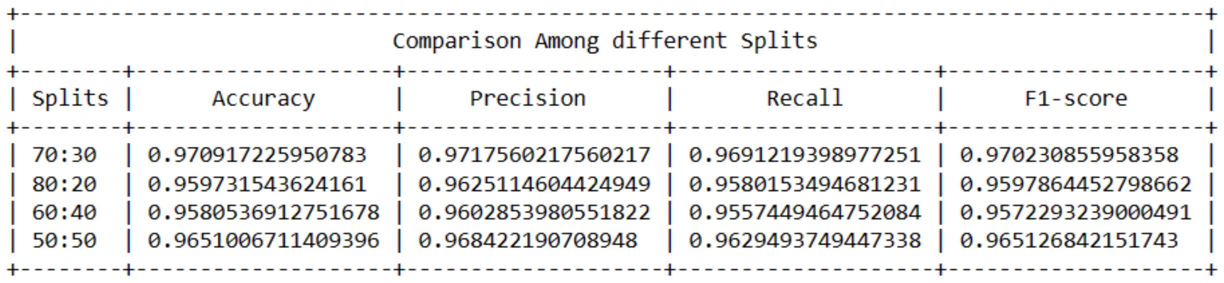
Next Split the dataset into training and testing sets :

“ X_train, X_test, y_train, y_test = train_test_split(tf_icf_matrix, data['Category'], test_size=0.3, random_state=42) ”.

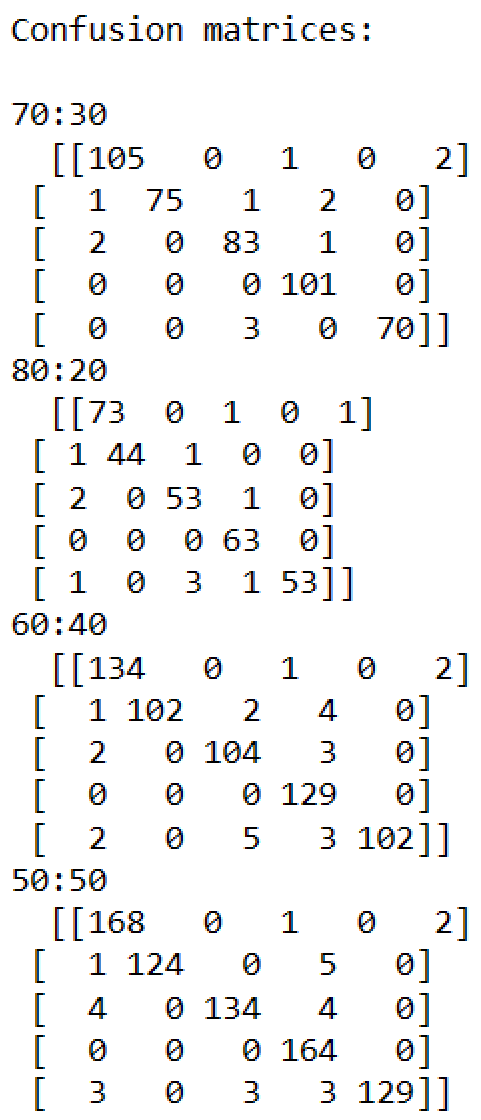
“ train_test_split” function splits a dataset into two sets - a training set and a testing set.
“ tf_icf_matrix “, which is the feature matrix data['Category'], which is the target variable. The test_size parameter is set to 0.3, which means that 30% of the data will be allocated to the testing set, and the remaining 70% will be allocated to the training set.

After executing this code, the following four objects will be created:
- X_train: A matrix of training features
- X_test: A matrix of testing features
- y_train: A vector of training target values
- y_test: A vector of testing target values

Training the Naive Bayes classifier with TF-ICF and Testing the Naive Bayes classifier with TF-ICF :



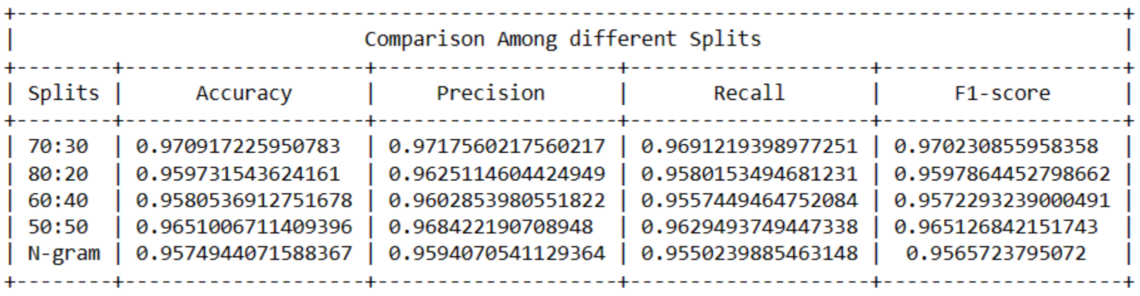
```
+--------------------------------------------------------------------------------------------------------+
|                                      Comparison Among different Splits                                |
+--------+--------------------+--------------------+--------------------+--------------------+
| Splits |      Accuracy      |     Precision      |       Recall       |      F1-score      |
+--------+--------------------+--------------------+--------------------+--------------------+
| 70:30  | 0.970917225950783  | 0.9717560217560217 | 0.9691219398977251 | 0.970230855958358  |
| 80:20  | 0.959731543624161  | 0.9625114604424949 | 0.9580153494681231 | 0.9597864452798662 |
| 60:40  | 0.9580536912751678 | 0.9602853980551822 | 0.9557449464752084 | 0.9572293239000491 |
| 50:50  | 0.9651006711409396 | 0.968422190708948  | 0.9629493749447338 | 0.965126842151743  |
+--------+--------------------+--------------------+--------------------+--------------------+
```



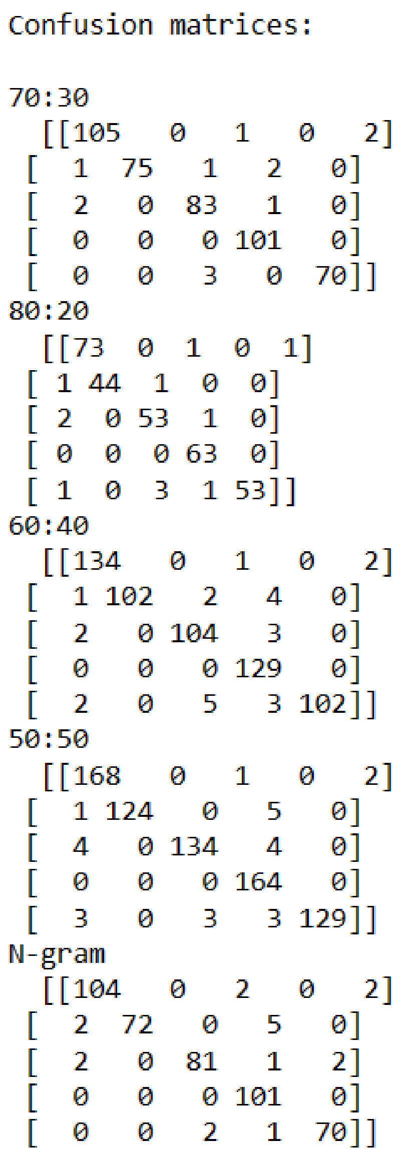
```
Confusion matrices:

70:30
  [[105   0   1   0   2]
 [  1  75   1   2   0]
 [  2   0  83   1   0]
 [  0   0   0 101   0]
 [  0   0   3   0  70]]
80:20
  [[73  0  1  0  1]
 [ 1 44  1  0  0]
 [ 2  0 53  1  0]
 [ 0  0  0 63  0]
 [ 1  0  3  1 53]]
60:40
  [[134   0   1   0   2]
 [  1 102   2   4   0]
 [  2   0 104   3   0]
 [  0   0   0 129   0]
 [  2   0   5   3 102]]
50:50
  [[168   0   1   0   2]
 [  1 124   0   5   0]
 [  4   0 134   4   0]
 [  0   0   0 164   0]
 [  3   0   3   3 129]]
```

Improving the classifier :



```
+------------------------------------------------------------------------------------------------+
|                                Comparison Among different Splits                               |
+--------+--------------------+--------------------+--------------------+--------------------+
| Splits |      Accuracy      |     Precision      |       Recall       |      F1-score      |
+--------+--------------------+--------------------+--------------------+--------------------+
| 70:30  | 0.970917225950783  | 0.9717560217560217 | 0.9691219398977251 | 0.970230855958358  |
| 80:20  | 0.959731543624161  | 0.9625114604424949 | 0.9580153494681231 | 0.9597864452798662 |
| 60:40  | 0.9580536912751678 | 0.9602853980551822 | 0.9557449464752084 | 0.9572293239000491 |
| 50:50  | 0.9651006711409396 | 0.968422190708948  | 0.9629493749447338 | 0.965126842151743  |
| N-gram | 0.9574944071588367 | 0.9594070541129364 | 0.9550239885463148 |  0.9565723795072   |
+--------+--------------------+--------------------+--------------------+--------------------+
```



```
Confusion matrices:

70:30
  [[105   0   1   0   2]
 [  1  75   1   2   0]
 [  2   0  83   1   0]
 [  0   0   0 101   0]
 [  0   0   3   0  70]]
80:20
  [[73  0  1  0  1]
 [ 1 44  1  0  0]
 [ 2  0 53  1  0]
 [ 0  0  0 63  0]
 [ 1  0  3  1 53]]
60:40
  [[134   0   1   0   2]
 [  1 102   2   4   0]
 [  2   0 104   3   0]
 [  0   0   0 129   0]
 [  2   0   5   3 102]]
50:50
  [[168   0   1   0   2]
 [  1 124   0   5   0]
 [  4   0 134   4   0]
 [  0   0   0 164   0]
 [  3   0   3   3 129]]
N-gram
  [[104   0   2   0   2]
 [  2  72   0   5   0]
 [  2   0  81   1   2]
 [  0   0   0 101   0]
 [  0   0   2   1  70]]
```

Ans 3 :

1. First of all, reading a file called "Dataset_Question3.txt" using the pandas library in Python and assigning it to a variable called "df". Overall loading a text file into a pandas dataframe.

2. Only the queries with qid=4 are considered using the command:
   df [df [1] == “qid:4”].

3. storing all unique relevance judgment labels in relevanceJudgementLabel

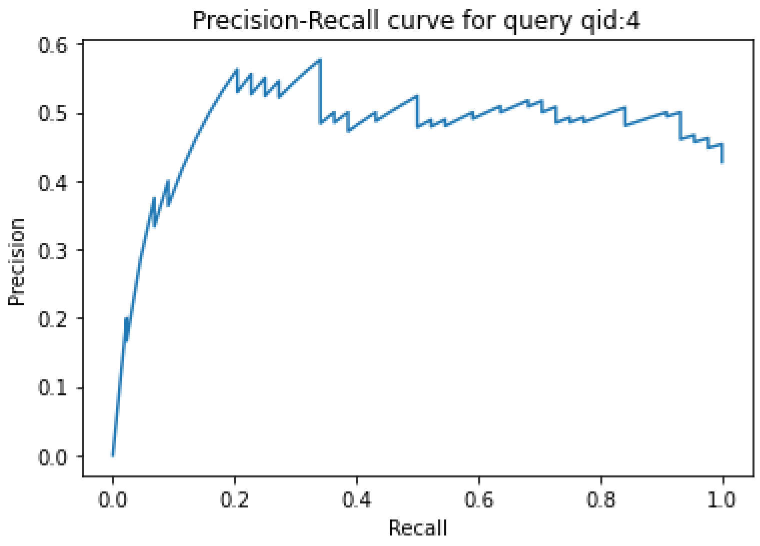
4. Calculating final sorted dataframe and save to CSV:
   Totalfiles:
   19893497375938370599826047614905329896936840170566570588205180311
   2704857992695193482412686565431050240000000000000000000000

5. Now using function DFG() to calculate nDCG of 50 data and to calculate nDCG of the whole dataset.

```
nDCG of 50 data 0.35612494416255847
nDCG of whole dataset 0.5784691984582591
```

6. Sort the relevance labels based on the feature_75 scores in descending order, calculating precision and recall for each document in the sorted list.

7. Plotting the precision and recall curve.


Precision-Recall curve for query qid:4
Precision
Recall
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
0.2
0.4
0.6
0.8
1.0

8.