ASSIGNMENT 2

Q1)

Jaccard Coefficient:

The goal is to find the Jaccard coefficient between a given query and the document.

1) Using the same data given in assignment 1, the same preprocessing steps as mentioned before is carried out.
    - The text is first converted into lower case.
    - The punctuations in the text are removed.
    - Stopwords are then removed from the text.
    - The text is converted to tokens.
    - The blank tokens are removed.

2)
    - Query is received and the same preprocessing steps are performed on it.
    - Query is converted into a set with query tokens.
    - Now each document is traversed using a for loop where intersection and union between each document and the query is carried out and then length of intersection set is divided by length of union set which is the jaccard coefficient.

3) Then from the scores obtained we find the top 5 documents with the highest jaccard coefficient.

TF-IDF Matrix:

The goal is to generate a TF-IDF matrix for each word in the vocab and obtain a TF-IDF score for a given query.

1) The same data as above is used and again the same preprocessing steps as above is done.
2) A matrix of size no. of documents vs vocab size is created.
3) A function to calculate tf-idf is created and for each word vs document, the tf-idf is computed and stored in the matrix.
4) The query is received and the same preprocessing steps are performed. It is then converted into a one hot vector of vocab size where if a word in the query is present in the vocab, then it is represented as 1 or else it is represented as 0.
5) Then the dot product of the query vector with each row in the matrix is performed, which gives the tf-idf score of the query with each document. The top 5 documents with the tf-idf scores are then taken.
6) All the five weighting schemes are performed and tf-idf is performed using each scheme.
7)
    - Binary weighting scheme:
      Pros: It is simple to implement and easy to compute.
      Cons: It does not show which document is more relevant.

    - Raw count:
      Pros: It is simple to implement and gives the sense which document is more relevant.
      Cons: Relevance does not increase proportionally with term frequency, thus it cannot show how much relevant the document

    - Term frequency:
      Pros: It is simple to implement and gives the sense which document is more relevant. It also shows the score in normalized value to minimize the effect of long vs. short documents and reflect the true importance of a keyword.
      Cons: Relevance does not increase proportionally with term frequency, thus it cannot show how relevant the document is.

- Log Normalization:
  Pros: The values obtained are not as large as term frequency and it is simpler to compute.It also shows the score in normalized value to minimize the effect of long vs. short documents and reflect the true importance of a keyword.
  Cons: It does not take into account the semantics.

- Double Normalization:
  Pros: It also shows the score in normalized value to minimize the effect of long vs. short documents and reflect the true importance of a keyword.
  Cons: It also does not take into account the semantics.

Question 2:

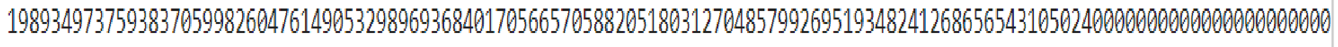
First, the CSV file was converted into DataFrame.

1. Only the queries with qid=4 are considered using the command:

   ```
   df[df[1] == "qid:4"]
   ```

2. All the unique relevance judgment labels were stored in relevance.

3. The total number of files on rearranging was calculated.

   ```
   19893497375938370599826047614905329896936840170566570588205180312704857992695193482412686565431050240000000000000000000000
   ```

4. Now nDCG at 50 and for the whole dataset was calculated using:

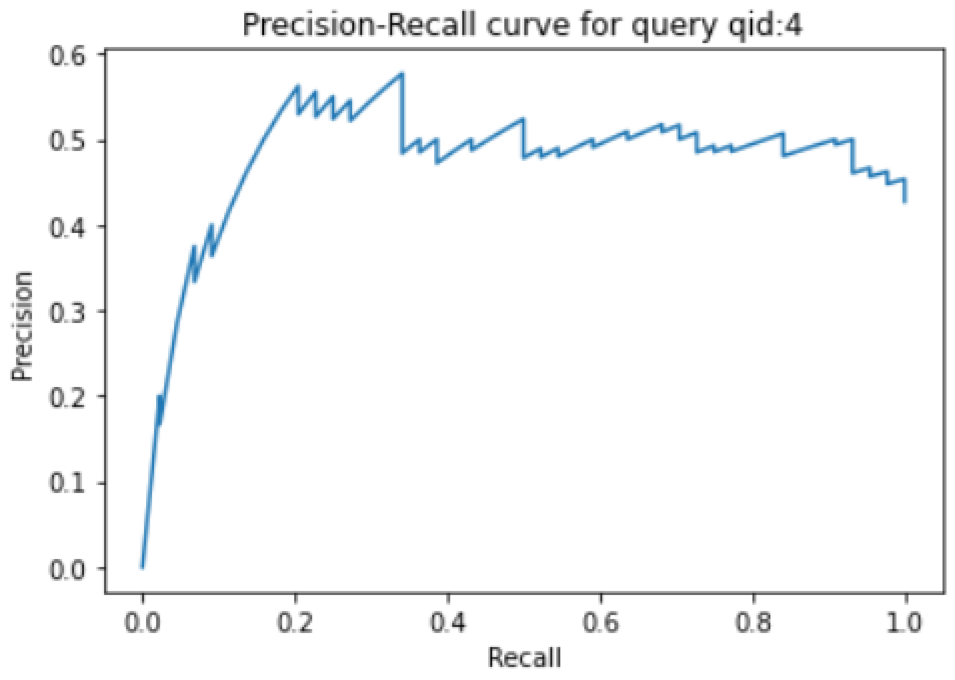
nDCG = DCG /ideal DCG

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i}-1}{log(1+i)}$$

```python
print("nDCG at 50:",DCG(df2,50)/DCG(final,50))
print("nDCG for whole dataset:",DCG(df2,len(df2))/DCG(final,len(df2)))
```

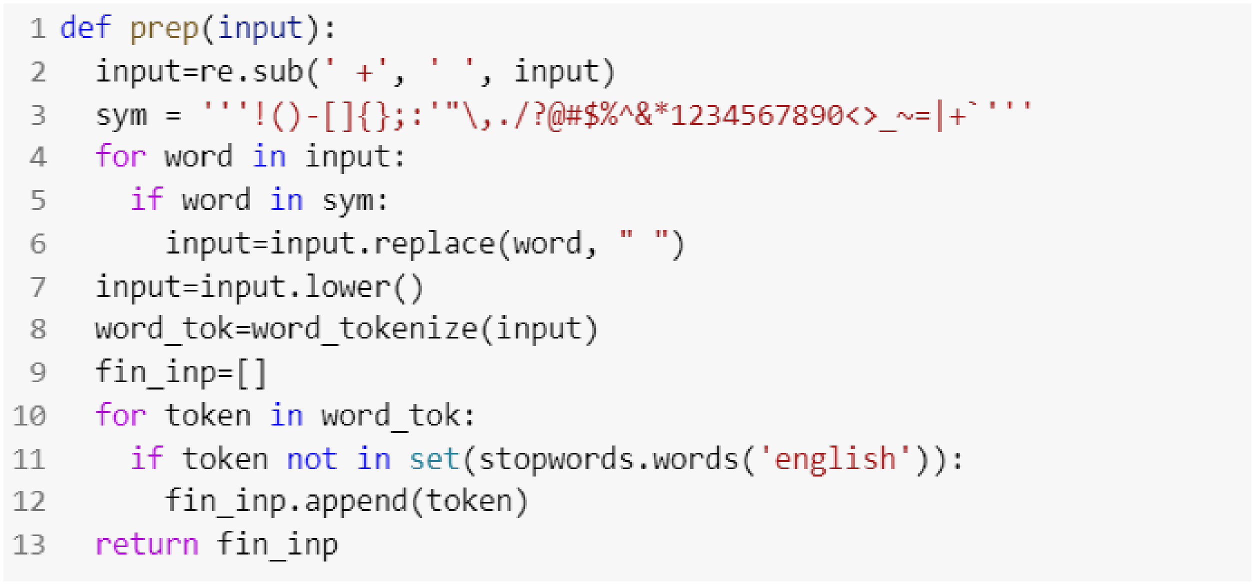
```
nDCG at 50: 0.35612494416255847
nDCG for whole dataset: 0.5784691984582591
```

5. The relevance labels were arranged on the basis of feature 75 for queries with qid:4 and relevance label > 0 and precision and recall were calculated and then the precision-recall curve was plotted.



Question 3:

- First, all the files were extracted and documents of comp.graphics, sci.med, talk.politics.misc, rec.sport.hockey, sci.space were picked.
- Next, all the necessary preprocessing was carried out.

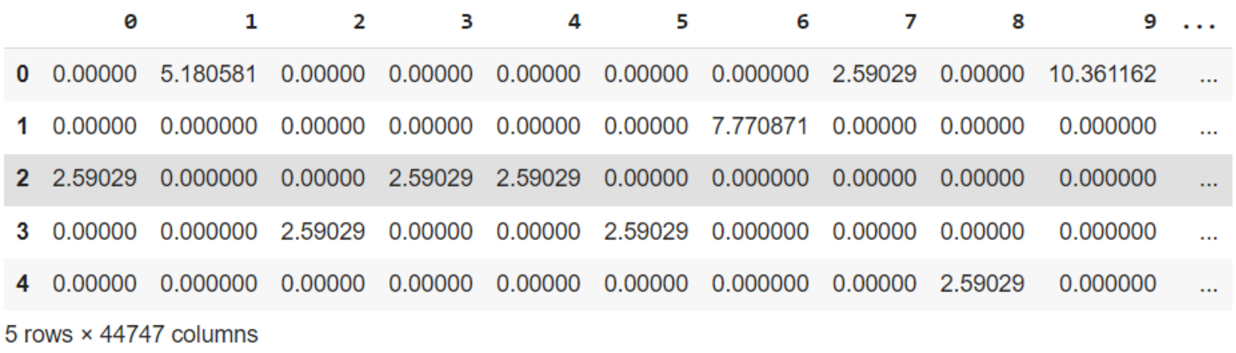


```python
def prep(input):
  input=re.sub(' +', ' ', input)
  sym = '''!()-[]{};:'"\,./?@#$%^&*1234567890<>_~=|+`'''
  for word in input:
    if word in sym:
      input=input.replace(word, " ")
  input=input.lower()
  word_tok=word_tokenize(input)
  fin_inp=[]
  for token in word_tok:
    if token not in set(stopwords.words('english')):
      fin_inp.append(token)
  return fin_inp
```

- Next, the dataset was split randomly into 50:50, 70:30, and 80:20.
C
- Next, term frequency for each term in each class was calculated in a dataframe.

- Next, for each term ICF was calculated and saved in a list.
ICF=log(Number of classes/Class Frequency)

- TF-ICF matrix was made and for each class top k features were extracted. k is user input.



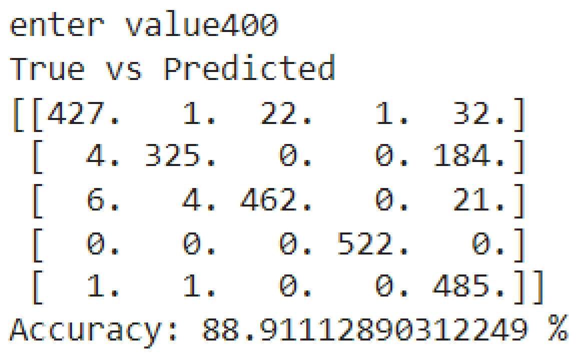
	0	1	2	3	4	5	6	7	8	9	...
0	0.00000	5.180581	0.00000	0.00000	0.00000	0.00000	0.000000	2.59029	0.00000	10.361162	...
1	0.00000	0.000000	0.00000	0.00000	0.00000	0.00000	7.770871	0.00000	0.00000	0.000000	...
2	2.59029	0.000000	0.00000	2.59029	2.59029	0.00000	0.000000	0.00000	0.00000	0.000000	...
3	0.00000	0.000000	2.59029	0.00000	0.00000	2.59029	0.000000	0.00000	0.00000	0.000000	...
4	0.00000	0.000000	0.00000	0.00000	0.00000	0.00000	0.000000	0.00000	2.59029	0.000000	...

5 rows × 44747 columns

- For each class, Naive Bayes Classifier was trained on the training data.

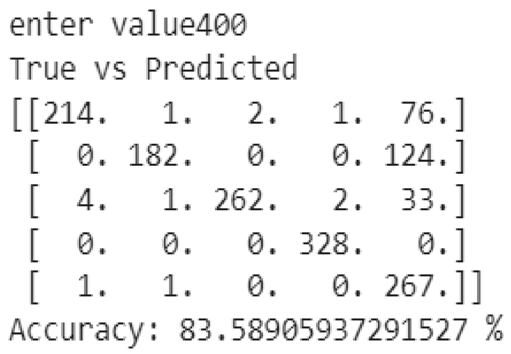
- The confusion matrix and overall accuracy for :

    - ☐ 50:50 split



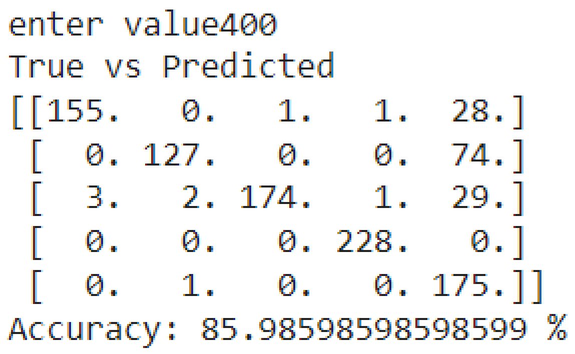
```
enter value400
True vs Predicted
[[427.   1.  22.   1.  32.]
 [  4. 325.   0.   0. 184.]
 [  6.   4. 462.   0.  21.]
 [  0.   0.   0. 522.   0.]
 [  1.   1.   0.   0. 485.]]
Accuracy: 88.91112890312249 %
```

    - ☐ 70:30 split



```
enter value400
True vs Predicted
[[214.   1.   2.   1.  76.]
 [  0. 182.   0.   0. 124.]
 [  4.   1. 262.   2.  33.]
 [  0.   0.   0. 328.   0.]
 [  1.   1.   0.   0. 267.]]
Accuracy: 83.58905937291527 %
```

    - ☐ 80:20 split



```
enter value400
True vs Predicted
[[155.   0.   1.   1.  28.]
 [  0. 127.   0.   0.  74.]
 [  3.   2. 174.   1.  29.]
 [  0.   0.   0. 228.   0.]
 [  0.   1.   0.   0. 175.]]
Accuracy: 85.98598598598599 %
```