IR Assignment 2
Group 70

Submitted by:
Utsav Baghela (MT21101)
Ashwini Dongre (MT21016)

Question 1:

Part A: Jaccard Coefficient:

1) Input query is taken
2) Preprocessing is performed and converted that query to a list of tokens
3) Intersection and union are performed with tokens present in each file.
4) Jaccard coefficient is calculated using the formula.
5) Sorted the documents based on their Jacccards coefficient
6) Top 5 documents printed

Input Query:

```
Enter the query:tiger cat lion
```

Output: Returned top 5 relevant documents

```
Document Name   Jaccard Coefficient
puzzles.jok   0.020833333333333332
netnews.10   0.011494252873563218
units.mea   0.008130081300813009
ohandre.hum   0.008064516129032258
smokers.txt   0.008
```

Part B: TF-IDF Matrix

1) Matrix is created with rows and columns size as the number of documents to the number of words in the corpus respectively.
2) IDF values are calculated for each of the words present in the corpus
3) TF values are calculated for each word present in the corpus corresponding to each document.
4) TF values are calculated using 5 different weighting schemes:
    a) Binary
    b) Raw count
    c) Term frequency
    d) Log normalization
    e) Double normalization

5) Input query is preprocessed and then TF- IDF values for all the tokens of that input strings are added for each of the documents
6) TF-IDF values for each of the documents are sorted in decreasing order
7) Top 5 documents are printed
8) This process is repeated for all the 5 weighting schemes

Pros and cons of scoring schemes:

	Pros	Cons
Jaccard coefficient	Performs better when the data is rare	Does not perform well when the duplication of data matters
TF-IDF	Performs better when the data does not contains duplication	It does not capture the position of text

Input query:

```
Enter the query:tiger lion cat
```

Output: Top 5 relevant documents printed

```
Enter the query:tiger lion cat
Top 5 Documents using Binary weigthing Scheme

Document Name   TF IDF Score
dthought.txt   7.633917440122534
deep.txt   7.633917440122534
mlverb.hum   5.9393217193481265
grospoem.txt   5.9393217193481265
lost.txt   5.9393217193481265
```

Question 2:

1. Consider only the queries with qid:4 and the relevance judgement labels as relevance score:

1. **Dataset was read with sep=" ".**
2. **Fetched all the queries with qid=4, rest all was dropped.**

	0	1	2	3	4	5	6	7	8	9	...	129	130	131	132	133	134	135	136	
0	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	128:2	129:9	130:124	131:4678	132:54	133:74	134:0	135:0	
1	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	128:0	129:8	130:122	131:508	132:131	133:136	134:0	135:0	
2	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	128:2	129:8	130:115	131:508	132:51	133:70	134:0	135:0	
3	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	128:82	129:17	130:122	131:508	132:83	133:107	134:0	135:10	136:
4	1	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	128:11	129:8	130:121	131:508	132:103	133:120	134:0	135:0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
98	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	128:35	129:1	130:153	131:4872	132:9	133:55	134:0	135:0	
99	1	qid:4	1:3	2:0	3:3	4:2	5:3	6:1	7:0	8:1	...	128:367	129:6	130:153	131:2383	132:18	133:99	134:0	135:16	136:11.31666666
100	2	qid:4	1:2	2:0	3:2	4:0	5:2	6:0.666667	7:0	8:0.666667	...	128:0	129:0	130:49182	131:26966	132:15	133:69	134:0	135:193	136:21.93555954
101	1	qid:4	1:2	2:0	3:2	4:0	5:2	6:0.666667	7:0	8:0.666667	...	128:0	129:1	130:42877	131:26562	132:12	133:24	134:0	135:56	136:62.92060423
102	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	128:1415	129:14	130:5334	131:6434	132:4	133:17	134:0	135:0	

103 rows × 139 columns

3. **Arranged the query-url pairs in order of max DCG. Then nu,mber of files were calculated as:**
4. **Number of files after rearranging: 19893497375938370599826047614905329896936840170566570588205180312704857992695193482412686565431050240000000000000000000000**

number_of_files = number_of_files* math.factorial(count)

```
Number of files after rearranging:  19893497375938370599826047614905329896936840170566570588205180312704857992695193482412686565431050240000000000000000000000
```

Further, Computed nDCG was calculator using formulae:

The traditional formula of DCG accumulated at a particular rank position p is defined as:[1]

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i+1)}$$

Reference: https://en.wikipedia.org/wiki/Discounted_cumulative_gain

```python
def DCG_Calculator(n, data):
    dcg_answer = 0;
    for i in range(1, n+1):
        dcg_answer = dcg_answer + (pow(2, data[0][i-1]) - 1)/(np.log2(i+1))
    return dcg_answer
```

(a) At 50:

```python
#nDCG at 50
dcg_at_df = DCG_Calculator(50,df)
dcg_at_df_sorted_on_dcg = DCG_Calculator(50,df_sorted_on_dcg)
print("nDCG at 50:",dcg_at_df/dcg_at_df_sorted_on_dcg)
```

```
nDCG at 50: 0.35612494416255847
```

(b) For the whole dataset

```
#nDCG at full Dataset
len_df = len(df)
len_sorted_df = len(df_sorted_on_dcg)
dcg_at_df_full = DCG_Calculator(len_df,df)
dcg_at_df_sorted_on_dcg_full = DCG_Calculator(len_sorted_df,df_sorted_on_dcg)
print("nDCG at Full Dataset:",dcg_at_df_full/dcg_at_df_sorted_on_dcg_full)
```

```
nDCG at Full Dataset: 0.5784691984582591
```

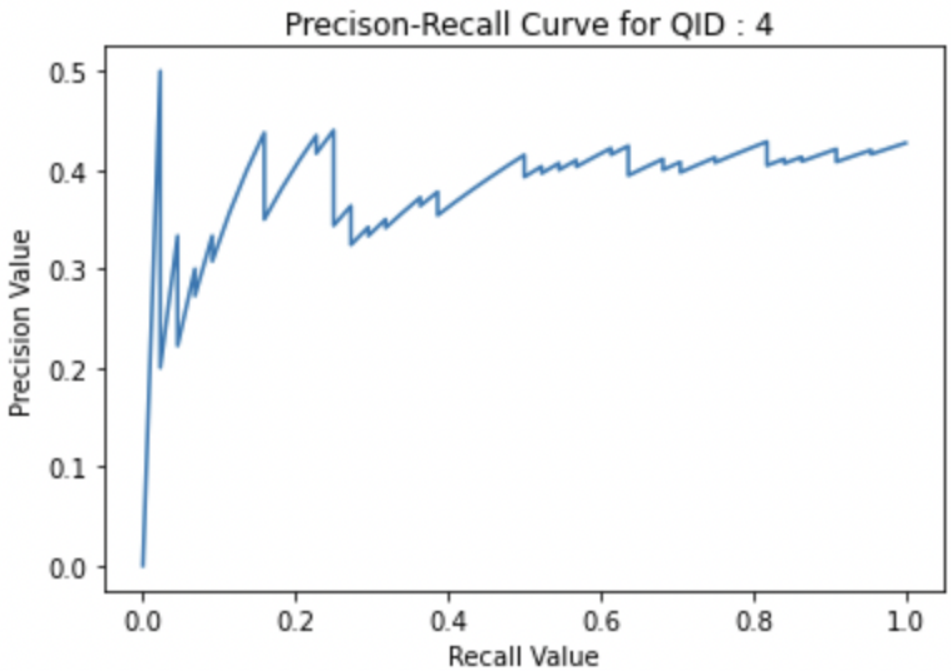
Made model ranks URLs on the basis of the value of feature 75 (sum ofTF-IDF on the whole document) i.e. the higher the value, the more relevant the URL. Assume any
non zero relevance judgment value to be relevant. Plot a Precision-Recall curve for query “qid:4”:

<u>Plot for Recall And Precision was made as follow:</u>

Here, formulae used to find values were:

precision_values= relevence_score/current_count)
Recall_values = relevence_score/jreleveance_Score)

```
plt.plot(recall_values,precision_values)
plt.xlabel("Recall Value")
plt.title(" Precison-Recall Curve for QID : 4")
plt.ylabel("Precision Value")
plt.show()
```



Question 3

1) Preprocessing is performed on the whole dataset.
2) Randomly split the dataset in an 80:20 ratio
3) TF ICF values are calculated for each of the words in the dataset
4) k features of each class are selected on the basis of TF-ICF values.
5) Union of all the features is taken to make new vocabulary
6) Trained the Naive Bayes classifier on the training data
7) The performance of the Naive Bayes Classifier is evaluated on different test ratios.

Results for Split 80:20

Accuracy:

```
Training Accuracy:
0.983
Testing Accuracy:
0.9723076923076923
```

Confusion Matrix

```
array([[460,   0,   0,   2,   1],
       [  3, 410,   2,  12,   8],
       [  1,   1, 451,   2,   4],
       [  1,   2,   0, 441,   3],
       [  0,   8,   3,  10, 450]])
```

Results for Split 50:50

Accuracy:

```
Training Accuracy:
0.9776
Testing Accuracy:
0.9737101544528426
```

Confusion Matrix:

```
array([[588,   0,   0,   2,   0],
       [  2, 582,   2,  21,   6],
       [  3,   1, 603,   2,   2],
       [  0,   2,   0, 592,   5],
       [  1,  10,   7,  14, 598]])
```

Results for Split 70:30

Accuracy:

```
Training Accuracy:
0.9854285714285714
Testing Accuracy:
0.9717400080742834
```

Confusion Matrix

```
array([[470,   0,   0,   3,   1],
       [  1, 449,   2,  11,  12],
       [  1,   3, 492,   1,   5],
       [  2,   0,   0, 516,   5],
       [  2,   8,   3,  10, 480]])
```

References

https://stackoverflow.com/questions/60969884/multinomial-naive-bayes-for-python-from-scratch