

IR

Assignment-2

Sristi Sharma (MT21098)

Parul Jain (MT21064)



Question - 1 Scoring and term Weighting


Preprocessing is done like:

1. Tokenization
2. Punctuation remove
3. Stop words removal
4. Converting into Lower case
5. Remove blanks

At the second we created Posting lists, A dictionary is created containing the document name and words that are present in the document and its frequency.

At first, Query entered by user is pre processed, and a query vector is made over that query.

Jaccard Coefficient : Results for Jaccard Coefficient-



	scores	document
783	0.038462	strattma.txt
643	0.027523	blkbnsrc.vgn
732	0.027397	oranchic.pol
756	0.021277	pasta001.sal
706	0.019417	egglenl.vgn
969	0.019231	garlpast.vgn
702	0.018692	jawgumbo.fis
749	0.018182	recipe.007
746	0.018018	margos.txt
771	0.017241	venison.txt

TF-IDF:

A vocab is created which contains all the unique words and its length is calculated which is : 83288

Calculating TF-IDF for each word present in the dictionary and created a TF-IDF matrix.

We have five methodologies for calculating TF-IDf matrix and extracting Top 5 relevant documents.

1. Binary:

Documents Matched using Raw Count:

```
[('appetiz.rcp', 123.51679195931202),  
 ('vegan.rcp', 111.28152322826028),  
 ('candy.txt', 78.64411125255762),  
 ('penndtch', 75.78468496729872),  
 ('jerky.rcp', 60.19579447299453),  
 ('chili.txt', 59.879415684550295),  
 ('roadpizz.txt', 50.9010430699485),  
 ('shuimai.txt', 36.562899643132155),  
 ('pizzawho.hum', 29.719809915989003),  
 ('bread.rcp', 27.235491576474963)]
```

2. Raw count :

Documents Matched using Raw Count:

```
[('appetiz.rcp', 123.51679195931202),  
 ('vegan.rcp', 111.28152322826028),  
 ('candy.txt', 78.64411125255762),  
 ('penndtch', 75.78468496729872),  
 ('jerky.rcp', 60.19579447299453),  
 ('chili.txt', 59.879415684550295),  
 ('roadpizz.txt', 50.9010430699485),  
 ('shuimai.txt', 36.562899643132155),  
 ('pizzawho.hum', 29.719809915989003),  
 ('bread.rcp', 27.235491576474963)]
```

3. Term frequency:

Documents Matched using Term Frequency:

```
[('pasta001.sal', 0.1057976527404703),  
 ('garlpast.vgn', 0.0969138803729499),  
 ('roadpizz.txt', 0.08540443468112163),  
 ('fish.rec', 0.07097770919968628),  
 ('fbipizza.txt', 0.06976488347948832),  
 ('oranchic.pol', 0.04851947460204402),  
 ('strattma.txt', 0.04416006826026164),  
 ('buffwing.pol', 0.039123716598520535),  
 ('venganza.txt', 0.03612975001752629),  
 ('capital.txt', 0.03526951787425185)]
```

4. Log Normalization:

Documents Matched using Log Normalization:

```
[('appetiz.rcp', 13.194647969425413),  
 ('vegan.rcp', 11.862101959901475),  
 ('penndtch', 10.299416766393662),  
 ('bread.rcp', 7.367469559273125),  
 ('chili.txt', 7.220851808013785),  
 ('jerky.rcp', 6.758764421366658),  
 ('shuimai.txt', 6.322109435008565),  
 ('pizzawho.hum', 5.9852168953708915),  
 ('top10.txt', 5.52321032775934),  
 ('fajitas.rcp', 5.498819181722448)]
```

5. Double Normalization :

Documents Matched using Double Normalization:

```
[('pasta001.sal', 3.3031654409718305),  
 ('garlpast.vgn', 3.29872355478807),  
 ('roadpizz.txt', 3.292968831942156),  
 ('fish.rec', 3.2857554692014386),  
 ('fbipizza.txt', 3.2851490563413397),  
 ('oranchic.pol', 3.274526351902617),  
 ('strattma.txt', 3.272346648731726),  
 ('buffwing.pol', 3.2698284729008553),  
 ('venganza.txt', 3.2683314896103584),  
 ('capital.txt', 3.267901373538721)]
```

Page 10 of 10

Given a data set to process upon.

a. Consider only the queries with qid:4 and the relevance judgment labels as relevance score.

1. All the files are first to read and converted into a DataFrame with column names Col1,Col2,Col3.....
2. All the files with qid:4 are retrieved

b. Make a file rearranging the query-URL pairs in order of max DCG. State how many such files could be made.

1. First calculated the lengths of documents according to the relevance.
2. Calculated factorial of lengths for each relevance.
3. There are 59 zeros, 26 1's , 17 2's, 1 3's and 0 4's.
4. Taking factorial of each length:

```
[ ] 1 import math
    2 math.factorial(1)
```

1

```
[ ] 1 math.factorial(17)
```

355687428096000

```
[ ] 1 math.factorial(26)
```

403291461126605635584000000

```
[ ] 1 math.factorial(59)
```

1386831185456898357379390197203894063459028767726874325408212949401600000000000000

PART-2

```
[ ] 1 math.factorial(1)*math.factorial(17)*math.factorial(26)*math.factorial(59)
```

198934973759383705998260476149053298969368401705665705882051803127048579926951934824126865654310502400000000000000000000

5. Total number of files : $59! * 26! * 17! * 1!$

```
[ ] 1 math.factorial(1)*math.factorial(17)*math.factorial(26)*math.factorial(59)
```

```
198934973759383705998260476149053298969368401705665705882051803127048579926951934824126865654310502400000000000000000000
```

```
19893497375938370599826047614905329896936840170566570588205180312704857992
695193482412686565431050240000000000000000000000000
```

C. Compute DCG : A DCG function is defined which takes relevances as parameters

a. At 50

```
1 def DCG(relevances,p):
2     print(relevances)
3     p=len(relevances)
4     res=0
5     for i in df_qid4['Relevance']:
6         num=relevances[i+1]
7         den=math.log((i+1),2)
8         res += num/den
9         #print(i)
10    res += relevances[0]
11    return res
12
```


DCG at 50

```
[ ] 1 ideal_dcg50=DCG(df_top50_sorted['Relevance'])
    2 print("Ideal DCG at 50:\t",ideal_dcg50)
    3
```

Ideal DCG at 50: 20.989750804831445

```
[ ] 1 dcg50=DCG(df_top50['Relevance'])
    2 print("DCG at 50:\t",dcg50)
```

DCG at 50: 7.39058096925802

b. For whole Dataset:

Ideal DCG at 50: 20.989750804831445

DCG at 50: 7.39058096925802

DCG for Entire Dataset

```
[ ] 1 ideal_dcg=DCG(df_qid4_sorted['Relevance'])  
    2 print("Ideal DCG:\t",ideal_dcg)
```

Ideal DCG: 20.989750804831445

```
[ ] 1 dcg=DCG(df_qid4['Relevance'])  
    2 print("DCG:\t",dcg)
```

DCG: 12.550247459532576

Ideal DCG: 20.989750804831445

DCG: 12.550247459532576

Assume a model that simply ranks URLs on the basis of the value of feature 75 (sum of TF-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”.

On the basis of Feature 75, all the documents are sorted in descending order that is, the highest the value the most relevant the document is.

Then a function called fun_precision is defined which calculates precision for all the relevances using the formula :

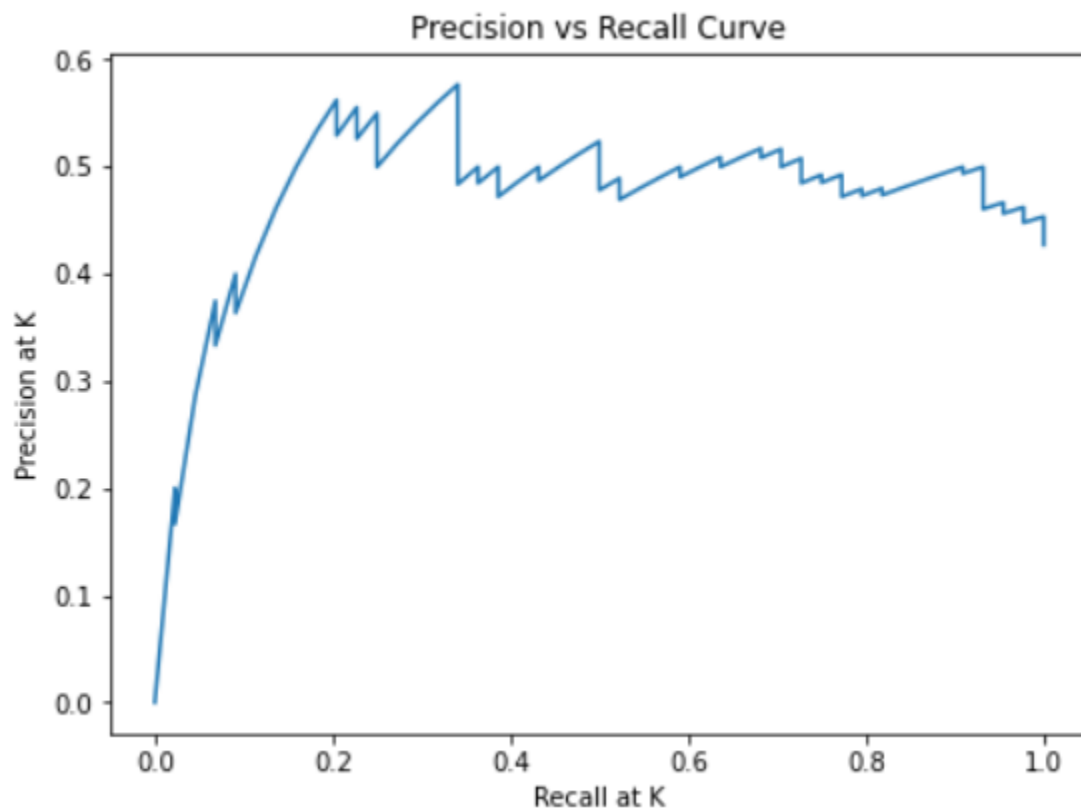
Precision = # of documents that are relevant / # of items recommended

0.0
0.0
0.0
0.0
1
0.2
0.16666666666666666
1
0.2857142857142857
1
0.375
0.3333333333333333
1
0.4
0.36363636363636365
1
0.41666666666666667
1
0.46153846153846156
1
0.5
1
0.5333333333333333
1
0.5625
0.5294117647058824
1
0.5555555555555556
0.5263157894736842
1
0.55
0.5238095238095238
0.5
1

Then a function `fun_recall` is defined, which calculates recall for all the relevances using the formula :

Recall = # of documents that are relevant / # of all the possible relevant items

The Precision VS Recall graph is plotted using matplotlib library.



Question-3: Naive Bayes Classifier

Given a dataset 20_newsgroup.

We need to pick the 5 mentioned classes for classification.

TC-ICF is used as a feature selection technique, which is an extension of TF-IDF

TF calculated as follows: Number of occurrences of a term in all documents of a particular class

CF calculated as follows: Number of classes in which that term occurs

Inverse-Class Frequency (ICF): $\log(N / CF)$, where N represents the number of classes

1. First pre-processing is performed over the files.
2. Dataset is being split into test and train data set on the basis of a random selection of documents.
3. Select top k features of each class for TF-ICF implementation
4. Training the Naive Bayes Classifier on the training data for each class
5. Test the test data by applying the classifier and find the Classification_report and Accuracy of the model.
6. Training and testing the dataset over the different ratios of train and test data.

Below are the results for the classification :

```

Accuracy: 0.9816643159379408
      precision    recall  f1-score   support

 comp.graphics      0.98      0.98      0.98       294
rec.sport.hockey     1.00      0.99      1.00       306
   sci.med          0.99      0.96      0.98       287
   sci.space        0.95      0.99      0.97       294
talk.politics.misc   0.99      0.98      0.99       237

   accuracy
 macro avg      0.98      0.98      0.98     1418
weighted avg      0.98      0.98      0.98     1418

array([[289,  0,  0,  5,  0],
       [ 1, 304,  0,  0,  1],
       [ 3,  0, 276,  7,  1],
       [ 1,  0,  2, 291,  0],
       [ 1,  0,  1,  3, 232]])

```

```

Accuracy: 0.977112676056338
      precision    recall  f1-score   support

 comp.graphics      0.97      0.99      0.98       122
rec.sport.hockey     1.00      0.99      1.00       120
   sci.med          1.00      0.94      0.97       128
   sci.space        0.93      1.00      0.96       107
talk.politics.misc   0.99      0.97      0.98        91

   accuracy
 macro avg      0.98      0.98      0.98       568
weighted avg      0.98      0.98      0.98       568

array([[121,  0,  0,  1,  0],
       [ 1, 119,  0,  0,  0],
       [ 2,  0, 120,  5,  1],
       [ 0,  0,  0, 107,  0],
       [ 1,  0,  0,  2,  88]])

```

TF-ICF : Results

```
tf1 = termfreq(word_set1,Xtrain1)
tf2 = termfreq(word_set2,Xtrain2)
tf3 = termfreq(word_set3,Xtrain3)
tf4 = termfreq(word_set4,Xtrain4)
tf5 = termfreq(word_set5,Xtrain5)

[38] tf1 = termfreq(word_set1,Xtrain1)

[39] print(tf1)

{'fombaron': 0, 'acceptable': 0, 'deep': 0, 'geneva': 0, 'conflict': 0, 'brakfast': 0, 'lobby': 0, 'baylor': 0, 'kitt': 0, 'multiprocessor': 0, 'ice': 0, 'harbor'}
```

```
print(tf1)

{'fombaron': 0, 'acceptable': 0, 'deep': 0, 'geneva': 0, 'conflict': 0, 'brakfast': 0, 'lobby': 0, 'baylor': 0, 'kitt': 0, 'multiprocessor': 0, 'ice': 0, 'harbor'}
```

```
def ICF(word set):
```

```
print(tf1)

{'fombaron': 2, 'acceptable': 6, 'deep': 4, 'geneva': 0, 'conflict': 1, 'brakfast': 0, 'lobby': 0, 'baylor': 0, 'kitt': 0, 'multiprocessor': 2, 'ice': 133, 'harbor'}
```

```
[48] tf2 = termfreq(word_set2,Xtrain2)

print(tf2)

{'hairline': 2, 'aggravated': 1, 'acceptable': 5, 'inland': 3, 'mixes': 2, 'deep': 17, 'resticted': 1, 'circumvents': 1, 'whew': 0, 'ovary': 2, 'kohen': 0, 'lobby'}
```

```
[50] tf3 = termfreq(word_set3,Xtrain3)
```

```
print(tf3)
```

```
{'overheard': 1, 'acceptable': 5, 'rushdie': 0, 'deep': 12, 'conflict': 4, 'whew': 2, 'supervising': 2, 'lobby': 4, 'baylor': 0, 'civilian': 5, 'scream': 4, 'bare
```