Information Retrieval Assignment 2

Question 1:

Data Preprocessing:

Preprocessing is done similarly to Assignment 1. We have preprocessed the relevant text by extracting the title and text and concatenating both fields. The data from tags is extracted using the BeautifulSoup library. We have performed the same for all the 1400 files and shown the text before and after the preprocessing for five such samples in our code file. Here, we are showing the output for one such file.

```
Before Extraction :  <doc>
<docno>
1223
</docno>
<title>
inviscid-incompressible-flow theory of static two-dimensional
solid jets, in proximity to the ground .
</title>
<author>
strand,t.
</author>
<biblio>
j. ae. scs. 1962, 170.
</biblio>
<text>
  the inviscid-incompressible-flow theory of static two-dimensional
solid jets impinging orthogonally on the ground is presented
using conformal mapping methods .
  it is shown that the thrust of a solid jet at constant power
initially decreases as the ground is approached .  the magnitude
of the thrust out of ground effect is regained only at a very
low height-to-jet width ratio (approximately 0.55) .  the maximuin
decrease is about 6 percent .  the ground effect on solid
jets is thus largely unfavorable .
</text>
</doc>
```

After Extraction :

```
inviscid-incompressible-flow theory of static two-dimensional
solid jets, in proximity to the ground .

  the inviscid-incompressible-flow theory of static two-dimensional
solid jets impinging orthogonally on the ground is presented
using conformal mapping methods .
  it is shown that the thrust of a solid jet at constant power
initially decreases as the ground is approached .  the magnitude
of the thrust out of ground effect is regained only at a very
low height-to-jet width ratio (approximately 0.55) .  the maximuin
decrease is about 6 percent .  the ground effect on solid
jets is thus largely unfavorable .
```

We have done the preprocessing specified in the assignment in the mentioned order:-

1. Convert the text to lowercase.
2. Perform tokenization on lowercase text input using nltk library.
3. Removed stopwords from the tokenized dataset using nltk library.
4. Removed punctuation using string library.
5. Removed leftover blank space tokens.

We have performed the above steps on the entire dataset, and the below screenshot shows the text files before and after preprocessing.

Before lower case text :

```
free-flight measurements of the static and dynamic

  charts have been prepared relating the thermodynamic properties of
air in chemical equilibrium for temperatures to 15,000 k and for pressures
from 10 to 10 atmospheres .  also included are charts showing
the composition of air, the isentropic exponent, and the speed of
sound .  these charts are based on thermodynamic data calculated by the
national bureau of standards .
```

After lower case text and before tokenization :

```
free-flight measurements of the static and dynamic

  charts have been prepared relating the thermodynamic properties of
air in chemical equilibrium for temperatures to 15,000 k and for pressures
from 10 to 10 atmospheres .  also included are charts showing
the composition of air, the isentropic exponent, and the speed of
sound .  these charts are based on thermodynamic data calculated by the
national bureau of standards .
```

```
After tokenization and before stopwords removal :  ['free-flight', 'measurements', 'of', 'the', 'static', 'and', 'dyn
amic', 'charts', 'have', 'been', 'prepared', 'relating', 'the', 'thermodynamic', 'properties', 'of', 'air', 'in', 'ch
emical', 'equilibrium', 'for', 'temperatures', 'to', '15,000', 'k', 'and', 'for', 'pressures', 'from', '10', 'to', '1
0', 'atmospheres', '.', 'also', 'included', 'are', 'charts', 'showing', 'the', 'composition', 'of', 'air', ',', 'th
e', 'isentropic', 'exponent', ',', 'and', 'the', 'speed', 'of', 'sound', '.', 'these', 'charts', 'are', 'based', 'o
n', 'thermodynamic', 'data', 'calculated', 'by', 'the', 'national', 'bureau', 'of', 'standards', '.']
After stopwords removal and before punctuations removal :  ['free-flight', 'measurements', 'static', 'dynamic', 'char
ts', 'prepared', 'relating', 'thermodynamic', 'properties', 'air', 'chemical', 'equilibrium', 'temperatures', '15,00
0', 'k', 'pressures', '10', '10', 'atmospheres', '.', 'also', 'included', 'charts', 'showing', 'composition', 'air',
',', 'isentropic', 'exponent', ',', 'speed', 'sound', '.', 'charts', 'based', 'thermodynamic', 'data', 'calculated',
'national', 'bureau', 'standards', '.']
After punctuations and before blank space token removal :  ['free-flight', 'measurements', 'of', 'the', 'static', 'an
d', 'dynamic', 'charts', 'have', 'been', 'prepared', 'relating', 'the', 'thermodynamic', 'properties', 'of', 'air',
'in', 'chemical', 'equilibrium', 'for', 'temperatures', 'to', '15,000', 'k', 'and', 'for', 'pressures', 'from', '10',
'to', '10', 'atmospheres', 'also', 'included', 'are', 'charts', 'showing', 'the', 'composition', 'of', 'air', 'the',
'isentropic', 'exponent', 'and', 'the', 'speed', 'of', 'sound', 'these', 'charts', 'are', 'based', 'on', 'thermodynam
ic', 'data', 'calculated', 'by', 'the', 'national', 'bureau', 'of', 'standards']
After blank space token removal :  ['free-flight', 'measurements', 'of', 'the', 'static', 'and', 'dynamic', 'charts',
'have', 'been', 'prepared', 'relating', 'the', 'thermodynamic', 'properties', 'of', 'air', 'in', 'chemical', 'equilib
rium', 'for', 'temperatures', 'to', '15,000', 'k', 'and', 'for', 'pressures', 'from', '10', 'to', '10', 'atmosphere
s', 'also', 'included', 'are', 'charts', 'showing', 'the', 'composition', 'of', 'air', 'the', 'isentropic', 'exponen
t', 'and', 'the', 'speed', 'of', 'sound', 'these', 'charts', 'are', 'based', 'on', 'thermodynamic', 'data', 'calculat
ed', 'by', 'the', 'national', 'bureau', 'of', 'standards']
********************************************************************************************
```

TF-IDF Matrix:

Methodology:

- Create raw term frequency for each term in each document:
    - We go through each document. Create a frequency dictionary for each doc.
    - Create an empty term-frequency dictionary that will store each term as a key and a nested dictionary corresponding to that term. The nested dictionary will have the doc ID as the key and term frequency in that document as the value
    - We iterate through each document and fill the term-frequency dictionary.
- Create inverse document frequency:
    - We iterate over each term and compute how many documents that term occurs.
    - Using the formula, compute IDF.
- Initialize empty tf-idf matrix of size no. of docs X vocab size.
- For each term and its document pair, we normalize the raw term frequency using the given weighting schemes and multiply by the IDF value.
- The tf-idf matrix construction is completed.
- We process the test query the same way as the token creation.
- Create the query vector of size vocab. Here the number of documents is one.
- Using the dot product, calculate the similarity between the tf-idf matrix and query vector. The dot product is then normalized by the norm of the tf-idf and the norm of the query vector.
- We sort and retrieve the 5 most relevant documents with their similarity score.

Pros and Cons of all weighting schemes-

- **Binary Count** - Just checks if a term appears in a document. It can help reduce the impact of very frequent terms which are not useful in context, such as stopwords. The approach is naive as we ignore each term's frequency.

- **Raw Count**- It takes into account how many times a term appears. It is helpful when the relevant term appears multiple times. It is more biased toward longer documents because they have a higher chance of repeating the same terms.

- **Term Frequency** - It gives an advantage to a term that is repeated multiple times. However, it helps reduce the bias toward longer documents as we divide by the total number of tokens in the document.

- **Log Normalization** - It mitigates the impact of terms that have a very high frequency in comparison to other terms in the query. The bias towards terms with higher frequency is dealt with by taking the log of the raw count. However, we can lose the information that comes with higher term frequency because the values are scaled down.

- **Double Normalization** - It helps mitigates the effect of higher frequency terms in the longer documents. The method is unstable and hard to tune. The value 0.5 might not be optimal in every case.

Results:

```
Enter the number of queries. 1
Query : Which document should this query belong to? Maybe our algorithm can help figure it out.
Query 1:  Which document should this query belong to? Maybe our algorithm can help figure it out.
Documents retrieved for query 1 for binary:
 +----+------------+---------------+
|    |      Score | Document      |
|----+------------+---------------|
|  0 | 0.00353159 | cranfield0153 |
|  1 | 0.00319108 | cranfield0928 |
|  2 | 0.00277138 | cranfield0962 |
|  3 | 0.00250726 | cranfield0798 |
|  4 | 0.00250726 | cranfield0499 |
+----+------------+---------------+
```

Documents retrieved for query 1 for raw count:

	Score	Document
0	0.00359466	cranfield0962
1	0.00307852	cranfield0928
2	0.00295812	cranfield0914
3	0.00253147	cranfield0798
4	0.00250967	cranfield1147

Documents retrieved for query 1 for term frequency:

	Score	Document
0	0.00454263	cranfield0153
1	0.00194453	cranfield0914
2	0.00184536	cranfield0130
3	0.00162415	cranfield0463
4	0.00155529	cranfield0457

Documents retrieved for query 1 for log normalization:

	Score	Document
0	0.00334915	cranfield0962
1	0.00325503	cranfield0928
2	0.00306321	cranfield0914
3	0.00298822	cranfield0153
4	0.00259431	cranfield0798

Documents retrieved for query 1 for double normalization:

	Score	Document
0	0.00385409	cranfield0153
1	0.00302948	cranfield0928
2	0.00267723	cranfield0962
3	0.00245303	cranfield0914
4	0.0023455	cranfield0499

Jaccard Coefficient:

Methodology:

- For each document, we find the unique tokens in it.
- We processed the test query the same way tokens were created.
- Now for each document,
    - We find the number of intersecting tokens in both the document and query.
    - Find the length of the intersection list.
    - We find the union of the number of unique tokens in both the document and query.
    - Find the length of the union list.
    - The jaccard coefficient is calculated by dividing the intersection length by the union list's length.
- We sort and retrieve the top 10 documents with the highest Jaccard coefficient.

Results:

```
Enter the number of queries.1
Query : Which document should this query belong to? Maybe our algorithm can help figure it out.
Query 1:  Which document should this query belong to? Maybe our algorithm can help figure it out.
Documents retrieved for query 1 using Jaccard Coefficient:
 +----+-----------+---------------+
|    |     Score | Document      |
|----+-----------+---------------|
|  0 | 0.1       | cranfield0281 |
|  1 | 0.0754717 | cranfield0835 |
|  2 | 0.0740741 | cranfield0153 |
|  3 | 0.0704225 | cranfield1275 |
|  4 | 0.0694444 | cranfield0637 |
|  5 | 0.0677966 | cranfield0855 |
|  6 | 0.0666667 | cranfield0909 |
|  7 | 0.0657895 | cranfield1388 |
|  8 | 0.0638298 | cranfield0506 |
|  9 | 0.0632911 | cranfield0469 |
| 10 | 0.0625    | cranfield1323 |
| 11 | 0.0615385 | cranfield0768 |
| 12 | 0.0615385 | cranfield0817 |
| 13 | 0.0612245 | cranfield1102 |
| 14 | 0.0606061 | cranfield0376 |
| 15 | 0.0597015 | cranfield0067 |
| 16 | 0.0595238 | cranfield1155 |
| 17 | 0.0588235 | cranfield0532 |
| 18 | 0.0576923 | cranfield0385 |
| 19 | 0.0574713 | cranfield0377 |
+----+-----------+---------------+
```

Question 2:

We define a function called **preprocess** for preprocessing text data. The function takes in a string of text and performs the following operations:

1. Converts the text to lowercase using the lower method.
2. Tokenizes the text into individual words using the word_tokenize method from the nltk library.
3. Removes stop words using a list of stop words.
4. Removes punctuation using the string.punctuation constant.
5. Lemmatizes each word using the lemmatize method from the nltk library.

We define a class called **TfICf** for calculating the tf-idf scores for each term in a corpus of documents based on their category.
The class has several methods:

1. **TF** : calculates the term frequency for each term in each category.
2. **CF** : calculates the class frequency for each term in the corpus.
3. **tficfScore** : calculates the tf-icf score for each term in each category.
4. **fit** : calls the TF, CF, and tficfScore methods to compute the final tf-icf scores for each term.

The class assumes that the corpus is a list of documents and the category is an array of labels corresponding to each document in the corpus.

We define a class called **NaiveBayes** for performing document classification using the Naive Bayes algorithm. The constructor initializes several variables, including the vocabulary, the term frequency scores for each term in each category, and the term frequency for each term in the corpus.
The class has two main methods:

1. **fit** : fits the Naive Bayes classifier to the training data. It takes in the training corpus and the corresponding labels and calculates the prior probability of each class and the likelihood of each term in each class.
2. **predict** : predicts the labels of the test data based on the fitted Naive Bayes classifier. It takes in the test corpus and returns the predicted labels.

The algorithm uses smoothing by adding 1 to the numerator and the vocabulary size to the denominator of the likelihood calculation to avoid zero probabilities.

We have tried different splits (50 : 50, 60 : 40, 70 : 30, 80 : 20) and different weighting methods (TF-IDF, TF-ICF).
The best model we get is for TF-ICF weighting method with a 50 : 50 split. The TF-IDF method does not work as well as TF-ICF because it does take into account the class information provided in the data.

50 : 50 Split with TF-ICF Score:

```
**********************************************************
TRAINING REPORT
**********************************************************
              precision    recall  f1-score   support

     business       0.96      0.92      0.94       168
entertainment       0.92      0.93      0.93       136
     politics       0.96      0.96      0.96       137
        sport       0.96      0.99      0.98       173
         tech       0.93      0.92      0.93       131

     accuracy                           0.95       745
    macro avg       0.95      0.95      0.95       745
 weighted avg       0.95      0.95      0.95       745

**********************************************************
TESTING REPORT
**********************************************************
              precision    recall  f1-score   support

     business       0.94      0.89      0.91       168
entertainment       0.91      0.99      0.95       137
     politics       0.93      0.92      0.93       137
        sport       0.96      0.99      0.98       173
         tech       0.93      0.88      0.91       130

     accuracy                           0.94       745
    macro avg       0.93      0.93      0.93       745
 weighted avg       0.94      0.94      0.94       745
```

Question 3(i)

Preprocessing:

I first filtered the dataset based on column value qid:4 as asked in the assignment, which resulted in 103 records.

Methodology:

- The first objective was to create a file that rearranges the query-URL pairs in order of the maximum DCG (discounted cumulative gain).
- To achieve the same, I have reversed sorted column 1 so that the query with the maximum relevance score is at the top, followed by queries with a lower score to maximize the DCG.

Results:

The number of ways this can be attained is: `1!*17!*26!*59!`

Question 3(ii)

Methodology:

- To calculate nDCG for the 50th document, I calculated the DCG of the 50th document and iDCG of the entire dataset and calculated the nDCG of 50th document.
- To calculated nDCG of the entire dataset, I calculated the DCG of entire dataset and used iDCG calculated in the above step to calculate nDCG of the entire dataset,

Results:

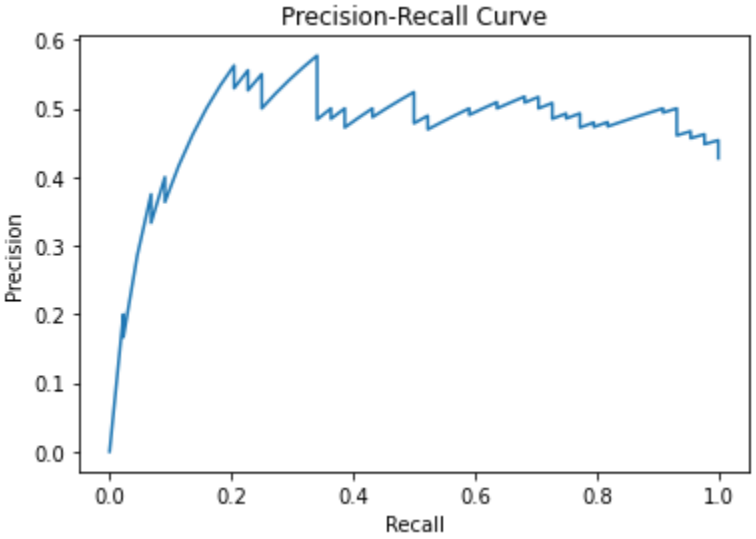
nDCG of the 50th document: 0.3521042740324887
nDCG of the entire dataset: 0.5979226516897831

Question 3(iii)

Methodology:

- For the third task, I have reverse sorted the dataset based on the value of the 75th feature
- I assigned a relevance score of 1 if the value of the relevance score is greater than 0 else, the value is kept at 0
- I have calculated the value of precision and recall for the entire dataset, and the precision-recall curve is as follows

Results:



We can see with this curve that recall increases when we keep retrieving the documents, and precision decreases. Precision is useful when we may want only the top few results, but not all the relevant results but recall is important if we want to get all the relevant results. So, there is always a tradeoff between the two depending upon the scenario. The best value of the recall is 0.93, while the precision value is 0.50, which shows a good balance between both metrics.