IR Assignment 2
Report

Team Members-

1. **Kartik Jain**
2. **Uttkarsh Singh**
3. **Darsh Parikh**

Group Number- 4

Q1.

Initially, the dataset is loaded into the program and then relevant text extraction and preprocessing are done similarly to the previous assignment.

To compute the Term Frequency, the frequency count of each term in every document is computed and stored as a nested dictionary for each document. The outer dictionary is indexed by the document name and the inner dictionary is indexed by the term. The value of the inner dictionary is the frequency count of the term in the document.

Then we create a posting list for each term that contains the list of documents in which the term occurs, and length of the term list. After this idf is created for each term.

Then tf-idf matrix is created by using 5 different methods as stated in the question. After this, we calculate each document's score for a given query using the cosine similarity.

```
# run for every query and return the top 10 documents for each query.
def run_query(query, tf_idf_matrix, idf_dict):
    scores = calculate_score(query, tf_idf_matrix, idf_dict)
    top_10 = np.argsort(scores)[::-1][:10]
    return top_10+1
```

```
os.chdir('CSE508_Winter2023_Dataset_XML')
query = preprocess('cranfield0001')
print(run_query(query, binary_tf_idf_matrix, idf_dict))
print(run_query(query, raw_count_tf_idf_matrix, idf_dict))
print(run_query(query, term_frequency_tf_idf_matrix, idf_dict))
print(run_query(query, log_normalization_tf_idf_matrix, idf_dict))
print(run_query(query, double_normalization_tf_idf_matrix, idf_dict))
```

```
[   1  484  453  601  927 1164 1064  225  173 1092]
[   1  484 1064 1092  453  673  696  695   42 1239]
[   1  484 1064  453  696  197  920 1094 1164 1075]
[   1  484 1064  453 1092 1164  927  225  695  197]
[   1 1064  601 1092  484  453  556 1218  572 1164]
```

Binary Weighting Scheme-

Pros-

1. Simple and easy to implement and helpful where document length does not matter
2. Can be effective for tasks where the presence of a term in a document is more important than its frequency

Cons-

1. Can result in loss of information
2. Does not take into account the frequency of occurrence of a term in a document

Raw Count Weighting Scheme-

Pros-

1. Simple and easy to implement
2. Takes into account the frequency of occurrence of a term in a document

Cons-

1. Document length may affect the relevance score, leading to the bias towards longer documents
2. This can lead to overemphasis on frequently occurring terms, potentially ignoring less frequent but important terms.

Term Frequency Weighting Scheme-

Pros-

1. Takes into account the frequency of occurrence of a term in a document
2. Can help address the bias towards longer documents by normalizing by document length

Cons-

1. This can lead to overemphasis on frequently occurring terms, potentially ignoring less frequent but important terms.

Log Normalization Weighting Scheme-

Pros-

1. Helps address the overemphasis on frequently occurring terms by scaling down the weight of terms that occur frequently
2. Can help reduce the impact of outliers

Cons-

1. This can result in the loss of some information, particularly in short documents
2. Requires a log transformation, which may increase the computational complexity of the algorithm

Double Normalization Weighting Scheme-

Pros-

1. Helps address the overemphasis on frequently occurring terms by normalizing document length and term frequency
2. Can help reduce the impact of outliers

Cons-

1. Requires more computational resources than other weighting schemes
2. May not be suitable for all tasks and datasets.

```python
#Jaccard Similarity
def jaccard_similarity(query, document):
    intersection = len(set(query).intersection(set(document)))
    union = len(set(query).union(set(document)))
    return intersection/union
```

```python
def get_top_10_jaccard(query):
    scores = calculate_score_jaccard(query)
    top_10 = np.argsort(scores)[::-1][:10]
    return top_10+1

query = '''simple shear flow past a flat plate in an incompressible fluid of small
viscosity . in the study of high-speed viscous flow past a two-dimensional body it
is usually necessary to consider a curved shock wave emitting from the
nose or leading edge of the body .  consequently, there exists an inviscid
rotational flow region between the shock wave and the boundary layer
.  such a situation arises, for instance, in the study of the hypersonic
viscous flow past a flat plate .  the situation is somewhat different
from prandtl's classical boundary-layer problem . in prandtl's
original problem the inviscid free stream outside the boundary layer is
irrotational while in a hypersonic boundary-layer problem the inviscid
free stream must be considered as rotational .  the possible effects of
vorticity have been recently discussed by ferri and libby .  in the present
paper, the simple shear flow past a flat plate in a fluid of small
viscosity is investigated .  it can be shown that this problem can again
be treated by the boundary-layer approximation, the only novel feature
being that the free stream has a constant vorticity .  the discussion
here is restricted to two-dimensional incompressible steady flow .'''
top10 = get_top_10_jaccard(query)
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1400.0), HTML(value='')))
[   2  664  310  134    4 1251    3  389  334  308]
```

Q2.

Initially we read the BBC news testing and training data. Then we preprocess the data.

```
df['Text'] = df['Text'].apply(preprocess)
text = df['Text']
text.head()
```

```
0    worldcom ex-boss launch defenc lawyer defend f...
1    german busi confid slide german busi confid fe...
2    bbc poll indic econom gloom citizen major nati...
3    lifestyl govern mobil choic faster better funk...
4    enron bos payout eighteen former enron directo...
Name: Text, dtype: object
```

Then we created a word_dict which will store the word and corresponding frequency of the word in different categories.

```
{'worldcom': {'business': 54},
 'ex-boss': {'business': 2},
 'launch': {'business': 41,
  'tech': 120,
  'entertainment': 25,
  'sport': 8,
  'politics': 40},
 'defenc': {'business': 17,
  'sport': 37,
  'entertainment': 1,
  'tech': 9,
  'politics': 15},
```

We then find the number of documents in each category.

```
{'business': 336,
 'tech': 261,
 'politics': 274,
 'sport': 346,
 'entertainment': 273}
```

After this, we again created a word_dict which will store the word and corresponding frequency of the word in different categories and then calculated the inverse class frequency for each word and category. Then we calculated the TF-ICF score for each word and category using the ICF data.

```
{1833: {'Answer': 'business',
  'business': 774.7450052123727,
  'tech': 27.76984706801326,
  'entertainment': 6.686648051173971,
  'sport': 9.504957345861829,
  'politics': 21.227062930956663},
 154: {'Answer': 'business',
  'business': 477.66408898562685,
  'sport': 5.898544104902195,
  'politics': 73.24243949261998,
  'entertainment': 8.566409824949016,
  'tech': 82.5829447929965},
 1101: {'Answer': 'business',
  'business': 374.97427638853577,
  'tech': 17.271276750217016,
  'politics': 98.41473444723054,
  'entertainment': 6.6079569122599455,
  'sport': 6.164606673778912},
 1976: {'Answer': 'tech',
  'tech': 1264.5156250942982,
  'entertainment': 119.80282660223186,
  'business': 128.1297322605036,
  'politics': 40.94402117276413,
  'sport': 8.476785775879657},
 917: {'Answer': 'business',
```

```
Modified_df = pd.DataFrame.from_dict(tf_icf_scores,orient='index')
Modified_df.head()
```

	Answer	business	tech	entertainment	sport	politics
1833	business	774.745005	27.769847	6.686648	9.504957	21.227063
154	business	477.664089	82.582945	8.566410	5.898544	73.242439
1101	business	374.974276	17.271277	6.607957	6.164607	98.414734
1976	tech	128.129732	1264.515625	119.802827	8.476786	40.944021
917	business	272.912149	27.396586	20.113906	17.597321	20.113906

Then we split the data into 70:30 ratio for training and testing and built a naive Bayes model using Gaussian. This model gave an accuracy of 94.4% and we also calculated the probabilities of each category based on this model.

```
def calculate_category_prob(train):
    category_prob = {}
    total = len(train)
    for i in train.value_counts().index:
        category_prob[i] = train.value_counts()[i]/total
    return category_prob
prob = calculate_category_prob(y_train)
print('Category probabilities: ',prob)
```

```
Accuracy: 0.9440715883668904
              precision    recall  f1-score   support

    business       0.88      0.95      0.92       103
entertainment       0.96      0.98      0.97        89
    politics       0.99      0.81      0.89        81
       sport       0.98      1.00      0.99        97
        tech       0.94      0.96      0.95        77

    accuracy                           0.94       447
   macro avg       0.95      0.94      0.94       447
weighted avg       0.95      0.94      0.94       447

-----------------------------------------
Category probabilities:  {'sport': 0.23873441994247363, 'business': 0.2233940556088207, 'politics': 0.1850431447746884, 'entertainment': 0.176414189837
00863, 'tech': 0.17641418983700863}
```

Then we split the data into 60:40 ratio for training and testing and built a naive Bayes model using Gaussian. This model gave an accuracy of 93.28%

```
def calculate_category_prob(train):
    category_prob = {}
    total = len(train)
    for i in train.value_counts().index:
        category_prob[i] = train.value_counts()[i]/total
    return category_prob
prob = calculate_category_prob(y_train)
print('Category probabilities: ',prob)
```

```
Accuracy: 0.9328859060402684
              precision    recall  f1-score   support

    business       0.89      0.90      0.89       145
entertainment       0.95      0.97      0.96       116
    politics       0.93      0.83      0.88       103
       sport       0.98      1.00      0.99       131
        tech       0.91      0.95      0.93       101

    accuracy                           0.93       596
   macro avg       0.93      0.93      0.93       596
weighted avg       0.93      0.93      0.93       596

-----------------------------------------
Category probabilities:  {'sport': 0.24049217002237136, 'business': 0.21364653243847875, 'politics': 0.1912751677852349, 'tech': 0.1789709172259508, 'e
ntertainment': 0.1756152125279642}
```

Then we split the data into 80:20 ratio for training and testing and built a naive Bayes model using Gaussian. This model gave an accuracy of 95.97%

```
def calculate_category_prob(train):
    category_prob = {}
    total = len(train)
    for i in train.value_counts().index:
        category_prob[i] = train.value_counts()[i]/total
    return category_prob
prob = calculate_category_prob(y_train)
print('Category probabilities: ',prob)
```

```
Accuracy: 0.959731543624161
              precision    recall  f1-score   support

    business       0.92      0.95      0.94        64
entertainment       0.98      0.97      0.98        63
    politics       0.98      0.87      0.92        53
       sport       0.98      1.00      0.99        65
        tech       0.93      1.00      0.96        53

    accuracy                           0.96       298
   macro avg       0.96      0.96      0.96       298
weighted avg       0.96      0.96      0.96       298

-----------------------------------------
Category probabilities:  {'sport': 0.23573825503355705, 'business': 0.22818791946308725, 'politics': 0.18540268456375839, 'entertainment': 0.1761744966
442953, 'tech': 0.174496644295302}
```

Then we calculated the feature probability for each category. Firstly we calculated the total sum of TF-ICF values for each category and then using this sum we calculated the probability of each category.

```
#Calculate the total sum of tf-icf values for each category
category_totals =  {}
for article in tf_icf_scores:
    category = tf_icf_scores[article]['Answer']
    if category not in category_totals:
        category_totals[category] = 0
    for feature in tf_icf_scores[article]:
        if feature != 'Answer':
            category_totals[category] += tf_icf_scores[article][feature]

#Calculate probability of each feature given each category
feature_prob = {}
for feature in word_dict:
    feature_prob[feature] = {}
    for category in class_count:
        category_total = category_totals[category]
        category_feature_total = 0
        if category in word_dict[feature]:
            category_feature_total = word_dict[feature][category]
        if category_total>0:
            feature_prob[feature][category] = category_feature_total / category_total
        else:
            feature_prob[feature][category] = 0

print(feature_prob)
```

```
{'worldcom': {'business': 0.00017450708082539628,
  'tech': 0.0,
  'politics': 0.0,
  'sport': 0.0,
  'entertainment': 0.0},
 'ex-boss': {'business': 6.463225215755417e-06,
  'tech': 0.0,
  'politics': 0.0,
  'sport': 0.0,
  'entertainment': 0.0},
 'launch': {'business': 0.0,
  'tech': 0.0,
  'politics': 0.0,
  'sport': 0.0,
  'entertainment': 0.0},
 'defence': {'business': 0.0,
  'tech': 0.0,
  'politics': 0.0,
  'sport': 0.0,
  'entertainment': 0.0},
 'lawyer': {'business': 0.0,
  'tech': 0.0,
  'politics': 0.0,
  'sport': 0.0,
  'entertainment': 0.0},
 'defending': {'business': 8.961060358060046e-07,
  'tech': 2.5544160657162264e-07,
  'politics': 1.2912802669279867e-06,
  'sport': 8.656976244339831e-06,
  'entertainment': 0.0},
```


Q3.

First of all the dataset is read in the program and then necessary preprocessing is done which includes selecting the data for which qid:4 after which the data is sorted according to the relevance score and then the maximum DCG is calculated.

```
Maximum DCG is => 20.989750804831445
```

The number of zeroes, ones, twos, threes and fours is calculated from the relevance score to find out the total number of files that can be made.

```
# calculate the number of files that can be made
files = factorial(zeroes)*factorial(ones)*factorial(twos)*factorial(threes)*factorial(fours)

print("Number of files that can be made are: ", files)

Number of files that can be made are:  19893497375938370599826047614905329896936840170566570588205180312704857992695193482412686565431050240000000000000000000000
```

Then nDCG is calculated using IDCG and DCG. We calculate the nDCG for position 50 and for the whole dataset.

```
# Calculating nDCG at 50

ndcg50 = float(data_dict_copy[0]['score'])
max_ndcg50 = float(data_dict[0]['score'])
# print(ndcg50)
# print(max_ndcg50)
for i in range(1, 51):
    temp1 = (float(data_dict_copy[i]['score'])/log2(i+1))
    ndcg50 = ndcg50 + temp1
    temp2 = (float(data_dict[i]['score'])/log2(i+1))
    max_ndcg50 = max_ndcg50 + temp2

ndcg_50 = ndcg50/max_ndcg50
print("nDCG at position 50 is ", ndcg_50)
```

```
nDCG at position 50 is  0.3521042740324887
```

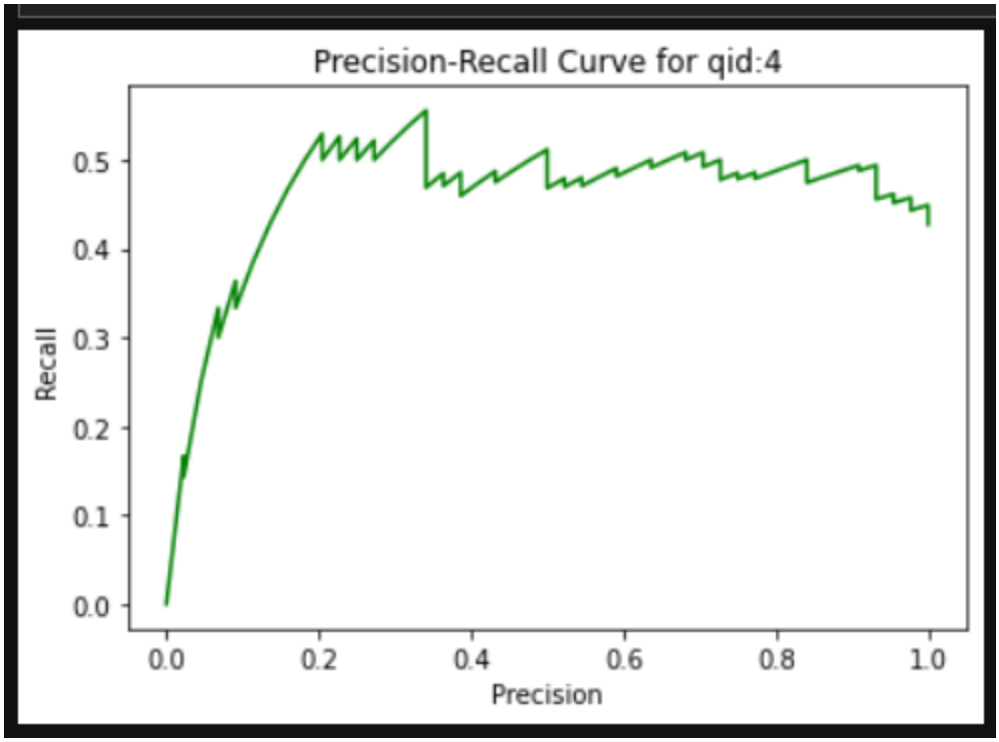
```
maxnDcg = float(data_dict[0]['score'])
for i in range(1, len(data_dict)):
    temp1 = (float(data_dict[i]['score'])/log2(i+1))
    maxnDcg = maxnDcg + temp1

ndcg_val = float(data_dict_copy[0]['score'])
for i in range(1, len(data_dict_copy)):
    temp2 = (float(data_dict_copy[i]['score'])/log2(i+1))
    ndcg_val = ndcg_val + temp2

ndcg_whole = ndcg_val/maxnDcg
print("nDCG for Whole Dataset is", ndcg_whole)
```

```
nDCG for Whole Dataset is 0.5979226516897831
```

After this, a precision-recall plot is made by using the model which uses variable 75 as the relevance scores.


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