CSE508: Information Retrieval
Assignment 2

Rishi Raj 2019090
Robin Garg 2019092

Q1

a. First, we performed some preprocessing steps (using nltk python library) as required.

1. First converted the text to lower case
2. Then I did word tokenization
3. Removed stopwords from tokens
4. Removed punctuation marks from tokens
5. Removed blank space tokens

b. Then I created the final list of tokens for calculating the Jaccard Coefficient. I applied the formula

$$\text{Jaccard Coefficient} = \text{Intersection of (doc,query)} \;/\; \text{Union of (doc,query)}$$

To find the coefficient and then sorted all the given documents using the coefficient found. At last, I ranked the documents and printed top 5 relevant documents according to the Jaccard coefficient.

INPUT

The Input is given as present in the input file which is first the number of queries and then next n lines contains the query to be processed using Jaccard Coefficient.

OUTPUT

The output is first five relevant documents found by ranking using Jaccard coefficient

Sample Input:

*Q1_input - Notepad

File Edit Format View Help

```
12
american dream
lion king
A series of death at a hospital for elderly patients in shanghai is underscoring the dangerous consequences of china's
daily music television show guests
still living in the fifties
dumb rich ugly guys
This gives it an atomic
Born and raised
and Adam was named
I've never my honey
And their French cousin
days to go to completion
```

Sample Output for Jaccard:

```
TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
oxymoron.jok
ozarks.hum
fajitas.rcp
psalm_nixon
psalm23.txt

TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
lion.jok
lines.jok
lion.txt
epi_bnb.txt
pukeprom.jok

TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
readme.bat
aclamt.txt
fascist.txt
deadlysins.txt
aussie.lng

TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
popmach
abbott.txt
hate.hum
robot.tes
commutin.jok

TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
acronym.lis
ambrose.bie
psalm_nixon
psalm23.txt
prover_w.iso
```

```
TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
acronym.lis
flattax.hum
insult.lst
collected_quotes.txt
oxymoron.jok

TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
electric.txt
addrmeri.txt
lampoon.jok
policpig.hum
polemom.txt

TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
welfare
t-shirt.hum
aeonint.txt
libraway.txt
weights.hum

TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
abbott.txt
acronyms.txt
lifeonledge.txt
tribble.hum
gd_frasr.txt

TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
oilgluts.hum
normal.boy
nintendo.jok
lawskool.txt
cucumber.jok
```

```
TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
recipe.001
t-10.hum
arnold.txt
chung.iv
making_y.wel

TOP 5 DOCUMENTS BASAED ON JACCARD COEFFICINT
is_story.txt
gameshow.txt
skippy.txt
whitbred.txt
bunacald.fis
```

c. The next goal was to create a TF-IDF matrix. For that we needed two things, first was tf i.e., term frequency and second IDF i.e., document frequency of each word. So, we calculated the document frequency using the posting list of the input dataset and then calculated the term frequency of each word in each document using all the 5 different variants given in the question. After finding both the terms, we created the TF-IDF matrix of size no of documents X vocabulary. Then we found the Tf-score of the input query using the TF-IDF matrix found and ranked the documents using Tf-score in decreasing order of the score.

The pros and cons of different variants are:

1. Binary scheme make it simpler to calculate but may miss important information due to unavailability of number of terms.
2. Raw count is the most general way but it can be biased for documents of varying length like too short or too long documents.
3. Term frequency overcome the issue in raw count by dividing frequency with total frequency but it may assign low values to the words that are relatively more important.
4. We can use log normalization to handle large data but after a certain of time even if number of terms increase dramatically it will not change the TF-IDF score much.
5. Double normalization is also another way to normalize the large terms present but it is much more dependent on the maximum occurring frequency hence it can be biased for a very short document.

INPUT

The Input is given as present in the input file which is first the number of queries and then next n lines contains the query to be processed using TF-score.

OUTPUT

The output is first five relevant documents found by ranking using the TF-score.

Sample Input:

Q1_input - Notepad

File Edit Format View Help

```
1
A series of death at a hospital for elderly patients in shanghai is underscoring the dangerous consequences of china's
```

Sample Output:

```
A series of death at a hospital for elderly patients in shanghai is underscoring the dangerous consequences of china's stuborn
pursuit of a zero-COVID approach

TOP 5 DOCUMENTS BASED ON tf_idf COEFFICINT for variant: raw_tf
score:  148.09094171375654 dcoument name:  acne1.txt
score:  92.41812005625235 dcoument name:  antibiot.txt
score:  85.20447411346699 dcoument name:  stuf11.txt
score:  83.35619184998033 dcoument name:  humor9.txt
score:  74.75182466495792 dcoument name:  stuf10.txt

TOP 5 DOCUMENTS BASED ON tf_idf COEFFICINT for variant: long_norm_tf
score:  29.09361496457005 dcoument name:  idaho.txt
score:  26.48074051971883 dcoument name:  mog-history
score:  20.31994089023259 dcoument name:  marriage.hum
score:  17.194980411584098 dcoument name:  prooftec.txt
score:  15.60353053984451 dcoument name:  variety2.asc

TOP 5 DOCUMENTS BASED ON tf_idf COEFFICINT for variant: binary_tf
score:  22.671642488757556 dcoument name:  mog-history
score:  20.157018921461656 dcoument name:  idaho.txt
score:  18.35148451769332 dcoument name:  prooftec.txt
score:  18.140002906126952 dcoument name:  marriage.hum
score:  16.57004380466664 dcoument name:  epi_.txt

TOP 5 DOCUMENTS BASED ON tf_idf COEFFICINT for variant: double_norm_tf
score:  11.506671864996768 dcoument name:  mog-history
score:  10.382729139015426 dcoument name:  idaho.txt
score:  9.474089636850277 dcoument name:  prooftec.txt
score:  9.310848114507015 dcoument name:  marriage.hum
score:  8.925499235807983 dcoument name:  variety2.asc

TOP 5 DOCUMENTS BASED ON tf_idf COEFFICINT for variant: term_freq_tf
score:  0.13879188539246162 dcoument name:  adameve.hum
score:  0.11763925432304143 dcoument name:  recepies.fun
score:  0.09078681583626023 dcoument name:  feggaqui.txt
score:  0.07305780241600975 dcoument name:  appetiz.rcp
score:  0.06464433724097629 dcoument name:  badday.hum
```

Q2

Preprocessing – none required as it is a precompiled dataset

1. Only qid:4
2. Number of possible query URL pairs -

Only descending order of relevance scores will give max DCG. Queries with the same relevance scores can be rearranged amongst themselves without changing the DCG.

Therefore, total number of such files is 59!*26!*17!*1!

```
Counter({'0': 59, '1': 26, '2': 17, '3': 1})
Number of such files that can be made are
19893497375938370599826047614905329896936840170566570588205180312704857992695193482412686565431050240000000000000000000000
```

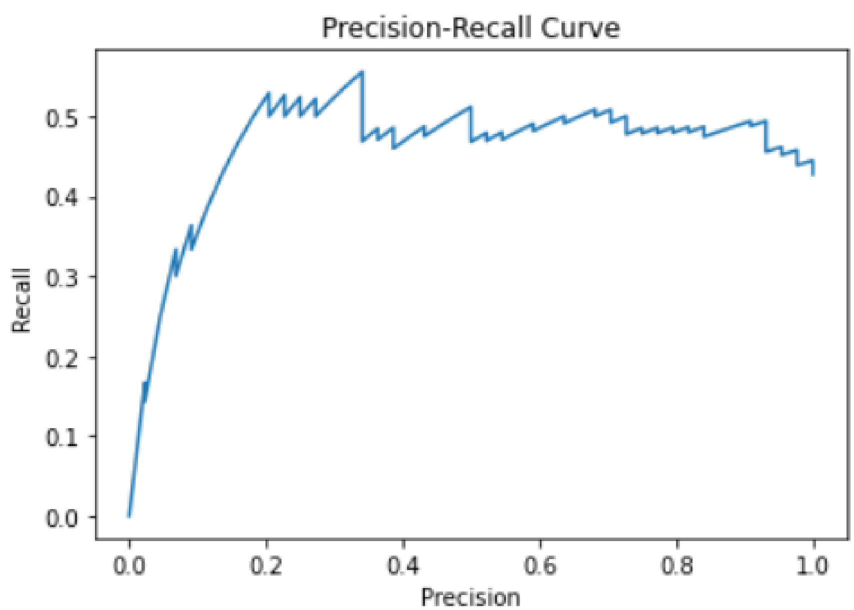
3. nDCG

Using formula

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

```
[('0', 59), ('1', 26), ('2', 17), ('3', 1)]
Whole dataset
DCG = 16.768935581665193 , IDCG = 28.98846753873482 , NDCG = 0.5784691984582591
NDCG at 50
DCG = 10.323516383590077 , IDCG = 28.98846753873482 , NDCG = 0.35612494416255847
```

4. Precision recall curve



First sort all the URLs by tf-idf in descending order, then calculate the precision and recall from top.

Q3 Naive Bayes classifier
Assumptions -

- Class frequency = “Number of classes in which that term occurs” i.e., value of class frequency is between 1 and 5 (inclusive).
- We had to select top k features, the value of k taken here is 100

Preprocessing -

- Removing punctuations
- Tokenizing the text
- Converting all tokens to lower case
- Removing English stop words
- Stemming tokens
- Lemmatizing tokens

Method

1. Calculate cf using to store the labels(classes) corresponding to each word if word is in said label.
2. Use cf, number of labels to get icf.
3. Using tf-icf to sort by descending, select top k tokens.
4. Get frequency and class of top k tokens for naive bayes algorithm.
5. In naive bayes algorithm, probabilities added using log.

Results

```
Train:Test = 80:20
Accuracy = 0.971
Confusion Matrix
[[207   1   1   0   1]
 [  0 215   0   0   1]
 [  5   1 175   4   7]
 [  2   1   1 185   1]
 [  1   1   0   1 189]]
```

```
Train:Test = 70:30
Accuracy = 0.9713333333333334
Confusion Matrix
[[295   1   2   1   2]
 [  1 312   0   0   3]
 [  5   1 269   4  11]
 [  3   1   1 300   4]
 [  1   1   0   1 281]]
```

```
Train:Test = 50:50
Accuracy = 0.9744
Confusion Matrix
[[475   1   0   1   1]
 [  2 501   1   0   2]
 [ 13   2 462   5  10]
 [ 10   3   1 494   8]
 [  1   1   0   2 504]]
```

Performance analysis
Very small accuracy difference (~0.3%) for different train test ratios.