Information Retrieval
Assignment 2

Priyanshu 2019473

Himanshu Sehrawat 2019468

Question 1: TF-IDF & Jaccard

Preprocessing

- The program processes one file at a time.
- It transforms the file into a single string of lowercase letters, and then performs word tokenization.
- Next, it removes stopword tokens using the nltk corpus.
- The program uses a regular expression ([^\w\s]) to eliminate punctuation.
- Finally, it gets rid of blank spaces and empty tokens to prepare for data structure implementation.

Methodology

Jaccard Coefficient

- The program stores the tokens of each document in advance and converts them into sets to allow for union and intersection operations.
- The input query is also preprocessed and converted into a list of tokens.
- The program then performs intersection and union operations over the tokens of both the document and the query.
- To calculate the Jaccard coefficient, the program divides the number of tokens that are present in both the document and the query by the total number of tokens that are present in both.

Query Processing

- When a query is received, we preprocess it using the same approach as we do for files.
- This returns a list of all the terms contained in the query.
- Next, we calculate the Jaccard coefficient for each document in the corpus.
- We then sort the document IDs based on the score and report the top five files.

TF-IDF Matrix

- The program generates five distinct matrices using various weighting techniques.
- Each matrix utilizes IDF, which is determined by taking the logarithm of the total number of documents divided by the smoothed document frequency of a given word.
- Document frequency refers to the number of documents in which a term appears.

Query Processing

- The program processes queries in the same way as it does for documents.
- For each term in the query, we calculate the tf-idf score for every document by adding the tf-idf scores of the words contained in the token.
- We then sort the documents in decreasing order of their scores and report the top five documents that match the query.

Assumptions

- The preprocessing steps given in the question are exhaustive.
- The matrices take a long time to build so they have been stored as CSV files in the drive.

Pros and Cons of different weighting schemes.

Binary

Pros

- This makes it easier to determine the document's relevancy without doing a lot of maths. For instance, if a document's topic is "machine learning," any question about the topic will be entirely relevant.

Cons

- Yet, it does not account for the word's frequency. Just having a word there does not mean the document is pertinent to the query.
- So, the type of question determines its importance.

Raw Count

Pros

- In comparison to other weighing techniques, it is computationally cheap.
- Simple extraction of the raw count from the term's posting list is possible.

Cons

- As the raw count does not actually represent the meaning of the term, it cannot be held accountable for the purported relevance.

Term Frequency

Pros

- It takes into account the normalized version of the raw count because the simply raw count can be a false impression of the document being relevant.
- Every weight is the probability of the occurrence of the word in the document.

Cons

- It is computationally expensive as we need to extract the length of every document after preprocessing.
- It lacks in consideration the true significance of the term and does not take into account the length of the documents to report relevance.

Log

normaliza

tion Pros

- Since every occurrence of a word does not truly carry its significance, we need to go beyond the frequency of the term and this accounts for that. It is the frequency and hence denotes the significance of the term.

Cons

- However, this does not take into account the long documents in which a word without adding to the relevance can be repetitive.

Double normalization

Pros

- The weighting scheme takes into account the anomaly that long documents have a higher frequency of words as they repeat the same words.

Cons

- However, a change in the stop word can dramatically alter the term weights.

- A document may contain a large number of a word which is not representative of the content of the query and hence does not make it more relevant.

Result:

TF - IDF:

```
Input the query
Yourr query is : removing finding doing stuff and forgetting who peoples are \t
 Preprocessed query : ['removing', 'finding', 'stuff', 'forgetting', 'peoples']
Unmatched Token Error : stuff
Unmatched Token Error : forgetting
Unmatched Token Error : peoples
cranfield0738 7.129283016944966
cranfield1323 5.910832407407448
cranfield0745 5.3469622627087245
cranfield1205 5.171978356481518
cranfield0556 4.752855344629977
```

Jaccard coefficient

```
Input the phrase query
 Your query : solutions thethree particular casesi propose give general solutionto problem indicatebriefly obtained onedimensional transient heat temperature doublelayer
 Preprocessed query : ['solutions', 'thethree', 'particular', 'casesi', 'propose', 'give', 'general', 'solutionto', 'problem', 'indicatebriefly', 'obtained', 'onedimensi
cranfield0006 0.7446808510638298
cranfield0091 0.13333333333333333
cranfield0582 0.11594202898550725
cranfield0005 0.10526315789473684
cranfield0981 0.08536585365853659
```

Question 2 : Naive Bayes with TF-ICF

Preprocessing

- Unzip all files in the folder. Extract them. Now, iterate through every file and read its text
- Removal of punctuation marks
- Word tokenizer to form tokens of all words in text
- Stemming
- Removing white spaces of each token
- Converting all letters in token to lowercase.
- Removed stop words from the list of tokens

Splitting into train and test set

- Now, for every file in the dataset, we make a list recording the paths of the files.
- The size of the list will be 5000 in our case.
- We use random.reshuffle to reshuffle the list randomly, and then pick up the first x percent elements depending on the requirement of the ratio of train test.
- For example for an 80: 20 split the result is 4000 training samples, and 1000 test samples

Implementing TF-ICF scoring

- For every term that occurs in all documents, we will compute its term frequency for each class. By this we mean, the number of times the term occurs in documents of the class. This is TF

- Now we find the number of classes in which the term occurs and store it in the dictionary. This is the CF.
- Now, for each term, we calculate its ICF.
- Now, for each term, we store its TF-ICF value by multiplying its TF and ICF value.
- For each class, we find the terms with top k TF-ICF values. These are added to the set called vocabulary which we later use in the Naive Bayes implementation.

Using the vocabulary created using TF-ICF scoring in Naive Bayes

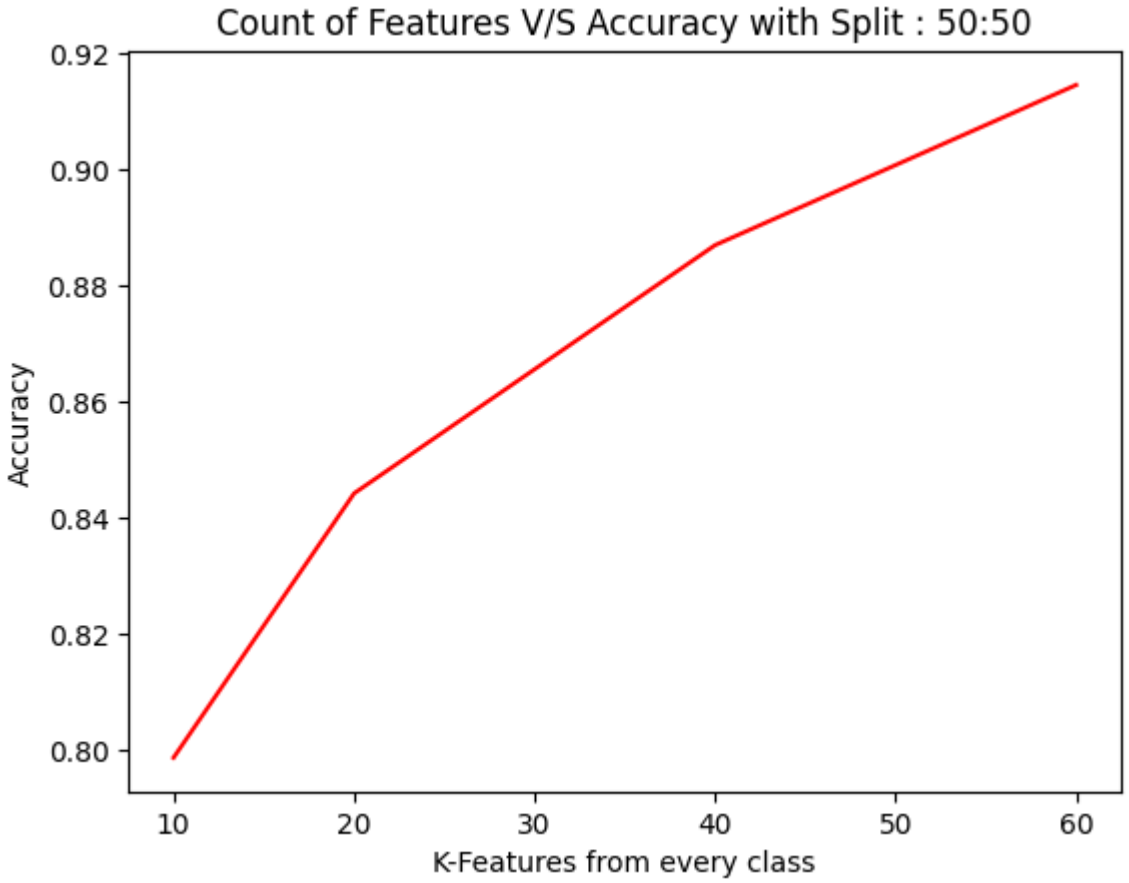
- For each document, and for each possible class ci :
- we calculate the P(c = class i | w = word) for each word in the document provided it is in the vocabulary.
- To do this we use Bayes formula: P(c = class i | w = word) = P(w = word | c = class i)*P(c = class i)/P(w = word)
- All the terms on the RHS are calculated using simple probability formulae

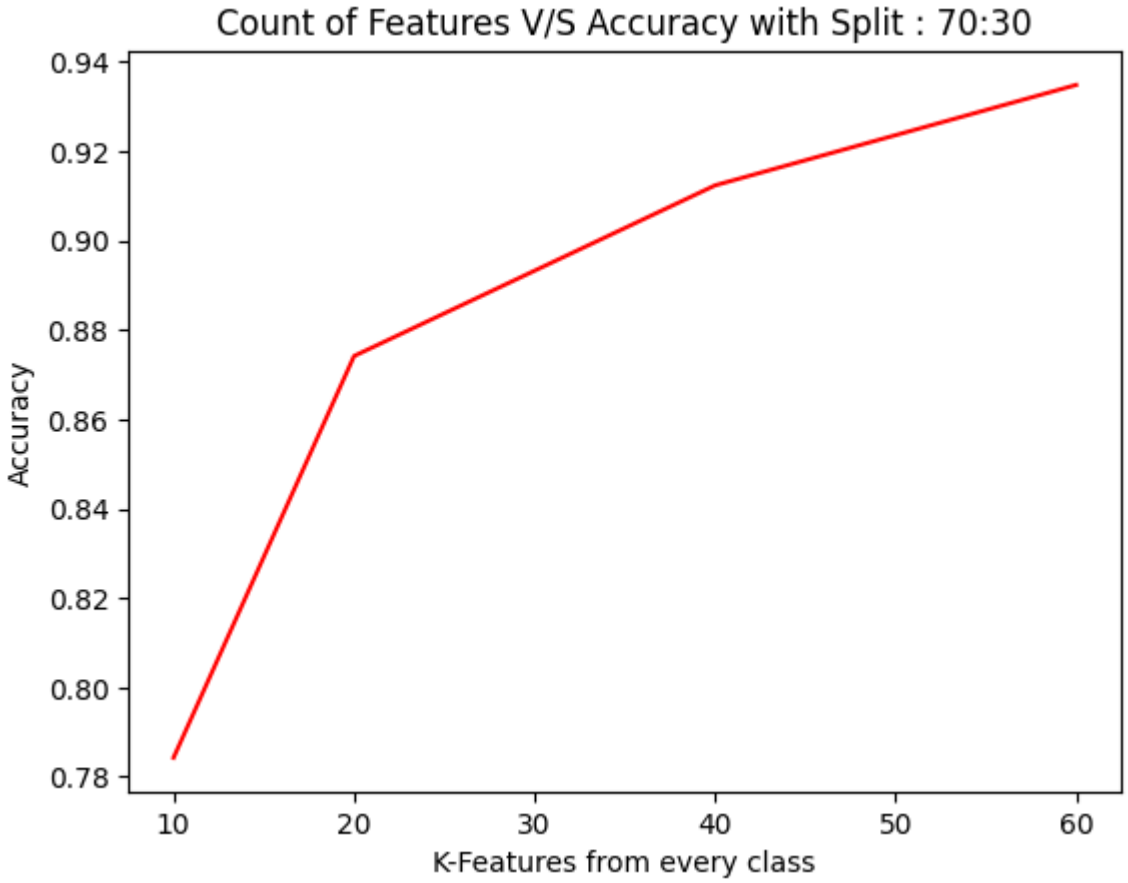
Accuracy on checking for 50: 50, 70:30, 80:20 with k = 100

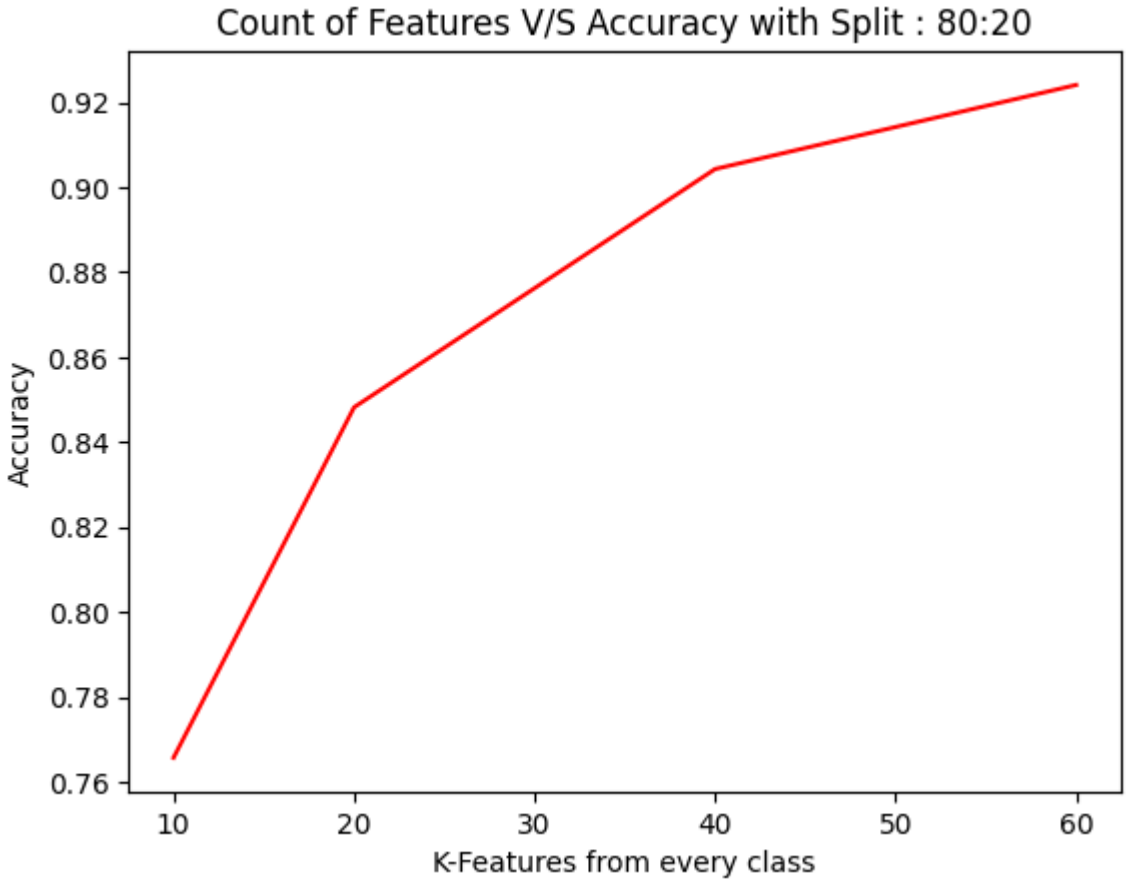
- For 80: 20 the accuracy is 92.1 percent
- For 70:30 the accuracy is 91.1133 percent
- For 50:50 the accuracy is 89.16 percent

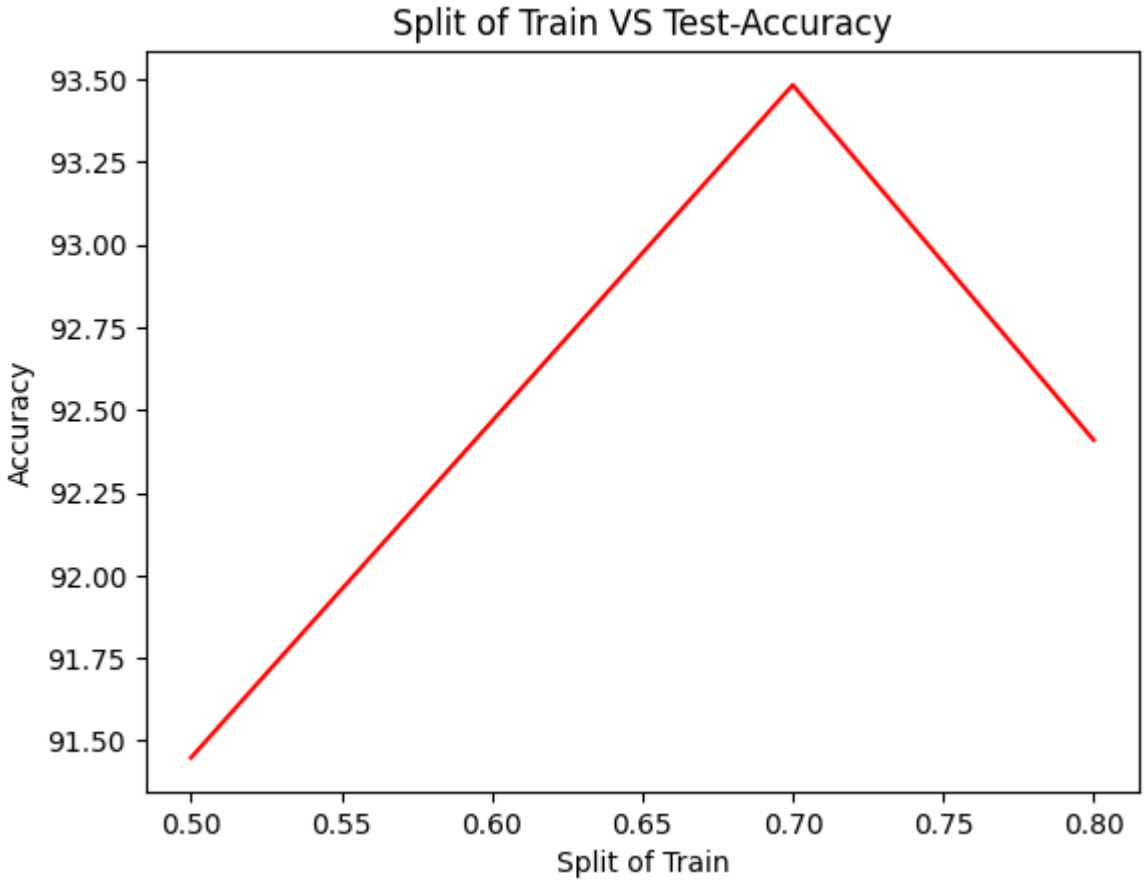
Thus when the training set size increases, the accuracy increases this is because the naive Bayes classifier sees more examples with an increase in the size of the training set. However during the demo, there is a possibility that the accuracy might slightly differ because the shuffling of the dataset is random.

Results:


Count of Features V/S Accuracy with Split : 50:50
0.92
0.90
0.88
0.86
0.84
0.82
0.80
Accuracy
10
20
30
40
50
60
K-Features from every class


Count of Features V/S Accuracy with Split : 70:30
Accuracy
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
10
20
30
40
50
60
K-Features from every class


Count of Features V/S Accuracy with Split : 80:20
Accuracy
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
10
20
30
40
50
60
K-Features from every class


Split of Train VS Test-Accuracy
93.50
93.25
93.00
92.75
92.50
92.25
92.00
91.75
91.50
Accuracy
0.50
0.55
0.60
0.65
0.70
0.75
0.80
Split of Train

Question 3: Information Retrieval System(DCG,nDCG)

Preprocessing

- Each file is preprocessed one at a time.
- Only qid:4 query URL pairs are worked upon.

Methodology

(2)

- We get max DCG when the pairs are in descending order of their relevance score.
- All the qid:4 query-URL pairs are stored in a data frame and then sorted.
- The data frame is then stored in a CSV file

(3)

- The DCG is calculated from the relevance score they are present in.
- The formula used is score1 + (score2/log(2) + score(3/log(3) +....
- The max DCG is calculated when the query-URL pairs are sorted in descending order.
- The nDCG is reported by dividing the DCG by the max DCG.

(4)

- Extract the value of the 75th feature for all qid:4.
- Sort the pairs according to the score and sort the relevance scores simultaneously.
- Consider all the pairs with scores> 0 as relevant.
- Calculate precision and recall @ k for all the pairs.
- Plot the precision-recall graph.

Assumptions

- No preprocessing of the dataset is done.

Results:

```
Maximum DCG ordering possible for queries with qid:4 : 19893497375938370599826047614
nDCG Value at 50: 0.35612494416255847
nDCG for the Whole Dataset: 0.5784691984582591
```