Assignment-2 Report

Harjeet Singh Yadav2020561
Varun Parashar2020482

Preprocessing:-

Step 1:- lowering the data to lowercase to maintain uniformity.

Step 2:- Used the NLTK library's word tokeniser to tokenise the string.

Step 3:- Used the NLTK library to remove the stopwords and punctuations from the tokens.

Step 4:- Used the regEx (**[^\w\s]**) to remove the chars other than alphabets and numbers. E.g. **removed -, ., /, \, | etc.**

Step 5:- Finally, join the tokens to form a string and write back to the same file.

Assumption:- No emojis are present in the files

Part 1)

TF-IDF stands for "Term Frequency-Inverse Document Frequency". It is a numerical statistic used to measure the importance of a term in a document or corpus, by considering the frequency of the term in the document and the frequency of the term in the corpus.

Binary:
Pro: Simple to calculate and interpret, and can help reduce the impact of frequently occurring words.
Con: Ignores the frequency of a term within a document, which can result in a loss of information.

Raw count:
Pro: Simple to calculate and useful for comparing the frequency of terms across documents.
Con: Gives more weight to longer documents, which may not necessarily be more relevant than shorter ones.

Term frequency:

Pro: Accounts for the frequency of a term within a document, and can help identify important terms in the document.
Con: Fails to account for the frequency of a term in the corpus, which can lead to less accurate results in cases where a term appears frequently across the corpus.

Log normalization:
Pro: Helps to reduce the impact of very frequent terms while still retaining their importance, and is useful for long-tailed distributions.
Con: Does not account for the frequency of a term in the corpus, and may not be appropriate for all types of documents.

Double normalization:
Pro: Helps to address the issue of long documents having more weight than short ones, while still accounting for the frequency of a term in the corpus.
Con: Can be more complex to calculate and interpret, and may not be necessary for all types of documents.

Jaccard Coefficient

The Jaccard coefficient is a measure of similarity between two sets of data. It is defined as the ratio of the size of the intersection of the sets to the size of their union. In other words, it measures the proportion of elements that are common to both sets, compared to the total number of elements across both sets. The Jaccard coefficient can be used to compare the similarity of documents, images, or other types of data represented as sets of features. A Jaccard coefficient of 1 indicates that the sets are identical, while a value of 0 indicates that they have no elements in common. The Jaccard coefficient is often used in information retrieval and data mining applications, such as clustering, classification, and recommendation systems.

Part 2)

Increasing the size of the training set and decreasing the size of the test set can have different effects on the performance of a machine learning model, depending on the specific context and the characteristics of the data. In general, however, the following effects can be expected: Better model training: With more training data, the model can potentially learn better representations of the underlying patterns in the data, and reduce the risk of overfitting to specific examples. This can lead to higher accuracy and generalization performance on the test set.

Naive Bayes classifier is a probabilistic machine learning algorithm that is based on Bayes' theorem. The algorithm is called "naive" because it makes the simplifying assumption that the features or attributes of a data point are independent of each other, given the class label.

TF-IDF weights perform better than TF-ICF (Term Frequency-Inverse Class Frequency) weights for text classification because they take into account the frequency of a term in the document and the inverse frequency of the term in the entire corpus, whereas TF-ICF only considers the frequency of the term in its respective class. TF-IDF weights are better suited for text classification because they give more weight to terms that are both frequent in the document and rare in the entire corpus, which are likely to be more informative in distinguishing between different classes. On the other hand, TF-ICF weights give more weight to terms that are frequent in the document and rare in its respective class, which may not necessarily be the most discriminative features for text classification. Additionally, TF-IDF weights can be used for both binary and multi-class classification problems, while TF-ICF is mainly designed for binary classification problems. Therefore, TF-IDF weights are generally considered to be a more effective feature weighting scheme for text classification tasks.

Pre-processing techniques play an important role in text classification tasks, as they can significantly impact the accuracy of the classification models. Different preprocessing techniques can lead to different vocabulary size and words in the vocabulary. Vocabulary may also change depending on whether we are using stemming or lemmetization in our preprocessing step. Stemming just chops the suffix of the word e.g having will change to hav, while lemmetization will convert having to have.

I have done +1 to all values of tf-icf values for smoothing.

Accuracies:-
Without smoothing :- 60%
60-50 training split :- 58%
With smoothing:- 80%
TF-IDF:- 95%
Bigram :- 95%
Trigram:- 94%

Part 3)

3. In this question, we needed to filter the data where the query id was 4. Firstly, we had to arrange the rows to maximise the overall DCG of the retrieved URLs. So, we sorted the rows in decreasing order of their relevance scores since DCG is a weighted sum of the relevance of the retrieved items where the weights decrease as we go down in ranking. This way, the maximum relevance values will have the maximum weights leading to the maximum DCG value. The number of such orderings was equal to the number of permutations possible for rows with identical relevance scores, i.e. permutations(rel = 3) * permutations(rel = 2)*permutations(rel=1)*permutations(rel=0).

Now, we computed nDCG for the ordering initially given to us. To calculate the nDCG, we calculated the DCG for the sorted and initial ordering. Then, we divide the DCG values of the initial arrangement by the DCG values of sorted ordering, giving us the nDCG value at each row. Hence, we can calculate the nDCG value at row 50 and the whole document.

Now, we get a new ordering based on feature 75, so we sort our dataset in decreasing order of the value of feature 75. We take the value of relevance as one if the relevance score is greater than 0 and zero otherwise. After, we got the binary relevance of the order. We calculate precision as (number of the relevant documents retrieved) / (number of documents retrieved) and recall as (number of the relevant documents retrieved) /(total number of relevant documents). Then we plotted these values.