Report

Q1

Preprocessing

For preprocessing, we perform the following steps (in the given order):

1. **Converting text to lowercase:** We first convert all the text in training data to lower case. This ensures that we don't differentiate between words if they only differ in their case. E.g. hello, Hello, HELLO, and HeLlO are all same words and converted to hello.
2. **Removing punctuations:** We remove all the punctuations from the text. The punctuations removed are !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
3. **Tokenization:** We token the text data i.e. break up sentences into tokens where each token is a single word.
4. **Filtering stopwords:** We remove all the stopwords that do not contribute any information to the sentence. These include words like "I", "me", "here", "there", etc.

Methodology

We calculate the Jaccard coefficient. Higher value of Jaccard coefficient between given query and the document implies document is relevant for the query. Formula for Jaccard coefficient is:

Jaccard Coefficient = Intersection of (doc,query) / Union of (doc,query)

This is the ratio of the number of terms in the intersection of the token set of the document and query to the number of terms in their union. This value always lies between 0 and 1.

TF-IDF Matrix

Preprocessing

Same as that done in the last part.

Methodology

Using the various document-wise tokens we have after preprocessing, we create the whole vocabulary set. In order to create matrices, we also map every term to a term id since each cell in a matrix is identified by a pair of indices that correspond to pairs of (document id, term id) pairs.

In order to construct the TF-IDF Matrix, two calculations must be made: the term frequency (TF) and the inverse document frequency (IDF). We first determine the total number of words in each text in order to calculate the term frequency. This is kept as a nested dictionary, with the document id serving as the key for the outer dictionary and another dictionary holding the words and their counts for the terms found in that document as the value.

We are provided with the raw term frequency for each phrase used in each document as a result. Now that we have the document frequency (DF) for each word in the lexicon, we can compute the IDF. We create posting lists for each phrase in the lexicon to determine this, and the length of the posting list is equal to the frequency of the term's occurrence in documents. We determine the IDF for each phrase using the algorithm after calculating the document frequency for each term:

IDFterm = log(Ndocs/(DFterm+1))

where Ndocs is the overall number of documents in our dataset and DFterm represents the term's document frequency.

The TF-IDF matrix is now created for each of the 5 TF weighting schemes:

The size of the 2-dimensional TF-IDF matrix is (No. of docs x no. of terms in vocab). The TF-IDF value of the word with term id 'j' in document id I corresponds to the I j)th element of the matrix. Weight(TFj, I x IDFj is equal to the value in the cell.

where TFj,i is the term frequency value for the term with term id "j" in the document with document id "i."

Five distinct TF-IDF matrices, one for each weighting method, are thus produced.

Pros and Cons

Jaccard Coefficient

- Simpler since always assigns a score between 0 and 1
- Does not consider ordering of words
- Does not consider term frequency
- Query representations can be of different sizes

TF-IDF Matrix

- More complex as compared to Jaccard coefficient
- Also doesn’t consider order of words
- Can be slower for very large vocabularies

- No semantic similarity is taken into account

Q2

Preprocessing

For preprocessing, we perform the following steps (in the given order):

5. **Converting text to lowercase:** We first convert all the text in training data to lower case. This ensures that we don't differentiate between words if they only differ in their case. E.g. hello, Hello, HELLO, and HeLlO are all same words and converted to hello.
6. **Removing punctuations:** We remove all the punctuations from the text. The punctuations removed are !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
7. **Tokenization:** We token the text data i.e. break up sentences into tokens where each token is a single word.
8. **Filtering stopwords:** We remove all the stopwords that do not contribute any information to the sentence. These include words like "I", "me", "here", "there", etc.
9. **Lemmatization:** We reduce all the words in the tokens to their root form by using a lemmatizer provided by nltk library. E.g. the word "corpora" is reduced to its base form "corpus".

TC-ICF

To compute the TC-ICF matrix, we first compute the TC matrix which gives us the number of occurrences of each term for each class. This is a C x V matrix where C is number of classes and V is vocabulary size.

We then find the ICF matrix by first finding the CF matrix which is the number of classes in which each term of the vocabulary appears. ICF is found by computing log(C / CF) where C is the number of classes. The shape of ICF matrix is 1 x V.

Finally, we find the TC-ICF matrix by element-wise multiplication of each row of TC matrix with ICF matrix to get a C x V TC-ICF matrix.

Performing Naive Bayes

To perform, we first need to find the probability of each class occurring in the training set. This can be done by dividing the frequency of a class by the total number of samples in the training set.

We then find the class-wise probability of each term in the vocabulary. These two probability matrices are used in Naive Bayes to find the prediction. Note that we multiply the posterior with TC-ICF weights which gives us a weighted Naive Bayes. Also, we use log probabilities to keep the calculations numerically stable.

Results

The results obtained on 70:30 split is:

	precision	recall	f1-score	support
0	0.98	0.96	0.97	108
1	1.00	0.99	0.99	79
2	0.97	0.98	0.97	86
3	1.00	1.00	1.00	101
4	0.96	0.99	0.97	73
accuracy			0.98	447
macro avg	0.98	0.98	0.98	447
weighted avg	0.98	0.98	0.98	447

80:20 split:

	precision	recall	f1-score	support
0	0.97	0.97	0.97	75
1	1.00	0.98	0.99	46
2	0.96	0.96	0.96	56
3	1.00	1.00	1.00	63
4	0.97	0.98	0.97	58
accuracy			0.98	298
macro avg	0.98	0.98	0.98	298
weighted avg	0.98	0.98	0.98	298

60:40 split:

	precision	recall	f1-score	support
0	0.99	0.96	0.97	137
1	1.00	0.97	0.99	109
2	0.97	0.98	0.98	109
3	1.00	1.00	1.00	129
4	0.95	0.99	0.97	112
accuracy			0.98	596
macro avg	0.98	0.98	0.98	596
weighted avg	0.98	0.98	0.98	596

50:50 split:

	precision	recall	f1-score	support
0	0.98	0.94	0.96	171
1	0.98	0.98	0.98	130
2	0.97	0.97	0.97	142
3	1.00	1.00	1.00	164
4	0.95	0.99	0.97	138
accuracy			0.98	745
macro avg	0.98	0.98	0.98	745
weighted avg	0.98	0.98	0.98	745

Since we are getting similar results on all splits, we choose 50:50 split since that uses minimum amount of training data and predicts for the maximum number of data samples.

We try 50:50 split with both stemming and lemmatization:

Stemming:

	precision	recall	f1-score	support
0	0.98	0.94	0.96	171
1	0.98	0.98	0.98	130
2	0.97	0.97	0.97	142
3	1.00	1.00	1.00	164
4	0.95	0.99	0.97	138
accuracy			0.98	745
macro avg	0.98	0.98	0.98	745
weighted avg	0.98	0.98	0.98	745

Lemmatization:

	precision	recall	f1-score	support
0	0.98	0.94	0.96	171
1	0.98	0.98	0.98	130
2	0.97	0.97	0.97	142
3	1.00	1.00	1.00	164
4	0.95	0.99	0.97	138
accuracy			0.98	745
macro avg	0.98	0.98	0.98	745
weighted avg	0.98	0.98	0.98	745

We obtain similar results for both case. Note that we get a very high accuracy, precision, recall and F1-score on both.

Q3

- The dataset contained a large number of query-URL pairings together with their relevance score. First, we only extract queries with the qid:4 value. We receive 103 of these questions. The first integer in each question line, which represents their relevance judgement scores, is used as their individual relevance scores. We now have the queries with qid:4 and will utilise them for the next sections.

- We are aware that we must arrange the query-URL pairs in descending order of relevance scores in order to create a file with query-URL pairings organised in the order of maximum DCG. In order to create an arrangement that corresponds to the maximum DCG arrangement, we repeat the process to arrange the query-URL pairs in decreasing order of their relevance scores.
- The query-URL pairs are sorted in decreasing order of the value of feature 75 since it is assumed that the greater the value, the more relevant the URL. We extract the value of feature 75 and then rank the query-URL pairings on the basis of that feature. We obtain the necessary rating by sorting the data in decreasing order depending on the value of feature 75. Moreover, it is assumed that query-url combinations with relevance judgement scores greater than zero are relevant. As a result, by employing this, we can determine how many relevant pairings there are overall.