IR ASSIGNMENT 2 REPORT

Ques.1

a) TFIDF:
Import all the necessary libraries for the preprocessing process.
We have the raw files with us. We will be doing the preprocessing process on these files and get the filtered data.
Then we will make a list of all the words present in the file and then add this list to another empty list.
Then we take the input from the user and then do the preprocessing process on the input.
Then many functions are made to calculate the Term Frequency and Inverse Document Frequency.
We find the TF for weighting schemes such as Binary, raw count, Term Frequency, Log normalization, Double normalization. Then we made separate functions for each weighting scheme to calculate the TF-IDF values.
After getting all the TF-IDF values for all the weighting schemes we then calculate the TF-IDF score between the contents of the file and the input given by the user and store it in a dictionary.
We then sort the dictionary in decreasing order of the TF-IDF score and display the Top 5 relevant files for all the weighting schemes.

b) Jaccard Coefficient:
We have the preprocessed data which we created from the previous parts.
Then we took an input from the user and did the tokenization process on it and we also did the tokenization process on the content of the file in order to calculate the Jaccard coefficient.
Jaccard coefficient is Intersection/Union. So we wrote the code for Intersection and Union in different functions.
Then we calculated the Jaccard coefficient between the query and all the contents of the file and stored the values and file name in the dictionary.
Then we sorted the dictionary in decreasing value of the Jaccard coefficient and then at last print the Top 10 relevant files.

Ques 2.
Pre-processing of data

1. Stripped the data for each files
2. Converted the text to lower case
3. Tokenized the data and generated tokens
4. Converting the number into word like 12 -> twelve
5. Removed punctuation marks
6. Removed single alphabet terms
7. Removed stop words
8. Applied lemmatization

Ques. 2

Words are commonly used words such as "the," "a," "an," "in," etc., that are usually filtered out from text data because they do not carry much meaning. This function uses the nltk library to obtain a set of English stop words, tokenizes the input string s into individual words, and removes the stop words from it.

The remove_punc function takes a list of words s as input and removes the punctuation marks from each word in the list. This function uses the string library's translate method to remove all punctuation marks from the words.

The remove_space function takes a list of words s as input and removes any leading or trailing white spaces from each word in the list. This function uses the join method to remove spaces between words and the split method to split the words into a list.

The lowercase function takes a string s as input and converts all the characters in it to lowercase. This function first applies the remove_stop function to remove stop words, then converts the remaining text to lowercase.
The extract function takes a file path as input and reads the contents of the file. It then splits the contents into individual words and appends them to a list l2. Finally, it appends this list to another list l1, which contains the list of words in each file. This function is intended to be used to read all the files in a directory and extract the words from them.
The "process_file" function takes three arguments: "file_path," which is a string representing the path of a file, "files_list," which is a list of strings containing the names of files to be processed, and "folder_path," which is a string representing the path of a folder. The function tries to split the "file_path" into the name and the tail of the file using the "os.path.split" method. Then, it initializes an empty list called "path_list" and two counters, "i" and "j," to iterate over the files list and the list of files in the folder. The function loops through each file in the folder and checks whether its tail matches the "tail_of_file" from the input "file_path." If there is a match, the function appends the path of the file to "path_list." Finally, the function returns "path_list" containing the paths of files with matching names.
The "paths_creation" function takes one argument, "data_path," which is a string representing the path of a directory containing the files to be processed. The function calls three helper functions: "get_files_list," "get_folder_path," and "get_file_paths," to get a list of files to be processed, the folder path, and a list of paths to each file in the directory, respectively. Then, the function initializes an empty list called "path_list" and another list called "LA," which will contain the file names. The function loops through each file in the directory using the "tqdm" module to display a progress bar. For each file, the function calls the "process_file" function to get the paths of all files with matching names and appends them to "path_list." Additionally, the function appends the file name (obtained using the "os.path.split" method) to the "LA" list. Finally, the function returns a tuple containing "path_list" and "LA."

This code defines several functions that are used to perform text analysis on a set of documents from different classes. Here is a brief explanation of each function:

1. count_unique_words(word_dict): This function counts the number of unique words in each class and returns a set of all unique words and the number of unique words.
2. find_class_frequency(word_dict): This function counts the frequency of each class for every word in the corpus and returns a dictionary that maps each word to its frequency.
3. inverse_class_frequency(w, cf_values): This function calculates the inverse class frequency of a word w based on the number of classes it appears in, as specified by the dictionary cf_values.
4. tf_icf(word_dict, cf_values): This function calculates the TF-ICF (term frequency - inverse class frequency) score for each word in each class, based on the word frequency dictionary word_dict and the inverse class frequency values calculated by the inverse_class_frequency function. It returns a dictionary that maps each class to a dictionary of words and their corresponding TF-ICF scores.

The code provided contains two functions, find_top_k_features and count_frequencies, both of which are used in the process of feature selection and frequency counting for text classification tasks.
The find_top_k_features function takes in a dictionary of TF-ICF values (term frequency–inverse document frequency) for each word in each class, and an integer k which specifies the number of top features to be selected for each class. The function returns a dictionary k_features_dict that contains the top k features for each class, and a list final_features_list that contains all the selected features across all classes.
The count_frequencies function takes in a list of classes, a dictionary tf_icf_features that contains the TF-ICF values for each word in each class, and a dictionary k_features that contains the top k features for each class. The function returns two dictionaries class_one and class_two. class_one contains the frequency of each top feature for each class, while class_two contains the total frequency of all top features for each class.
To count the frequencies of each top feature, the function loops through each class in the class_list, and for each class, it counts the frequency of each top feature in the k_features dictionary using the Counter method, and stores it in the class_one dictionary. It also adds the frequency of each top feature to the total frequency of all top features for the current class in the class_two dictionary.
The nbAlgoFunction function performs the Naive Bayes algorithm on the test data. It takes in several inputs including the number of distinct words in the data (disctintWordsC), the proportion of data in each class (classTrainSplit), training data (trainData), test data (testData), all possible classes (allClasses), the frequency of each word in each class (classOne), and the total frequency of each class (classTwo). The function iterates through each test instance and calculates the probability of the instance belonging to each class using the Naive Bayes formula. It then predicts the class with the highest probability for the instance and stores both the predicted and true values in separate lists.
The accuracy_evaluation function takes in the predicted and true values and calculates the accuracy of the classification by dividing the number of correct predictions by the total number of predictions.
The evaluate_confusion_matrix function takes in the predicted and true values and the list of all possible classes and generates a confusion matrix that shows the number of instances classified correctly and incorrectly for each class. The confusion matrix is stored as a 2D numpy array.

Ques 3.

The code imports the Pandas, Matplotlib, Numpy, and Math libraries using the import statement. These libraries provide functions for data manipulation, graphing, numerical computing, and mathematical operations, respectively.

1. The code reads in a text file named 'IR-assignment-2-data.txt' using the pd.read_csv function from the Pandas library. It sets the delimiter to a space and sets the header to

None, indicating that the data does not contain a header row. The resulting data is stored in a Pandas DataFrame called data.

2. The code creates an empty dictionary called database_dict and populates it with the values from the 'qid:4' column of data. The keys of the dictionary are the row indices where the 'qid:4' value appears in the '0' column of data.
3. The code creates a new DataFrame called temp that only contains the rows in data where the key is in database_dict. It then saves this new DataFrame to a text file called 'query4max.txt' using the np.savetxt function from the Numpy library.
4. The code creates a list called unsortedDb containing the items from database_dict in the form of (key, value) tuples. It then sorts database_dict in descending order based on the value of each item using the sorted function with a lambda function as the key argument. The sorted database_dict is then reassigned to the original variable name.
5. The code defines two helper functions, check and calc, that are used in the findTotalFiles function. check takes a dictionary and counts the number of occurrences of the values 0, 1, 2, and 3. calc takes a list of counts and returns the factorial of each count multiplied together. findTotalFiles calls check and calc to calculate the total number of possible files that could be retrieved from the database_dict.
6. The code defines a function called run_findDCG that calculates the Discounted Cumulative Gain (DCG) for a given set of data and length. It uses a formula that calculates the sum of the relevance scores for each item, with a logarithmic discount factor applied to the position of each item in the list. The findDCG function calls run_findDCG to calculate the DCG for database_dict and unsortedDb, and then calculates the normalized DCG (nDCG) for each of these sets of data.
7. The code initializes some variables and defines two functions, plot_me and ff, that are used in the getPrecisionValueAndRecallValue function. plot_me takes two lists and plots them on an x-y graph using the plt.plot function from the Matplotlib library. ff takes a sorted list of (key, value) pairs and calculates the precision and recall values for each item. It then appends these values to the precision val and recall val lists, respectively.
8. The getPrecisionValueAndRecallValue function takes a dictionary of (key, value) pairs as input and sorts the pairs by value in descending order. It then calls ff to calculate the precision and recall values for each item and plots these values using plot_me.