Coursework 1: Sentiment analysis from tweets

Objective: In this coursework, you will implement a Support Vector Machine classifier (or SVM) that classifies tweets according to their sentiment, i.e. positive or negative. You will derive features from the text of the tweets to build and train the classifier on part of the dataset. You will then test the accuracy of your classifier on an unseen portion of the dataset. Much of the background for this part is in Lecture 2 and Lab 2 on Text Classification, though you should use all of your knowledge across the module.

Question 1

Simple data input and pre-processing:

In the initial phase of sentiment analysis for tweets, this task involves simple data input and pre-processing. The objective here is to extract relevant information from the dataset and prepare the raw text for further analysis

The parse_data_line function takes a line of tab-separated data as input and returns a tuple containing the label(either positive or negative) and the statement(text). The label and text are extracted from the second and third elements of the data_line list.

The pre_process function is used for cleaning and structuring the raw text data. It performs the follow crucial steps:

1. Separates punctuation at the beginning and ends of strings to ensure proper tokenization
2. Tokenizes the text into individual words using regular expressions to split on whitespace
3. Normalizes the tokens by converting them to lowercase.

These preprocessing steps are designed to enhance the quality of the text data and to make it more manageable to feature extraction and analysis

Question 2

Simple Feature Extraction: We focus on feature extraction from a text using a binary representation and training a sentiment classifier using a linear support vector machine

1. Initialized a global dictionary(global_feature_dict) to keep track of all the features in the entire dataset
2. Created a feature vector function which takes a list of tokens as input. It counts the occurrences of each token using ‘Counter’ and updates the global feature dictionary with the current token counts.

   Then it creates a binary feature vector where each token is represented by 1 if present and 0 if absent
3. Defined train_classifier to train a sentiment classifier. The classifier is created using a linear support vector machine(LinearSVC) within a pipeline
4. The SklearnClassifier is then used to train the classifier on the provided data

Question 3

Cross-validation on training data: The objective here is to do cross validation to assess the classifier's performance across different subsets of the dataset, ensuring reliable evaluation and minimizing the risk of overfitting

The cross_validate function implements k-fold cross validation(k=10) on the training data, thus evaluating the model's performance on each fold and then aggregating the results

1. The predict_labels function predicts labels for a set of samples using the trained classifier and the predict_label_from_raw function predicts the label for raw text by first preprocessing and then using the classifier
2. The main section of the code initializes global lists, loads and preprocesses the data, splits it into training and test sets and performs cross validation
3. We use the performance metrics for this including precision, recall, f1-score and accuracy which are printed for each fold and averaged over all fold

Execution Details:

1. The code initiates cross validation with ten folds. Each fold covers a subset of the dataset, with approximately 2684 items in each fold
2. For each fold, the classifier is trained on the corresponding training subset
3. After processing all folds, it prints the cross validation results, including precision, recall, f1-score and accuracy
4. The obtained average scores are as follows
   Precision: 0.8517370942024269
   Recall: 0.8529904974846281
   f1-score: 0.8521313658960432
   Accuracy: 0.8529904974846281

Question 4

Error analysis: The objective is to assess the performance of a text classifier through quantitative analysis including the generation of a confusion matrix and identification of False Positives and False Negatives as this can offer valuable insights into specific areas for improvement and further refinement of the classifier

1. The confusion_matrix_heatmap function visualizes the classifier's performance by generating a visually informative confusion matrix heatmap. This heatmap is based on the predicted labels(preds) and the true labels(y_test). We use matplotlib library for creating this representation
2. The classification_report function generates a comprehensive classification report. This report provides precision, recall, f1-score and accuracy metrics for each class(positive and negative)
3. For each misclassified instance, we extract relevant information including the original text, true label and predicted label and then we store it as separate lists(false_positives and false_negatives) for later analysis
4. We use the 'with open' statement to write the identified False Positives and False Negatives to text files(false_positives.txt and false_negatives.txt) as printing the errors to files rather than within the notebook is preferred to avoid massive notebooks which can cause you memory issues.
5. To provide a visual representation of classifier performance on one fold of the data, the code selects a subset of the test data(test_data_first_fold). It then predicts labels for this subset and generates a confusion matrix heatmap using the confusion_matrix_heatmap function

Question 5

Optimising pre-processing and feature extraction:

Pre-processing:

Enhanced pre-processing by incorporating additional steps such as

1. Separation of punctuation at both ends of the strings
2. Tokenization using nltk and stopword removal
3. Lowecasing and Lemmatization

Feature Extraction:

Improved feature extraction with additional options such as

1. Binary or term frequency weights with additional options
2. Stylistic feature: words per sentence
3. Included bigrams

Cross validation

Improved cross validation with dynamic fold sizes and multiple iterations

1. Iterates through the dataset with dynamic fold sizes
2. Repeats the process for improved statistical reliability
3. Also, utilizes train_test_split for train-test separation

4. After processing all folds, it prints the cross validation results, including precision, recall, f1-score and accuracy
5. The obtained average scores are as follows
   Precision: 0.8602440361641175
   Recall: 0.8617477175330726
   f1-score: 0.8601676040026278
6. Accuracy: 0.8617477175330726

Confusion Matrix and Analysis:

1. Confusion matrix heatmap provides a visual representation of predictions
2. Classiification report offers precision, recall, f1-score and support metrics
3. Identification of storage of false positives and false negatives

Optimizations made and Potential Impact on Accuracy:

1. The additional steps in pre-processing(punctuation separation, lemmatization, stopword removal) contribute to a cleaner and more normalized text representation
2. Improved tokenization may help capture more meaningful features potentially enhancing the model's ability to distinguish sentiment
3. Inclusion of bigrams captures more complex relationships between words
4. Stylistic features like words per sentence introduce additional info
5. The flexibility to choose binary or term frequency weights allows better feature representation
6. Dynamic fold sizes and multiple iterations enhance the reliability of cross validation results. Improved statistical robustness may lead to a better estimation of model performance
7. Detailed metrics in the classification report offer a more refined understanding of the model's strengths and weaknesses

In summary, the optimizations focus on refining both the preprocessing and feature extraction stages along with enhancing the evaluation process. These improvements contribute to a more comprehensive analysis of the model's performance potentially leading to a higher accuracy by addressing limitations observed in the earlier version.