Report

Question 1

Method

Input:
The *parse_data_line()* function takes a list corresponding to a data line as input and returns a tuple <(label, statement)>.

Preprocessing:
In the *pre_process()* function, we import the regular expression library *'re'*. The *re.sub()* method is used to separate words from punctuations occuring at the end and beginning respectively. The *re.split()* method is used to split the text into a list of tokens. We finally convert all tokens to lowercase and return the list of tokens.

Question 2

Method

The *to_feature_vector()* function is used to implement simple feature extraction. We create a dictionary '*feature_vector*' and use a for loop to parse through every unique word in the tokens list. The dictionary then maps each word to a weight value of '1'. We also update the '*global_feature_dict*' everytime the function is called. We finally return the '*feature_vector*' dictionary.

Question 3

Method

We implement cross validation on training data by dividing the entire training set into 10 folds. In the *cross_validate()* function, we iterate through each fold and use the current fold for validation purpose while training the classifier on the remaining 9 folds. We then predict the labels for the validation set and generate a classification report comparing the true labels and the predicted labels in terms of various performance metrics such as precision, recall, f1 for each class and the accuracy scores. We store the scores generated in each iteration to the corresponding lists and finally calculate the average across all folds for each class and return them in a dictionary.

Question 4

Method

We split the training set into 10 folds and use the first fold for our error analysis while training the classifier on the other folds. We plot a confusion matrix heatmap of the true labels against the predicted labels. From the plot we can infer that the 'positive' class has a higher advantage as the classifier is more accurately able to correctly classify the positive samples (True Positive Rate=91.5%) and less accurately correctly classify negative samples (True Negative Rate=72.1%)

Error Analysis of False Positives: (Prec=0.882)

Here are few instances where the classifier might have got confused:

i. 'With Zayn joining ISIS it is hard to maintain their family friendly image'. Since unigram tokens do not take context into account, the word 'friendly' must have confused the classifier in misclassifying this negative statement as positive. Also, it doesn't know what 'ISIS' stands for!
ii. 'I am out of my favorite Tea' is another example where 'favorite' misleads the classifier.

Error Analysis of False Negatives: (Recall=0.915)

Here are few instances where the classifier might have got confused:

i. 'Any evil twins you may have won't be able to break Windows 10's facial recognition security. So that's reassuring'. The word 'evil' and 'break' might have confused the classifier in predicting this as negative
ii. 'A Monday without teen wolf is a Monday with less stress and anxiety'. 'stress' and 'anxiety' are not taken in context along with the word 'less', hence the misclassification.

Question 5

Modifications

1. **Feature Extraction**
   We use a dictionary of frequency distribution of the tokens instead of just simply assigning a weight value of '1' to every token.

Initial Accuracy: **0.8483**
Improved Accuracy: **0.8506**

(All reported accuracy scores in this and subsequent steps are the average across all folds)

2. It is observed that preprocessing techniques such as lemmatizing, stop word removal, using n-gram tokens reduce the overall accuracy of the model. Hence, these are not included in the code.

3. **Hyperparameter Tuning**
   We tune the hyperparameters of the LinearSVC classifier by setting *C=0.05* & *max_iter=100*

Initial Accuracy: **0.8506** (after the first modification)
Improved Accuracy: **0.8658**