EECS763P Assignment 1 Report

Name: Bheki Maenetja
Student No.: 230382466

Please note:

- The code for questions 1 to 4 is found in the *.ipynb* file named 230382466-Bheki-Maenetja-CW1. In this report this notebook will be referred to as the "CW1" notebook.
- The code for questions 5 is found in the *.ipynb* file named 230382466-Bheki-Maenetja-CW1-Q5. In this report this notebook will be referred to as the "Q5" notebook.

Question 1

The "pre_process" function implemented in question 1 (see code block 4 in the CW1 notebook) uses [SPACE] characters to separate the elements of the text into a list. The list of elements is then returned along with a "</s>" character at the end to signify the end of the sequence.

Question 2

The feature weighting used in question 2 is a simple count weighting. For each token in a given sequence of tokens, the token's weighting is simply the number of times that it occurs in the sequence; this representation is commonly known as the bag-of-words (BoW) representation. The function to_feature_vector (see code block 5 in the CW1 notebook) makes use of the built-in "count" method for python lists as well as a dictionary comprehension to return the feature representation.

Question 3

The function "cross_validate" performs k-fold cross-validation on a given dataset. For each of the n number of folds (n being the folds parameter) it builds a training set by putting together all samples in the dataset except those in the current fold; the current fold (specified by dataset[i:i+fold_size]) is used as the test set. The true values are then extracted from the test set and stored in the variable "true_vals". Model training and inference is then done, using the "train_classifier" and "predict_labels" functions; this creates a classifier, trains it, and then generates predictions from it. Predictions are then compared with the true values using the "precision_recall_fscore_support" function from scikit-learn; this calculates precision, recall and fscore metrics. The metrics are calculated for each label and their average is weighted by the number of "positive" values for each label; this guards against an imbalance of the distribution of labels in the dataset. Accuracy (i.e. the ratio of correct predictions observations to the total observations) is also calculated for each fold. The results from each fold are aggregated in the variable "cv_results". The variable is then divided by the given number of folds to yield the average for each metric across all folds.

Question 4

The error analysis performed for this question (see code blocks 12 and 13) was performed on a 1-tenth fold (the 1st fold used for cross validation) of the training set. The analysis yields some interesting insights. From the confusion matrix we see that majority (2217) of predictions are either true positives or true negatives; indicating that the classifier is, at the very least, more accurate than the flip of coin. The rest of the predictions are a fairly even distribution of false positives (255) and false negatives (212).

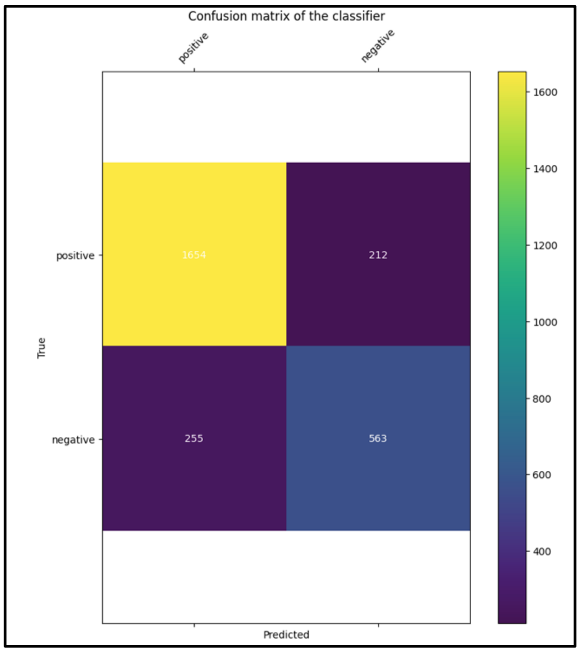
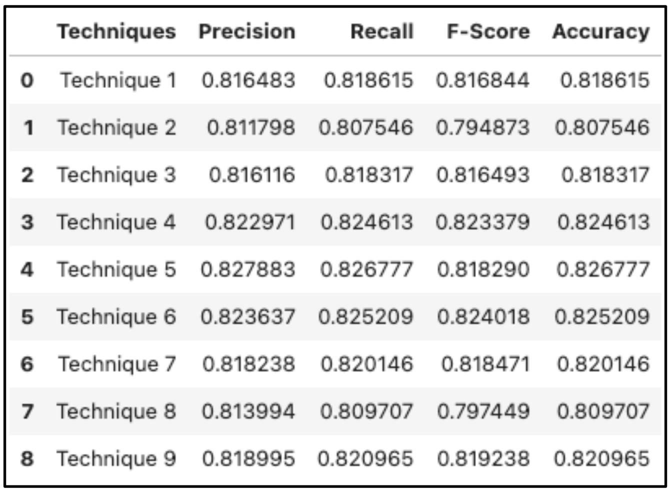


Figure 1 Confusion matrix for 1st fold of training set used for cross validation.

If we look deeper into the nature of the errors (see error-analysis.txt), we observe that a common cause of the errors appears to stem from the classifier focusing too much on “negative sounding” words and not on the context in which they are used. For example, item #39 in error-analysis.txt is a false positive error that seems to be caused by an abundance of positive words like “good” and “favourites”. The classifier has ignored the context in which they are being used (why the person keeps drinking unhealthy beverages) and has instead judged the statement only on the connotations of the words it contains. We see the reverse of this for false negative errors such as the one seen in item #646.

Question 5

The implement for question (see the Q5 notebook) looks at how various pre-processing techniques can be combined to build an optimal classifier. There are two processing tasks that are examined specifically: tokenisation and feature vectorisation. The notebook looks at 3 tokenisation techniques and 3 feature vectorisation techniques. All combinations (a total of 9) of these techniques were explored, with a classifier being cross validated for each combination. Whilst there are only marginal differences between each technique combination, the results (see table 1 or code block 38 in the Q5 notebook) show that the best approach for tokenisation appears to be removing all punctuation and stopwords and taking the stem of the remaining tokens. The best approach for feature vectorisation – when using the optimal tokenisation technique is a toss-up between the term frequency, normalised term frequency (dividing term frequency by the number of tokens) and the log of term frequency (i.e. taking the log of term frequency) approach. The results for all these combinations (techniques 4, 5 6 in the table) are very close, with technique 5 having the slight edge for average precision, recall and accuracy.



	Techniques	Precision	Recall	F-Score	Accuracy
0	Technique 1	0.816483	0.818615	0.816844	0.818615
1	Technique 2	0.811798	0.807546	0.794873	0.807546
2	Technique 3	0.816116	0.818317	0.816493	0.818317
3	Technique 4	0.822971	0.824613	0.823379	0.824613
4	Technique 5	0.827883	0.826777	0.818290	0.826777
5	Technique 6	0.823637	0.825209	0.824018	0.825209
6	Technique 7	0.818238	0.820146	0.818471	0.820146
7	Technique 8	0.813994	0.809707	0.797449	0.809707
8	Technique 9	0.818995	0.820965	0.819238	0.820965

Table 1 Cross validation performance of various tokenisation and feature vectorisation techniques.