Sentiment Analysis

Implementation

A Naïve Bayesian sentiment analysis model program has been created for analysing movie reviews. The model is implemented across four files accessed from a main file called **NB_sentiment_analyser.py**. The four files are:

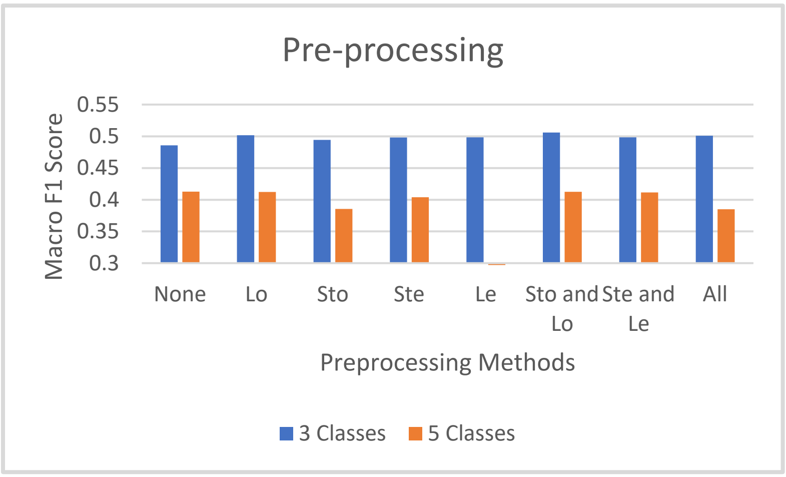
preprocessSentences.py, which pre-processes the sentences by removing punctuation, stop words, and performing lemmatization and stemming. The pre-processing step also rescales the data depending on the number of classes needed, it scales down a five-class sentiment system to a three-class sentiment system.

featureExtraction.py, which extracts features by calculating a sentiment score for each feature token and sorting the scores in a list. A certain amount of the highest scoring features is taken from the list based on a featurePercentage. The scale for the sentiments in the **train**, **dev**, and **test** tsv data files uses a [0,1,2,3,4] system whilst the predicted sentiments that we want can be negative and positive values. This means that the sentiment score most be calculated on a scale of [-2,-1,0,1,2] for the five classes or [-1,0,1] for the three classes so the end predictions are correct. Further feature extraction is done using a count vectorizer function.

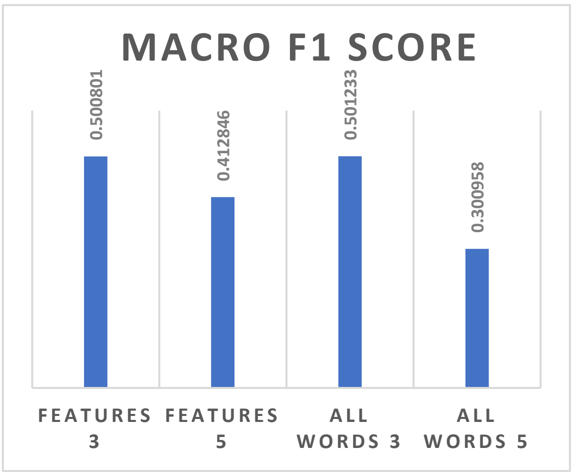
naïveBayes.py, which trains the corpus using naïve bayes modelling on a labelled dataset.

evaluate.py, which evaluates the predictions using **Macro F1 scores** and plots a confusion matrix.

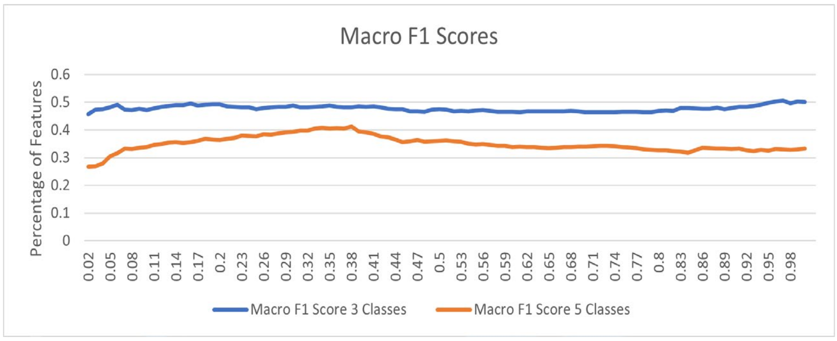
Evaluation of Results



This graph shows the macro f1 scores using the best features selection but using different pre-processing steps such as **Lowercase**(lo), **Stoplist**(Sto) stop-word removal, **stemming**(Ste) and **lemmatisation**(Le). All data has gone through **tokenisation** so it can be used in other classes. I have used various combinations of these pre-processing steps and found that pre-processing always improves the results of the three-class model but always decreases the results of the five-class model. The three-scale results are improved the most by the lowercase processing and the least by the Stoplist on their own, whilst the best results are achieved with only Stoplist, and lowercase are used. Lemmatisation and stemming do not increase the result further when added onto these processes. The five-class sentiment scale achieves the best results with no pre-processing and the worst results when all pre-processing steps are used. The pre-processing likely negatively effects the five classes negatively because more classes make it more sensitive to changes in the data due to having more class boundaries, this means that if any words that are important for expressing sentiment are removed via pre-processing it could have a large negative effect on the result. The three-class scale however has less class boundaries and is therefore less sensitive to changes in the data, therefore, it follows the expected effect of having improved results when pre-processing steps are applied. Also, the graph shows that the three classes were affected less by the pre-processing than the five classes.



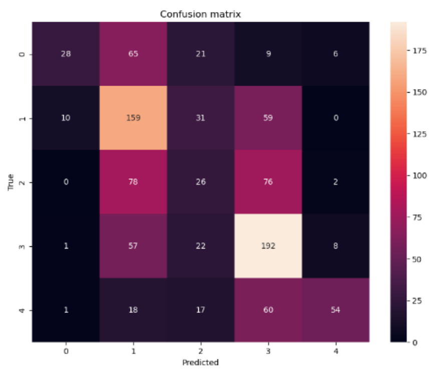
This graph shows the macro f1 scores for the sentiment analysis when using the feature extraction or using all the words as the features for both of the classes. As the graph shows, feature extraction always performs better than using all the features. However, the five classes improve much more with feature extractions vs the three classes. This is because the three classes were found to perform its best when using 97% of the features as shows in the paragraph which is not much different from using all the features whilst the five classes drastically improve with feature extraction as its best performing feature amount was 38%, this is very different from using all words as features. These results are explained further in the next paragraph.

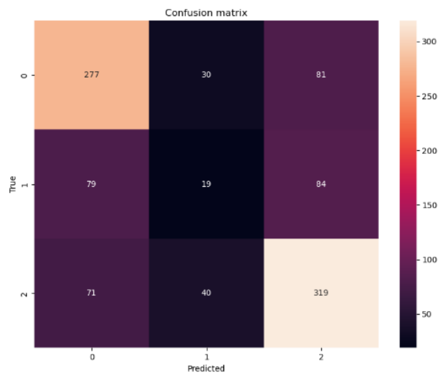


Using the optimal pre-processing processes for each set of classes as well as naïve bayes classifier I performed feature extraction on the feature tokens. I did this by calculating their sentiment bias, ordering them, and then taking a percentage of the best features. I experimented with changing the percentage of the features I used for each class and found that the optimal results for the three classes is 97% of the feature tokens whilst the five classes produced best results when 38% of the feature tokens where used. I calculated this by iterating by 1% each time from 1% of the features to 100% recording my findings. I found that the three-sentiment class also had a local maximum result when the feature percentage was around 35% as well showing that this number of features is still good in both number of classes, however, the three classes performed better when almost all features were used which was the global maximum for these results. The highest Macro F1 score for the three classes using this feature extraction is 0.506002 whilst the best for the five classes is 0.412846.

These scores of **0.50600** and **0.412846** are not as high as would be needed for a sentiment analysis software which is likely due to the way naïve Bayesian classifiers treat features as **independent** which is not realistic to real world scenarios such as the movie reviews. This leads to poor performance as it cannot capture the relationships of the words. Also, naïve bayes is also sensitive to redundant features in the dataset, our dataset is small and therefore, not trained to a high degree leaving many redundant features inside after training and leading to lower results. **Laplace Smoothing** has been used to increase the results of the sentiment analysis training as it solves some of the limitations of the algorithm. It avoids zero probabilities and improves the accuracy of the model and reduces the effects of bias. Also, by adding a small constant value to the counts of each word, Laplace smoothing smooths out outliers in the data improving overall performance.

The confusion matrices below show my best performing results, the confusion matrix visualises the performance of the sentiment analysis in terms of true positive, true negative, false positive, and false negative predictions. This helps us understand how well the trained model is performing and predicting the sentiments. This allows us to search for areas of improvement. For example, when the confusion matrix showed many false-negatives I tuned the classification thresholds to improve the results. The left shows the five classes matrix, and the right shows the three classes matrix.





A count vectorizer function was created to attempt to improve the performance of the data but it didn't end up improving the result and was also slow compared to my other feature extraction method. This is likely due to the similarities between the count vectorizer and the percentage of features extraction with the sentiment biases producing similar results and thus not affecting the outcome. Both count the occurrence of tokens and calculate the features based off this.

Another feature extraction method that yielded no improvement was the use of NLTK POS tags which has been removed from the code, I believe this did not increase the performance as there are many different part-of-speech tags and no way to evaluate the tag counts for this dataset in a meaningful way. This would require extensive evaluation of the data and the POS tags to create improvements in the results.

Finally, I attempted to implement a token sentiment weight list which is a list of words with sentiment weights which I would multiply the sentiment biases of my tokens by to hopefully produce better features on its own it produced decent results improving the performance of the five classes result by 0.3 but was overshadowed by my other feature extraction methods. This likely didn't produce good results as a much larger list produced by a large set of users is needed to take advantage of 'wisdom of the crowd' and produce a useful dataset. Overall, the Naïve Bayes sentiment analyser performs better than majority class baseline by roughly 0.3 for the three classes and over 0.32 for the five classes.