NLP – Assignment 1 Report

Q1: I implemented the parse_data_line simply to extract the label and statement of the input csv from each line that is read. In this process, I converted the labels to a binary label of “FAKE” or “REAL”.

I start off with a simple pre_process function that splits on word boundaries (anything that isn’t a letter, digit, or underscore). I have ignored punctuation and will later see how this affects the model's performance. I have made sure to remove all whitespace from the token list as well.

Q2: My to_feature_vector function simply implements a BOW model of the pre-processed tokens with no other features. We will see the consequences of this later. This is a good baseline to measure my final model’s performance against.

Q3: Performing 10-fold cross-validation on our simple BOW model, we see that our performance is lacking. The accuracy of the model is not that high at only around 56%. This value is clearly skewed by the fact that we have a lot more training instances of “REAL”. This means that the model has more training data to work with and as such, it should perform better, in this case, it does. Looking at our f1-score for “FAKE” and “REAL”, we see this is clearly evidenced, with around 49% and 61%, respectively. For my analysis, I will use all the performance metrics but comment mostly on the macro avg and accuracy. I have decided on this because accuracy is an important metric, but it does not give the full picture. I chose macro avg because it gives equal weighting to all classes whereas weighted avg is skewed by the class with the higher value for that metric. I think this gives a more complete picture. The model for this has the following performance metrics. This will be my baseline,

| Macro Precision | Macro Recall | Macro f1-score | Accuracy |
|---|---|---|---|
| 0.552722054948136 | 0.5527250383890114 | 0.5524228100458656 | 0.5604523609275501 |

Q4: Taking the “FAKE” class as positive, we see that there are much fewer True Positives than True Negatives. As such, the False Positives and False Negatives skews the Precision and Recall for the positive class, “FAKE” a lot more. I have created 4 new files for this question, each beginning with “Q4”. They contain all instances of TP, FP, TN, and FN on the first fold of the CV. The Rows are the Inputs, and the Columns are the Features, which are sorted by frequency of occurrence. We can see that the most frequent word features are ambiguous words and function words. These appear in all sorts of contexts and are common in language. So, I will try to look beyond these and at the more unique features amongst them. Looking at “Q4_False_Positives.csv” and “Q4_True_Positives.csv”, some common features are, “Obama”, “Barack”, “health”, “voted”. Having looked at the TNs as well, I see that most of these words are more common in the TP class and as such are good discriminators for the TP class. So, as a lot of FPs contain these features, they are mistakenly identified as the True class. Looking at “Q4_False_Negatives.csv” and “Q4_True_Negatives.csv”, some common features are, “000”, “million”, “jobs”, “tax”, “Obama”. And as before, most of these are more common in the Negative class. So, we get FPs and FNs due to the discriminator features of a class occurring in instances of the other class. One feasible way to deal with this is to discard these features. Although this way would be good, it would not be the best as some of these features will be the best discriminators for that class and as such, getting rid of these will result in a worse performance. The best way to deal with this is with feature selection.

Q5: The methods I tried were Case folding, Leaving in Punctuation, Stop Word Removal (online lexicon under “stop_words.csv”), Famous People Lexicon (online lexicon under “famous_people.csv”), using varying degrees of nGrams, Negation Words Lexicon (lexicon under “negation.csv”), Average Word

Length, Types Per Tokens, Sentiment (lexicon under "sentiment_lexicon.csv"), Mutual Information, SVC Hyperparameter tuning. I did my analysis on each of these individually (from the base model) as looking at them together would have taken too long computationally. The results of this analysis are available at "Q5_Hyperparameter_Analysis.csv".

The greatest increase in performance (accuracy and macro avg f1-score) was in n-grams (specifically 5-gram). The higher the n-gram, the better the performance. So I have used a 5-gram in my final model. I did not venture further than this as using a high n-gram on small corpora can result in overfitting. I also implemented Pointwise Mutual Information (and average PMI) between the features, this resulted in disappointment as the calculation (under "Q5_average_pointwise_mutual_information.csv") can result in infinities in average PMI due to sparse features. To resolve this, I considered doing smoothing but depending on my smoothing factor, I got vastly different results and with computation time already being a problem, this would have been difficult to execute. So, I used the PMI for each class (under "Q5_pointwise_mutual_information_FAKE/REAL.csv") and each feature in that class. I used this to do feature selection but it didn't result in any noticeable performance increase (actually, macro avg f1-score decreased greatly), so this wasn't included in the end. Using case folding, Famous People Lexicon, Average Word Length, Types Per Token, and Sentiment all resulted in a performance increase. Using Stop Word Removal, negating words, and punctuation resulted in a performance decrease.

Based on the analysis above, my final model uses Case Folding, 5-grams, Average Word Length, Types Per Token, and Sentiment, with no feature reduction from mutual information. The model uses a Hinge Loss and regularization factor of 0.2 (I.e., less regularization). The Model at the end of this section has the following performance metrics:

| Macro Precision | Macro Recall | Macro f1-score | Accuracy |
|---|---|---|---|
| 0.5977743288810194 | 0.5951880124148362 | 0.5953048767241269 | 0.6072149465337018 |

When compared to the baseline, we around an 8% performance increase.

Q6: I have encoded all the extra information in order by s1-10 so there is not a conflict with other features. Performing feature analysis (under "Q6_New_Feature_Analysis.csv"), I observe that each of these features improves on the model from Q5, as such, I will use all these as features. I also computed the mutual information for these features.

When testing the performance on the test set, note that I have computed the Macro average as well. Final Model's Performance on test set:

| Macro Precision | Macro Recall | Macro f1-score | Accuracy |
|---|---|---|---|
| 0.695275 | 0.690138 | 0.691185 | 0.6974133723767691 |

This is an improvement of about 24% on the Initial model.