**Data Wrangling Assignment**

Part B

Myles Neilson

[mylesneilson@gmail.com](mailto:mylesneilson@gmail.com)

Matric Number: 98100378

The model is based on Pandas and NLTK packages.

Pre-processing

For the model I imported the data using Pandas. After this I created a function to extract the word features. As rows are separated by newlines it was fairly simple to split the data into sentences, put to lower case, then tokenize the words in the sentences using NLTK word tokenise.

I created one function with further pre-processing, this included using NLTK stop word remover and NLTK porter stemmer. Adding this pre-processing did improve the accuracy of the model.

The function then returned a simple dictionary which maps the words in the sentence to ‘True’ if the word exists in the data these are the features for the model. I also renamed the columns of the data frame to make the inputs for the model more obvious.

Model selection

I decided to use Naïve Bayes as my classifier. It is relatively simple to implement, NB can also train quickly, has no tuneable hyper parameters and is also widely used in text classification.

Applying the model

The sentiment column of the data frame contained 0 for negative sentiment and 1 for positive sentiment these were split into ?lists? of ‘neg’ and ‘pos’ sentences.

Then the words from each sentence were extracted as negative (negreviews) and positive (posreviews) features. We now have two dictionaries of negative and positive feature-label pairs.

These reviews have to be combined into one dataset to train the model. The reviews are then put into random order so that training can be more effective. The dataset was then split into training and test data. I used a 80% / 20% training/test split, this seemed to be a common split in the literature for model training. This was done by dividing the dataset at the record that applied to 80% of the data, in this case 5534.

The NaiveBayesClassifier was then applied to the training data. And this was assigned to a classifier variable.

Model evaluation

I initially used NLTK, nltk.classify.util.accuracy(classifier, test) for accuracy evaluation.

This is a simple metric to evaluate a classifier. Accuracy measures the percentage of inputs in the test set that the classifier correctly labelled.

I also investigated the metrics of Precision, Recall and F-measure. These measures make more inquiry into well the results are relevant.

Precision indicates how many of the items that were identified were relevant. Which is true positives/true positives + false positives. Recall which is how many of the relevant items that were identified. Which is true positives/true positives + false negatives. F-measure combines the precision and recall to give a single score. Which is the ‘harmonic mean’ of the precision and recall values ( 2 x Precision x Recall) / (Precision + Recall)

I have to initiate referenceSets and testSets, to be used shortly. referenceSets will contain the actual values for the testing data (which we know because the data is prelabeled) and testSets will contain the predicted output.

To do this Next, for each one of the testFeatures (the reviews that need testing), I iterate through three things: an arbitrary ‘i’, so be used as an identifier, and then the features (or words) in the review, and the actual label (‘pos’ or ‘neg’).

This gives me a big list of identifiers in referenceSets[‘pos’], which are the reviews known to be positive (and the same for the negative reviews). It also gives me a list of identifiers in testSets[‘pos’], which are the reviews predicted to be positive (and similarly for predicted negatives). What this allows me to do is to compare these lists and see how well the predictor did

The essence of those two terms is that precision is a measure of false positives — a higher precision means fewer reviews that aren’t in the desired label get labeled as being in there. A high recall means fewer reviews that are in the desired label get put in the wrong level. As you can imagine, these metrics correlate very closely. Here’s the code (again from the NLTK library) to print out the positive and negative recall and precision, as well as the accuracy (a less-specific measure just showing what percentage the classifier got right). NLTK also has a cool function that shows the features (words) that were most helpful to the classifier in determining whether a review was positive or negative. So here’s the code: