Sierra Schwellenbach
7799323

Bayesian Classifier Report

Architecture:

My program is implemented as one main class called DecisionTreeClassifier which contains a main method and a separate method which is in charge of classifying the data. This method is used to classify both the training and testing reviews and returns the accuracy. I also have a separate class called “Word” which stores the number of positive and negative documents the word occurs in as well as the calculated probability value which is needed for calculating the probability of a document being negative or positive later.

I use a hashtable to store all the words which occur in the training documents. The table uses a String (the word) as the key and the Word object (containing the number of occurrences and probability values) as the values. This Hashtable is then referenced as the program runs to retrieve the Probabiltity of the document being positive given that it contains the word.

Preprocessing:

For each input review, I use a regular expression to remove all of the punctuation (.,;:/?’”&%$!). I also take all of the line breaks (represented as “
 </br>” and replace it with “
</br> “. This allows it to be treated by the classifier as a “word” without adding “
” to the end or start of random words thereby creating one instance of, for example, “good
” which decreases the accuracy of the word “good” as an instance of it isn’t counted and adds on a word which now has only one occurrence (and is therefore less relevant). After that, I split the input string into words on the space character. From here, I go through my new list of words which occur in the current review and either add them to the hashtable of words (if the word isn’t already there) with one occurrence of either positive or negative to start with, increment either the positive or negative counter on the word (if the word is already in the table), or ignore the word (if it has already appeared in this review as kept track of by a hashset). After this is done with all of the reviews a hashtable will have been built up which contains all the words along with the number of positive and negative documents which the word occurs in.

Model Building:

I trained the Naive Bayes Classifier by calculating the number of positive and negative reviews each word occurs in. From here, it is a simple matter to calculate the percentage of positive and negative reviews each word occurs in. This is effectively the Probability that a review is positive or negative given that it contains a certain word. To calculate the most likely classification (positive or negative) of a certain document, we multiply the probability that a review is positive given that it contains a word for all of the words in the document. I then do the same thing for the probability that the document is

negative. From here, I compare these probabilities and classify the document as the greater of these two values.

Results:

The accuracy of my algorithm on the testing dataset is 82.4% for the given testing set and 91.3% for the given training set. The runtime of my algorithm is really small (less than a second with the given datasets). Important features for the positive class include: very, well, wonderful, unexpected, wit, troubled, reminiscent, showed, statement, space. Important features for the negative class include: they, just, been, so, nothing, not, movie, make, only, and stupid.

Challenges:

The main challenge I faced in this project was messing with small things and tweaking the algorithm to get good results. I was surprised by how much your results can change with just a small change in one number. The most important case of this in my program was figuring out how many occurrences of a word should be the minimum in the training set before I do not include it in the calculation of the probabilities. I solved this problem by simply testing a lot of different values and using the one which worked the best.

Another problem I ran into was making my code clean. I found that I was repeating a lot of code, but when I went to factor it out into different functions, I realized that to do so I would need to pass in a lot of parameters including the entire hashtable. This, however, is the method that I chose to use in the end as it was the “lesser of two evils”.

I also ran into issues with overfitting. I found that implementing stop words into my algorithm actually improved the accuracy of the training set but decreased the accuracy of the testing set. I ultimately decided that using stop words in this classification system would not be useful and simply removed it.

Weaknesses:

There are many parts of my algorithm which could be improved. For example, currently my algorithm only uses one word at a time for classification and doesn’t account for word pairs. If, for example, the phrase “not good” was used, my classifier will only read it as two separate words and classify it as such. I could improve this by using word pairs or even trios as part of my classification.

Also, my technique for smoothing could be improved. I did not follow the method discussed in class and made my own based on guessing and checking values. It could probably be improved by following a better method.