Machine Learning – Bayesian Classification

Assignment

Michael O' Sullivan

R00077764

DCOM4

# Naive Bayes Algorithm

I first ran the code by only using just regular expressions to clean the data. The results I achieved were:

--------- smallTest\pos\ ----------

Postive: 76%

Negitive: 24 %

--------- smallTest\neg\ ----------

Postive: 14.7 %

Negitive: 85.3 %

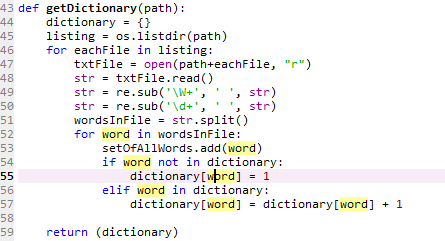
The regular expressions used to clean the data set were:

\W+ which removed special characters

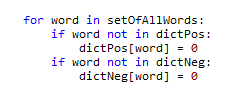
\d+ which removed any numbers

Code

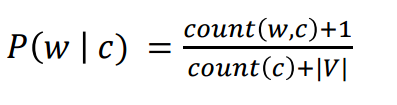
Insert the directory of the training data to a method called “getDictionary”. This method will go through all the files that is in the path passed in through the parameter. It will clean all the data of each text file using the regex stated above. The code will loop through each word in the file and add it to a set of unique words. I have a variable of type <dict> at the top of the method this will be used to store each word and their frequencies. Once it has looped through every word in every file it will return the dictionary. This method will be called twice. Once for positive review and once for negative reviews.



I now have a positive and negative dictionary, the key being the word and the value being the frequency. I ran into an issue because the length of these dictionaries were different when they should have been the same length as the number of unique words. To overcome this obstacle I looped through each word in the unique set of words and added them to the dictionaries, I set the value of each word to 0 so it wouldn’t have much of an effect on my model.



I now calculate the probability using the formula:

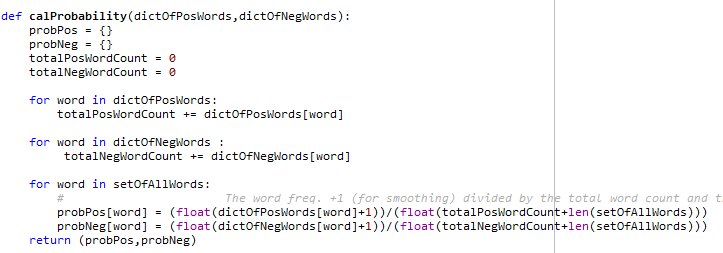


count(w, c) is the number of occurrences of the word w in all documents of class c.

count(c) The total number of words in all documents of class c (including duplicates).

|V| The number of words in the vocabulary.

I looped through all of the words in my dictionaries and calculated the total word count for each positive and negative word. I then used the formula above to get the probability and stored it in a new dictionary using the word as the key value.



I have my model now created I can now pass in some unseen data into the model and see the results. To test the accuracy I read in all the positive and negative test text files. I used the same regex as above to clean the data. Once the data was clean I looped through each word in the test text file, any word that was also in the unique word set was used because I already have the probability for these words. I accumulated these probabilities for both negative and positive. After completing the loop, If the positive accumulated value was higher than the negative then the review would be deemed positive or vice versa.



# Research

I am now going to attempt to increase the accuracy of my algorithm adding pre-processing techniques such as stopword removal and stemming.

## Stopwords

Stopwords are high-frequency words like the, to and also that we sometimes want to filter out of a document before further processing. Stopwords usually have little lexical content, and their presence in a text fails to distinguish it from other texts. [<http://www.nltk.org/book/ch02.html>].

I imported nltk.corpus and used the stopwords method to filter out these words that have no real features. When looping through each word in the files I used if statement to check if the word was not in the stopwords set. I also set all the words to lowercase, the main reason behind this was to ensure there was no duplicates of any words in my dictionaries. The results are as follows:

--------- smallTest\pos\ ----------

Postive: 78.1 %

Negitive: 21.9 %

--------- smallTest\neg\ ----------

Postive: 13.3 %

Negitive: 86.7 %

The results of not excluding any stopwords were that an increase of accuracy of 2.1% in the positive test file and 1.4% in the negative test file.

## Stemming

The use of stemming it removes morphological affixes from words. An example of this would be running -> run

Generously -> generous.

While doing the prepossessing I used the nltk.stem.snowball function to stem each word. Before I added this code I ran my program using the large database and printed out the total number of unique words which was 73417. I now ran my code using stemming. The number of words in my unique dataset was now reduced to 49083 which is roughly 33% reduction in unique words. With the reduction of word there was no change in accuracy in the positive test folder and a decrease of accuracy in the negative test files of 1.1%

--------- smallTest\pos\ ----------

Postive: 78.1 %

Negitive: 21.9 %

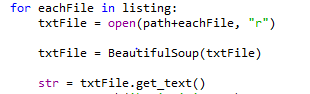
--------- smallTest\neg\ ----------

Postive: 14.4 %

Negitive: 85.6 %

## Cleaning Data Using BeautifulSoup

After looking through a few of the reviews manually in my review folders, I realized that there was a lot of html tags remaining in the reviews. I did a bit of research and found a python library called BeautifulSoup. BeautifulSoup is used for pulling data out of html websites. I am going to use this to remove any html tags remaining in the reviews. As you can see above what the current accuracy is, I now add the code to my file and ran to see if it improved the accuracy.



--------- smallTest\pos\ ----------

Postive: 78.4 %

Negitive: 21.6 %

--------- smallTest\neg\ ----------

Postive: 14.9 %

Negitive: 85.1 %

It increased the accuracy in the positive test files by .3% and lowered the accuracy of the negative test files by .5%. I also printed out the number of unique words were found while using BeautifulSoup which also increased from 49,083 to 50,017 which I found a bit odd.