COMP 8043 Machine Learning

Naive Bayes Algorithm

Darren Smith

BSc.(Hons.) in Software Development

Department of Computing

Student ID: R00117899

Lecturer: Dr. Ted Scully

Date: 27nd October 2016

Technical Description

Naïve Bayes classifiers are among the most successful know algorithms for learning to classify text documents. The implementation that I will use is Multinomial Naïve Bayes learning algorithm written in Python. The following are the steps taken to implement this algorithm.



Step 1: Get a dataset of pre-classified documents.

Step 2: Take out a sub-set from the dataset to be used to evaluate the model. From the remaining data, we can now build the model.

Step 3: Read in each remaining file that is negative/positive and repeat the following steps.

Step 3.1: Create an array of all the words in the file.



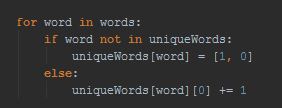
Step 3.2: Keep a count of the number of words.



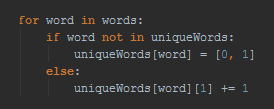


Step 3.3: Create a dictionary of unique words with a count of occurrences. 

(Negative version)

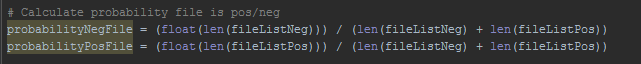


(Positive version)



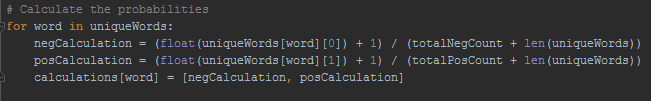
Step 5: Calculate the prior probabilities.





Step 6: Calculate the conditional probability for each unique word that it will appear in a positive and negative file with Laplace smoothing. The reason for Laplace smoothing is if there are zero occurrences in one type of file it won’t make the equation equal to zero.

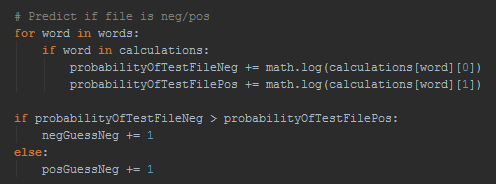




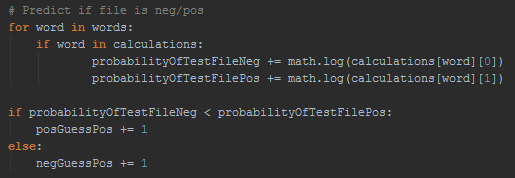
Step 6: Now that the model has been built, we can read in each test file to evaluate the accuracy of the model. To this we use the values calculated in the model with the words in the test files. To calculate the probability of the file being positive or negative we add the log of the prior probability class to the log of the probability that the word occurs for that class for each word. We for both positive and negative. Whichever one has the greater answer is my prediction. I then compare this prediction to the actual class.



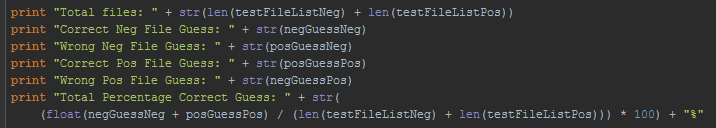
(Negative version)



(Positive version)



Step 7: Finally, I print out the results with a total percentage correct guess. I repeated this for +1/2 Smoothing and Laplace Smoothing. The results can be seen in CASE 1.



CASE 1

|  |  |  |
| --- | --- | --- |
|  | Option 1 | Option 2 |
| Read data | split by spacing | split by spacing |
| Cleaning | none | none |
| Smoothing | Laplace smoothing | +1 smoothing |
| Number of words in dictionary | 281131 | 281131 |
| Total files | 2000 | 2000 |
| Correct Neg File Guesses | 860 | 864 |
| Wrong Neg File Guesses | 140 | 136 |
| Correct Pos File Guesses | 784 | 777 |
| Wrong Pos File Guesses | 216 | 223 |
| Total Percentage of Correct Guesses | 82.2% | 82.05% |
| Continue with option | YES | NO (due to lower percentage) |
| Runtime | 11.73 seconds | 11.69 seconds |

# Conclusion

Option 1

Laplace smoothing is a type of smoothing to counteract the problem with zero occurrences of a word happing in a class. The formula is (1/number of words). The results after running were a lower correct negative file guesses but higher correct positive file guesses. It also had a higher total percentage of correct guesses. I decided to continue to use Laplace smoothing as it gave better results.

Option 2

+ 1/2 smoothing is another type of smoothing to counteract the problem with zero occurrences of a word happing in a class. The formula is (1/2). The results after running were a higher correct negative file guesses but lower correct positive file guesses. It also had a lower total percentage of correct guesses. I decided to not continue to use + 1/2 smoothing as it gave worse results.

Research

# NLTK Word Tokenize

The first step of my research I considered Pythons natural language toolkit (NLTK) (<http://www.nltk.org/>). NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to [over 50 corpora and lexical resources](http://nltk.org/nltk_data/) such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active [discussion forum](http://groups.google.com/group/nltk-users). NLTK has a function to split a string into words using a very smart algorithm.





After running my program using this function the following were my results:

|  |  |
| --- | --- |
| Read data | nltk word tokenize |
| Cleaning | none |
| Smoothing | Laplace smoothing |
| Number of words in dictionary | 135098 (-146033) |
| Total files | 2000 (0) |
| Correct Neg File Guesses | 851 (-9) |
| Wrong Neg File Guesses | 149 (+9) |
| Correct Pos File Guesses | 769 (-15) |
| Wrong Pos File Guesses | 231 (+15) |
| Total Percentage of Correct Guesses | 81.0% (-1.2) |
| Continue with option | YES |
| Runtime | 122.29 seconds (+110.56) |

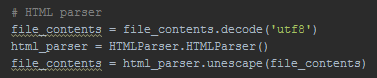
# Conclusion

Even though NLTK word tokenize decreases the total percentage of correct guesses by 1.2%, it also decreases the number of words in the dictionary by 146033 which is a 51.94% decrease. It also increases the running time by 110.56 seconds. I feel that these is are acceptable results and I will continue to use NLTK word tokenize.

# HTML Parser

Data obtained from web usually contains a lot of html entities like &lt; &gt; &amp; which gets embedded in the original data. Html parser converts &lt; to < and &amp; to &. There is an inbuilt library in Python to deal with this.





After running my program using this function the following were my results:

|  |  |
| --- | --- |
| Read data | nltk word tokenize |
| Cleaning | HTML parser |
| Smoothing | Laplace smoothing |
| Number of words in dictionary | 135095 (-3) |
| Total files | 2000 (0) |
| Correct Neg File Guess | 851 (0) |
| Wrong Neg File Guess | 149 (0) |
| Correct Pos File Guess | 769 (0) |
| Wrong Pos File Guess | 231 (0) |
| Total Percentage of Correct Guesses | 81.0% (0) |
| Continue with option | NO (due to remove punctuation) |
| Runtime | 123.44 seconds (0) |

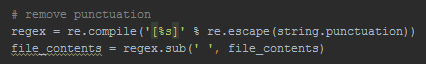
# Conclusion

Even though HTML parser does not improve the total percentage of correct guesses or the speed, it lowers the number of words in the dictionary by 3. The result is a small benefit with this dataset but is a good way to clean data for future datasets. I would continue to use HTML parser but the next type of cleaning which is removing punctuation will do this and more.

# Removing Punctuation

Punctuation marks don’t have any significate influence to whether a sentence is positive or negative so I decided to remove them. In the string library, there is a function to remove punctuations. With the help of regex, I can easily replace them with a space.





After running my program using this function the following were my results:

|  |  |
| --- | --- |
| Read data | nltk word tokenize |
| Cleaning | Punctuation |
| Smoothing | Laplace smoothing |
| Number of words in dictionary | 94427 (-40668) |
| Total files | 2000 (0) |
| Correct Neg File Guesses | 852 (+1) |
| Wrong Neg File Guesses | 148 (-1) |
| Correct Pos File Guesses | 767 (-2) |
| Wrong Pos File Guesses | 233 (+2) |
| Total Percentage of Correct Guesses | 80.95% (-0.05) |
| Continue with option | No (due to remove all but letters) |
| Runtime | 60.51 seconds (+62.93) |

# Conclusion

Removing all the punctuation decreases the number of words in the dictionary by 40668 which is a 30.10% decrease. It also decreases the total percentage of correct guesses by 0.05% but increases the runtime by over 50%. I feel that these are acceptable results and I would continue to use removing punctuation but the next type of cleaning which is only letters will do this and more.

# Letters Only

Due the purpose of this algorithm is to find out if a review is positive or negative, I decided to remove everything from the data except letters A -Z and a – z. With the help of regex, I can easily replace anything not a letter with a space.





After running my program using this function the following were my results:

|  |  |
| --- | --- |
| Read data | nltk word tokenize |
| Cleaning | Letters only |
| Smoothing | Laplace smoothing |
| Number of words in dictionary | 92335 (-2092) |
| Total files | 2000 (0) |
| Correct Neg File Guesses | 853 (+1) |
| Wrong Neg File Guesses | 147 (-1) |
| Correct Pos File Guesses | 760 (-7) |
| Wrong Pos File Guesses | 240 (+7) |
| Total Percentage of Correct Guesses | 80.65% (-0.30) |
| Continue with option | YES |
| Runtime | 59.93 seconds (0) |

# Conclusion

Removing everything but letters decreases the number of words in the dictionary by 2092 which is 2.22% decrease. It also decreases the total percentage of correct guesses by 0.30%. Even though this does not show positive results, I will continue to use this as it will help improve further cleaning.

# Lowercase Words

Changing all the words to lowercase will remove words that are duplicated e.g. good, Good and only keep the lowercase of the word.



After running my program using this function the following were my results:

|  |  |
| --- | --- |
| Read data | nltk word tokenize |
| Cleaning | Letters only + lowercase |
| Smoothing | Laplace smoothing |
| Number of words in dictionary | 73268 (-19067) |
| Total files | 2000 (0) |
| Correct Neg File Guesses | 869 (+16) |
| Wrong Neg File Guesses | 131 (-16) |
| Correct Pos File Guesses | 764 (+4) |
| Wrong Pos File Guesses | 236 (-4) |
| Total Percentage of Correct Guesses | 81.65% (+1.00) |
| Continue with option | YES |
| Runtime | 59.23 seconds (0) |

# Conclusion

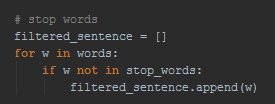
Lowercasing all the words decreases the number of words in the dictionary by 19067 which is a 20.65% decrease. It also increases the total percentage of correct guesses by 1%. These are good results and I will continue to use lowercase.

# Stop Words

Stop words play no importance to the meaning of a sentence. They are only used for humans to help structure a sentence for easier understanding and speaking. For this reason, I decided to remove all stop the stop words. NLTK has an inbuilt list of these stop words which can be used.







After running my program using this function the following were my results:

|  |  |
| --- | --- |
| Read data | nltk word tokenize |
| Cleaning | Letters only + lowercase + stop words |
| Smoothing | Laplace smoothing |
| Number of words in dictionary | 73115 (-153) |
| Total files | 2000 (0) |
| Correct Neg File Guesses | 867 (-2) |
| Wrong Neg File Guesses | 133 (+2) |
| Correct Pos File Guesses | 784 (+20) |
| Wrong Pos File Guesses | 216 (-20) |
| Total Percentage of Correct Guesses | 82.55% (+0.90) |
| Continue with option | YES |
| Runtime | 66.60 seconds (+7.37) |

# Conclusion

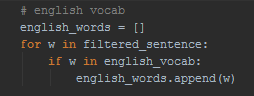
Removing stop words decreases the number of words in the dictionary by 153. It also increases the total percentage of correct guesses by 0.90%, but increased the runtime by 7.37 seconds. These are good results and I will continue to use removing stop words.

# English Words

After reviewing the data, I noticed firstly the words were English words and secondly there was a lot of misspelt words. To try solve this I decided to limit my unique words to words only in the English dictionary. NLTK has a list all the English words in the dictionary which I could use.







After running my program using this function the following were my results:

|  |  |
| --- | --- |
| Read data | nltk word tokenize |
| Cleaning | Letters only + lowercase + stop words + English words |
| Smoothing | Laplace smoothing |
| Number of words in dictionary | 28849 (-44266) |
| Total files | 2000 (0) |
| Correct Neg File Guesses | 855 (-12) |
| Wrong Neg File Guesses | 145 (+12) |
| Correct Pos File Guesses | 790 (+6) |
| Wrong Pos File Guesses | 210 (-6) |
| Total Percentage of Correct Guesses | 82.25% (-0.30) |
| Continue with option | YES |
| Runtime | 63.25 seconds (0) |

# Conclusion

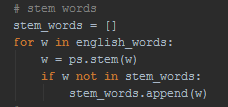
Using only English words decreases the number of words in the dictionary by 44266 which is a 60.54% decrease. It also decreases the total percentage of correct guesses by 0.30%. After reviewing the results, I decided to continue with using only English words as I felt the decrease in the dictionary was worth the decrease in the total percentage of correct guesses.

# Stemming

Stemming removes morphological affixes from words leaving only the word stem. For example, the loving will be changed to love. This will reduce the dictionary and hopefully increase the correct guesses. NLTK has an inbuilt function to stem words which I could use.







After running my program using this function the following were my results:

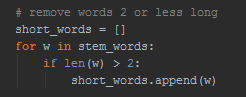
|  |  |
| --- | --- |
| Read data | nltk word tokenize |
| Cleaning | Letters only + lowercase + stop words + English words + stemming |
| Smoothing | Laplace smoothing |
| Number of words in dictionary | 21536 (-7313) |
| Total files | 2000 (0) |
| Correct Neg File Guesses | 852 (-3) |
| Wrong Neg File Guesses | 148 (+3) |
| Correct Pos File Guesses | 805 (+15) |
| Wrong Pos File Guesses | 195 (-15) |
| Total Percentage of Correct Guesses | 82.85% (+0.60) |
| Continue with option | YES |
| Runtime | 141.94 seconds (+78.69) |

# Conclusion

Using stemming decreases the number of words in the dictionary by 7313 which is a 25.35% decrease. It increased the total percentage of correct guesses by 0.60% but increased the runtime by 78.69 seconds. After reviewing the results, I decided to continue with stemming as I felt the overall benefit out weighted the drawbacks.

# Words Less than 2 Characters

After reviewing the data I noticed that I could try to remove words 2 characters or less due I believed that there were no positive or negative words this size.



After running my program using this function the following were my results:

|  |  |
| --- | --- |
| Read data | nltk word tokenize |
| Cleaning | Letters only + lowercase + stop words + English words + stemming + words < 2 |
| Smoothing | Laplace smoothing |
| Number of words in dictionary | 21389 (-147) |
| Total files | 2000 (0) |
| Correct Neg File Guesses | 850 (-2) |
| Wrong Neg File Guesses | 150 (+2) |
| Correct Pos File Guesses | 807 (+2) |
| Wrong Pos File Guesses | 193 (-2) |
| Total Percentage of Correct Guesses | 82.85% (0) |
| Continue with option | YES |
| Runtime | 143.51 seconds (0) |

# Conclusion

Using stemming decreases the number of words in the dictionary by 147. There was no other advantages or disadvantages. After reviewing the results, I decided to continue with removing words < 2 as there were only benefits.

# Only Positive and Negative Words

I finally came up with the idea to try use only positive and negative words. After looking online, I found a site with a list of positive and negative words from a twitter sentiment analysis.

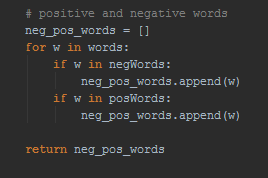
Negative words:

<https://github.com/jeffreybreen/twitter-sentiment-analysis-tutorial-201107/blob/master/data/opinion-lexicon-English/negative-words.txt>

Positive words:

<https://github.com/jeffreybreen/twitter-sentiment-analysis-tutorial-201107/blob/master/data/opinion-lexicon-English/positive-words.txt>





After running my program using this function the following were my results (As I started again I will compare the figures to basic algorithm with no cleaning):

|  |  |
| --- | --- |
| Read data | nltk word tokenize |
| Cleaning | Letters only + lowercase + positive and negative |
| Smoothing | Laplace smoothing |
| Number of words in dictionary | 5511 (-275620) |
| Total files | 2000 (0) |
| Correct Neg File Guesses | 839 (-21) |
| Wrong Neg File Guesses | 161 (+21) |
| Correct Pos File Guesses | 863 (+79) |
| Wrong Pos File Guesses | 137 (-79) |
| Total Percentage of Correct Guesses | 85.1% (3.1) |
| Continue with option | YES |
| Runtime | 51.24 seconds (39.51) |

# Conclusion

Using only positive and negative words decreases the number of words in the dictionary by 275620 which is a 98.04% decrease. This is an enormous decrease making the unique dictionary very small. The next figure to look at is the increase in the total percentage of correct guesses by 3.10%. This again is a very good and surprising increase. There was an increase in the runtime of 39.51 seconds, but this was to be excepted as there was no cleaning in the basic version. After reviewing my findings, I feel this is my best version of pre-processing and I will submit this version as my answer.

Declaration

I, the undersigned, declare that this report is entirely my own written work, except where otherwise accredited, and that it has not been submitted for a degree or other award to any other university or institution

Signed: \_\_\_\_\_\_\_\_\_Darren Smith \_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_27/10/2016 \_\_\_\_\_\_\_\_\_\_