MACHINE LEARNING

Bayesian Classification

David Coughlan
R00009964 DCOM4A

Contents

A Basic Evaluation

Cleaning the Dataset

On the first raw run of the model I achieved a result of 76% for positive and 84% for negative.

After the initial run I immediately implemented a basic method for cleansing the dataset as it was being read in from the files and into the set. As the dataset was quite dirty so I implemented one to get a somewhat workable reading on the accuracy even at an almost untouched pre-processing level.

I decided to use regular expressions with the "\W*" command to find a word in the text [1]. This matches a string consisting of a single character, where that character is alphanumeric (letters and numbers) an underscore or an asterisk. However, this turned out to be trouble-some to my code as it returns a list since it splits the words up on the regular expression and stores the results in an array. So, I added another line using translate from the string package that has the function punctuation [2] and a basic string function strip that removes trailing white space.

I realised that I need to clean the wording going into my vocab set as these unique words are used to populate my dictionaries for both frequency and probability, so it would have the greatest impact on the data coming into my training model. I then thought of the unseen reviews being read in. I decided since I was passing a review into the formula and then calculating each word. I could call the cleaning function there to make the best correlation between the words in the test data and the words in the training data.

I passed one word at a time in lowercase format into this method and split the text based on the regex filter thus giving me an array of the words in the string.

I placed a filter with in the method for loading in the files blocking any None values. I knew this, and my cleansing method was a basic way of filtering the dataset contents however I felt that I would be able to optimize the cleansing of the data later and wanted quantifiable results that had very little impact from the pre-processing methods I was implementing.

Performance

I initially ran the code on the supplied small test dataset having trained the model using the large IMDB data. On the positive reviews I got the overall reading of 77%accuracy. On the negative reviews I got the overall reading of 84%. When I ran the test without any cleaning of the data I was receiving 49% for positive and 77% for the negative.

The basic cleansing of the datasets produced a rise in the positive review predictions of 1% and in the negative predications I received an increase of 1%.

Even with such a small adjustment there was a very small increase over all on the accuracy in the model.

Research and Detailed Evaluation

I researched tips on how to get the most from the Naïve Bayes algorithm and found I was already meeting a lot of the suggestions [3]. I had already dealt with some of the key points. Firstly, missing data, I have dealt with this by not adding the word to the set if it is of type None. I also used logs when calculating joint probabilities as when they are multiplied together the floats often give a loss in precision due to under-runs. I searched the math API for python and found that I should be passing in the base I wanted. [4] The default is in base 'e' so I checked on the page and it said that a method log10() was more accurate than passing in the desired base. I tested both and passing in the base seemed to have more positive results. I made the changes to my formula.

I then found the tip of removing redundant features. [3] The performance of Naïve Bayes degrades if the data contains highly correlated features. This is due to the highly correlated features that are voted for frequently therefore inflating their importance. I decided to include a text file of stop words to refine the datasets by filtering the unimportant but regularly occurring words.

I found a link to suggested stop words from the lecture notes. [5] I put them in a text file and made a method to read them in and stored them in a set which I planned to use for cleaning my review test data. Once the set for my vocab was populated with cleaned words I could remove the stop words by passing the vocab set to a method I had made for removing any stop words present in the set. This would have my dictionaries optimized as well.

For the testing reviews I simply added a clause while processing each word that if it was in the set of stop words not to add its result to the calculation.

I set up a test with my punctuation filter, lowercase, and my stop words filter.

Upon running the positive predictions without the stop words my accuracy was 78.4% and for the negative predictions it resulted in 86.3%. I was happy with these results and thought that anything after would be a slight creep in accuracy.

I decided to research some of the algorithms used in the Natural Language Toolkit for Python. This is a leading platform for building Python programs to work with human language data. [6]

I researched the Porter Stemmer algorithm [8] that is written and maintained by Martin Porter. The Porter stemming algorithm is a process for removing the common morphological and inflexional endings from English words. This would be used to normalize the data being fed into the model.

The algorithm dissects the word and removes the appendages of the base word and then discards them.

I watched a YouTube video of how to install the NTLK as it only supposes 32bit systems. I followed the tutorial and was up and running. It was extremely easy to incorporate into my model. I just imported the package from ntlk.stem and then instantiated the Port Stemmer. I looked up the API for the package [8] and found the function stem which takes a token. This function gets the words stem.

I put this into my cleaning function and commented out other cleaning functions. Upon running this, I received the accuracies: 68% for positive and 88.2%. A massive drop for the positive and only a slight incline for the positive.

I googled which was better for classification of data either porter stemmer or word net lemmatize [9].

The conclusion was that stemmers are faster to run but they don't process the text as efficiently. They crudely chop off the ends of words and then hope for the desired goal to be achieved. Lemmatization does things properly by using vocabulary and analysis of words often able to return the base word from a dictionary.

I looked at incorporating Lemmatization hoping that it would help refine my model using some sort of intelligence.

I brought in WordNetLemmatizer in much the same way that I did with the Porter Stemmer.

I ran the model using lowercase, exclusion of stop words from a text file and the lemmatization.

I achieved 75.2% on the positive and 88.7% on the negative. Dramatically better results over the stemming technique with a 7.5% increase on the positive and a minute increase of 0.5% on the negative.

I decided to google if there was a way of dealing with the stop words from the NLTK as it had to be a common task in training natural language models. I quickly found the package stopwords in the corpus section of the NLTK

I copied what I had already implemented with the text file by passing the values into a set. I ran this with the lowercase words and the lemmatization. This gave me 75.5% on the positive and 89.4% on the negative.

Unfortunately, this was as far as I was able to explore as I simply didn't have enough time to try the negation and the n-grams as well as Ada Boost [10] that I came across on my research into techniques used.

Conclusion

Even though the increase were only ever quite small with the techniques I can appreciate that with a massive dataset or even working with Bigdata that it would have a massive impact. A small increase could mean thousands of predictions being categorised correctly or if the datasets were based on people's health the tuning of the model could have such life changing results.

As for the coding side of the assignment I feel far more comfortable with python now probably as I had to restart it so many times, but I found many powerful tools at my finger tips and the syntax made them quite easy to implement and manipulate once I started to get the hang of the language.

Citations:

1. Regex W*
   https://stackoverflow.com/questions/1576789/in-regex-what-does-w-mean
2. Translate function from string API
   https://stackoverflow.com/questions/265960/best-way-to-strip-punctuation-from-a-string-in-python

3.Tips on improving Naïve Bayes

https://machinelearningmastery.com/better-naive-bayes/

4.Math API Python

https://docs.python.org/2/library/math.html

5.Stop words text file

http://xpo6.com/list-of-english-stop-words/

6.Stemming

https://pythonprogramming.net/stemming-nltk-tutorial/

7.PortStemmer

https://tartarus.org/martin/PorterStemmer/

8.Stemmer API

http://www.nltk.org/api/nltk.stem.html

9. Stemming Vs Lemmatization

http://nlp.standford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

10. AdaBoost

https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/