### COMP8043 Machine Learning

### Assignment 1 – Naïve Bayes Classification

Student: Kevin O’Mahony

ID: R00105946

Date: 17/03/2017

# **Basic Naïve Bayes Classifier**

## Overview of code

The first step in the training phase is to build the word frequency counts for both our positive and negative movie reviews plus create our overall unique vocabulary set. This happens by calling the *trainNB()* function with the required (positive or negative) review file locations, the corresponding word frequency dictionary and a set in which to store the unique vocabulary. *updateVocabAndCounts()* called from *trainNB()* does the real work: reads the given review contents, performs whatever preprocessing is configured, adds the word to our vocabulary set and increments the count for the word in the appropriate dictionary. It was decided to do the word frequency dictionary updates and set updates in the one function for efficiency. After processing all positive and negative review files we need to update our word frequency dictionaries with the words they missed but that exist as unique words in the vocabulary set. Finally, we calculate the probabilities of each word occurring given each of the two classes:

*P( w | C ) = ( count of w for C + 1 ) / ( count of all words for C + size of vocab )*

for word, w and class, C. The probabilities are stored, like the word frequencies in two dictionaries. Laplace smoothing is used (adding one to the numerator and including our vocab size in the denominator ) to handle zero frequency words i.e. words we meet in test that we haven’t seen in the training phase.

Each unique word forms a feature for our classification. We have a large number of features ( > 200,000 ) and so are in danger of getting underflow in our probability calculations. Underflow can occur when multiplying several small numbers together to give a much smaller number, which then might be rounded off to zero by a given programming language/underlying hardware combination. In our context, this can be prevented by taking the log of the probabilities product used in classification. Since the log of a product evaluates to the sum of the logs of the individual terms:

log(ab) = log(a) + log(b)

- we take the log of the individual word probability calculated above and store it in the corresponding dictionary for later summing. This completes the training phase of the classification.

For the classification phase we call *classifyDoc()* which takes each unseen movie review filename, the two, word probability dictionaries and the standalone class probabilities. It reads the file contents and preforms whatever pre-processing we have configured. It then calculates two probability sums: one sum for the probabilities associated with each word given a negative class plus the probability of a negative class occurring and the second sum for the corresponding positive class. Whichever sum is the greater forms the predicted class of the given review.

Multiple enhancements were added to the initial basic program for the second part of the assignment. A .ini style configuration file (bayes.ini) was used to enable individual enhancements be turned on/off. This can be done by setting an enhancements configuration parameter to True/False in the .ini file. The built in *ConfigParser* module was used to read the file and access individual configuration settings. While checking the configuration multiple times for every review no doubt adds some overhead, the ability to mix and match enhancements without code change was felt to outweigh any downside. For this first section all enhancements were turned off.

## Results

The results from training our basic naïve Bayes classifier on the ‘LargeIMDB’ data set and testing on the ‘smallTest’ data set are shown in Table 1 below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Overall** | **Correct** | **Correct** | **Feature** |
| **Accuracy** | **Positive** | **Negative** | **Size** |
|  | *(out of 1,000)* | *(out of 1,000)* |  |
| 82.20 | 784 | 860 | 281,131 |

**Table 1: Results from executing basic Naïve Bayes Classifier**

**on IMDB test dataset.**

For just the basic Bayes algorithm 82.2% seems like a very respectable accuracy. It did better (~10%) at predicting negative movie reviews than positive reviews, perhaps the negative reviews have more significant words that are unique to negative sentiment than positive. We see we have a baseline of 218,131 unique features in our data set.

# **Enhanced Naïve Bayes Classifier**

There are many techniques that can be applied to the standalone Bayes algorithm to improve its accuracy. Below are a sample I have researched and implemented in the program. Each technique was added to the basic algorithm in turn and the results of running against the test data set shown in Table 2 below.

Most of the enhancements below are implemented in the function *preprocess()*. Each follows the same pattern: we check if the given technique is on or off, if on we run the required transform on our text. Each transform is implemented as a list comprehension as a convenient way of running potentially multiple transforms on our original word list.

## Punctuation and Digit Removal

In general, it is better to remove any characters that are not contributing towards the classification process, for example words like: beautiful, beautiful! beautiful!? etc. dilute the real probability that we should be associating with the word beautiful. There are options to separately remove punctuation and digits in bayes.ini. We see that we actually get a drop in overall accuracy of 0.3% for punctuation removal and 0.5% for digit removal. There is also a very significant drop in feature size (~100,000) for punctuation removal. Removing punctuation may be causing some loss of meaning in our context. For example: “the film was terrible!!!!” – here maybe we should retain ‘terrible!!!’ if it is a pattern in a reviewer’s writing as it carries more significant negative sentiment.

## Lowercase

Changing all words to lowercase is another common pre-processing step used to standardise the text [1]. One might think that for sentiment analysis capitalization would be relevant and therefore should be retained but on implementing lowercasing we see that we get a slight accuracy improvement, 0.15% and 30,000 less features. It’s possible that like punctuation removal we are boosting the counts for significant words that otherwise would be split over two or more counts.

## Stop words

Stop words are commonly used words which do not contain any contextual significance. They are often removed from queries by search engines in order to return only the most relevant results. Likewise for naïve Bayes stop words can be removed from the text to reduce the ‘noise’ in the feature space. A freely available stop word list [2] was used and saved to the file stop-words-standard.txt. A second stop word file, stop-words-custom.txt was also created that uses all the words in the standard file and also adds some domain specific words. We expect these words will frequently occur in the text but convey no added meaning for us. Examples of such words are: film, movie, director, actor, plot, script, title, scene, etc. Using the standard file gave a 0.65% accuracy improvement while the custom file added 0.9%. As expected we get a slight reduction in feature size for both.

## Features Selection

Many techniques have been applied to feature selection from using Decision Trees and C4.5 feature selection [3] to Chi-square and Mutual Information Feature Selection [4]. The motivation is again to reduce unnecessary noise in the feature space and minimize training and test execution times. Here we used a simple idea of reducing our feature set to the N highest frequency words. We took N=100,000; 20,000; 2,000 and reran our test set for each. All values of N produced very poor accuracy figures. Interestingly as we reduce the feature size we get proportionally a much smaller reduction in accuracy suggesting there is considerable redundancy in the feature set. The challenge remains capturing the most significant subset.

## Counting only one occurrence per document

We are using the Multinomial model for our naïve Bayes algorithm i.e. counting the total occurrence of words in our data set. The alternative Bernoulli model relies on counting the number of documents where the word occurs. As a hybrid of these two models I looked at counting the occurrence of a given word once per document, referred to as binary multinomial naïve Bayes [5]. This gives the biggest boost so far to the basic algorithm at 1.2%.

## Negation

Negation involves identifying the presence of negation words like ‘not’, ‘no’, etc. and replacing them with Not\_not, Not\_no, etc. My implementation continues this word replacement until we see the next punctuation mark. As a simple test I only considered the word set: ['Not','not','no','No'] for negation. It gave a 0.35% improvement on accuracy with a moderate increase in feature size of ~ 17,000, a larger negation word set would no doubt have seen a much bigger increase.

## Part of Speech

Part of speech classification involves assigning a part of speech tag to each word in the text. The tag is derived from the words definition and its context within the text. There is a standard set of tags [6] in common use. I used a third party library, TextBlob [7] which provides a wide range of NLP functionality including part of speech (POS) tagging. There are a variety of ways of using POS tagged text for sentiment analysis. I initially decided to use it to reduce the feature set to the more powerful words for our context, based on the tags:

**Tag Description**

JJ Adjective

JJR Adjective, comparative

JJS Adjective, superlative

NN Noun, singular or mass

NNS Noun, plural

NNP Proper noun, singular

NNPS Proper noun, plural

RB Adverb

RBR Adverb, comparative

RBS Adverb, superlative

RP Particle

VB Verb, base form

VBD Verb, past tense

VBG Verb, gerund or present participle

VBN Verb, past participle

VBP Verb, non-3rd person singular present

VBZ Verb, 3rd person singular present

This was implemented by checking the first character of the tag to be one of ‘JRVN’. This gave a reduction in accuracy over the basic algorithm to 80%. Using the first character tag set of ‘JRV’ reduced accuracy further to 78.15%. For both cases the overhead of POS tagging caused an order of magnitude increase in the runtime making the program practically unusable.

There are much more sophisticated methods of using POS tagging than our simple application, e.g. combining words based on strong adverb/verb or noun/adjective pairings to create new features. I didn’t investigate any further uses because of the impact POS had on program performance.

## N-grams

N-grams are continuous sequences of n items taken from a text sequence. They are widely used in natural language processing applications and have been the subject of extensive research [8]. I used my own implementation in *ngrams()* to generate space separated n-grams. Using n=2 produced an accuracy improvement of 4.6% and for n=3, we get +4.65%, the largest improvements seen to date. We do get the expected explosion in feature size, for n=2, almost 2 million and n=3, 4 million features. Going to n=4 gives an accuracy reduction of 0.25% so it would appear n=3 is optimal.

The *ngrams()* function does not differentiate punctuation, so it will combine words across a full stop. Intuitively this would seem undesirable as we would be generating unintended semantic combinations. I tested this with ngram generation, stopping at a full stop if within n words. Allowing for full stops gave an accuracy of 87.55% instead of the 89.1% when we ignore full stops. Despite being non-intuitive I left *ngrams()* as was – ignoring full stop.

## Combining Enhancements

As a final step I looked at what combinations produce the best accuracy figures. Table 2 below shows the best combinations found. For single enhancements we got to 87.15% using trigrams. Adding in negation, one occurrence per document and lowercase we move to 88.4%. Taking out negation and adding punctuation and digit removal gives a boost to 89.1%. This appear to be the maximum accuracy available with the current implementation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Overall** | **Correct** | **Correct** | **Feature** |
| **Additional Technique** | **Accuracy** | **Positive** | **Negative** | **Size** |
|  |  | *(out of 1,000)* | *(out of 1,000)* |  |
| Ngrams n=3 | 87.15 | 854 | 889 | 4,055,650 |
| Ngrams n=2 | 87.10 | 865 | 877 | 1,947,307 |
| SingleOccurrencePerDoc | 83.40 | 797 | 871 | 281,131 |
| StopWords (stop-words-custom.txt) | 83.10 | 800 | 862 | 280,484 |
| StopWords (stop-words-standard.txt) | 82.85 | 794 | 863 | 280,535 |
| Negation | 82.55 | 793 | 858 | 297,566 |
| Lowercase | 82.35 | 784 | 863 | 252,165 |
| **None** | **82.20** | **784** | **860** | **281,131** |
| Ngrams n=4 | 81.95 | 796 | 843 | 5,214,293 |
| RemovePunctuation | 81.90 | 783 | 855 | 181,350 |
| RemoveDigits | 81.70 | 782 | 852 | 275,878 |
| PartOfSpeech ( 'JRVN' ) | 80.00 | 770 | 830 | 118,765 |
| PartOfSpeech ( 'JRV' ) | 78.15 | 749 | 814 | 34,635 |
| ReduceFeature | 53.00 | 514 | 546 | 100,000 |
| ReduceFeature | 43.25 | 464 | 401 | 20,000 |
| ReduceFeature | 32.05 | 333 | 308 | 2,000 |

**Table 2: Results from executing basic Naïve Bayes Classifier with various additional techniques**

**on IMDB test dataset.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Overall** | **Correct** | **Correct** | **Vocab** |
| **Combinations** | **Accuracy** | **Positive** | **Negative** | **Size** |
|  |  | *(out of 1,000)* | *(out of 1,000)* |  |
| RemovePunctuation |  |  |  |  |
| RemoveDigits | 89.10 | 871 | 911 | 3,719,759 |
| Ngrams n=3 |  |  |  |  |
| SingleOccurrencePerDoc |  |  |  |  |
| Lowercase |  |  |  |  |
| Negation |  |  |  |  |
| Ngrams n=3 | 88.40 | 870 | 898 | 4,038,544 |
| SingleOccurrencePerDoc |  |  |  |  |
| Lowercase |  |  |  |  |
| Negation |  |  |  |  |
| Ngrams n=2 | 87.55 | 866 | 885 | 1,928,623 |
| SingleOccurrencePerDoc |  |  |  |  |
| Lowercase |  |  |  |  |

**Table 3: Results from executing basic Naïve Bayes Classifier with various combinations of additional techniques on IMDB test dataset.**

**Bibliography**

[1] K.-M. Schneider, “Techniques for Improving the Performance of Naive Bayes for Text Classification.”

[2] Unknown, “List of English Stop Words | XPO6.” [Online]. Available: http://xpo6.com/list-of-english-stop-words/. [Accessed: 19-Mar-2017].

[3] C. Ratanamahatana and D. Gunopulos, “Feature selection for the naive bayesian classifier using decision trees,” *Appl. Artif. Intell.*, vol. 17, no. 5–6, pp. 475–487, 2003.

[4] Datumbox, “Using Feature Selection Methods in Text Classification | Datumbox.” [Online]. Available: http://blog.datumbox.com/using-feature-selection-methods-in-text-classification/. [Accessed: 19-Mar-2017].

[5] L. Speech, Processing, Daniel, H. James, and Martin, “Naive Bayes and Sentiment Classification.”

[6] P. Treebank, “Penn Treebank P.O.S. Tags.” [Online]. Available: https://www.ling.upenn.edu/courses/Fall\_2003/ling001/penn\_treebank\_pos.html. [Accessed: 19-Mar-2017].

[7] TextBlob, “TextBlob: Simplified Text Processing — TextBlob 0.12.0 documentation.” [Online]. Available: https://textblob.readthedocs.io/en/dev/. [Accessed: 19-Mar-2017].

[8] J. Urnkranz, “A Study Using n-gram Features for Text Categorization.”