DJ Ambler

Course Project Report

Problem Description

For this problem, I will be taking a dataset consisting of 2000 movie reviews from IMBD, and seeing if the machine is capable of using sentiment analysis to properly predict whether the reviews are positive or negative by looking at the words used in each review. I will be using Naive Bayes classification to classify the reviews according to their overall sentiment (positive/negative).

Background:

In this problem I will be using sentiment analysis to extract or categorize the sentiment content of a piece of sample text. Sentiment analysis is the use of statistics, natural language processing and machine learning to extract or categorize the sentiment content of a piece of sample text. Sentiment analysis is a type of document or text classification, which is assigning a document to one or more classes or categories. This may be done "manually" (or "intellectually") or algorithmically (Katti). Documents may be classified according to their subjects or according to other attributes (such as document type, author, printing year etc.). Unfortunately, sentiment analysis is a field that is kind of touchy feely, which makes it somewhat difficult for computers to accurately compute the results sometimes. In this experiment, I will simply be seeing if the machine can accurately predict whether a movie review is positive or negative based on the words used in it. Generally speaking, sentiment analysis isn’t something that’s usually simple enough to divide the results into a simple yes or no like I will be doing, which may cause the results to be inaccurate. I will also be using multinomial Naive Bayes classification algorithm to classify 2000 movie reviews according to their overall sentiment (positive/negative). The Naive

Bayes classification is based on the Bayes' theorem in statistics. The theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event (An Intuitive). Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. All naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable (Naive Bayes Classifier). With a multinomial event model, samples (feature vectors) represent the frequencies with which certain events have been generated by a multinomial $(p_1, \dots \dots, p_n)$ where p_i is the probability that event *i* occurs (or *K* such multinomials in the multiclass case). A feature vector $x = (x_1, \dots\dots, x_n)$ is then a histogram, with x_i counting the number of times event *i* was observed in a particular instance. This is the event model typically used for document classification, with events representing the occurrence of a word in a single document. The likelihood of observing a histogram **x** is given by

$$p(x|C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i\,!} \prod_i^n p_{ki}^{x_i}$$

I will be using bag of words as the representation in this problem. In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. This is typically used to create a list of each word and how frequently it is used (Brownlee).

Methodology:

In this experiment I will be importing the data set into R then using the documents to create a corpus. After this, I will clean the corpus by removing stop words, punctuation, etc… I will then create a document term matrix using the corpus, then remove words that are not used frequently to make it easier. I will be using 1500 of the reviews as the training data, and the remaining 500 to test the machine. After both training and testing using the corpus and the document term matrix, I will create a table to see the predictions the machine made versus the actual results. Finally, I will create a confusion matrix and observe it to see how accurate the machine really is.

Assumptions:

The biggest assumption that I am making in this experiment is that the machine is capable of sentiment analysis to some extent. Since sentiment analysis usually takes a lot of time to sit through and analyze each document, having the machine go through and do this process in rapid succession may make lead the machine to being less accurate than a normal human would be.

Experimental Design:

In this problem I will be using the bag of words representation and training a multinomial Naive Bayes classifier on the data and testing the model's performance on a hold-out set. The data set is a set of 2000 movie reviews put together by Pang and Lee. The general process begins by randomizing the dataset. In this approach, I will represent each word in a document as a token (or feature) and each document as a vector of features. In addition, for simplicity, I will disregard word order and focus only on the number of occurrences of each word i.e., I will represent each document as a multi-set 'bag' of words. First I will create a corpus of all the documents in the dataset, then I will clean up the corpus by eliminating numbers, punctuation, white space, and by converting to lowercase. I will also be discarding common stop words such as "his", "our", and "couldn't". I will represent the bag of words tokens with a document term matrix (DTM). The rows of the DTM correspond to documents in the collection, columns correspond to terms, and its elements are the term frequencies. I will use a built-in function from the ‘tm’ package to create the DTM. Next, I will create 75:25 partitions of the data frame, corpus and document term matrix for training and testing purposes. The DTM contains a large number of features but not all of them will be useful for classification. I will be reducing the number of features by ignoring words which appear in less than five reviews. To do this, I will use ‘findFreqTerms’ function to identify the frequent words. I will then restrict the DTM to use only the frequent words using the ‘dictionary’ option. I will use a variation of the multinomial Naive Bayes algorithm known as binarized (boolean feature) Naive Bayes due to Dan Jurafsky. In this method, the term frequencies are replaced by Boolean presence/absence features. The logic behind this being that for sentiment classification, word occurrence matters more than word frequency. To train the model I will use the naiveBayes function from the ‘e1071’ package. Since Naive Bayes evaluates products of probabilities, I need some way of assigning non-zero probabilities to words which do not occur in the sample. I will use Laplace 1 smoothing to this end. I will then test the predictions and look at the confusion matrix to see how accurate the machine is.

Results and Discussion:

For this experiment, I decided to do the computations three times with three different randomized seeds. I will be documenting the results of each of these three attempts.

<u>Results of the first attempt:</u>

For the first try, I set the seed to 98. After randomizing the order of the documents, I created a corpus with them. I then used dplyr’s %>% (pipe) utility to clean the corpus. This function removes punctuation, numbers, capital letters, white space, and the most common stop words in the English language. Next, I created a document term matrix using the corpus then inspected it. I specifically looked at only 6 terms spread across 11 documents. It found the sparsity to be 65%. After this, I create 75:25 partitions of the dataframe, corpus and document term matrix for training and testing purposes. Inspecting the document term matrix again, I see that there are 38957 features. Most of these are not going to be useful, so I decided to reduce this number by ignoring words that occurred in less than five reviews. I then used the findFreqTerms function to identify the frequent terms, then recreate the document term matrix with only these words. Now it finds that there are 12129 features. I then use a variation of the multinomial Naive Bayes algorithm known as binarized (boolean feature) Naive Bayes due to Dan Jurafsky. In this method, the term frequencies are replaced by Boolean presence/absence features. The logic behind this being that for sentiment classification, word occurrence matters more than word frequency. After this I apply this new function to get the final training and testing document term matrices. I then train the machine using 1500 of the reviews as the training set. It took roughly 3 seconds for the machine to complete the training. Next is testing if the machine can predict properly. This function took 143 seconds to complete this time. I created a prediction vs. actual table, which looked like this:

	Actual	
Predictions	Negative	Positive
Negative	81	68
Positive	179	172

It predicted that there were 149 negative reviews, 81 of which were actually negative and 68 that were actually positive. It also predicted that there were 351 positive reviews, 172 of which were actually positive and 179 that were actually negative. According to the confusion matrix, the accuracy of this is 50.6%

<u>Results of the second attempt:</u>

For this attempt, I set the seed to 402 and started over again. After creating the corpus and the document term matrix, it found the sparsity to be 88%. Again, there were 38957 features in the document term matrix before reducing the number of words. After restricting it to the most frequent words, there were 12116. Training the machine took 3 seconds again, but testing it only 122 seconds this time. The prediction vs. actual was as follows:

	Actual	
Predictions	Negative	Positive
Negative	69	58
Positive	188	185

It predicted that there were 127 negative reviews, 69 of which were actually negative, and 58 that were actually positive. It also predicted that there were 373 positive reviews, 185 of which were actually positive, and 188 that were actually negative. According to the confusion matrix, the accuracy of this attempt was 50.8%.

<u>Results of the third attempt:</u>

For this final attempt, I used 1 as the seed. After creating the corpus and document term matrix, the machine found the sparsity to be 83%. Again, it found there to be 38957 features in the matrix before I reduced the words. After restricting the matrix down to only the frequent words, there were 12083 features. Training the machine took 3 seconds again, and testing it took 122 seconds. The prediction vs. actual table was as follows:

	Actual	
Predictions	Negative	Positive
Negative	88	93
Positive	153	166

It predicted that there were 181 negative reviews, 88 of which were actually negative, and 93 that were actually positive. It also predicted that there were 319 positive reviews, 166 of which were actually positive, and 153 that were actually negative. According to the confusion matrix, the accuracy of this is 50.8%.

Discussion:

The first thing that I would like to call attention to is that somehow between the three attempts, the machine found a different number of frequent terms each time. They were all relatively similar, but they were somehow different, even though they should have been the same each time. I do not think that the different was enough to change the end results, but I thought it was interesting enough to point out. Something else I found interesting was that the first attempt took approximately 20 seconds longer to test than the second and third attempt. Again, I don't think that this affected the results, but I thought it was interesting. It seems that in every attempt, the machine predicted that there were almost twice as many positive reviews as negative. It is possible that this is caused by some common negative words such as 'not' and 'no' being removed for being stop words. Each result had roughly the same accuracy at about 50.8%. Frankly, I was certainly hoping for a higher accuracy number, but this result does not surprise me, since I am not very adept at coding this kind of stuff, and the machine probably won't do a good job of sentiment analysis without much more fine tuning anyway.

Issues:

My biggest issue was the same as every other experiment, which is that I am not familiar with much of the code needed to do this experiment. I still don't fully understand all of the code that I used to do this project, which may be why the results aren't super great.

Conclusions and Future Work:

Unfortunately, it seems that my code does not make for a very good predictor of whether a movie review is positive or negative. Being just above 50% accuracy, you may as well just flip a coin instead. The most obvious future work would be to somehow refine the code to see if it's possible to make the predictions more accurate. Aside from this, other work could include trying to create a scale for results, rather than just negative or positive. Perhaps trying to see if the machine could accurately predict a movie review on a scale from 1 to 10 would be interesting future work.

Works Cited:

An Intuitive (and Short) Explanation of Bayes' Theorem. (n.d.). Retrieved November 15, 2017, from https://betterexplained.com/articles/an-intuitive-and-short-explanation-of-bayes-theorem/

Brownlee, J. (2017, August 15). A Gentle Introduction to the Bag-of-Words Model. Retrieved November 15, 2017, from https://machinelearningmastery.com/gentle-introduction-bag-words-model/

Katti, R. (2016, April 30). Naive Bayes Classification for Sentiment Analysis of Movie Reviews. Retrieved November 15, 2017, from https://rpubs.com/cen0te/naivebayes-sentimentpolarity

Movie Review Data. (n.d.). Retrieved November 16, 2017, from http://www.cs.cornell.edu/people/pabo/movie-review-data/

Naive Bayes Classifier. (n.d.). Retrieved November 15, 2017, from http://www.statsoft.com/textbook/naive-bayes-classifier