CMPS242 Final Project - A Comparison of Naive Bayes and Boosting

Brian Madden
December 11, 2009

1 Project Description

My final project was to implement and compare a number of Naive Bayes and boosting algorithms. For this task I chose to implement two Naive Bayes algorithms that are able to make use of binary attributes, the multivariate Naive Bayes and the multinomial Naive Bayes with binary attributes. For the boosting side of the algorithms I chose to implement AdaBoost, and its close bother AdaBoost*. Both of these algorithms were chosen for their simplicity yet amazingly accurate results; in fact, the IEEE International Conference on Data Mining (ICDM) listed both AdaBoost and Naive Bayes as top ten algorithms in data mining [3].

2 Data

The data used in this project was found on the LIBSVM website (http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html) and is binary classification data consisting of 123 features. There are 32,561 training examples and 16,281 test cases. Approximately 76% percent of the test cases contained negative classes (class 0) as opposed to a positive class (class 1).

3 Algorithms

3.1 AdaBoost

The first algorithm used was the AdaBoost algorithm. The premise of AdaBoost is to assign more weight to examples that are more difficult to classify. The AdaBoost algorithm uses a number of classifiers to make its predictions and as the algorithm runs these classifiers are grouped into weighted ensembles. These ensembles are groups of weak learners (a weak learner being defined as performing just better than random) that together are able to perform better than any single learner.
In the case of this project, the learners used for boosting were the features themselves. The algorithm used is as follows:

For each feature the edge was computed using:

\[ r = w \cdot u \]

where \( u = y \cdot h \) and \( y = \) actual value, \( h = \) feature

The max edge was then selected and a weight for the classifier that corresponds with that edge computed using:

\[ \alpha = \frac{1}{2} Lr \left( \frac{1 + r}{1 - r} \right) \]

For this algorithm, the number of classifiers in our ensemble was tuned using five fold cross validation. For every length of ensemble from 25 to 123 five linear step sizes was used. AdaBoost was trained on four fifths of the data, and then tested against one fifth. For each iteration the four fifths and one fifth was changed and the results averaged across the five runs. This was all done ten times and the results averaged.

3.2 AdaBoost*

The AdaBoost* algorithm is very similar to the standard AdaBoost algorithm but with an added parameter when calculating the \( \alpha \) parameter:

\[ \alpha = \frac{1}{2} \ln \left( \frac{1 + r}{1 - r} \right) - \frac{1}{2} \ln \left( \frac{1 + p}{1 - p} \right) \]

where \( p \) is computed as:

\[ p = r - v \]

, and \( v \) is a parameter chosen and passed as input to the algorithm. Knowing little about this parameter I tried a number of methods to tune it which will be presented in the results section.

3.3 Multi-variate Naive Bayes

The multi-variate Naive Bayes algorithm is a member of the Bayesian family of algorithms, but is special because of the naive assumptions it makes about the data. The first naive assumption that the algorithm makes is the same in all naive Bayes algorithms – that the features of the data are all independent. In a more complex Bayesian model features may be seen as conditional; that is, that one condition may be more or less likely given the presence of a previous feature. The second naive assumption that the multi-variate Naive Bayes algorithm makes is that the features of the data are independent given the class [1]. These naive assumptions aside, the algorithm is still able to classify at good rates.

2
The algorithm is as follows:

A probability of a feature occurring in a given class is calculated for each feature:

\[ p(\vec{x}|c) = \prod_{i=1}^{m} p(t_i|c)^{x_i} \cdot (1 - p(t_i|c))^{(1-x_i)} \]

where \( m \) is the number of features, \( \vec{x} \) is a vector of example features, and \( c \) is a class (binary).

Classification becomes a matter of:

\[ p(c_a) \cdot \prod_{i=1}^{m} p(t_i|c_a)^{x_i} \cdot (1 - p(t_i|c_a))^{(1-x_i)} > \text{threshold} \]

and each \( p(t|c) \) is estimated using a Laplacian prior as:

\[ p(t|c) = \frac{1 + M_{t,c}}{2 + M_c} \]

\( M_{t,c} \) being the number of training examples of category \( c \) containing token \( t \), while \( M_c \) is the total number of training examples of class \( c \) [1].

The Laplacian prior is necessary to prevent 0 probabilities for features that occur infrequently, or even never in the training data. This is a simple example of smoothing.

### 4 Multinomial Naive Bayes with Boolean Attributes

The multinomial Naive Bayes is similar to the multi-variate case, but with a different calculation of \( p(\vec{x}|c) \). Whereas the multi-variate case calculated probabilities based on the occurrence of a word in the example the multinomial case uses the frequency of occurrence as a probability. Thus \( p(\vec{x}|c) \) becomes:

\[ p(\vec{x}|c) = \frac{p(|d|)}{|d|!} \cdot \prod_{i=1}^{m} \frac{p(t_i|c)^{x_i}}{x_i!} \]

and the criterion for classifying becomes:

\[ \frac{p(c_a) \cdot \prod_{i=1}^{m} p(t_i|c_a)^{x_i}}{\sum_{c \in \{c_a, c_b\}} p(c) \cdot \prod_{i=1}^{m} p(t_i|c)^{x_i}} > \text{threshold} \]

where \( p(t|c) \) is calculated using a Laplacean prior as:

\[ p(t|c) = \frac{1 + N_{t,c}}{m + N_c} \]

and \( N_{t,c} \) is the number of times a feature is present in the training examples of category \( c \), and \( N_c = \sum_{i=1}^{m} N_{t_i,c} \), or in other words, \( N_c \) is the total number of features present in a category \( c \).
5 Comparators

The primary mode of comparison that I used was accuracy. Since these are all algorithms that will be used for classification an accuracy comparison seems most appropriate and most useful in a “real world” sense.

6 Results

6.1 AdaBoost

AdaBoost was a good performer on this data set classified with an average of around 84% accuracy with just the 123 base attributes. The following graph shows the accuracy of the AdaBoost algorithm over ten trials with only 123 attributes used and an ensemble length of 94.

In an attempt to coerce AdaBoost to perform better I also tried ten runs with the negations of the features included. This afforded me slightly better performance, but ultimately only about 1% in the average of the 10 runs. The following graph shows these ten runs.
The ensemble length of 94 was selected by performing 5 fold cross validation with increasing ensemble lengths ranging from 25 to 123 in 5 linear steps. The following graph illustrates the average training error and average testing error over these five folds as ensemble length grows.
The only real thing to note about these accuracies is that there does not appear to be any over fitting. Over fitting would be easy to spot as the training accuracy would continue to grow as the testing accuracy would tend to decline. Or in other words, the training error would decrease as testing error increased. AdaBoost is particularly resilient to over fitting though so this was a somewhat expected result.

The final tuning I tried with AdaBoost was to increase the maximum ensemble length to 2000 to see if there was just a plateau in accuracy for the ensemble lengths of 94−123, however this was not the case. As there were no interesting results from this no graph is provided.

6.2 AdaBoost*

The AdaBoost* algorithm was also a decent performer and classified with an average accuracy of 83% using an ensemble length of 94 and an accuracy parameter of \( v = .1 \). This result was obtained after tuning the accuracy parameter \( v \) over 10 runs with a set ensemble length of 94. The following graph illustrates the tuning of the \( v \) parameter over 10 runs.
Unfortunately I didn’t quite understand how the $v$ parameter was supposed to be chosen, but based on these results it seems as the lower $v$ is a better performer. Based on the results of tuning the $v$ parameter I believe that it may work by specifying how close to the maximum margin that would like to be achieved. A small value (such as $v = .1$) would cause more weight to be allowed on each example, whereas a larger $v$ would put less allowable weight on each example. This may cause the algorithm to converge faster (as it can be a bit less precise) but would lead to less accurate results (as seen in the graph). This tuning parameter may be useful if the AdaBoost algorithm was taking a long time to converge as there is very little performance degradation until $v > .6$.

Similarly to the AdaBoost algorithm I tuned the ensemble length over five folds. Below is a graph of the training error vs testing error as the ensemble lengths are allowed to grow. Again, as with AdaBoost there is no evidence of over fitting.
An interesting thing to note about this particular graph is that there is little accuracy gained after ensemble length of about 70. This reinforce the fact that the $v$ parameter is able allow AdaBoost* to converge faster as the AdaBoost* algorithm needed a smaller ensemble size to attain close to its best results.

The next graph shows the average accuracy over 10 runs of the AdaBoost* algorithm with $v = .1$ and ensemble length of 94.
6.3 Multi-variate Naive Bayes

The multi-variate Naive Bayes algorithm was also a strong performer for this classification task. Unlike AdaBoost and AdaBoost* there wasn’t any reason to average results over 10 runs. Due to the nature of the algorithm permutations of the data have no effect on the probability distribution that the algorithm creates and uses to classify.

The accuracy of the algorithm however does change based on the threshold value used. The follow graph presents an ROC curve that shows how the true positive vs false positive rate changes as the threshold value changes.
The points each represent a true positive vs false positive rate for a linearly increasing threshold from .01 to 1 in increments of .01.

Initially I started with a threshold value of .5 as seems fitting with a binary classifier. When using a threshold of .5 the accuracy of the algorithm was just about 80%. The confusion matrix of results obtained with a threshold of .5 is as follows:

<table>
<thead>
<tr>
<th></th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>3056</td>
<td>10035</td>
</tr>
<tr>
<td>False</td>
<td>2400</td>
<td>790</td>
</tr>
</tbody>
</table>

Upon examining the ROC curve for the algorithm I did notice that as the threshold increases the accuracy also tends to get better. This makes sense as it is essentially forcing the algorithm to be more certain about a class to assign it to the positive class. The following graph shows the accuracy of the algorithm as the threshold grows from .1 to .9 in steps of .1.
Re-running the algorithm with a threshold of .9 about 3% accuracy was gained (from 80% to 83%). The confusion matrix for the algorithm with threshold of .9 is as follows:

<table>
<thead>
<tr>
<th></th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>2365</td>
<td>11228</td>
</tr>
<tr>
<td>False</td>
<td>1207</td>
<td>1481</td>
</tr>
</tbody>
</table>

6.4 Multinomial Naive Bayes with Boolean Attributes

Right out of the gate the multinomial Naive Bayes with boolean attributes was better than its multi-variate brother. Using a threshold value of .5 the algorithm was able to classify at 83% accuracy. The confusion matrix is as follows:

<table>
<thead>
<tr>
<th></th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>1792</td>
<td>11811</td>
</tr>
<tr>
<td>False</td>
<td>624</td>
<td>2054</td>
</tr>
</tbody>
</table>

Just as multi-variate Naive Bayes is tuned with the threshold value, so is the multinomial case. The following ROC curve shows true positive rates vs false positive rates for threshold values of .01 to 1 in steps of .01.
Whereas the multi-variate Naive Bayes performed better as the threshold grew, this does not seem to be the case with the multinomial case. As shown in the graph below, as the threshold value grows the performance of the multinomial Naive Bayes decreases. The optimal threshold in this case seems to be .3, where the algorithm hits 84% accuracy.
Running the algorithm with a threshold of .3 attains an 84% accuracy with the following confusion matrix:

<table>
<thead>
<tr>
<th></th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>2182</td>
<td>11502</td>
</tr>
<tr>
<td>False</td>
<td>933</td>
<td>1664</td>
</tr>
</tbody>
</table>

7 Conclusions

After implementing and playing with all of the algorithms they seem to be on equal footing. With the data set used all of the algorithms performed about on par, giving accuracies within a few percent each other (after proper tuning). The vanilla AdaBoost performed slightly better than the AdaBoost* algorithm, but took slightly longer to converge than the AdaBoost*. The multinomial Naive Bayes performed better with no tuning than the multi-variate case, but both performed within one percent of each other after tuning.

Anecdotally in a real world scenario I would use multinomial Naive Bayes as a first algorithm due to its simplicity to implement and the fact that it runs much faster and requires less tuning than AdaBoost. I would however probably use AdaBoost if accuracy was paramount and time was of no real factor.

Based on readings and prior experience with the algorithms I had expected a bit better performance from both sides, but I believe that the nature of the data
may have had an impact on each algorithm. To get better performance from AdaBoost having more features may have helped, for example in homework 3 AdaBoost had a much higher accuracy which may be attributed to the data itself (being easier to classify) or the fact that there were many more attributes available to work with.

The same is true for Naive Bayes; having prior experience with these algorithms (though not with binary labels) I believe that more features may have made the data easier to classify. For example, many Naive Bayes spam classifiers have thousands of attributes rather than just over one hundred.

8 References

