Movie Review Sentiment Classification Using Naive Bayes

Behavioral Data Mining, Assignment 1, Spring 2012

Eric Battenberg
February 5, 2012

1 Introduction

In this assignment, we were to apply a simple Naive Bayes Classifier (NBC) to text sentiment classification. NBC is a probabilistic classification model that is termed “naive” due to the fact that each observed feature contributes independently to the overall class probability (a naive assumption). This independence assumption can be seen in eq. 1, where the probability that document $d$ belongs to class $c$ is proportional to the prior probability of class $c$ times a product of independent feature probabilities conditioned on class $c$.

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c) \quad (1)$$

For text classification, a document is composed of $n_d$ terms, where $t_k$ represents the $k$th term.

In order to estimate the various conditional term probabilities, $P(t_k|c)$, a simple term frequency count is computed on the training data for each class. This is shown in eq. 2, where $T_{ct}$ is the number of times term $t$ occurs in the training data for class $c$, and $V$ is the set of all terms in the vocabulary.

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}} \quad (2)$$

A practical problem with eq. 2 is that if a term does not occur in any of the training data but does occur during classification, a probability of zero will be computed in eq. 1, regardless of the probabilistic strength of the other terms. A simple approach to dealing with this is “add-one smoothing”, in which one is added to each term count (shown in eq. 3). This eliminates zero term counts, instead assigning a very small probability to terms absent from the training data.

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} \quad (3)$$
2 Implementation

The above model was implemented in the Python programming language with the help of the Numpy numerical library[1] and the Natural Language Toolkit[2]. The FreqDist class, which is basically a hash table that counts the occurrences of terms, is the main contribution of the Natural Language Toolkit. The script is made up of approximately 180 lines of code. Add-one smoothing was used, as zero probabilities occurred much too frequently without it. The presence of terms were modelled as multinomial, wherein repetitions of the same term in a single document are counted as independent features, each with the same conditional probability. An alternative approach, binary modelling, simply uses the presence or non-presence of a term in a document, ignoring the number of times a term is repeated.

Probabilities were computed in the log domain to avoid floating point underflow and to allow classification to be performed using a dot product between log-domain term weights and term counts. The prior class probability, \( P(c) \), was ignored because there was an equal number of positive and negative class examples in the dataset.

The implementation includes stopword removal before term counting. Stopwords are common words; such as “the”, “and”, and “to”; that are not likely to be useful in text classification. I used the list of 127 English stopwords provided by the Natural Language Toolkit.

3 Results

Accuracy of the classifier was evaluated using 10-fold cross-validation on the provided set of 1000 positive and 1000 negative movie reviews. Table 1 shows the F1 score and the Matthews Correlation Coefficient (MCC). The MCC is basically the correlation between the predicted class label, 1, \(-1\), and the actual class label, and is shown in eq. 4.

\[
MCC = \frac{tp \times tn - fp \times fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}}
\]

(4)

Where \( tp, fp, tn, \) and \( fn \) are the numbers of true positives, false positives, true negatives, and false negatives, respectively.

The MCC was chosen to supplement the F1 score due do the fact that it was designed for binary classification, whereas the F1 score was designed to measure the accuracy of a binary test.

<table>
<thead>
<tr>
<th></th>
<th>F1 Score</th>
<th>MCC</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Stopword Removal</td>
<td>0.812</td>
<td>0.629</td>
<td>79 sec</td>
</tr>
<tr>
<td>Stopword Removal</td>
<td>0.813</td>
<td>0.632</td>
<td>59 sec</td>
</tr>
</tbody>
</table>

Table 1: Accuracy and execution time with and without stopword removal.
Stopword removal resulted in a very slight increase in accuracy (3 additional correct classifications out of 2000). The main improvement was in training time. The removal of a large number of common words from the training data resulted in a significant speed improvement.

4 Discussion

4.1 Terms with largest weights

We can examine the trained classifiers by looking at which terms were given the largest weights (class conditional probabilities). Since we are training a movie review sentiment classifier, the hope would be that strongly weighted terms of the positive model would express positive sentiment and vice-versa for the negative model. The lists shown in 4.1 and 4.2 come from the models created using the entire dataset as training data along with stopword removal.

Top 10 weighted positive terms:
film, one, movie, it’s, like, story, also, good, even, time.

Top 10 weighted negative terms:
film, movie, one, like, it’s, even, good, time, would, get.

These top terms do not really point to anything interesting as far as discriminating between the classes is concerned. They are just a list of the most commonly occurring terms.

4.2 Terms with top ratios

A more interesting analysis can be made using the ratio between the positive weight and the negative weight (and vice-versa). This way we gain information about which terms most strongly affect the likelihood ratio.

Top 10 positive/negative ratios:
shrek, gattaca, mulan, ordell, truman’s, flynt, guido, leila, sweetback, taran.

Top 10 negative/positive ratios:
&nbsp, seagal, brenner, sphere, stigmata, 1900, bye, silverman, supergirl, musketeer.

These lists also are not very enlightening as they just contain lists of terms specific to particular movies included in only the positive or negative training sets. This does show how the models could be improved with a much larger training set, since it is an invalid assumption that any review containing the word e.g. “musketeer” is a bad review.

4.3 Terms with top significance

To discover the terms that actually affected classification the most, we can accumulate each term’s overall contribution to all likelihood ratios computed for every classification during cross-validation. We are basically multiplying the
ratios between the positive and negative weights by the number of times the term occurs in each test set.

Top 10 positive/negative significance:
life, film, also, great, best, world, many, jackie, mulan, truman.

Top 10 negative/positive significance:
bad, movie, worst, plot, *, even, stupid, boring, like, there’s.

Here we finally start to see words of accolade appearing in the positive list with only a sprinkling of movie-specific terms, and the negative list definitely contains a strong showing of demeaning words.

Beyond movie-specific term problem pointed out in 4.2, which could be solved with a much larger training dataset, it is likely that the inclusion of bi-gram training would yield additional improvement, but it could also significantly increase the execution time of the model.

References
