CS 294-1: Assignment 1
Naive Bayes Classification with Improvements

Shaunak Chatterjee
Computer Science Division
University of California
Berkeley, CA 94720
shaunakc@cs.berkeley.edu

Abstract

The main objective of this assignment was to implement a Naive Bayes classifier and attempt certain improvements upon the vanilla version. A major challenge was to implement the classifier in Scala using the two libraries - scalala and scalanlp. This report presents details regarding the different experiments I tried out, namely varying the smoothing parameter, feature selection, n-gram models and mixture of models.

1 Introduction

The Naive Bayes (NB) Classifier is one of the most popular classifiers due to its simplicity and robustness. Given a set of documents \( D \) and a set of classes \( C \), the probability that a document \( d \) lies in class \( c \) is given by:

\[
P(c|d) \propto p(c) \prod_{1 \leq k \leq N_d} p(t_k|c)
\]

where \( p(t_k|c) \) is the probability that the \( k^{th} \) token appears in document class \( c \). \( p(c) \) is the prior probability of class \( c \). \( N_d \) is the number of tokens in document \( d \). The salient (and naive) assumption of this classifier is that the token probabilities are independent. For classification, we find the most likely class given a document. This Maximum A Posteriori (MAP) estimate is found using:

\[
c_{MAP} = \arg\max_{c \in C} \hat{p}(c|d) = \arg\max_{c \in C} \hat{p}(c) \prod_{1 \leq k \leq N_d} \hat{p}(t_k|c)
\]

where \( \hat{p} \) denotes an estimate of the probability distribution \( p \). People mostly work with \( \log \) of the probabilities for computational efficiency and numerical stability.

Parameters for an NB classifier are easy to learn (they are simply term frequencies; no iterations required) and can be estimated in a decoupled manner. It is also known to work quite well for sparse data (i.e. with unseen words/tokens).

However, its weakness lies in its over-simplistic assumption of independence between each term.

This assignment involved implementing a Naive Bayes Classifier to classify movie reviews into one of two classes — “Positive” or “Negative”. The dataset (available at http://www.cs.cornell.edu/People/pabo/movie-review-data/) comprised of 1000 positive and 1000 negative movie reviews in English.

2 Experimental Setup

10-fold cross validation was performed for generating most of the results. The review files are numbered 0 through 999. I divided them up into 10 sets based on their last digit. In each “fold”, I used 9 of these sets for training and the other one for testing. For experiments requiring validation, I used 8 of these sets for training, 1 for validation and 1 for testing (in each fold). In some cases (and this will be explicitly mentioned), I trained with smaller amount of data to see what effect that would have.

The next few sections describe the different experiments that I performed and some of my relevant observations.

3 Getting started: Unigrams and smoothing

The baseline model I constructed at the outset used individual words (unigrams) as tokens. I will refer to this model as \( M_{Uni} \) in the rest of this report. Throughout this
exercise, I have used a multinomial model. This means that if a token $t$ occurs in a document $k$ times, then its log-likelihood is $k \times \log(p(t|C))$. I did not choose the Bernoulli model (where we ignore the counts altogether, and only distinguish the presence and absence of a word in a document) because counts are important in not-so-short articles. Five occurrences of “great movie” is a stronger indicator of a positive review than one occurrence.

I recall that a “different version” of the multinomial model was discussed in the tutorial section (My definitions correspond to the course slides). In this alternate definition, we would try to explicitly estimate the probability of a particular token $t$ occurring $k$ times in a document from the training data. I did not choose this model because firstly, you would need a lot of data to train such a model. Secondly, the parameters would be dependent on the length of the document (since a longer document is more likely to have multiple occurrences of a token than a shorter document), thus making them even more difficult to estimate.

### 3.1 Effect of smoothing parameter

I tried different values of the smoothing parameter, $\alpha$, to check its effect on accuracy. By and large, I found that the classification accuracy was quite insensitive to $\alpha$ in the range of $[0.25, 1.0]$. To understand why, we have to look at the role of the smoothing constant in the overall classification.

$$ \hat{p}(t|c) = \frac{T_{ct} + \alpha}{(\sum_{t' \in V} T_{ct'}) + B\alpha} $$

where $t$ is the token, $c$ is a class, $T_{ct}$ is the term count of token $t$ in class $c$ and $B$ is the vocabulary size.

The change in likelihood of $\hat{p}(t|c)$ for a change in $\alpha$ depends on the vocabulary size $B$. For our vocabulary size, this change in likelihood for $\alpha \in [0.25, 1.0]$, is just the right amount so as to not affect the classifier too much. This explains the insensitivity of the classifier to the $\alpha$ value in that range. For smaller values of $\alpha$, the performance actually deteriorates a bit. The reason being that the smoothing is not sufficient and thus biases the classifier against unseen words. The results are shown in the first row of Table 2.

### 3.2 Amount of training data

I also wanted to see how the accuracy of the classifier changed with increasing training data. For this experiment, I only performed $1-$fold validation. Table 1 shows that as expected, the performance improves with more training data. However, I was surprised to find that we could get almost 0.70 accuracy with only 100 training examples. Also, there is a slight performance dip from 500 to 800 and in general a slower rate of increase in accuracy with increase in the size of the training data.

<table>
<thead>
<tr>
<th>#Training Files</th>
<th>100</th>
<th>500</th>
<th>800</th>
<th>900</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>0.696</td>
<td>0.805</td>
<td>0.788</td>
<td>0.825</td>
</tr>
</tbody>
</table>

Table 1: Effect on accuracy with increasing training data

### 4 Richer structure: Playing with n-grams

The initial experiments suggest that considering words independently ignores some vital information. For instance, consider the two phrases “very bad” and “not bad”. While the former is strongly indicative of a negative review, the latter is mildly indicative of a positive review. Upon breaking the phrases down to their constituent words, this contextual information is completely lost. Therefore, in order to use this information, I considered the use of bigrams and trigrams — sliding windows of two and three words respectively.

The posterior class likelihood given the document using the bigram model (hereafter $M_{Bi}$) is as follows:

$$ p_{M_{Bi}}(c|d) \propto p(c) \prod_{1 \leq k \leq N_d-1} p(<t_k,t_{k+1}>|c) $$

where $<t_k,t_{k+1}>$ is the bigram composed of the two words $t_k$ and $t_{k+1}$. Similarly the trigram model’s ($M_{Tr}$) posterior likelihood is defined as:

$$ p_{M_{Tr}}(c|d) \propto p(c) \prod_{1 \leq k \leq N_d-2} p(<t_k,t_{k+1},t_{k+2}>|c) $$

One downside of using n-grams is that the vocabulary size increases exponentially in $n$. This leads to memory issues for large tasks. But for this exercise, I did not face any memory issues for either $M_{Bi}$ or $M_{Tr}$. However, the likelihood computation time for any document also increases with an increasing vocabulary. Thus, evaluation was slower.

The upside was a significant performance improvement. In $10-$fold cross validation using the same method as before, $M_{Bi}$ achieved an average accuracy of **0.850** (for $\alpha = 1$). The detailed results are shown in Table 2. Looking into the examples that $M_{Bi}$ classified correctly and $M_{Uni}$, I noticed phrases like “most pleasant” and “most charming” (in positive review 725). “most” is a neutral word without the adjective following it. Hence, $M_{Uni}$ loses the context of strong positivity here by decoupling the words, while $M_{Bi}$ retains it and that in turn helps it to classify the example correctly. Another frequently occurring bigram which helped in many correct classifications is “not funny” — in fact, it is a high-ranked feature (to be discussed later).

$M_{Tr}$ is theoretically better than $M_{Bi}$ in capturing context. However, $M_{Tr}$ needs a lot more training data and our limited training data is the most likely reason for $M_{Tr}$ to be actually slightly worse than $M_{Uni}$ as well.
<table>
<thead>
<tr>
<th>Model</th>
<th>α = 1</th>
<th>α = .5</th>
<th>α = .1</th>
<th>α = .05</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_{Uni}$</td>
<td>0.808</td>
<td>0.804</td>
<td>0.794</td>
<td>0.792</td>
</tr>
<tr>
<td>$M_{Bi}$</td>
<td>0.850</td>
<td>0.844</td>
<td>0.843</td>
<td>0.841</td>
</tr>
<tr>
<td>$M_{Tri}$</td>
<td>0.810</td>
<td>0.786</td>
<td>0.792</td>
<td>0.795</td>
</tr>
</tbody>
</table>

Table 2: Performance of the various n-gram models with varying $\alpha$

5 Mixing it up: Mixture model

Both the unigram model ($M_{Uni}$) and the bigram model ($M_{Bi}$) have their own strengths. Therefore, I wanted to see if I could create a mixture model of $M_{Uni}$ and $M_{Bi}$ which would in some way combine the goodness of the two models. The formulation of the mixture model was as follows:

$$p(c|d) = \beta p_{M_{Uni}}(c|d) + (1 - \beta)p_{M_{Bi}}(c|d)$$

where $0 \leq \beta \leq 1$. In order to find a good $\beta$, I performed a line search between 0 and 1 at increments of 0.1. My training set for each fold was 800 files. I used 100 files for validation (i.e. finding the optimal value of $\beta$) and the remaining 100 files were used for testing.

The average accuracy (over 10-fold cross validation) of this mixture model was 0.843. The average value of $\beta$ (the weight of the unigram model $M_{Uni}$) was 0.07. Thus, most of the weight was placed on $M_{Bi}$. Thus, it comes as no surprise that the performance was very similar to the bigram model (0.85). The small difference is most likely caused by the reduction in training set size (from 900 files to 800). It is unlikely to be a random effect since there aren’t any randomized components in either the training or the test pipeline.

6 Less is more: Feature selection

The number of unique tokens (e.g. distinct words in a corpus when dealing with unigrams) increases “almost” linearly with the corpus size. This is a direct consequence of Zipf’s law which states that the frequency of a word is inversely proportional to some power of its frequency rank (http://en.wikipedia.org/wiki/Zipf%27s_law). A large percentage of the tokens occur only a few times (about half the words occur only once!). These rarely occurring words generally do not add much to the accuracy of the classifier. Thus, it should be possible to build a reasonably accurate classifier by only considering a few “top-ranked” tokens. These tokens should have maximum discriminative power. I used mutual information as the metric to rank the tokens.

6.1 Mutual Information: Feature-ranking

The mutual information $I(U_i, U_C)$ of a token $t$ and a class $C$ is defined by:

$$I(U_i, U_C) = \sum_{U_i \in \{0,1\}} \sum_{U_C \in \{0,1\}} p(U_i, U_C) \log_2 \frac{p(U_i, U_C)}{p(U_i)p(U_C)}$$

Here, $U_i$ is an indicator variable which is 1 when the token occurs in a document and 0 otherwise. $U_C$ is also an indicator variable which is 1 when a document is in class $C$ and 0 otherwise. Thus, $p(U_i = 1|U_C = 1) = \#docsc_C(t)/\#docsc_C$. Here, $\#docsc_C(t)$ represents the number of documents in class $C$ containing the token $t$.

An important point to note here is that in a 2-class classifier, the mutual information $I(U_i, U_C)$ is equal to $I(U_i, U_{C_{\bar{C}}})$. Thus, the top-ranked tokens for the two classes are identical. They represent the set of tokens which are expected to be most useful in classifying the documents correctly.

6.2 Top-ranked Features

Here is a list of the top 15 unigrams and bigrams (see Table 3). Most of the unigrams are negative review words (e.g. “bad”, “worst”, “stupid”). There are a few positive review words too (e.g. “outstanding”, “wonderfully”, “memorable”) and some seemingly neutral words (“life”, “supposed”). The corpus indicated “life” to be strongly indicative of a positive review and “supposed” was deemed strongly negative.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Unigram</th>
<th>$I_{uni}$</th>
<th>Bigram</th>
<th>$I_{bi}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bad</td>
<td>.0568</td>
<td>the worst</td>
<td>.0315</td>
</tr>
<tr>
<td>2</td>
<td>worst</td>
<td>.0559</td>
<td>should have</td>
<td>.0275</td>
</tr>
<tr>
<td>3</td>
<td>stupid</td>
<td>.0452</td>
<td>waste of</td>
<td>.0255</td>
</tr>
<tr>
<td>4</td>
<td>boring</td>
<td>.0362</td>
<td>supposed to</td>
<td>.0228</td>
</tr>
<tr>
<td>5</td>
<td>wasted</td>
<td>.0344</td>
<td>the best</td>
<td>.0218</td>
</tr>
<tr>
<td>6</td>
<td>mess</td>
<td>.0333</td>
<td>he is</td>
<td>.0209</td>
</tr>
<tr>
<td>7</td>
<td>ridiculous</td>
<td>.0314</td>
<td>as the</td>
<td>.0205</td>
</tr>
<tr>
<td>8</td>
<td>outstanding</td>
<td>.0309</td>
<td>bad movie</td>
<td>.0175</td>
</tr>
<tr>
<td>9</td>
<td>life</td>
<td>.0298</td>
<td>this mess</td>
<td>.0171</td>
</tr>
<tr>
<td>10</td>
<td>waste</td>
<td>.0296</td>
<td>be funny</td>
<td>.0158</td>
</tr>
<tr>
<td>11</td>
<td>awful</td>
<td>.0284</td>
<td>absolutely no</td>
<td>.0158</td>
</tr>
<tr>
<td>12</td>
<td>wonderfully</td>
<td>.0259</td>
<td>portrayal of</td>
<td>.0149</td>
</tr>
<tr>
<td>13</td>
<td>supposed</td>
<td>.0243</td>
<td>at least</td>
<td>.0149</td>
</tr>
<tr>
<td>14</td>
<td>memorable</td>
<td>.0231</td>
<td>is also</td>
<td>.0148</td>
</tr>
<tr>
<td>15</td>
<td>subtle</td>
<td>.0227</td>
<td>is perfect</td>
<td>.0145</td>
</tr>
</tbody>
</table>

Table 3: Top-ranked unigrams and bigrams with corresponding mutual information

Top-ranked bigrams are less correlated to human intuition than the unigrams. Corpus artifacts result in “he is”, “as
the” and “at least”. Bigrams occurring more than twice in every review on average (equivalent to stopwords) were not considered as a token for ranking. Despite this pruning, some ambiguous bigrams still make their way to this list. Interesting notes about nouns: “War”, “Cameron” and “Tom Hanks” are generally part of good reviews. So are “niro” (Robert De Niro) and “obi-wan” (Star Wars character). Also “poker” and “italian”. Not so lucky are “fred-die”, (Hank?) “azaria” and the “wild west” — these are seemingly synonymous with bad reviews!

6.3 Accuracy

The accuracy of the classifier using 100, 200, 500 and all the features (i.e. the original classifier) is plotted in Figure 1. Two key observations:

- A small number of features can also be used to get decent accuracy. The accuracy however does not increase monotonically with the number of features (most likely due to the inclusion of noisy features). The original classifier with all the features performs the best for both $M_{Un}$ and $M_{Bi}$.

- The features of $M_{Un}$ are better than $M_{Bi}$. The reasons for this were discussed earlier and the numbers support the hypothesis.

The total number of features in $M_{Un}$ and $M_{Bi}$ was well over 150K and 250K respectively. $M_{Tr}$ was not included in the mixture model since its individual performance was not so promising.

The reason for this is quite obvious upon looking at the ranked features. A majority of the discriminating features are strong indicators of a good review. Hence, any positive review is likely to have a bunch of these and get correctly classified. On the other hand, a negative review will contain very few ranked features and hence is much more susceptible to a random choice and thereby, error.

7 Synopsis

- Best classifier: Simple Bigram Model (Average Accuracy = 0.850)
- Smoothing analysis
- Classifiers implemented:
  - Unigram model
  - Bigram model
  - Trigram model
  - Mixture of n-gram models
  - Feature-selected models

8 Conclusion

Looking back at the various experiments, the accuracy of the classifiers are of course limited by the Naive Bayes assumption of token independence. The sentence “... does not make this a good movie” cannot be correctly utilized unless we are using pentagrams or parsing sentences to analyze deeper meaning.

I discovered another not-so-obvious cause of failure by looking at the dataset. I was looking at different reviews which had been correctly classified by one model but misclassified by another to hunt for interesting snippets when I observed this pattern. Every review’s first half is dominated by a plot summary. This portion rarely reveals the positive or negative nature of the review. In fact, often a positive review about a dark film (or a cult film) will contain plenty of “negative” words in the plot summary. This actually confounds the classifier. In datasets like these, maybe assigning higher weights to the latter half of the review (or completely ignoring the first part) could lead to better results. In fact, from empirical observations, it seems like just looking at the last sentence of each review could give us a fantastic NB-style classifier! Sample final sentences (from examples misclassified by both $M_{Un}$ and $M_{Bi}$):

“by grabthar’s hammer , this is a hell of a movie .”
“the bottom line : chocolat is pure pleasure .”
“many critics have claimed that this movie will change the way action movies are made . . . . . i certainly hope so .”
“consequently , the audience has an equally difficult time trying to keep up .”
“this is an emotionless costume epic.”
“on the granger movie gauge of 1 to 10 , ’the watcher’ is an appallingly awful , amateurish 2 . the real torture is watching it .”

Wishlist

It would be great if some sample solution (optimized for computational efficiency) using different components of the libraries is posted. It is a pity that I could not fully use these wonderful libraries. I hope that with time as these libraries become more popular, the documentation will become more user-friendly.

Acknowledgements

Understanding Scala was made easier via discussions with Mobin Javed and Anupam Prakash.