COMPSCI 589 Homework 2 - Spring 2023

Due March 26, 2023, 11:55pm Eastern Time

Introduction

I am training a Naive Bayes algorithm capable of classifying movie reviews as positive or negative. My algorithm will be trained using the IMDB Large Movie Review dataset, developed by Maas et al. in 2011. Below, I first present a summarized description of how the Multinomial Naive Bayes algorithm works, discuss the structure of the dataset that I will be exploring, and, finally, present the experiments and analyses that I will conduct.

1 Multinomial Naive Bayes for Document Classification

A Multinomial Naive Bayes model is trained based on a set of training *documents*, each one belonging to a class. In this project, each document corresponds to one movie review made by a user and posted on IMDB. I will use a training set and a test set containing examples of positive movie reviews and negative reviews. Recall that in the Multinomial Naive Bayes model, each document is represented by a Bag-of-Words vector **m** composed of integer entries; each entry in this vector indicates the frequency of a word in the document (see Fig. 1 for an example).

m = (2,1,0,1,0,1,0)

	password	expired	send	review	conference	account	paper
doc1	**2**	1	0	1	0	1	0

Figure 1: Example of how a document is represented by the Multinomial Naive Bayes algorithm.

Assume a classification problem with c classes. Let y_i be the i-th class in the problem, where $1 \leq i \leq c$. Let **Doc** be a document, $len(Doc)$ be the number of unique words appearing in **Doc**, and w_k be the k-th unique word in the document, where $1 \leq k \leq len(Doc)$. Let $\Pr(y_i)$ be the prior probability of class y_i and $\Pr(w_k \,|\, y_i)$ be the probability that word w_k appears in documents of class y_i. Recall that when computing the latter probability, the Multinomial Naive Bayes model takes into account the *frequency* with which the word appears in documents of that class, and not just *whether* the word appears in documents of that class (as done, for example, by the Binomial Naive Bayes algorithm).

To classify a new document **Doc**, the algorithm computes, for each class y_i, the following probability:

$$\Pr(y_i \,|\, Doc) \;=\; \Pr(y_i) \prod_{k=1}^{len(Doc)} \Pr(w_k \,|\, y_i). \tag{1}$$

The algorithm then classifies **Doc** as belonging to the class that maximizes the probability in Eq. (1). To train the algorithm, one needs to estimate $\Pr(y_i)$, for each possible class, and $\Pr(w_k \,|\, y_i)$, for each

possible word and class, given a training set. The prior probability of a class can be estimated, using examples from the training set, as follows:

$$\Pr(y_i) = \frac{N(y_i)}{N}, \tag{2}$$

where N is the total number of documents available in the training set and $N(y_i)$ is the number of documents in the training set that belong to class y_i. Furthermore, the conditional probability of a word given a class can be estimated, using examples from the training set, as follows:

$$\Pr(w_k \,|\, y_i) = \frac{n(w_k, y_i)}{\sum_{s=1}^{|V|} n(w_s, y_i)}, \tag{3}$$

where $n(w_k, y_i)$ is the frequency (total number of occurrences) of word w_k in documents that belong to class y_i, V is the vocabulary containing all unique words that appear in all documents of the training set, and $|V|$ is the length of the vocabulary.

2 The IMDB Dataset of Movie Reviews

In this project, I will train a machine learning model capable of classifying movie reviews as positive or negative. my algorithm will be trained on the IMDB Large Movie Review dataset. This dataset is partitioned into 2 folders: 'train' and 'test', each of which contains 2 subfolders ('pos' and 'neg', for positive and negative reviews, respectively). In order to train the Multinomial Naive Bayes algorithm, each review/document first needs to be converted to a Bag-of-Words representation, similar to the one shown in Fig. 1. To do so, it is necessary to go through all the examples in the 'train' folder (including both positive and negative reviews) and construct the vocabulary V of all unique words that appear in the training set. Furthermore, each review that will be analyzed by my algorithm—either during the training process or when classifying new reviews—needs to be pre-processed: all words in the review should be converted to lowercase, stopwords (*i.e.*, common words such as "the" and "of") should be removed from reviews, etc.

The dataset has one fixed training set and one test set. Each of these sets contains 12,500 examples of positive reviews and 12,500 examples of negative reviews. This means that there is a total of 50,000 movie reviews available, 25,000 of which will be used to train the Multinomial Naive Bayes algorithm, and 25,000 of which will be used to test it.

Furthermore, the functions that load the movie reviews (documents) which will be used as training and test sets are: `load_training_set()` and `load_test_set()`, respectively. It can be found in the `utils.py` file. Both of them take as input two parameters: the percentage of positive and negative examples that should be randomly drawn from the training (or test) set, respectively. As an example, if I call `load_training_set(0.5, 0.3)`, the function will return a data structure containing approximately 50% of the existing positive training examples, and approximately 30% of the existing negative training examples. The function `load_test_set()` works similarly: calling `load_test_set(0.2, 0.1)` will return a data structure containing approximately 20% of the existing positive test examples and approximately 10% of the existing negative test examples. This capability is useful for two reasons: *(i)* it allows me to work with very small datasets, at first, while I am still debugging my code, which makes it easier and faster to identify and solve problems; and *(ii)* it

will allow me, later, to quantify the impact that unbalanced datasets (*i.e.*, datasets with significantly more examples of one of the classes) have on the performance of the model.

2.1 Example

Consider, for example, the file `train/pos/9007_10.txt`. This is one of the 12,500 examples of a positive review included as part of the training set. Its contents are:

```
This is the best movie I've seen since White and the best romantic comedy I've
seen since The Hairdresser's Husband.  An emotive, beautiful masterwork from
Kenneth Branagh and Helena Bonham Carter.  It is a tragedy this movie hasn't
gotten more recognition.
```

After pre-processing it (using the `preprocess_text()` function) to remove punctuation, stop-words, etc., it becomes:

```
best movie ive seen since white best romantic comedy ive seen since hairdressers
husband emotive beautiful masterwork kenneth branagh helena bonham carter tragedy
movie hasnt gotten recognition
```

Assume, for simplicity, that the training set contains only two movie reviews:

(1)

```
The movie itself was ok for the kids.  But I gotta tell ya that Scratch, the
little squirrel, was the funniest character I've ever seen.  He makes the movie
all by himself.  He's the reason I've just love this movie.  Congratulations
to the crew, it made me laugh out loud and always will!
```

and

(2)

```
What can I say?  An excellent end to an excellent series!  It never quite got
the exposure it deserved in Asia, but by far, the best cop show with the best
writing and the best cast on televison.  EVER! The end of a great era.  Sorry
to see you go...
```

In this case, the data structures returned by the function `load_training_set()` would look something like this:

```
[
  ['movie', 'ok', 'kids', 'gotta', 'tell',
  'ya', 'scratch', 'little', 'squirrel',
  'funniest', 'character', 'ive', 'ever',
  'seen', 'makes', 'movie', 'hes',
  'reason', 'ive', 'love', 'movie',
  'congradulations', 'crew', 'made',
  'laugh', 'loud', 'always'],
 ['say', 'excellent', 'end', 'excellent',
  'series', 'never', 'quite', 'got',
  'exposure', 'deserved', 'asia', 'far',
```

```
  'best', 'cop', 'show', 'best',
  'writing', 'best', 'cast', 'televison',
  'ever', 'end', 'great', 'era', 'sorry',
  'see', 'go']
]
```

That is, a vector where each element corresponds to one of the reviews. Each element of the returned vector, in particular, is a Bag-of-Words vector containing the words that appear in the corresponding movie review, after all pre-processing steps have been completed.

3 Analysis

I ran an experiment to evaluate how the performance of the Naive Bayes algorithm is affected depending on whether *(i)* I classify instances by computing the posterior probability, $\Pr(y_i \mid Doc)$, according to the standard equation (Eq. (1)); or *(ii)* I classify instances by performing the log-transformation trick and compare log-probabilities, $\log(\Pr(y_i \mid Doc))$, instead of probabilities. Estimating the posterior probability using Eq. (1) might cause numerical issues/instability since it requires computing the product of (possibly) hundreds of small terms—which makes this probability estimate rapidly approach zero. To tackle this problem, instead of using $\Pr(y_1 \mid Doc)$ and $\Pr(y_2 \mid Doc)$ to compare, I used corresponding log-probabilities: $\log(\Pr(y_1 \mid Doc))$ and $\log(\Pr(y_2 \mid Doc))$. Taking the logarithm of such probabilities transforms the product of hundreds of terms into the sum of hundreds of terms—which avoids numerical issues. Importantly, it does not change which class is more likely according to the trained model. When classifying a new instance, the log-probabilities that should be compared, for each class y_i, are as follows:

$$\log(\Pr(y_i \mid Doc)) = \log\left(\Pr(y_i) \prod_{k=1}^{len(Doc)} \Pr(w_k \mid y_i)\right) \tag{4}$$

$$= \log(\Pr(y_i)) + \sum_{k=1}^{len(Doc)} \log(\Pr(w_k \mid y_i)). \tag{5}$$

In this experiment, I will use 20% of the training set and 20% of the test set; *i.e.*, call the dataset-loading functions by passing 0.2 as their parameters. First, I performed the classification of the instances in the test set by comparing posterior probabilities, $\Pr(y_i \mid Doc)$, according to Eq. (1), for both classes. Then, I reported *(i)* the accuracy of the model; *(ii)* its precision; *(iii)* its recall; and *(iv)* the confusion matrix resulting from this experiment. Then I repeated the same experiment above but classify the instances in the test set by comparing log-probabilities, $\log(\Pr(y_i \mid Doc))$, according to Eq. (5), for both classes. I have discussed whether classifying instances by computing log-probabilities, instead of probabilities, affects the model's performance. I have also mentioned which affects the performance more strongly: the model's accuracy, precision, or recall.

Using Posterior Probabilities:
(i) Accuracy: 0.618
(ii) Precision: 0.759
(iii) Recall: 0.363
(iv) Confusion Matrix: [[931, 1633], [296, 2191]]

Using Log Probabilities:
(i) Accuracy: 0.725
(ii) Precision: 0.739
(iii) Recall: 0.709
(iv) Confusion Matrix: [[1818, 746], [640, 1847]]

Note:

- If the probability of the document being a positive review is strictly greater than the probability of it being a negative review, the document is declared as positive review
- Used a very small value of α=0.00001

Impact of model's performance:

Recall:

- Recall value has increased drastically from 0.36 to 0.7 when using log method.
- This is because the false negatives has reduced significantly from 1633 to 746 and the true positives have also amplified from 931 to 1818 when compared to posterior method.
- This overall has increased the recall when using log probability, being the most impacted performance parameter.

Accuracy:

- Accuracy has increased from 0.62 to 0.76 when using log probability.
- This is because, in posterior probability, I multiply large number of small probabilities resulting in values that are so small that it gets represented as 0 due to numerical underflow.
- In case of log probability, this is avoided because the multiplication operation is replaced with addition which avoids the numerical underflow.
- This has resulted in majority of the test cases being assigned the right class, that is both true positives and true negatives has increased, thereby increasing the overall accuracy.

Precision:

- Precision value has very slightly decreased in case of log method when compared to posterior method.
- In the posterior method, the number of classes correctly identified as positive itself is very low. Within that, the precision of the classes being true positive is high simply because the denominator is small.
- In the log method, the true positive is very close to the actual positive test cases in comparison to the posterior method. But there is a minor increase in false positives too leading to a slight reduction in the precision.

An issue with the original Naive Bayes formulation is that if a test instance contains a word that is not present in the vocabulary identified during training, then $\Pr(word|label) = 0$. To mitigate this issue, one solution is to employ Laplace Smoothing. To do so, I replaced the standard way of estimating the probability of a word w_k, given a class y_i, with the following equation:

$$\Pr(w_k \mid y_i) = \frac{n(w_k, y_i) + 1}{\sum_{s=1}^{|V|} n(w_s, y_i) + |V|}. \tag{6}$$

More generally, Laplace Smoothing can be performed according to a parametric equation, where instead of adding 1 to the numerator, I adjusted the probability of a word belonging to a class by adding a user-defined parameter α to the numerator, as follows:

$$\Pr(w_k \mid y_i) = \frac{n(w_k, y_i) + \alpha}{\sum_{s=1}^{|V|} n(w_s, y_i) + \alpha|V|}. \tag{7}$$

Intuitively, setting $\alpha = 0$ results in the standard formulation of Naive Bayes—which does not tackle the problem of words that do not appear in the training set. Suppose, alternatively, that we set $\alpha = 4$. This is equivalent to adding 4 "fake" occurrences of that word to the training set, in order to avoid the zero-frequency problem. Using $\alpha = 1000$, on the other hand, is equivalent to pretending we have seen that word 1000 times in the training set—even though we may have seen it, say, only 8 times. Although this solves the problem of zero-frequency words, it also strongly biases the model to "believe" that that word appears much more frequently than it actually does; and this could make the predictions made by the system less accurate. For these reasons, although it is important/necessary to perform Laplace Smoothing, we have to carefully pick the value of α that works best for our dataset. Using $\alpha = 1$ is common, but other values might result in better performance, depending on the dataset being analyzed.

In this experiment, I use 20% of the training set and 20% of the test set; *i.e.*, call the dataset-loading functions by passing 0.2 as their parameters. I first reported the confusion matrix, precision, recall, and accuracy of my classifier (when evaluated on the test set) when using $\alpha = 1$. Now, I vary the value of α from 0.0001 to 1000, by multiplying α with 10 each time. For each value, I recorded the accuracy of the resulting model when evaluated on the test set. Then, I created a plot of the model's accuracy on the test set (shown on the y-axis) as a function of the value of α (shown on the x-axis). The x-axis represents α values in a log scale. Finally I analyzed this graph and discussed why the accuracy suffers when α is too high or too low.
For $\alpha = 1$, below are the performance parameters.
(i) Confusion Matrix: [[1935, 600], [286, 2250]]
(ii) Accuracy: 0.825
(iii) Precision: 0.871
(iv) Recall: 0.763

Low values of α:

- For low values of α the model is over-fitting on the training data.
- For vocabulary that occurs in test data and not in training data I am using Laplace smoothing. Even if the word has occurred more number of times in the test data, it is not considered and is given a very low alpha value.

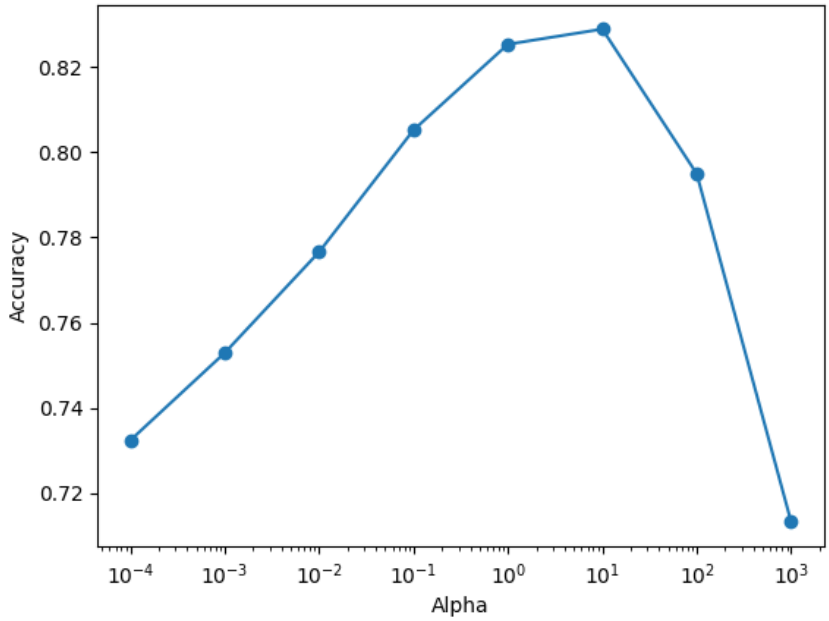

Figure 2: Accuracy for different values of Alpha

- For this data set, the lower values of α is negligible compared to the frequency of other words in the training data.
- This has resulted in low accuracy values for lower values of α

High values of α:

- For high values of α the model is under-fitting on the training data.
- For vocabulary that occurs in test data and not in training data, I am adding a very high number of "fake" occurrences for those vocabulary.
- Because of these fake occurrences, the model becomes strongly biased towards these words even though it occurred only once or twice.
- This has resulted in low accuracy values for higher values of α

Next I investigated the impact of the training set size on the performance of the model. The classification of new instances, here, is done by comparing the posterior log-probabilities, $\log(\Pr(y_i \mid Doc))$, according to Eq. (5), for both classes. I used the value of α that resulted in the highest accuracy according to my experiments. I use 100% of the training set and 100% of the test set; I reported *(i)* the accuracy of my model; *(ii)* its precision; *(iii)* its recall; and *(iv)* the confusion matrix resulting from this experiment. With $\alpha = 10$ and 100% training data,
(i) Confusion Matrix: [[9992, 2508], [1430, 11070]]
(ii) Accuracy: 0.842
(iii) Precision: 0.875
(iv) Recall: 0.799

Now I repeated the experiment above but used only 50% of the training instances. *The entire test set was used.* I have reported the same quantities as in the previous experiment. I have also discussed whether using such a smaller training set had any impact on the performance of my learned model. I have analysed the confusion matrices (of this experiment and the previous one) and discussed whether one particular class was more affected by changing the size of the training set.
Number of positive training instances: 6237
Number of negative training instances: 6288
With $\alpha = 10$ and 50% training data,
(i) Confusion Matrix: [[9968, 2532], [1426, 11074]]
(ii) Accuracy: 0.841
(iii) Precision: 0.874
(iv) Recall: 0.797

The use of a smaller training set (50%) did not have much impact on the performance of the learned model. The performance is quite similar when using both 100% and 50% of the training set. There does seem to be a very minor variation of 0.1% in accuracy and precision and a 0.2% variation in recall.

Analyzing the confusion matrix, it can be seen that the positive class has been affected by the change in the size of the training set. The true positives have decreased from 9992 to 9968 increasing the false negatives from 2508 to 2532. This could be due to the slightly higher number of negative training instances (55) in comparison to the positive training data.

Finally, I will study how the performance of the learned model is affected by training it using an unbalanced dataset (*i.e.*, a dataset with significantly more examples of one of the classes). The classification of new instances, here, is done by comparing the posterior log-probabilities, $\log(\Pr(y_i \mid Doc))$, according to Eq. (5), for both classes. I used the value of α that resulted in the highest accuracy according to my experiments. I will now conduct an experiment where I use only 10% of the available *positive* training instances and that uses 50% of the available *negative* training instances. *The entire test set is used.* I have shown the confusion matrix of my trained model, as well as its accuracy, precision, and recall.

Number of positive training instances: 1287
Number of negative training instances: 6252
With $\alpha = 10$ and 10% positive training instances and 50% negative training instances,
(i) Confusion Matrix: [[2, 12498], [0, 12500]]
(ii) Accuracy: 0.50008
(iii) Precision: 1.0
(iv) Recall: 0.00016

When comparing with the confusion matrix of question 4, the true positives has dropped to 2 and the rest of 12498 positive reviews is falsely classified as negative. For negative class, all the instances of negative test data are correctly classified and hence false positives is 0 and true negatives is 12500. Since the model is trained on large values of negative reviews, the model is biased to predict the new test data also as negative reviews.

Accuracy: For imbalanced dataset, accuracy has dropped to 0.5 from 0.841 in question 4. This is because the model is biased towards negative dataset and all the negatives are classified correctly but almost all of the positive data are incorrectly classified.

Precision: The precision is high when compared to the previous case, because none of the negative class is falsely classified as positive. So tp/(tp+fp) is 100%.

Recall: The recall drops to nearly zero when compared to balanced case, because out of 12500 positive reviews, 12498 reviews are falsely categorized as negatives. So tp/(tp+fn) is extremely low.