COMPSCI 589 Homework 2 - Spring 2024

Due March 12, 2024, 11:55pm Eastern Time

1 Instructions

- This homework assignment consists of a programming portion. While you may discuss problems with your peers, you must answer the questions on your own and implement all solutions independently. In your submission, do explicitly list all students with whom you discussed this assignment.
- We strongly recommend that you use LaTeX to prepare your submission. The assignment should be submitted on Gradescope as a PDF with marked answers via the Gradescope interface. The source code should be submitted via the Gradescope programming assignment as a .zip file. Include with your source code instructions for how to run your code.
- We strongly encourage you to use Python 3 for your homework code. You may use other languages. In either case, you *must* provide us with clear instructions on how to run your code and reproduce your experiments.
- You may *not* use any machine learning-specific libraries in your code, e.g., TensorFlow, PyTorch, or any machine learning algorithms implemented in scikit-learn (though you may use other functions provided by this library, such as one that splits a dataset into training and testing sets). You may use libraries like numpy and matplotlib. If you are not certain whether a specific library is allowed, do ask us.
- All submissions will be checked for plagiarism using two independent plagiarism-detection tools. Renaming variable or function names, moving code within a file, etc., are all strategies that *do not* fool the plagiarism-detection tools we use. If you get caught, all penalties mentioned in the syllabus *will* be applied—which may include directly failing the course with a letter grade of "F".

  → Before starting this homework, please review this course's policies on plagiarism by reading Section 14 of the syllabus.

- The tex file for this homework (which you can use if you decide to write your solution in LaTeX), as well as the datasets and starting source code, can be found here.
- The automated system will not accept assignments after 11:55 pm on March 12.

Programming Section (100 Points Total)

In this homework, you will be implementing the Multinomial Naive Bayes algorithm. **Notice that you may not use existing machine learning code for this problem: you must implement the learning algorithms entirely on your own and from scratch.** You will, in particular, train a Naive Bayes algorithm capable of classifying movie reviews as positive or negative. Your algorithm will be trained using the IMDB Large Movie Review dataset, developed by Maas et al. in 2011. Below, we first present a summarized description of how the Multinomial Naive Bayes algorithm works, discuss the structure of the dataset that you will be exploring, and, finally, present the experiments and analyses that you should conduct.

You can download the dataset and starting source code here.

2 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 assignment, each document corresponds to one movie review made by a user and posted on IMDB. You will be given 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 $\Pr(w_k \mid y_i)$ 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:

$$\Pr(y_i \mid Doc) = \Pr(y_i) \prod_{k=1}^{len(Doc)} \Pr(w_k \mid y_i). \quad (1)$$

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 $\Pr(w_k \mid y_i)$, 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:

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

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.

3 The IMDB Dataset of Movie Reviews

In this assignment, you will train a machine learning model capable of classifying movie reviews as positive or negative. Your algorithm will be trained on the IMDB Large Movie Review dataset. This dataset (which is provided to you here) 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 your 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. We provide you with a function that performs such pre-processing tasks: `preprocess_text()`, in the `utils.py` file.

Since the dataset to be used in this assignment is relatively large, and you will have to conduct many experiments, we will not ask you to repeatedly manually split the dataset into training and test sets, conduct experiments many times, and evaluate the average performance. Instead, we will provide you with one pre-computed split of the dataset; *i.e.*, with 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, we also provide you with functions to load the movie reviews (documents) that will be used as training and test sets: `load_training_set()` and `load_test_set()`, respectively. These functions 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 you 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 you to work with very small datasets, at first, while you are still debugging your code, which makes it easier and faster to identify and solve problems;

and *(ii)* it will allow you, 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. You can find an example of how to use the above-mentioned functions in the file `run.py`, included as part of this assignment.

3.1 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.  Congradulations
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.

4 Questions

Hint: while you are still implementing and testing/debugging your solution, we recommend that you load just a small part of the training and test sets. If you call the `load_training_set` and `load_test_set` functions passing as parameters, say, 0.0004, that will return a training set containing approximately 0.04% of the training and test sets. This corresponds to approximately 10 training instances and approximately 10 test instances. This should make it much easier and faster to debug your code.

Q.1 (18 Points) You will first run an experiment to evaluate how the performance of the Naive Bayes algorithm is affected depending on whether *(i)* you classify instances by computing the posterior probability, $\Pr(y_i \mid Doc)$, according to the standard equation (Eq. (1)); or *(ii)* you classify instances by performing the log-transformation trick discussed in class and compare log-probabilities, $\log(\Pr(y_i \mid Doc))$, instead of probabilities. Recall, in particular, that 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, we often compare not $\Pr(y_1 \mid Doc)$ and $\Pr(y_2 \mid Doc)$, but the 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, you should 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, perform 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, report *(i)* the accuracy of your model; *(ii)* its precision; *(iii)* its recall; and *(iv)* the confusion matrix resulting from this experiment. Now repeat 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. Report the same quantities as before. Discuss whether classifying instances by computing log-probabilities, instead of probabilities, affects the model's performance. Assuming that this transformation does have an impact on performance, does it affect more strongly the model's accuracy, precision, or recall? Why do you think that is the case?

Q.2 (18 Points) 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, as discussed in class, we replace 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, we adjust 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, you should 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. You should first report the confusion matrix, precision, recall, and accuracy of your classifier (when evaluated on the test set) when using $\alpha = 1$. Now, vary the value of α from 0.0001 to 1000, by multiplying α with 10 each time. That is, try values of α equal to 0.0001, 0.001, 0.01, 0.1, 1.0, 100, and 1000. For each value, record the accuracy of the resulting model when evaluated on the test set. Then, create 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 should represent α values and use a log scale. Analyze this graph and discuss why do you think the accuracy suffers when α is too high or too low.

Q.3 (18 Points) Now you will investigate the impact of the training set size on the performance of the model. The classification of new instances, here, should be done by comparing the posterior log-probabilities, $\log(\Pr(y_i \mid Doc))$, according to Eq. (5), for both classes. You should use the value of α that resulted in the highest accuracy according to your experiments in the previous question. In this question, you should use 100% of the training set and 100% of the test set; *i.e.*, call the dataset-loading functions by passing 1.0 as their parameters. Then, report *(i)* the accuracy of your model; *(ii)* its

precision; *(iii)* its recall; and *(iv)* the confusion matrix resulting from this experiment.

Q.4 (18 Points) Now repeat the experiment above but use only 50% of the training instances; that is, load the training set by calling `load_training_set(0.5, 0.5)`. *The entire test set should be used.* Report the same quantities as in the previous question. Discuss whether using such a smaller training set had any impact on the performance of your learned model. Analyze the confusion matrices (of this question and the previous one) and discuss whether one particular class was more affected by changing the size of the training set.

Q.5 (10 Points) In this application (i.e., accurately classifying movie reviews), would you say that it is more important to have high accuracy, high precision, or high recall? Justify your opinion.

Q.6 (18 Points) Finally, you 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, should be done by comparing the posterior log-probabilities, $\log(\Pr(y_i \mid Doc))$, according to Eq. (5), for both classes. You should use the value of α that resulted in the highest accuracy according to your experiments in the previous questions. You will now conduct an experiment where you use only 10% of the available *positive* training instances and that uses 50% of the available *negative* training instances. That is, use `load_training_set(0.1, 0.5)`. *The entire test set should be used.* Show the confusion matrix of your trained model, as well as its accuracy, precision, and recall. Compare this model's performance to the performance (according to these same metrics) of the model trained in question Q.4—that is, a model that was trained under a *balanced* dataset. Discuss how training under an unbalanced dataset affected each of these performance metrics.