Homework 3: Naive Bayes Classification

CS 585, UMass Amherst, Fall 2016

due Oct 7^{th} 11:55pm

Overview

In this assignment you will build a Naive Bayes classifier that can classify movie reviews as either positive or negative. This assignment also asks you to evaluate and analyze your system. The goal is for you to begin to understand the Naive Bayes model, its strengths and weaknesses, how its parameters affect its accuracy and how to use the model to do some exploratory data analysis.

Dataset

Unlike the previous two homework, you'll be working with the **Full** IMDB Large Movie Review Dataset (You only work on a small subset of movie reviews in the hw1 and hw2). A copy of the dataset is available on the website.

The dataset includes 25,000 movie reviews for training and 25,000 movie reviews for testing. The root directory of the dataset contains a `train` directory and a `test` directory. Each of these directories has a `pos` and a `neg` directory containing positive and negative movie reviews respectively. Make sure that when training your model you use the files in the `train` directory; make sure that when testing your model you use the files in the `test` directory.

Running and Testing Code

We've partially implemented `nb.py` for you. Fill in the missing code as directed by this assignment. The end of the file contains a runnable main function (Notice that we don't use `jupyter notebook` in this assignment, so please type `python nb.py` in the command line to run the code). The code calls `produce_hw1_results()` which you can continue to implement to produce your results. Feel free to implement whatever other helper functions you like.

By default it is set to only use 20 documents: see the `num_docs` parameter for the training function. This is to make it easy to very rapidly test code. For all results you report in your solutions, always use the full dataset.

Deliverables and Due Date

You should submit a zipped directory named hw1_YOUR-USERNAME that contains:

- a document containing your responses to all of the questions in this assignment sheet (preferably in PDF format). Make sure to include any and all plots. If you have to write any of the answers by hand, please scan them and include them in your write up.

- any code you wrote for this assignment. This should at least include your completed `nb.py` file and may also include a completed `produce_hw1_results()` function.

Your work must be submitted via moodle no later than midnight on **Friday, Oct. 7th 11:55pm**. Our course's collaboration policy is specified on the website.

1 Bag-of-words

Before building the model, you'll need to get the text into a representation that the model can handle. Recall from lecture that the Naive Bayes model makes use of a *bag-of-words representation*. Naive Bayes is order-independent in that it doesn't care about the order of the words in the documents it classifies; it only keeps track of the number of each word type it encounters.

In the nb.py, we have provided `tokenize_doc` function to build a mapping (dictionary) of each lower-cased token to its frequency in the document. Please use our Naive tokenization implementation in `tokenize_doc` to answer the problems in this homework. At the end of the homework, you would have chance to use the better tokenizer you developed at the homework 1, and see whether it could actually improve the classifier performance.

1. **(5 pts)** Implement the `update_model` function. Before you start, make sure to read the function comments so you know what to update. Also review the `NaiveBayes` class variables to get a sense of which statistics are important to keep track of. Run the `train_model` function. What is the size of the vocabulary used in the training documents? You'll need to provide the path to the dataset you downloaded to run the code.

2. **(2.5 pts)** Let's begin to explore the count statistics stored by the `update_model` function. Use the provided `top_n` function to find the top 10 most common words in the positive class and top 10 most common words in the negative class. Will the top 10 words of the positive/negative classes help discriminate between the two classes? Do you imagine that processing other English text will result in a similar phenomenon?

2 Word Probabilities and Pseudocounts

The Naive Bayes model assumes that all features are conditionally independent given the class label. For our purposes, this means that the probability of seeing a particular word in a document with class label y is independent of the rest of the words in that document.

1. **(5 pts)** Implement the `p_word_given_label` function. This function calculates $P(w|y)$ (i.e., the probability of seeing word w in a document given the label of that document is y).

2. **(5 pts)** Use your function to compute the probability of seeing the word "fantastic" given each sentiment label. Repeat the computation for the word "boring." Which word has a higher probability given the positive class? Which word has a higher probability given the negative class? Is this what you would expect?

3. **(2.5 pts)** What happens if you try to compute the probability of a word that exists in the positive training data but not in the negative training data (and vice versa)? Explain what is going wrong.

4. **(5 pts)** We can address this issue with *psuedocounts*. A psuedocount is a fixed amount added to the count of each word stored in our model. Psuedocounts are used to help smooth calculations involving words for which there is little data. Implement `p_word_given_label_and_psuedocount`. **Hint:** look at the slides on 9/20 (slide 20).

3 Prior and Likelihood

As noted before, the Naive Bayes model assumes that all words in a document are independent of one another given the document's label. Because of this we can write the *likelihood* of a document as:

$$P(w_{d1}, \cdots, w_{dn}|y_d) = \prod_{i=1}^{n} P(w_{di}|y_d)$$

where w_{di} is the i^{th} word in document d and y_d is the label of document d.

However, if a document has a lot of words, the likelihood will become extremely small and we'll encounter numerical underflow. Underflow is a common problem when dealing with probabilistic models; if you are unfamiliar with it, you can get a brief overview on Wikipedia: `https://en.wikipedia.org/wiki/Arithmetic_underflow`. To deal with underflow, a common transformation is to work in log-space.

1. **(5 pts)** Derive the log of the likelihood function above.
2. **(5 pts)** Implement the `log_likelihood` function. **Hint:** it should make calls to the `p_word_given_label_and_psuedocount` function.
3. **(2.5 pts)** Implement the `log_prior` function. This function takes a class label and returns the log of the fraction of the training documents that are of that label.

4 Normalization and the Decision Rule

Naive Bayes is a model that tells us how to compute the posterior probability of a document being of some label (i.e., $P(y_d|\mathbf{w_d})$). Specifically, we do so using bayes rule:

$$P(y_d|\mathbf{w_d}) = \frac{P(y_d)P(\mathbf{w_d}|y_d)}{P(\mathbf{w_d})}$$

In the previous section you implemented functions to compute both the log prior ($\log[P(y_d)]$) and the log likelihood ($\log[P(\mathbf{w_d}|y_d)]$). Now, all your missing is the *normalizer* ($P(\mathbf{w_d})$).

1. **(5 pts)** Derive the normalizer.
2. **(5 pts)** Derive the log of the posterior probability by taking the log of the equation above.
3. **(5 pts)** One way to classify a document is to compute the *unnormalized log posterior* for both labels and take the argmax (i.e., the label that yields the higher unnormalized log posterior). The unnormalized log posterior is the sum of the log prior and the log likelihood of the document. Why don't we need to compute the log normalizer here?

4. **(2.5 pts)** Implement the `unnormalized_log_posterior` function.

5. **(5 pts)** Implement the `classify` function. The classify function should use the unnormalized log posteriors but should not compute the normalizer.

5 Evaluation

After training our model and implementing the `classify` function we'd like to evaluate its accuracy.

1. **(12.5 pts)** Implement the `evaluate_classifier_accuracy` function. This function should classify all of the instances in the test set and report the fraction of instances that are classified correctly. Report your classifier's accuracy (with psuedocount parameter 1.0).

2. **(7.5 pts)** Experiment with the effect of varying the psuedocount parameter on classifier accuracy. Plot classifier accuracy as a function of the psuedocount parameter. We have provided you with some sample code (the function `plot_psuedocount_vs_accuracy`) to help get you started with plotting. You may want/need to modify this function.

3. **(7.5 pts)** Find a review that your classifier got wrong. Why do you think your system misclassified this example? What improvements could you make that may help your system classify this example correctly?

6 Exploratory Analysis

Our trained model can be queried to do exploratory data analysis. We saw that the top 10 most common words for each class were not very discriminative. Often times, a more descriminative statistic is a word's likelihood ratio. A word's likelihood ratio is defined as

$$LR(w) = \frac{P(w|y = \text{pos})}{P(w|y = \text{neg})}$$

A word with $LR = 5$ is five times more likely to appear in a positive review than it is in a negative review; a word with $LR = 0.33$ is one third as likely to appear in a positive review than a negative review.

1. **(2.5 pts)** What is the range of the LR function?

2. **(2.5 pts)** Implement the `likelihod_ratio` function. This function takes a word and computes the likelihood ratio as defined above.

3. **(2.5 pts)** What are $LR(\text{“fantastic”})$ and $LR(\text{“boring”})$? Compare these to the likelihood ratio of some of the words in the top 10 lists generated above. For example, compare them to $LR(\text{“the”})$ and $LR(\text{“to”})$.

4. **(5 pts)** Explain how the word LRs are related to the Naive Bayes classifier model. If a word has LR=1, does that mean the word is or is not important for the NB classifier? If a word has LR very far from 1 (for example, LR=0.01, or LR=100) does that mean the word is or is not important for the classifier? What does an LR=0.01 word indicate, as compared to a LR=100 word, for the operation of the classifier? Explain.

7 Bonus

1. **(up to 15 pts)** Test different preprocessing steps to find a better bag-of-word representation for these movie reviews. You can use the code you wrote in homework 1 or external library such as `nltk` to perform tokenization, text normalization (e.g., lower-casing or even stemming), word filtering, etc.

   Roughly speaking, the larger performance improvement, the more extra credit. We will also give points for the effort in the evaluation and analysis process. For example, you can split the training data into training and validation set to prevent overfitting, and report results from trying different versions of features. You can also provide some qualitative examples you found in the dataset to support your choices on preprocessing steps. Whatever you choose to try, make sure to describe your method and the reasons that you hypothesize for why the method works.

   Finally, please don't modify the original `tokenize_doc` function while implement your improvements—create your own function(s) for the preprocessing steps. Please make sure that your submitted code can still generate the results based on the original `tokenize_doc`.

2. **(4 pts)** In Naive Bayes classifier, the features are word probabilities. That means you can't hand engineer specific features. On the other hand, log-linear classifiers (e.g., logistic regression) can use more complex, hand-engineered features. Describe 2 features you guess that would help classify movie reviews, and provide examples in the IMDB dataset to explain why the features can correct the mistakes made by your Naive Bayes classifier.

Often times we care about *multi-class classification* rather than *binary classification*.

3. **(2 pts)** How would the count statistics that we are storing change if the model were modified to support multi-class classification?

4. **(2 pts)** How would the normalizer change?

5. **(2 pts)** What would be the new decision rule (i.e., how would the classify function change)?