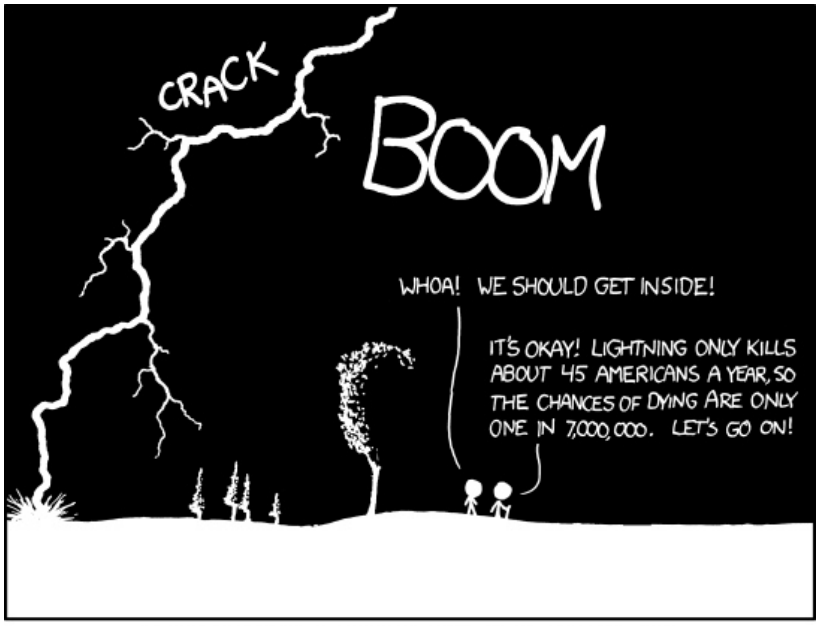
CS159 - Assignment 7 (last one!!)

Due: Wednesday, November 13 at 11:59pm



`http://xkcd.com/795/`

For our last (yay!) assignment, we will implement a multinomial Naive Bayes Classifier. You may, and I would encourage you to, work in pairs on this assignment if you'd like. As always, read through the **entire** document before starting.

Starter

The goal of this assignment is to build a multinomial NB classifier that can determine the sentiment of reviews, in this case, just positive and negative reviews.

As usual, go to the following link to get a copy of the starter project:

`https://classroom.github.com/a/b2E_F2tx`

Data

In the starter, there are two datasets to play with:

- **movies.data**: This data set consists of approximately 14K reviews collected from `www.rateitall.com`. The reviews come from a variety of domains including movies, music, books, and politicians. The actual data has ratings from 1 to 5, but I have cleaned these up and left only 1s and 5s, denoted those with a 5 score as "positive" and those with a 1 as "negative", resulting in a binary sentiment prediction task.
- **simple.data**: I've also provided a hand-made, smaller example to play with. You can use this for testing your algorithm and I'll also be asking you to submit some results on this data set.

Both data sets have one example per line of the form:

```
positive/negative[tab]text
```

The Model

The Multinomial Naive Bayes classifier models each class with a multinomial, which is a probability distribution over all of the words in the vocabulary (see the note below about vocabulary). Specifically, for each class you need to learn two things:

– $p(label)$: how likely this label is over the training data (note these will sum up to one over all labels).

– $p(w_j|label)$: how likely word w_j is to occur in a given label/class. Note that this is a probability distribution over all words in the vocabulary. For notational convenience, we often will write θ_j, that is $\theta_j = p(w_j|label)$

For this assignment, we will be learning two probability distributions, one for the positive class and one for the negative.

Implementation Details

- **Input**

In your `code` directory, create a script called `nb-classify.sh` that takes *three* parameters:

```
nb-classify.sh <training_data> <test_sentences> <lambda>
```

where `<training_data>` is a file formatted as described above with label and text, one per line, `<test_sentences>` is a normal text file with one example/text per line (without a label) and `<lambda>` is the lambda value to be used for smoothing.

- **Output**

  The program should train a multinomial NB model on the training data using the λ value for smoothing and then print one line for each line in the `<test_sentences>` file with the classification and the log-probability of that prediction:

  ```
  positive/negative[tab]log probability
  ```

  (see below for why the log of the probability and not the actual probability)

  Note: in the real world you would likely separate out the training and testing phases. During the training phase, you would generate the counts and then store these (or the actual θs to a file) and then read this representation (rather than retraining) during classification. For our purposes, though, we'll keep things simple and train the model each time.

- **Preprocessing**

  The data sets we'll be playing with have already been tokenized and lowercased: *just split the words based up on whitespace*. This will include some funny "words" like periods, etc., but that's fine; it will keep the answers consistent.

- **Underflow**

  When dealing with probabilities, you have to be careful about underflow. Like we've done before, the most common solution is to do everything in log-space and then sum the log-probs rather than multiplying probabilities. In our case, our general probability calculation per label is:

$$p(x_1, x_2, ..., x_m, label) = p(label) \prod_{j=1}^{m} \theta_j^{x_j}$$

  where x_j is the number of times word j occurred in the text and $\theta_j = p(w_j|label)$ (see notes about the model above). To avoid overflow, you should calculate the log-prob:

$$\log_{10} p(x_1, x_2, ..., x_m, label) = \log_{10} p(label) + \sum_{j=1}^{m} x_j \log_{10} \theta_j$$

- **Efficiency**

  Your program should be efficient both during training and classification. During training, you should only make ideally one (or two in the worst case) passes over the data, aggregating counts as you go. As always, you should pick an efficient data structure for this. (Hashtables are our new best friend in this class!)

  During classification, you should only have to iterate over the words that are in the particular example you're trying to classify, not all possible words you saw during training.

- **Probability answers**

  Technically speaking, the probability of a particular example is:

$$p(x_1, x_2, ..., x_m, label) = p(label)\frac{n!}{\sum_{j=1}^{m} x_j!}\prod_{j=1}^{m}\theta_j^{x_j}$$

  where $n = \sum_{j=1}^{m} x_j$. However, to keep things simple (and to avoid extra computation) what you should be outputing along with the classification prediction is just the log-prob as described above *without* the count/factorial component.

- **Ties**

  If you happen to have a tie between positive and negative, choose positive.

- **Vocabulary**

  The number of features in the model (m) will depend on the number of unique words that you see during training. Smoothing will handle cases where you see a word in training in one label, but not in the other (e.g. a word that occurs in a positive review, but not in a negative review). However, during testing, you might also encounter words that you've never seen before (in any of your training examples). There are many ways to handle this, but the easiest is just to ignore these words. This is also intuitive, since if you've never seen a word during training, it shouldn't bias your model towards either of the labels.

Writeup

In addition to your script and the associated code, you should also submit a short writeup that has your answers to the questions below. You don't need to (and shouldn't) submit any extra code that you write to get these answers.

Put your names at the top of the writeup file.

On `simple.data`:

1. What is the output from your system trained on `simple.data` on the following two sentences with $\lambda = 0$:

   ```
   i loved it
   i thought i hated it but loved it
   ```

   Include 1-2 sentences stating whether or not these seem reasonable.

2. Print out all of the probabilities for your model when you train on `simple.data` with $\lambda = 0$, specifically:

   - $p(negative)$ and $p(positive)$

- $p(*|positive)$ and $p(*|negative)$ – these are our multinomial distributions for each label, which shoulbe be distributions over all words in the vocabulary

On `movies.data`:

3. What are the top 10 most predictive features for each label (positive and negative)? To determine this, calculate:

$$p(w_i|positive)/p(w_i|negative)$$

using $\lambda = 0$ and for all *features that had occurrences in both labels*, i.e. have non-zero probabilities with $\lambda = 0$. The largest values will be those most predictive for the positive label and the smallest values those most predictive for the negative label. Include 1-2 sentences stating whether or not these seem reasonable.

4. Show the output for two sentences in each label category (4 total) from your system after training on the movie data. You should try and find one example each where it intuitively makes the correct decision and one where it intuitively makes the incorrect decision.

Extra Credit

Evaluating the model (2 points)

Split the `movies.data` file into 80% train, 10% dev and 10% test and calculate:

1. The accuracy of your approach on the dev set for varied λ. Accuracy is the proportion of examples that the classifier got correct.
2. The accuracy of your approach on the test set for the same set of λs

Bernoulli NB (4 points)

Implement the Bernoulli NB model. Compare it's performance based on accuracy for the *same* train/dev/test data to that of the multinomial NB model and include a short analysis of your results.

When you're done

All of your code should be in the `code` directory along with your script. You should also include a writeup in the base of your project.

If the naming of the files isn't obvious about where various things are being done, please include a `README` file explaining your organization.

Commenting and code style

Your code should be commented appropriately (though you don't need to go overboard). The most important things:

- Your name (or names) and the assignment number should be at the top of each file you modify.
- Each class and method should have appropriate JavaDoc comments (doc strings if you do it in Python).
- If anything is complicated, put a short note in there to help me out if there are any issues.

This is a non-trivial assignment and it can get complicated, which makes code style and comments very important so that I can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.

Grading

Part	points
Correctness	30
Efficiency	5
Writeup	15
Comments/style	5
extra credit	6
total	55 + 6 extra