4100/5100 Assignment 3: Bayesian Tomatoes
Due Thursday May 31 9PM

HackerRank site:
`https://www.hackerrank.com/cs41005100-hw3-bayesian-tomatoes`

In this assignment, you'll implement a "sentiment analysis" classifier that looks at sentences and decides how positive or negative the speaker is feeling. In fact, you'll implement two: a Naive Bayes classifier, and a cross between a Naive Bayes classifier and a Markov model. This assignment will also serve as your introduction to machine learning.

The data we'll be using consists of sentences pulled from the Rotten Tomatoes movie review aggregator website. Someone from the machine learning competition site Kaggle has helpfully gone through and rated a bunch of sentences:

0 - negative
1 - somewhat negative
2 - neutral
3 - somewhat positive
4 - positive

They've also gone through and "space-delimited" all the words and punctuation in the sentence, making it easier to separate words from punctuation, although they didn't standardize the capitalization. This training data is provided in the file train.tsv.

The provided file BayesianTomatoes.java does some of the heavy lifting of getting the counts you need from this file to do Bayesian reasoning. Reading a file of the train.tsv format, it populates the following data structures:

wordCounts - an ArrayList of 5 HashMaps that look up words to get word counts. There's one lookup table for each sentiment class.
bigramCounts - similar to wordCounts, but this counts two-word phrases. The words are separated by a space.
bigramDenoms - similar to bigramCounts, but only counts how often a word appears as the first word of a bigram. (This is slightly different from the overall word count because words at the ends of sentences aren't counted here.) The name comes from the fact that they'll be used as the denominators in some conditional probabilities.
sentimentCounts - 5 element array, count of all sentences with a particular sentiment.
totalWords - 5 element array, total count of words that were classified with a particular sentiment.
totalBigrams - 5 element array, total count of bigrams seen for each sentiment.

These are global-ish variables partly because it would be annoying to pass them all into the relevant functions, and partly because the question of how to use these counts is part of the assignment.

1) Finish the method naiveBayesClassify, which should use some of the above populated data structures to classify an input sentence with the correct sentiment using Naive Bayes classification. Assume the sentence is space-delimited, and use String.split(" ") to separate the sentence into tokens. Convert words to lower case before looking them up in the provided tables. Don't forget to factor in the prior probability of each sentiment. You should return a Classification object that contains the most likely classification and the log probability that led you to that conclusion. (You do not need to scale the classification probabilities to sum to 1 in this assignment. Don't do that, or you will fail the tests. Just add the relevant log probabilities.)

2) Finish the method markovModelClassify. This method is very similar to the naive Bayes model, but the probability of a word is conditioned on two things: the sentiment, and the word that came before it.

In a normal Markov chain model of language, a chain of words has some probability that can be calculated by multiplying out conditional probabilities:

$\Pr(w_1=\text{“foo”} \wedge w_2=\text{“bar”} \wedge w_3=\text{“baz”}) = \Pr(w_1=\text{“foo”})\Pr(w_2=\text{“bar”}|w_1=\text{“foo”})\Pr(w_3=\text{“baz”}|w_2=\text{“bar”})$

The conditional probability of a word given its predecessor can be calculated by dividing the number of times the bigram "w_1 w_2" appears by the number of times the start of the bigram, "w_1" appears as the first word of a bigram. (Thus if the training corpus consisted of "happy happy joy joy," Pr(joy | happy) is 0.5 because one "happy" is followed by "happy", and the other is followed by "joy." One count of "happy joy," two starting counts of "happy.")

We can perform our bigram classification by training one such model for each sentiment classification. The probability of the sentiment is then proportional to the prior probability of the sentiment (like Naive Bayes, based purely on how often the sentiment occurs in the training data) multiplied by the likelihood of the word chain given the sentiment's Markov chain.

Notice that the first word of the chain is still chosen according to the single-word probability of the word - that probability doesn't use a bigram.

The reason for wanting to use a Markov model here is that sometimes word meanings only become clear when the words are near their neighbors. "not good" is more informative than knowing "not" and "good" appear somewhere in the sentence; a Markov model can think "not good" indicates negative sentiment, but a model just using single word probabilities has to guess whether "not" is an overall good or bad thing.

For either model, when you encounter a word or bigram that has not been seen in the training data for a particular sentiment, use the hardcoded OUT_OF_VOCAB probability.

In your input file, the training data is separated by "---" from the test sentences. For each test sentence, your program should print, on separate lines:

Naive Bayes best classification
Naive Bayes negative log likelihood
Markov model best classification
Markov model negative log likelihood

Two tests are provided: one where you could calculate the values by hand to check your work if needed, and the other, the large Rotten Tomatoes file, train.tsv, which has four test sentences at the end as well. train.tsv is too large to be a sample test on Rotten Tomatoes, but that's what the second test there is. It's provided along with the desired test output for your debugging convenience.

Note: You will probably find it useful to print words and probabilities as debug information, as well as create your own test inputs. Consult the "Java Refresher" files or come to office hours if you're not used to working with input and output redirection at the command line.

"I thought this Movie would be Hella Lit; but considering the Evidence, it actually Hella Bit" - Rev. Bayes (1701-1761)