# 4100/5100 Assignment 3: Bayesian Tomatoes
# Due Friday May 29 9PM

*Provided files: bt.py (stub), bayesTest.txt, bayesTest.out, train.tsv, train.out, smallTest.txt, Python notes*

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

This training data is provided in the file `train.tsv`. (They also rated a lot of phrases within the sentences, but we'll ignore these on loading the file.)

The provided file `bt.py` 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 counts the following and stores the counts in a ModelInfo object:

`word_counts` - a list of 5 dicts that look up words to get word counts. There's one lookup table for each sentiment class.
`bigram_counts` - similar to word_counts, but this counts two-word phrases. The bigrams are generated by nltk.util.bigrams(), which stores them as two-element tuples.
`bigram_denoms` - similar to bigram_counts, 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.
`sentiment_counts` - 5 element list, count of all sentences with a particular sentiment.
`total_words` - 5 element list, total count of words that were classified with a particular sentiment.
`total_bigrams` - 5 element list, total count of bigrams seen for each sentiment.

Using input redirection similar to the previous assignments will populate the ModelInfo for the given file using all data above the "---" line. The sentences below the line are for classification.

A) Install the nltk module, if it's not installed. If you're using Anaconda, the command is

```
conda install -c anaconda nltk
```

Otherwise, look up how to install given your configuration on the Internet.

We don't *really* need nltk for this assignment, because the person who put this dataset together already made it easy to separate the sentences into words (tokens) and pairs of words (bigrams). But if you're using Python and you're interested in NLP, it's a good idea to get used to using NLTK, which has tools for less-preprocessed language.

B) Typing

```
import nltk
```

at the Python prompt should now result in no errors. But our *tokenizer* will still want to download some data, so type

```
nltk.download('punkt')
```

to do that. (You should only need to do this once.)

C) Finish the method naiveBayesClassify, which should use the data in the ModelInfo object to classify an input sentence with the correct sentiment using Naive Bayes classification. Use the provided tokenize() function to break strings into words in a way that matches how the original data was counted. The function should return two comma-separated values: the most likely sentiment number, and the log probability of that sentiment. (Note that you should *not* scale the probabilities to force them to sum to 1 before taking this log. In Naive Bayes classification, this is typically not done when all you want is the most likely class. This lets you add log probabilities as you go to calculate this final number. Also, don't forget the prior, which is the probability that a sentence in general belongs to a particular sentiment class.)

Use OUT_OF_VOCAB_PROB when dealing with a word that was never seen for a particular sentiment level. Notice that this isn't a log probability.

D) 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.

Assume the first word of the sentence is chosen using the same model as Naive Bayes, but every word after depends on both the word that came before it and the sentiment. We estimate the probability of each new word as:

$$\frac{\text{[count of bigram within this sentiment]}}{\text{[count of times first word appeared as first word of a bigram in this sentiment]}}$$

So, for example, if within sentiment #4 the words "very important" appear 3 times as a bigram, and the other bigrams including "very" as a first word are "very serious" and "very entertaining", appearing 7 times each, then Pr("important" | "very" previous word and sentiment 4) is 3/(3+7+7). Note that bigram_denoms already counts this slightly fiddly denominator for you, given a particular starting word and sentiment.

In this model, use OUT_OF_VOCAB_PROB for either bigrams not seen in this sentiment, or single words at the start of the sentence never seen by the sentiment.

E) You can test your classifier on two provided files -- one small, and one large.

In each 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 small file, `bayesTest.txt`, 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. Check that your outputs match the expected outputs for both files, `bayesTest.out` and `train.out`. `bayesTest` is small enough that you should be able to compute its log probabilities by hand with the help of a calculator; if you still have trouble debugging, consider making your own test files that are even smaller.

We have one last test file that you should run on as well, `smallTest.txt`. Submit your output on this file as `smallTest.out`, along with your bt.py code, to Blackboard.

Submission Checklist
bt.py
smallTest.out

*"Stuck inside with all the movie theaters closed for months? I have ... sentiments about this."*
*--Rev. Bayes*