4100/5100 Assignment 6: Semi-Supervised Tomatoes
Due June 21 9PM

For this final assignment, you'll run Expectation Maximization on the "Bayesian Tomatoes" data of assignment 3, doing Expectation Maximization on Naive Bayes models. Utterances with weak or neutral sentiment have been removed from the training data, leaving only extreme like or extreme dislike, to make our model-building easier. Most sentences won't be labeled, but we'll leave a tiny number of labels on (50 sentences or phrases) to prime the learning and make sure our clusters mean what we want them to mean; this is called *semi-supervised learning* since we're providing a few examples labeled with the desired output, but most will be unlabeled at first, and will have their classifications revised as the model does its work.

Our actual data will only have 2 classes, but you should write the code as if CLASSES could be a larger number.

In the UnsupervisedTomatoes.java file, you'll implement the following (and any needed helpers):

1) The *update()* method of the NaiveBayesModel class. Given a sentence and an ArrayList of class probabilities, you will update the *classCounts*, *totalWords*, and *wordCounts* variables by treating each observation as a "fractional" observation, based on the class probabilities. For example, if my probability vector is [0.3, 0.7], and I see the sentence "I like fun," then *classCounts[0]* will increase by 0.3, *totalWords[0]* will increase by 0.3 * 3, *wordCounts[0]["I"]* should increase by 0.3, and so on; while *classCounts[1]* will increase by 0.7, and so on for class 1.

(What we are doing is computing the **expected values** of all these variables; in this example, if there's 0.3 chance I saw class 0 and a 0.7 chance I didn't, the expected number of observations is 0.3 * 1 + 0.7 * 0 = 0.3, the same as the probability. By the principle E[X+Y] = E[X] + E[Y], the expected counts across all observations will be equal to the sum of these "fractional observations.")

2.) The *classify()* method of the NaiveBayesModel class, which should just apply the Naive Bayes model to a sentence and return an ArrayList of probabilities that the sentence belongs to each class. The probabilities should sum to 1.

(Note that you are multiplying instead of adding here, so that you can scale to sum to 1, and that means you're risking underflow. If a probability would be 0 because of multiplication, make it Double.MIN_NORMAL instead, a very small nonzero fraction.)

3.) The main EM loop, which will alternate between the E step (calling classify on all sentences to get probability ArrayLists for each) and the M step (throwing out the old counts of everything and using update() to get new counts, based on the E step classifications).

The way that the classifications are initialized treats the semisupervised examples as being definitely negative (class 0) or positive (class 1), while treating all other examples as being almost equiprobably class 0 or class 1, with a random nudge toward one or the other.

The sample data, trainEMsemisup.txt, has 50 marked examples and 5319 total examples. It also contains two "test sentences." On finishing training, the algorithm will print the 10 tokens with the highest Pr(class | token) for each class, to give you an idea of what the cluster means. It will also run a normal Naive Bayes calculation on each of the test sentences, printing the probability of each class. A working program will look plausible but not perfect for the 20 cluster words, and will classify the test sentences correctly.

If you want to make your own small test cases, adjust MIN_TO_PRINT to the number of times you'd expect important words to appear in the corpus on average. (If the value is too low, the words printed will be rare, but not really representative.)

If you're curious, you can see what happens if you set SEMISUPERVISED to false, so there are no starting hints and two arbitrary clusters are created. You may want to set FIXED_SEED to false as well to see the variety of clusters that can come about. EM only finds locally good solutions, so without starting near the goal, a range of results are possible.

Don't overthink this one, and don't be intimidated by the textbook's EM equation. The main thing that isn't straightforward in EM is the idea of using the expected values to build models - conceptually a little tricky, but you'll find the the code required will be relatively small as these things go.

There are no tests provided[1], and there are no questions to answer here, so turn your code into Blackboard once you think it's working on trainEMsemisup.txt. You're almost done with the course! And now you've implemented EM and a semisupervised algorithm, both of which should sound fancy to employers while also opening up a variety of machine learning techniques that rely on these ideas.

[1] I've used HackerRank successfully in Algorithms and Data, but I think I've decided that it's a poor fit for the latter half of an AI course. If you disagree, chime in on the TRACE evaluations!