CS-540 Homework Assignment #5.5: "A Little Room Confining Mighty Men": Naïve Bayes Implementation

Assigned: Thursday, April 27th
Due: Wednesday, May 3rd

Hand-In Instructions

This assignment includes only a programming portion in Java. The programming portion may be done with a partner, while the written portion must be done individually. Hand in all parts electronically by uploading them in *a single zipped file* to the assignment page on Canvas.

For the programming problem, put *all* files needed to run your program, including ones you wrote and ones provided, into a folder called <wisc NetID>-HW5.5.

Every student should turn in at least a README.txt on Canvas that mentions your username, your name, and the corresponding information of your partner, if you have one. If you don't have a partner, just write "I worked alone."

Once you are finished, put your programming component folder and your PDF file into a single directory. Zip it, name it <wisc NetID>-HW5.5, and upload it to the assignment Canvas page.

Make sure your program compiles on CSL machines this way! Your program will be tested using several test cases of different sizes.

Collaboration Policy

You are to complete this assignment individually or with at most one partner. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions.

You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems.

But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Problem 1: [50] "A Little Room Confining Mighty Men"

Who was it, that thus cried? why worthy Thane,
You do unbend your Noble strength, to think
So braine-sickly of things: Go get some Water,
And wash this filthy Witness from your Hand.
Why did you bring these Daggers from the place?
They must lie there: go carry them, and smeare
The sleeping Grooms with blood.

Shakespeare's *Macbeth* **(Tragedy)**

We are glad the Dauphin is so pleasant with us,
His Present, and your pains we thank you for:
When we have matched our Rackets to these Balls,
We will in France (by God's grace) play a set,
Shall strike his father's Crown into the hazard.
Tell him, he has made a match with such a Wrangler,
That all the Courts of France will be disturbed
With Chases. And we understand him well,
How he comes over us with our wilder days,
Not measuring what use we made of them.

Shakespeare's *Henry V* **(History)**

My Oberon, what visions have I seen!
Me-thought I was enamoured of an Ass.

There lies your love.

How came these things to pass?
Oh, how mine eyes does loath this visage now!

Shakespeare's *A Midsummer Night's Dream* **(Comedy)**

Shakespeare's First Folio, considered the definitive collection of his plays by scholars, was published posthumously in 1632. It divides thirty-six plays among three genres: Tragedy, Comedy, and History. While scholars continue to debate whether certain plays truly belong to their assigned genre, these broad, well-established genres spark interesting conversations and, more importantly, can be used as labels with which to train classifiers.

This assignment focuses on a corpus of Shakespeare's works among those of his Early Modern contemporaries. These plays were written between the late 15th and early 17th centuries, the period in which English evolved into the "modern" language we use

today. **For this assignment, you will build a Naïve Bayes classifier that predicts the genre of an Early Modern play given the words of its dialogue.**

For training and test data, I've provided a sample of Early Modern plays from the Early Modern Drama corpus. The plays' texts have been standardized and stripped of new line characters, so each training/test file has one play per line. If you would like to learn more about how the EMD corpus was standardized in detail, feel free to read the documentation.

Implementation Details

We have provided skeleton code for you that will open a file, parse it, tokenize into a list of strings, pass it to your classifier, and output the results. After reading over the skeleton, you will implement your classifier in *NaiveBayesClassifierImpl.java*.

Unlike the previous assignments, **won't have to make your model generalize to new labels or not dataset formats.** In other words, it will be able to guess genre and only genre. It be tested for hardcoded printing on a resampling of the source of the training and testing set.

Methods to Implement:

1. *void train(Instance[ ] trainingData)*

This method should train your classifier with the training data provided.

2. *double p_l(Label label)*

This method should return the **prior probability** of the label in the training set. In other words, return P(HISTORY) if label == Label.HISTORY or P(COMEDY) if label == Label.COMEDY and so on. You should compute this by dividing the number of training instances with the label passed in as an argument by the total number of training instances.

3. *double p_w_given_l(String word, Label label)*

This method should return the conditional probability of word w given the label. That is, return P (word|label). To compute this probability, you will use smoothing. Please read the note on how to implement this below.

4. *Label classify(Instance i)*

This method returns the label resulting from classification of a single Instance (document).

We have also defined two class types to assist you in your implementation.

- The *Instance* class is a data structure holding the label and the processed document as an array of words.

- The *Label* class is an enumeration of our class labels: TRAGEDY, COMEDY, and HISTORY. You can reference the constant using Label.TRAGEDY and so on.

The only provided file you are allowed to edit is NaiveBayesClassifierImpl.java. But you are allowed to add extra class files if you like.

Smoothing

There are two concepts we use here:

- Word *token*: an occurrence of a given word.
- Word *type*: a unique word as a dictionary entry.

For example, "the dog chases the cat" has 5 word tokens but 4 word types; there are two tokens of the word type "the". Thus, when we say a word "token" in the discussion below, we mean the number of words that occur and NOT the number of unique words.

The conditional probability P (w|l), where w represents some word token and l is a label, is a multinomial random variable. If there are V possible word types that might occur, imagine a V-sided die. P(w|l) is the likelihood that this die lands with the w-side up. You will need to estimate, for each word token, the probability of P(w|TRAGEDY), P(w|COMEDY) and P(w|HISTORY).

When testing our model, we will inevitably encounter words we've never seen before. To avoid multiplying or dividing by zero, we will pretend we actually did see some (possibly fractionally many) tokens of the new word. This goes by the name of Laplace smoothing or add-δ smoothing, where δ is a parameter. Compute P(w|l) as:

$$P(w|l) = \frac{C_l(w) + \delta}{|V|\delta + \sum_{v \in V} C_l(v)}$$

- |V| is the size of *combined vocabulary* (unique words) of all words in the dataset.
- $C_l(w)$ is the total number of w tokens (ie number of times "and" appears, across all documents with label l in the training set.
- The summation term at the bottom of the denominator is the total number of tokens of all documents with label l in the training set.
- For this assignment, use the value δ = 0.00001.

Use the equation above for P(w|l) to calculate the conditional probabilities in your implementation.

Log Probabilities

The second gotcha that any implementation of a Naive Bayes classifier must contend with is under- flow. Underflow can occur when we take the product of a number of very small floating-point values. Fortunately, there is a workaround. Recall that a Naive Bayes classifier computes

$$f(w) = \arg\max_{l} \left[P(l) \prod_{i=1}^{k} P(w_i|l) \right]$$

Because maximizing a formula is equivalent to *maximizing the log value of that formula*, f (w) computes the same class as

$$g(w) = \arg\max_{l} \left[\log P(l) + \sum_{i=1}^{k} \log P(w_i|l) \right]$$

What this means for you is that in your implementation you should compute **the g(w) formulation of the function above rather than the f(w) formulation.** Use the Java function log(x) which computes the natural logarithm of its input. This will result in code that avoids errors generated by multiplying very small numbers. **Note, however, that your methods p(l) and p(w|l) should return the true probabilities themselves and NOT the logs of the probabilities.**

Testing

We will test your program on multiple training and testing sets, and the format of testing commands will be:

java HW5 <modeFlag> <trainingFilename> <testFilename>

where trainingFilename and testFilename are the names of the training and testing data set files, respectively. modeFlag is an integer from 0 to 3, controlling what the program will output:

0: Prints out the number of documents for each label in the training set. 1: Prints out the number of words for each label in the training set.

2: For each instance in test set print out a line displaying the predicted class and the log probabilities for both classes.

In order to facilitate your debugging, we are providing to you a sample input file and its corresponding output files for each mode. They are called train.bbc.txt and test.bbc.txt in the zip file. So, here is an example command:

java HW5 0 play_train.txt play_test.txt

As part of our testing process, we will unzip the file you turn in, remove any class files, call javac *.java to compile your code, and then call the main method HW5 with parameters of our choosing. Make sure your code runs on one of the computers in the department because we will conduct our tests on these computers.