Brian Cullinan

11/8/2010

# Assignment 3

## Solution Overview:

When I first began the project, I was immediately reminded why I don’t like Java. It is an incredibly restrictive language, with semantics that are unnecessarily difficult to understand. Nuances such as using the size() function on ArrayLists, but using the .length property on actual arrays are very annoying.

Data structures are very hard to manage unlike other languages. For example, a class must be created and some sub-structure for each type of input for each class in the training data. It other languages, string compares or enumerate types would suffice, but in Java, everything must be laid out in a very exhaustive way. In this case, I created a class to hold gene data, and each gene was stored in an ArrayList. Java is not a concise language like Perl is, nor is it elegant like Ruby, nor is it simple and loosely types like PHP. All of these features of a language would be helpful when dealing with complex and large amount of data.

First the training data is loaded, then the testing data, and finally the “keys” data. The training data is enumerated and the Bayesian values are calculated for each Localization class. The entire training set is enumerated for each testing gene.

### How to Run:

This can be run by compiling and running the Main class. The input data must also be stored within the package. There are no command line options, and the result is saved in “output.txt”.

The program first loads all the training data and the testing data. It classifies the testing data and then loads the keys and checks how many are correct. It outputs the percent of completion and finally the number of correct items classified to standard out.

### Classifier:

The classifier I chose to implement is naive Bayesian classification. This seemed like a fairly simple classifier. It iterates through each field in the data set and counts the training data to obtain the maximum probability for the class each of the testing genes falls in to. The classifier that I implemented also uses the Laplacian correction. This sets the value to 1 if any of the fields occur 0 times, which allows a non-zero value for every probability that is calculated.

### Missing Data:

For the missing data I used the assumption that the gene was a part of that class and filled in the missing values. This made the classifier a lot more accurate, about 3 times more accurate than if it is just ignored.

### Accuracy:

I used cross-validation to check the accuracy of the results. These results are stored in the output.txt file. The accuracy resulted in 34%. This is not very high. I did not use the interactions table, and I did find it interesting that within the set of testing data, there are multiple genes with the same label.