Assignment 5

Assigned: Thu Mar 22, 2001
Due: Thu April 5, 2001

In this assignment, you will implement an ensemble technique known as AdaBoost. The AdaBoost algorithm is described in the Schapire paper I'm handing out, and more information on AdaBoost can be found on his web page (http://www.research.att.com/~schapire/boost.html).

Although you can build ensembles composed of any type of machine learning model, I want you to focus on a particularly simple model, a *decision stump*, which is simply a decision tree with a single branch. A single decision stump is a *weak learner*—it does not perform particularly well—but an ensemble of decision stumps can perform as well as or better than a full-blown decision tree.

Code organization

One approach to this problem is to break it into three separate programs—one that creates decision stubs, one that produces weightings for AdaBoost, and one that produces predictions and accuracy estimates for the test set.

The decision stump code should (1) create a decision stump based on a training set and a file containing weightings; (2) generate an output file containing one line per training example, including its target classification, its classification by the decision stub, and the current weighting of the example; and (3) generate an output file containing one line per test example, including the target and actual classifications.

The boosting code should read in the output file related to the training set, compute the weightings for the next iteration, and output them to a file for use by the decision stump code.

The prediction code should read in all of the test set files and combine the predictions from the individual stumps to produce final predictions, which can be compared to the target classification to produce an error rate.

Decision stump

You will first have to write a program that creates a decision stump based on some training data, and then classifies a test data set. You should be able to modify your decision tree program to implement the decision stump, or you can write a new program re-using routines from your decision tree software. Here are the important differences between the decision tree and the decision stump you will need for this assignment:

- The decision stump has only one level of branching. Thus, it is a decision tree with max_depth = 1. You can either remove the recursion from your decision tree software, or leave the software unchanged and force max_depth = 1.
- The data set you used for the decision tree included attributes that all had the values "y" or "n". In this assignment each attribute has a different set of values, and some attributes have more than two values. (Sorry, I tried to find another data set with binary-valued attributes, and there just weren't any interesting ones.)
- You may need to handle the case where decisions are based on an attribute dimension which has values in the test set that weren't contained in the training set. In this case, you should classify the example according to the majority value in the root node.
- In AdaBoost, classifiers are created based on a set of *weighted* training examples. Thus, your code for training the stump should read in a set of weights associated with the training examples. (If you put the weights in the same order as the training examples in the data file, you don't need to figure out the cor-

respondence.) The weights play into the decision stump algorithm in two ways. First, when you are computing, for a given set of examples, the fraction that are + or – (which you need to compute the entropy), you must use the weightings, i.e., $p_+ = w_+ / (w_+ + w_-)$, where w_+ is the total weight of the positive examples, and w_- is the total weight of the negative examples. Second, when you are computing the fraction of examples that are in one split i (which you need to compute the average entropy across splits), you should use $|S_i| / |S| = (w_{i+} + w_{i-}) / \Sigma_j (w_{j+} + w_{j-})$.

Boosting

The boosting code should read in the file produced by the decision stump for the training set, compute ε, α, and the new weightings. For this piece of code to be self contained, you will either need to read in the file containing the weightings from the current iteration, or you will need to write those weightings into the output file.

Prediction

After you have constructed a boosted set of decision stumps, and used those stumps to make predictions for the test set, you will need to combine them to get a final set of predictions for the test set and evaluate their performance. Assuming you make the predictor a separate program, and you have written the test set predictions for each boosting iteration to a file, you can loop through the test set files, combining the classifications weighted by the α's. (Note: you will have to have saved the alpha values some place at the time you built the stubs. You could put them in the test data output file.)

Data set

The data set for this assignment consists of descriptions of mushrooms drawn from the Audubon Society Field Guide. Each mushroom is described by 22 physical characteristics (e.g., scaly, yellow) and also by whether it is poisonous or edible. Information about the data base can be obtained at

`ftp://ftp.ics.uci.edu/pub/machine-learning-databases/mushroom`

The file `agaricus-lepoita.names` contains information about the data base and the 22 attributes, and the file `agaricus-lepiota.data` contains the actual data. The data set contains some records with missing values (indicated by question marks). I have removed these items from the data set, and split the data into a training set and a test set. You should retrieve the data from my site:

`ftp://ftp.cs.colorado.edu/users/mozer/5622/mushroom.train`
`ftp://ftp.cs.colorado.edu/users/mozer/5622/mushroom.test`

The first column of the data contains the class label, “p” for poisonous and “e” for edible.

What to hand in

Hand in your code as well as a plot showing the test set performance as a function of the number of boosting iterations. If you want, you can also show the training set performance as a function of the number of boosting iterations, but to compute this you'll need to treat the training data as you did the test data. Try 1 to 50 iterations of boosting.

Bells and Whistles

Without modifying your code, you should be able to run two other data sets we've used this semester: the credit approval and voting record data sets. If you have written really general purpose code (i.e., your initials are S. R.), you can explore a boosted ensemble of neural networks, maybe using the digits data base. Each neural net can be relatively small and therefore quick to train.