Assignment 3

Assigned: Tue Feb 13, 2001
Due: Thu Feb 22 2001

In this assignment, you will implement a decision-tree learning system, much like ID3 described in the textbook. You will also use cross validation to estimate the performance of the decision tree. The decision tree will have a parameter—the maximum depth of the tree, or *maxdepth*—which will control the complexity of the resulting decision tree. You will explore how varying *maxdepth* affects the decision tree's performance on both the training set and the test set.

The data set available for this assignment is based on the U.S. congress voting record from 1984. The data set consists of the votes (*yes* or *no*) on sixteen issues for each of the 435 members of congress. From the voting record, the task of the machine learning system is to predict whether the member of congress is a Republican or a Democrat. The voting data set is described in the UCI Machine Learning Repository. However, I have cleaned up the data set and made it available at

`ftp://ftp.cs.colorado.edu/users/mozer/5622/votes.tar`

The original data set contained many *missing values*, i.e., votes in which a member of congress failed to participate. Dealing with missing values is tricky, so I made your task simpler by inserting—for each absent vote—the voting decision of the majority. The result is that each record in the data base looks something like the following:

```
D y y y n n n y y y n n n n n y y
```

The `D` (or `R`) indicates the individual is a Democrat (Republican), and the symbols `y` and `n` denote yes and no votes. The column in which the `y` or `n` appears corresponds to the particular issue being voted on. Because all input attributes are binary, the resulting decision tree will also be binary (two branches from every non-leaf node).

The `votes.tar` file contains five copies of the data, each split into training and test sets. The training sets are named `votes-train[0-4].data` and the test sets are named `votes-test[0-4].data`. The union of the five test sets is the entire data base. Thus, training and testing on each of the five sets corresponds to five-fold cross validation. I've split up the data in this manner to make your task easier, and to ensure that everyone should obtain the same result.

Your decision tree program should take three inputs: (1) the name of the training file, (2) the name of the test file, and (3) *maxdepth*. The tree should be prevented from going deeper than *maxdepth* (*maxdepth=1* means a root node—level 0—and leaves—level 1). The program should build a decision tree subject to the *maxdepth* constraint, and then output the classification accuracy on both the training and test sets. You should perform this experiment for the five hold-out (test) sets, and compute the total proportion correct on the training and test sets for a particular value of *maxdepth*. Repeat this procedure for *maxdepth* ranging from 0 to 8.

Hand in code for your decision tree, a table and possibly a graph showing performance on the training and test sets as *maxdepth* is varied.

To begin constructing your tree, start with a root node. Associate all training examples with the root node. Consider branching the tree along each attribute dimension, and choose the dimension that yields the greatest gain, as quantified in equation 3.4 of the text. Create two child nodes—the two branches from the root—and associate with each child node the training examples that would be passed to that node. Repeat this process, stopping with the maximum tree depth is reached, when all examples associated with a node have the same classification, or when all examples associated with a node have the exact same voting record. For each of these leaf nodes, label the node as `D` or `R`, based on whether a majority of the training examples are Democrats or Republicans. If there's exactly the same number of Democrats and Republi-

cans at a leaf node, you can label the node whichever way you prefer. It might make more sense to label the node `D` because a majority of members of congress were Democrats in 1984.

For the voting record data set, each attribute is binary, and is either `y` or `n`. The output of the decision tree is one of two classes (`D` or `R`). You can make your code specific to this case (binary branches from nodes, all labeled `y` or `n`, and binary classification). However, it will be nice to generate code that is more flexible—allowing *n*-way branches and *n*-way classification. We will have the option of using this code for a future assignment, and the future assignment will require *n*-way branches and *n*-way classification.

Milestones

I strongly suggest that you do not put off this assignment until the last minute. I am giving you 16 days for the assignment, and if you work on it steadily, you should avoid last-minute panic. Here are some target dates you might shoot for:

Feb 13: download data set, understand how data set has been partitioned, and understand format of individual records. Write code that reads in the training set and stores it in a data base.

Feb 14: write function that computes class entropy over a subset of the training data; debug function; specify data structures used for representing decision tree.

Feb 16: write function that computes gain for a given attribute and a given subset of the training data, and another function that loops over all the attributes and determines which attribute should be chosen for branching based on maximizing the gain

Feb 18: implement main loop of decision tree constructor that considers one node at a time, and determines whether the node should be a leaf node or whether it should branch along some attribute dimension.

Feb 19: write function that, given a decision tree and an input example, determines the classification of the example. run this function on both the training and test data for a given data split.

Feb 20: write a shell script that loops over the 5 data splits and the various values of *maxdepth* to evaluate performance for each condition.

This assignment could be implemented in C, C++, Java, or perl; it shouldn't matter much which platform you use. My perl implementation of this assignment is about 200 lines long.

Bells and Whistles

Once you have the basic assignment completed, you might be curious to implement a different measure for attribute selection, such as the gain ratio described in section 3.7.3 of the text. You could also experiment with other complexity measures instead of *maxdepth*, such as requiring the gain ratio to be above a certain threshold in order to perform a split, or limiting the total number of nodes in the tree. Finally, if you are still looking for a way to make your code more general, modify it to handle missing attribute values, in one of the ways described in section 3.7.4 of the text. To test your code for missing attributes, grab the original data from the UCI repository; in this data set, missing attributes are indicated by question marks.