36-350: Data Mining

Lab 11

Date: November 8, 2002 **Due: end of lab**

---

1 Introduction

This lab teaches you how to construct and prune classification trees.

There are 5 questions. For each one, submit your commands and a response from R demonstrating that they work. (Only hand in commands relevant to the question.) To submit a plot, click on the plot window and select

```
File -> Save as -> Postscript...
```

This saves the plot to a file which can be printed, incorporated into a Word document, or mailed to us as an attachment.

2 Starting R

Start R as in lab 1. On the class web page, go to "computer labs" and download the files for lab 11 into your work folder. Read the special functions into your running R application via the commands

```
source("lab11.r")
```

If this fails, check that the files were downloaded correctly.

3 The data

The dataset used in this lab is the repayment behavior of 1000 individuals who acquired loans from a bank. The bank would like to use this data to decide which customers in the future are likely to repay a loan. Each individual is described by 21 variables, the most important being `Class` which classifies the loan as good or bad. Load this data via

```
load("Credit.rda")
```

This defines a matrix of training data called `x.tr` and a matrix of test data called `x.te`. Your job is to construct a classifier from the training data which has high accuracy in predicting `Class` on the test data.

Several of the variables are categorical. In the matrices, they have been numerically coded as follows:

```
Checking :
   <0 0-200  >200  None
 0.51  0.61  0.78  0.88
History :
     None Okay.here Until.now    Delays  Critical
     0.38      0.43      0.68      0.68      0.83
Purpose :
          Education               Other             New car
               0.56                0.58                0.62
            Repairs            Business           Appliance
               0.64                0.65                0.67
Furniture/equipment            Radio/TV            Used car
               0.68                0.78                0.83
         Retraining
               0.89
Savings :
    <100  100-500        0 500-1000    >1000
    0.64     0.67     0.83     0.83     0.88
Employed :
  <1    0  1-4   >7  4-7
0.59 0.63 0.69 0.75 0.78
Gender.Status :
 Male.divorced Female.married   Male.married    Male.single
          0.60           0.65           0.73           0.73
Guarantor :
Co.applicant         None    Guarantor
        0.56         0.70         0.81
Property :
          None            Car Life.insurance    Real.estate
          0.56           0.69           0.69           0.79
Other.plans :
  Bank Stores   None
  0.59   0.60   0.72
Housing :
Free Rent  Own
0.59 0.61 0.74
Job :
  Manager      None   Skilled Unskilled
     0.66      0.68      0.70      0.72
Telephone :
  No  Yes
0.69 0.72
Foreign :
 Yes   No
0.69 0.89
```

4 Constructing a tree

The command to construct a tree is `tree`. It is used the same way as `lm` and `loess`. You provide a formula and a data matrix and it returns a tree object. This tree object can be plotted by giving it to the function `plot.graph.tree`.

```
tr <- tree(Class~.,x.tr)
plot.graph.tree(tr)
```

To evaluate the tree's accuracy, there is the function `misclass`. Give it the tree object and a data matrix to classify. It returns the number of times the classifier chose a class different than the true class given in the matrix.

```
misclass(tr,x.tr)
misclass(tr,x.te)
```

> **Question 1:** Submit code to convert the number of errors above into accuracy rates. Use it to report classifier performance in the following questions.

To prune a tree, there is the function `prune.misclass`. Give it the tree object and a data matrix on which to evaluate different prunings. It returns the number of errors that each pruned tree makes. For illustration, try giving it the test data, to get an idea of what the "ideal" pruning of the tree should look like:

```
p = prune.misclass(tr,newdata=x.te)
plot(p,type="o")
```

The "size" axis here is the number of leaves in the tree. You should find that pruning the tree a little does help its performance. Of course, in a real situation you wouldn't be able to use the test data this way. In the next section, you will use cross-validation to try to approximate the ideal pruning.

> **Question 2:** Why isn't `prune.misclass` useful on the training data?

5 Cross-validation pruning

The function for pruning by cross-validation is `best.size.tree`. It takes a tree as input. It splits the training set for this tree into 10 blocks, constructs 10 new trees, and prunes each tree based on a block not used to construct the tree. It averages the errors on each of the 10 blocks and plots them, similarly to `prune.misclass`. Then it takes the tree size with lowest average errors and prunes the provided tree to have that size, returning a new tree.

```
tr.pruned <- best.size.tree(tr)
```

> **Question 3:** (a) Compare the cross-validation plot to the "ideal" pruning plot. Do they prefer the same size trees? (b) Which tree, `tr` or `tr.pruned`, performs better on the training set? Which performs better on the test set? (c) Which tree has a larger discrepancy between its training accuracy and its test accuracy (i.e. is overfitting the training set)?

Instead of splitting the training set into 10 blocks, `best.size.tree` can use any number of blocks, e.g. just two:

```
tr.pruned <- best.size.tree(tr,2)
```

You will find that this runs a lot faster, since only two new trees are being constructed.

> **Question 4:** Cross-validation is an inherently random process, because the splitting is chosen randomly each time. Run cross-validation with 2 blocks multiple times, and compare to using 10 blocks multiple times. (a) Which number of blocks is more stable? (b) When there are 2 blocks, how much of the training data is used to build each of the 2 trees? When there are 10 blocks, how much of the training data is used to build each of the 10 trees?

6 Nearest-neighbor classification

To build a nearest-neighbor classifier, use the function `knn.model`. It works the same way as `tree`, and returns a `knn` object. This object can be given to `misclass` to compute misclassifications, just like a tree object.

Cross-validation can also be used to improve a nearest-neighbor classifier. For each test point, a 1-nearest-neighbor classifier answers the class of the nearest training point. This is the default. But you can also make a 3-nearest-neighbor classifier; it finds the three nearest training points and votes their classes. This often gives better performance than using one nearest neighbor. More generally, a k-nearest-neighbor classifier finds the k nearest training points and votes their classes.

To illustrate the effect of k, the function `test.k.knn` evaluates different k's on the test set:

```
test.k.knn(nn,x.te)
```

Here `nn` is a `knn` object from `knn.model`. You will find that the errors change in an irregular way as you change k, though the overall dip shape should be similar to tree pruning. Of course, you can't use the test set this way, but you can try to come close using cross-validation.

The function `best.k.knn` is similar to `best.size.tree`; it splits the training set into 10 blocks, constructs 10 new nearest-neighbor classifiers, and tests each one on a block not used to construct the classifier. When testing, it tries various different values for k, and reports the average error for each one. Then it takes the k with lowest average errors and returns a new `knn` object which uses that value of k.

> **Question 5:** (a) Construct a 1-nearest-neighbor classifier from the training set and report the accuracy rate on the test set. (b) Use cross-validation to choose a better k, and evaluate this k on the test set. It is better than $k = 1$? (c) Compare the cross-validation plot to the "ideal" plot. Do they prefer the same value of k?