Decision trees

Preparing the data

The Wisconsin Breast Cancer dataset is available as a comma-delimited text file on the UCI Machine Learning Server (http://archive.ics.uci.edu/ml). The dataset contains 699 fine-needle aspirate samples, where 458 (65.5%) are benign and 241 (34.5%) are malignant. The dataset contains a total of 11 variables and doesn't include the variable names in the file. Sixteen samples have missing data and are coded in the text file with a question mark (?).
The variables are as follows:

- ID
- Clump thickness
- Uniformity of cell size
- Uniformity of cell shape
- Marginal adhesion
- Single epithelial cell size
- Bare nuclei
- Bland chromatin
- Normal nucleoli
- Mitoses
- Class

The first variable is an ID variable (which you'll drop), and the last variable (class) contains the outcome (coded 2=benign, 4=malignant). For each sample, nine cytological characteristics previously found to correlate with malignancy are also recorded. These variables are each scored from 1 (closest to benign) to 10 (most anaplastic). But no one predictor alone can distinguish between benign and malignant samples. The challenge is to find a set of classification rules that can be used to accurately predict malignancy from some combination of these nine cell characteristics. See Mangasarian and Wolberg (1990) for details. In the following listing, the comma-delimited text file containing the data is downloaded from the UCI repository and randomly divided into a training sample (70%) and a validation sample (30%).

The training sample has 499 cases (329 benign, 160 malignant), and the validation sample has 210 cases (129 benign, 81 malignant). The training sample will be used to create classification schemes using a decision tree. The validation sample will be used to evaluate the effectiveness of these schemes.

```
loc <- "http://archive.ics.uci.edu/ml/machine-learning-databases/"
ds <- "breast-cancer-wisconsin/breast-cancer-wisconsin.data"
url <- paste(loc, ds, sep="")
breast <- read.table(url, sep=",", header=FALSE, na.strings="?")
names(breast) <- c("ID", "clumpThickness", "sizeUniformity",
"shapeUniformity", "maginalAdhesion",
"singleEpithelialCellSize", "bareNuclei",
"blandChromatin", "normalNucleoli", "mitosis", "class")
df <- breast[-1]
df$class <- factor(df$class, levels=c(2,4),
labels=c("benign", "malignant"))
set.seed(1234)
train <- sample(nrow(df), 0.7*nrow(df))
df.train <- df[train,]
df.validate <- df[-train,]
table(df.train$class)
table(df.validate$class)
```

Decision trees

Decision trees are popular in data-mining contexts. They involve creating a set of binary splits on the predictor variables in order to create a tree that can be used to classify new observations into one of two groups. In this section, we'll look at two types of decision trees: classical trees and conditional inference trees.

Classical decision trees

The process of building a classical decision tree starts with a binary outcome variable (benign/malignant in this case) and a set of predictor variables (the nine cytology measurements). The algorithm is as follows:

1. Choose the predictor variable that best splits the data into two groups such that the purity (homogeneity) of the outcome in the two groups is maximized (thatis, as many benign cases in one group and malignant cases in the other as possible). If the predictor is continuous, choose a cut-point that maximizes purity for the two groups created. If the predictor variable is categorical (not applicable in this case), combine the categories to obtain two groups with maximum purity.
2. Separate the data into these two groups, and continue the process for each subgroup.
3. Repeat steps 1 and 2 until a subgroup contains fewer than a minimum number of observations or no splits decrease the impurity beyond a specified threshold. The subgroups in the final set are called *terminal nodes*. Each terminal node is classified as one category of the outcome or the other based on the most frequent value of the outcome for the sample in that node.
4. To classify a case, run it down the tree to a terminal node, and assign it the modal outcome value assigned in step 3.
5.

Unfortunately, this process tends to produce a tree that is too large and suffers from overfitting. As a result, new cases aren't classified well. To compensate, you can prune back the tree by choosing the tree with the lowest 10-fold cross-validated prediction error. This pruned tree is then used for future predictions.

In R, decision trees can be grown and pruned using the rpart() and prune() functions in the rpart package. The following listing creates a decision tree for classifying the cell data as benign or malignant.

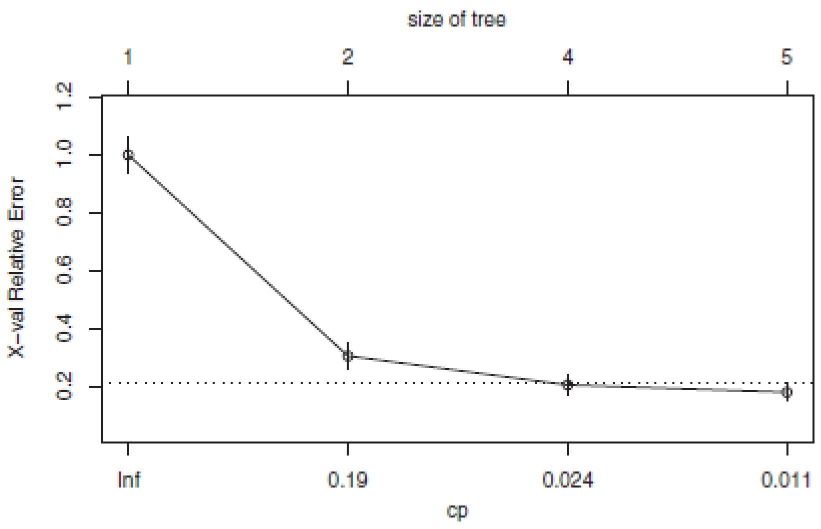
```
> library(rpart)
> set.seed(1234)
> dtree <- rpart(class ~ ., data=df.train, method="class",
parms=list(split="information"))
> dtree$cptable
CP nsplit rel error xerror xstd
1 0.800000 0 1.00000 1.00000 0.06484605
2 0.046875 1 0.20000 0.30625 0.04150018
3 0.012500 3 0.10625 0.20625 0.03467089
4 0.010000 4 0.09375 0.18125 0.03264401
> plotcp(dtree)
> dtree.pruned <- prune(dtree, cp=.0125)
> library(rpart.plot)
> prp(dtree.pruned, type = 2, extra = 104,
fallen.leaves = TRUE, main="Decision Tree")
> dtree.pred <- predict(dtree.pruned, df.validate, type="class")
> dtree.perf <- table(df.validate$class, dtree.pred,
dnn=c("Actual", "Predicted"))
> dtree.perf
Predicted
Actual            benign malignant
benign            122   7
malignant   2     79
```

First the tree is grown using the rpart() function. You can use print(dtree) and summary(dtree) to examine the fitted model (not shown here). The tree may be too large and need to be pruned.
In order to choose a final tree size, examine the cptable component of the list returned by rpart(). It contains data about the prediction error for various tree sizes. The complexity parameter (cp) is used to penalize larger trees.

Tree size is defined by the number of branch splits (nsplit). A tree with *n* splits has *n* + 1 terminal nodes. The rel error column contains the error rate for a tree of a given size in the training sample. The cross-validated error (xerror) is based on 10-fold cross validation (also using the training sample). The xstd column contains the standard error of the crossvalidation error.

The plotcp() function plots the cross-validated error against the complexity parameter (see figure 17.1). A good choice for the final tree size is the smallest tree whose cross-validated error is within one standard error of the minimum crossvalidated error value.

The minimum cross-validated error is 0.18 with a standard error of 0.0326. In this case, the smallest tree with a cross-validated error within 0.18 ± 0.0326 (that is, between 0.15 and 0.21) is selected. Looking at the cptable table in listing 17.3, a tree with three splits (cross-validated error = 0.20625) fits this requirement.

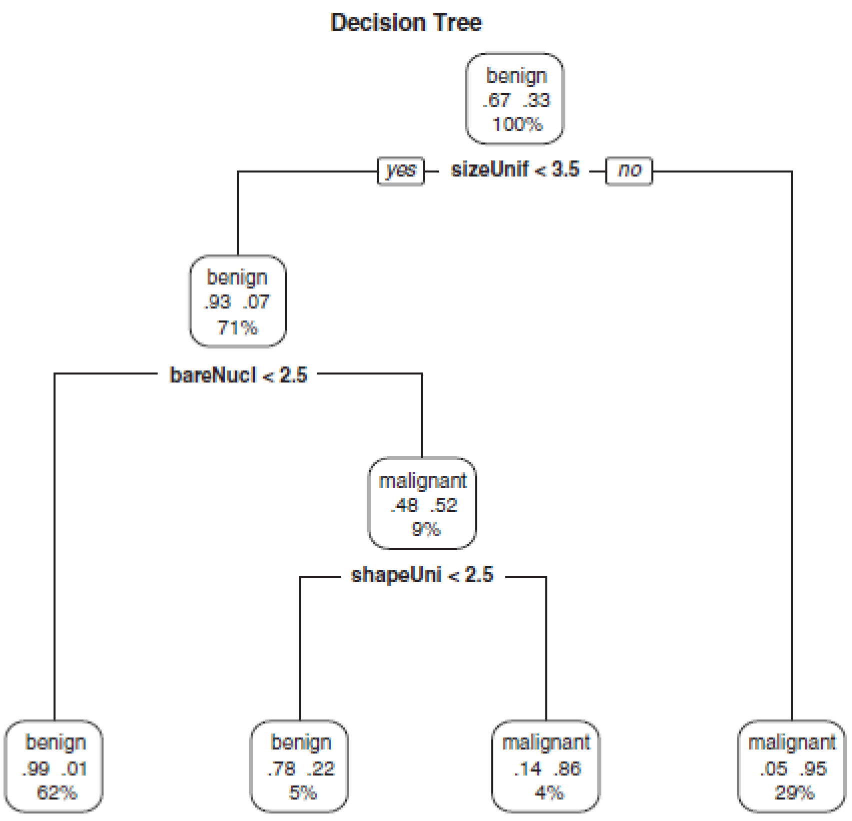


Complexity parameter vs. cross-validated error. The dotted line is the upper limit of the one standard deviation rule (0.18 + 1 * 0.0326 = .21). The plot suggests selecting the tree with the leftmost **cp** value below the line.

Equivalently, you can select the tree size associated with the largest complexity parameter below the line in the figure. Results again suggest a tree with three splits (four terminal nodes).

The prune() function uses the complexity parameter to cut back a tree to the desired size. It takes the full tree and snips off the least important splits based on the desired complexity parameter. From the cptable in listing 17.3, a tree with three splits has a complexity parameter of 0.0125, so the statement prune(dtree, cp=0.0125) returns a tree with the desired size .

The prp() function in the rpart.plot package is used to draw an attractive plot of the final decision tree (see figure 17.2). The prp() function has many options (see ?prp for details). The type=2 option draws the split labels below each node. The extra=104 parameter includes the probabilities for each class, along with the percentage of observations in each node. The fallen.leaves=TRUE option displays the terminal nodes at the bottom of the graph. To classify an observation, start at the top of the tree, moving to the left branch if a condition is true or to the right otherwise. Continue moving down the tree until you hit a terminal node. Classify the observation using the label of the node.



Traditional (pruned) decision tree for predicting cancer status. Start at the top of the tree, moving left if a condition is true or right otherwise. When an observation hits a terminal node, it's classified. Each node contains the probability of the classes in that node, along with the percentage of the sample.

Finally, the predict() function is used to classify each observation in the validation sample. A cross-tabulation of the actual status against the predicted status is provided. The overall accuracy was 96% in the validation sample. Note that decision trees can be biased toward selecting predictors that have many levels or many missing values.

Conditional inference trees

Before moving on to random forests, let's look at an important variant of the traditional decision tree called a *conditional inference tree*. Conditional inference trees are similar to traditional trees, but variables and splits are selected based on significance tests rather than purity/homogeneity measures. The significance tests are permutation tests.

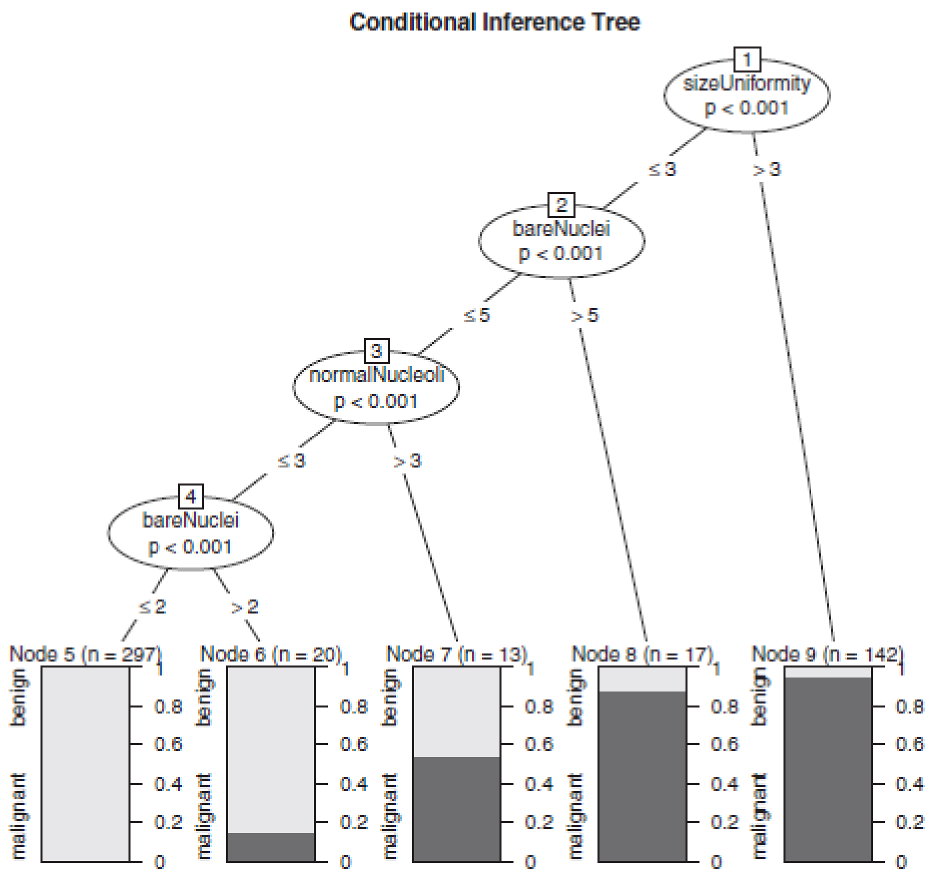
1. In this case, the algorithm is as follows:
2. Calculate p-values for the relationship between each predictor and the outcome variable.
3. Select the predictor with the lowest p-value.
4. Explore all possible binary splits on the chosen predictor and dependent variable (using permutation tests), and pick the most significant split.
5. Separate the data into these two groups, and continue the process for each subgroup.
6. Continue until splits are no longer significant or the minimum node size is reached.

Conditional inference trees are provided by the ctree() function in the party package.
In the next listing, a conditional inference tree is grown for the breast cancer data.

```
library(party)
fit.ctree <- ctree(class~., data=df.train)
plot(fit.ctree, main="Conditional Inference Tree")
> ctree.pred <- predict(fit.ctree, df.validate, type="response")
> ctree.perf <- table(df.validate$class, ctree.pred,
dnn=c("Actual", "Predicted"))
> ctree.perf
```

```
Predicted
Actual benign malignant
benign 122 7
malignant 3 78
```

Note that pruning isn't required for conditional inference trees, and the process is somewhat more automated. Additionally, the party package has attractive plotting options. The conditional inference tree is plotted in the figure below. The shaded area of each node represents the proportion of malignant cases in that node. The decision trees grown by the traditional and conditional methods can differ substantially. In the current example, the accuracy of each is similar. In the next section, a large number of decision trees are grown and combined in order to classify cases into groups.


Conditional Inference Tree
1
sizeUniformity
p < 0.001
≤ 3
> 3
2
bareNuclei
p < 0.001
≤ 5
> 5
3
normalNucleoli
p < 0.001
≤ 3
> 3
4
bareNuclei
p < 0.001
≤ 2
> 2
Node 5 (n = 297)
Node 6 (n = 20)
Node 7 (n = 13)
Node 8 (n = 17)
Node 9 (n = 142)
benign
malignant
1
0.8
0.6
0.4
0.2
0