Decision Tree with R

This lab manual is to demonstrate how to build and prune a Decision Tree in R. The package that we will be using is Recursive Partitioning and Regression Trees (rpart).

To build a tree, use

rpart(formula, data, method, control)

formula	a formula in the form of outcome ~ predictor1 + predictor2 + … + predictor*N* A predictor is an attribute.
data	A data frame in which to interpret the predictors named in the formula
method	Specifies a “class” or “anova” for classification or regression respectively.
control	Optional parameters for controlling tree growth.

Building the tree

We will be using the diabetes dataset. The dataset contains 11 attributes (columns) as follows.

- age: Age of the patient
- sex: Gender of the patient
- bmi: Body mass index
- bp: Blood pressure
- S1, S2, S3, S4, S5, S6: Six blood serum measurements
- Diabetes: A quantitative measure of disease progression one year after baseline

Download the diabetes dataset. Load the package as follows.

```
> install.packages("rpart")
> library(rpart)
> install.packages("Metrics")
> library(Metrics)
```

Load the data into R.

```
> diabetes <- read.csv("diabetes.csv")
```

To view the data frame

```
> View(diabetes)
```

Let’s examine the columns in the data frame.

```
> str(diabetes)
```

```
'data.frame':   442 obs. of  11 variables:
 $ age     : num  0.03808 -0.00188 0.0853 -0.08906 0.00538 ...
 $ sex     : num  0.0507 -0.0446 0.0507 -0.0446 -0.0446 ...
 $ bmi     : num  0.0617 -0.0515 0.0445 -0.0116 -0.0364 ...
 $ bp      : num  0.02187 -0.02633 -0.00567 -0.03666 0.02187 ...
 $ s1      : num  -0.04422 -0.00845 -0.0456 0.01219 0.00393 ...
 $ s2      : num  -0.0348 -0.0192 -0.0342 0.025 0.0156 ...
 $ s3      : num  -0.0434 0.07441 -0.03236 -0.03604 0.00814 ...
 $ s4      : num  -0.00259 -0.03949 -0.00259 0.03431 -0.00259 ...
 $ s5      : num  0.01991 -0.06833 0.00286 0.02269 -0.03199 ...
 $ s6      : num  -0.01765 -0.0922 -0.02593 -0.00936 -0.04664 ...
 $ diabetes: num  151 75 141 206 135 97 138 63 110 310 ...
```

We would like to predict if a patient diabetes progression given the attributes. As we can see column "diabetes" is in num (continuous value).

Then, we split the data into training dan test sets, in a ratio of 70:30. The training set is used for training and creating the model. The test set is to evaluate the accuracy of the model.

```
> sample_ind <- sample(nrow(diabetes), nrow(diabetes)*0.7)
> train <- diabetes[sample_ind,]
> test <- diabetes[-sample_ind,]
```

Assuming we want to predict the patients' diabetes given all the attributes. Now, let's build a regression tree by calling the rpart function. To build a tree, we begin with a small cp or cp equals 0. Certain attributes can be specified as follows e.g. `diabetes ~ age + sex + bmi + bp + s1`.

```
> reg_tree <- rpart(diabetes ~ ., train, method="anova",
  control=rpart.control(cp=0))
```

We can plot the decision tree using plot() and text() functions. The cex argument is to specify the size of the text.

```
> plot(reg_tree)
> text(reg_tree, cex=0.5)
```

Now, let's evaluate the tree by performing prediction on the test set. The argument "type" is to specify the character string denoting the type of predicted value returned. If the rpart object is a classification tree, then the default is to return prob predictions, a matrix whose columns are the probability of the first, second, etc. class. Otherwise, a vector result is returned.

```
> test$pred <- predict(reg_tree, test)
> rmse(test$diabetes, test$pred)
```

As we can see, the root mean squared error of the prediction is 73.6882. Let's prune the tree to reduce the error.

Pruning the tree

There are types of pruning: pre-pruning and post-pruning.

Pre-pruning or also known as early stopping criteria can be performed by specifying these three parameters.

maxdepth: This parameter is used to set the maximum depth of a tree. Depth is the length of the longest path from a Root node to a Leaf node. This parameter will the stop the tree building when the depth is equal to the value set for maxdepth.

minsplit: This parameter is used to set the minimum number of samples that must exist in a node for a split to be attempted. For example, if minsplit is set to 5, then a node can be further split when the number of samples is more than 5.

minbucket: This parameter is used to set the minumum number of samples that can be present in a node. For example, if minbucket is set to 5, then a node should have at least 5 samples. We should also take care of not overfitting the tree by specifying too small value of minbucket. If only one of minbucket or minsplit is specified, the code either sets minsplit to minbucket*3 or minbucket to minsplit/3, as appropriate.

Let's build the classification tree by specifying the parameters. Then we evaluate the tree using the test set.

```
> reg_tree_es <- rpart(diabetes ~ ., train, method="anova",
  control=rpart.control(cp=0, maxdepth=6, minsplit=70))

> test$pred2 <- predict(reg_tree_es, test)

> rmse(test$diabetes, test$pred2)
```

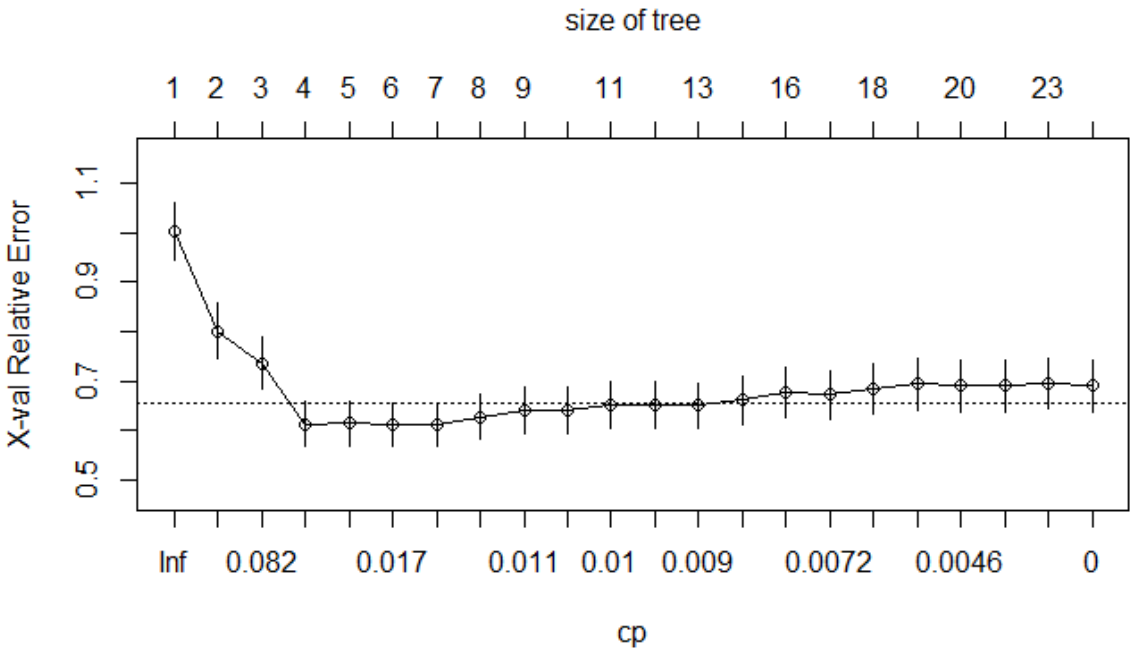
Use the plot() function to display the tree. We can see that the error has been reduced to 69.9848.

The second type of pruning is post-pruning. The method calculate the cost of complexity of building the tree. Then, a simpler (pruned) tree is selected based on the cost complexity (cp) parameter. To display the cp table of the tree, use printcp() function as follows.

```
> printcp(reg_tree)

Regression tree:
rpart(formula = diabetes ~ ., data = train, method = "anova",
    control = rpart.control(cp = 0))

Variables actually used in tree construction:
[1] age bmi bp  s2  s3  s4  s5  s6  sex

Root node error: 1795916/309 = 5812

n= 309

           CP nsplit rel error  xerror     xstd
1  0.3020824      0   1.00000 1.00453 0.058033
2  0.0944334      1   0.69792 0.80219 0.056212
3  0.0707153      2   0.60348 0.73697 0.053715
4  0.0246859      3   0.53277 0.61402 0.044772
5  0.0180307      4   0.50808 0.61433 0.043851
6  0.0166934      5   0.49005 0.61347 0.043695
7  0.0124328      6   0.47336 0.61201 0.044062
8  0.0122479      7   0.46093 0.62760 0.045525
9  0.0107084      8   0.44868 0.63985 0.046721
10 0.0104724      9   0.43797 0.64136 0.046411
11 0.0097205     10   0.42750 0.65188 0.046704
12 0.0092005     11   0.41778 0.65141 0.046665
13 0.0087960     12   0.40858 0.65031 0.046956
14 0.0075866     13   0.39978 0.66134 0.048851
15 0.0074964     15   0.38461 0.67688 0.049982
16 0.0068864     16   0.37711 0.67250 0.049690
17 0.0049056     17   0.37022 0.68538 0.050620
18 0.0048798     18   0.36532 0.69478 0.051973
19 0.0042703     19   0.36044 0.69030 0.051830
20 0.0037356     20   0.35617 0.69033 0.051715
21 0.0031197     22   0.34870 0.69510 0.051793
22 0.0000000     23   0.34558 0.69080 0.051855
```

We can also display the cp value against the cross-validated error using plotcp() function.

```
> plotcp(reg_tree)
```



The relative error (rel error) is $1 - R^2$, similar to linear regression. The xerror is cross-validated error and xstd and standard deviation of the cross-validated error.

Here, we want the cp value (with a simpler tree) that minimizes the xerror.

```
> bestcp <-
  reg_tree$cptable[which.min(reg_tree$cptable[,"xerror"]),"CP"]
```

Now, using the best cp value, we build the tree using rpart by specifying the best cp value. The evaluate the tree using the test set.

```
> reg_tree_prune <- rpart(diabetes ~ ., train, method="anova",
  control=rpart.control(cp=bestcp))

> test$pred3 <- predict(reg_tree_prune, test)

> rmse(test$diabetes, test$pred3)
```

As we can see, the error has been further reduced by about 0.6.

We create a postscript of the tree as follows.

```
> post(reg_tree_prune, file = "reg_tree_prune.ps", title = "Regression
  tree for Diabetes dataset")
```

The created ps file can be used to generate a pdf file of the tree.