**R: CART Tutorial**

*Goal:* Use the classification and regression tree (CART) model to predict baseball players’ salaries (regression tree) and heart disease status (classification tree).

*Data:* Hitters dataset from the ISLR package

**Table of Contents**

1 --- Hitters Data: Setup and Description

2 --- Regression Tree

2 --- Unpruned

3 --- Pruning with Cross-validation

5 --- Heart Disease Data: Setup and Description

6 --- Classification Tree

6 --- Unpruned

7 --- Pruning with Cross-validation

**Hitters Data: Setup and Description**

First, install the following packages. Don’t forget to use install.packages(“*packagename*”) if using for the first time. **ISLR** provides the hitters data, while **rpart** and **rpart.plot** provide decision tree models and plotting. **ggplot2** is used for plotting.

library(ISLR)

library(rpart)

library(rpart.plot)

library(ggplot2)

Next, we load the data, clean by removing NA and scaling the hitter salary, and perform a train-test split. In the Hitters dataset, each observation represents a batter, with 20 variables recorded that represent different aspects of their performance. All 20 variables described using the summary command.

# Set the seed

set.seed(200)

# Load the data

data(Hitters)

Hitters <- na.omit(Hitters) #Remove NA for demo

Hitters$Salary <- log(Hitters$Salary)

# Perform train-test split

hitter.split <- sample(1:nrow(Hitters), size=nrow(Hitters) \* 0.7)

h.train <- Hitters[hitter.split,]

h.test <- Hitters[-hitter.split,]

# Summarize data

summary(Hitters)

A close-up of a document

Description automatically generated with low confidence

**Regression Tree**

With the data cleaned, we can construct a CART model and output a graphical representation of the tree using the following code. The performance of the model at each “level” of the tree is displayed in the table, which evaluates MSE using 10-fold cross-validation.

Use “?rpart” and “?rpart.control” to query detailed help documents for definitions of each parameter.

Parameters of the **rpart** function:

* method = “anova” – a technique for splitting and indicates y is a continuous variable. Anova splitting has no parameters and is simple. You could choose “poisson” if you have count variables. See details in the R help document.
* control = rpart.control(minbucket = 6) – “minbucket” indicates the minimum number of observations required to be in a terminal leaf node. “rpart.control” can be used to specify other parameters as well.

Output results:

* CP = “complexity parameter”. With anova splitting, this means that the overall R-squared must increase by cp at each step. The main role of this parameter is to save computing time by pruning off splits that are obviously not worthwhile. Essentially, the user informs the program that any split which does not improve the fit by cp will likely be pruned off by cross-validation, and that hence the program need not pursue it.
* nsplit = numbers of splits (number of leaf nodes - 1)
* rel\_error = SSE at each CP relative to the root node SSE.
* xerror = cross-validated SSE for different numbers of splits (nsplit) (default k = 10);
* xstd = standard error of xerror.

# Construct Regression Tree on training set

example.cart <- rpart(formula = Salary ~ Years + Hits + RBI + PutOuts + Runs + Walks, data = h.train, method = "anova", control = rpart.control(minbucket = 6))

# Plot the regression tree from rpart

prp(example.cart, roundint = FALSE)

Diagram

Description automatically generated

# View the performance at each level of the tree

example.cart$cptable

CP nsplit rel error xerror xstd

1 0.46163334 0 1.0000000 1.0087334 0.08122696

2 0.11779898 1 0.5383667 0.5440761 0.06041888

3 0.04780978 2 0.4205677 0.4422747 0.06154261

4 0.02987700 3 0.3727579 0.4542102 0.06302094

5 0.02105090 4 0.3428809 0.4517177 0.06786611

6 0.01746041 5 0.3218300 0.4378974 0.07019741

7 0.01326326 6 0.3043696 0.4324530 0.07018232

8 0.01313420 7 0.2911063 0.4219593 0.06722881

9 0.01152560 8 0.2779721 0.4125124 0.06692748

10 0.01000000 9 0.2664465 0.3977695 0.06660225

It is preferable to “prune” the tree by limiting its evaluation at a point where performance does not significantly improve. Below, we test the tree’s performance on the 70-30 split of training and testing compared to the 10-fold cross-validation.

# Set up a grid of 10 potential alpha values

cp.param <- example.cart$cptable

train.mse <- double(10)

cv.mse <- double(10)

test.mse <- double(10)

# Calculate the average MSE on the train set, the test set, and the CV

for (i in 1:10) {

alpha <- cp.param[i, 'CP']

train.mse[i] <- mean((h.train$Salary - predict(prune(example.cart, cp=alpha), newdata = h.train))^2)

cv.mse[i] <- cp.param[i, 'xerror'] \* cp.param[i, 'rel error']

test.mse[i] <- mean((h.test$Salary - predict(prune(example.cart, cp=alpha), newdata = h.test))^2)

}

# Print MSE values

train.mse

[1] 0.7798341 0.4198367 0.3279730 0.2906893 0.2673902

[6] 0.2509740 0.2373578 0.2270146 0.2167721 0.2077841

test.mse

[1] 0.8282354 0.4874365 0.3975210 0.3940544 0.4086056

[6] 0.4172264 0.4044754 0.4089485 0.4571686 0.4564165

cv.mse

[1] 1.0087334 0.2929124 0.1860064 0.1693104 0.1548854 0.1409285 0.1316255

[8] 0.1228350 0.1146669 0.1059843

# Plot training, CV and testing errors at # of Splits

matplot(cp.param[,'nsplit'], cbind(train.mse, cv.mse, test.mse), pch=19, col=c("red", "black", "blue"), type="b", ylab="Mean Squared Error", xlab="# of Splits")

legend("right", c('Train', 'CV', 'Test'),col=seq\_len(3),cex=0.8,fill=c("red", "black", "blue"))

Chart, line chart

Description automatically generated

plotcp(example.cart)

Chart

Description automatically generated

# Print pruned and unpruned train, test, and all error

# Error at the maximum splits

train.mse[10] # Unpruned train error

[1] 0.2077841

test.mse[10] # Unpruned test error

[1] 0.4564165

cv.mse[10] # Unpruned all error

[1] 0.1059843

# Error at n=2 splits

train.mse[3] # pruned train error

[1] 0.327973

test.mse[3] # pruned test error

[1] 0.397521

cv.mse[3] # pruned all error

[1] 0.1860064

At the 3rd complexity value, n = 2, the tree attains the best performance on the 30% test set. We also observe a local minimum here on the graph of 10-fold cross validation error. Hence, we will use a pruned tree with n = 2, which we plot below and compare to n = 3 splits

Visualization is necessary to ensure the tree is not creating extraneous, trivial regions with few points being predicted. As we see, a tree with 3 splits creates a small trivial box in the center which does not majorly improve predictions.

# Plot the actual partitions of the tree

region <- ifelse(h.train$Years < 4.5, 'Region 3', ifelse(h.train$Hits < 118, 'Region 1', 'Region 2'))

ggplot(data=h.train) + geom\_point(aes(x=Years, y=Hits, color=Salary, shape=region)) +

geom\_vline(xintercept = 4.5) +

geom\_segment(x = 4.5, y= 118, xend=30, yend=118) +

theme\_minimal()

Chart, scatter chart

Description automatically generated

# Result of adding an additional split; creates trivial region

region <- ifelse(h.train$Years < 4.5, 'Region 4', ifelse(h.train$Hits < 111, 'Region 1', ifelse(h.train$Hits < 118, 'Region 2', 'Region 3')))

ggplot(data=h.train) +

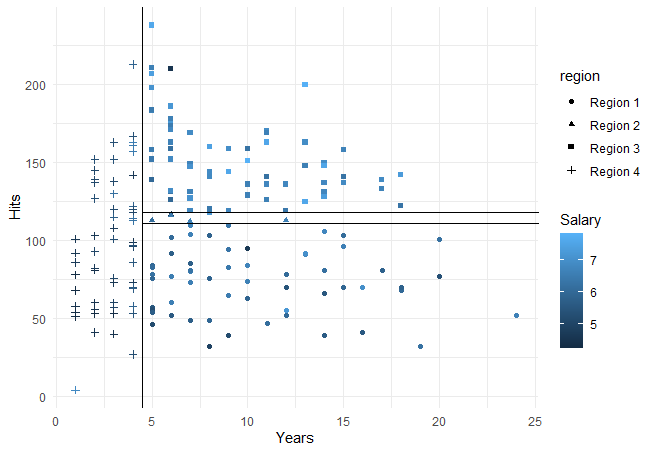
geom\_point(aes(x=Years, y=Hits, color=Salary, shape=region)) +

geom\_vline(xintercept = 4.5) +

geom\_segment(x = 4.5, y= 118, xend=30, yend=118) +

geom\_segment(x = 4.5, y= 111, xend=30, yend=111) +

theme\_minimal()



The table below compares the unpruned performance to the pruned performance for the regression tree, as output from the **train.mse, test.mse,** and **cv.mse** previously.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Unpruned Tree | Pruned Tree |
| Splits | | 9 | 2 |
| MSE | Train | 0.208 | 0.328 |
| Test | 0.456 | 0.397 |
| All (Cross-validated) | 0.106 | 0.186 |

**Heart Data: Setup and Description**

The CART algorithm can also be used for classification. For this, we will use the Heart dataset, which contains observations of patients with variables representing medical information related to heart health. This will rely on the packages previously loaded from the Hitters data. First, we load in the data, set the random seed, remove NA values, and summarize the variables.

Heart <- read.csv('https://www.statlearning.com/s/Heart.csv')

Heart <- na.omit(Heart) # Remove NA

set.seed(490)

heart.split <- sample(1:nrow(Heart), size=nrow(Heart) \* 0.7)

heart.train <- Heart[heart.split,]

heart.test <- Heart[-heart.split,]

summary(Heart)

####

A screenshot of a computer

Description automatically generated with low confidence

**Classification Tree**

Next, we formulate the decision tree as a classification model and plot a graphical representation. The performance of the model at each “level” of the tree is displayed in the table, which evaluates accuracy using 10-fold cross-validation.

class.cart <- rpart(formula = AHD ~ ., data = heart.train, method = "class", control = rpart.control(minbucket = 2, xval = 10))

prp(class.cart, roundint = FALSE)

Diagram, schematic

Description automatically generated

cp.class.param <- class.cart$cptable

cp.class.param

CP nsplit rel error xerror xstd

1 0.53684211 0 1.0000000 1.0000000 0.07546786

2 0.04736842 1 0.4631579 0.4631579 0.06195999

3 0.02631579 3 0.3684211 0.4421053 0.06090569

4 0.02105263 6 0.2842105 0.4526316 0.06143946

5 0.01052632 8 0.2421053 0.5263158 0.06482254

6 0.01000000 18 0.1368421 0.5368421 0.06525873

Again, we can “prune” the tree by limiting its evaluation at a point where performance does not significantly improve. Below, we examine the tree’s performance on the 70-30 split of training and testing and compare to the 10-fold cross-validation from the output of our model. We see that testing error is minimized at the 5th complexity observation in the table (or a depth of 6), which we use to prune and graph the optimal tree.

# if the below throws an error, change 6 to 7

train.err <- double(6)

cv.err <- double(6)

test.err <- double(6)

for (i in 1:nrow(cp.class.param)) {

alpha <- cp.class.param[i, 'CP']

train.cm <- table(heart.train$AHD, predict(prune(class.cart, cp=alpha), newdata = heart.train, type='class'))

train.err[i] <- 1-sum(diag(train.cm))/sum(train.cm)

cv.err[i] <- cp.class.param[i, 'xerror'] \* cp.class.param[i, 'rel error']

test.cm <- table(heart.test$AHD, predict(prune(class.cart, cp=alpha), newdata = heart.test, type='class'))

test.err[i] <- 1-sum(diag(test.cm))/sum(test.cm)

}

# Print classification error (1 – accuracy) values

train.err

[1] 0.45893720 0.21256039 0.16908213 0.13043478

[5] 0.11111111 0.06280193

test.err

[1] 0.4666667 0.2888889 0.2444444 0.1888889 0.2111111

[6] 0.2555556

# Plot training, CV and testing errors at # of Splits/depth

matplot(cp.class.param[,'nsplit'], cbind(train.err, cv.err, test.err), pch=19, col=c("red", "black", "blue"), type="b", ylab="Loss/error", xlab="Depth/# of Splits")

legend("right", c('Train', 'CV', 'Test') ,col=seq\_len(3),cex=0.8,fill=c("red", "black", "blue"))

Timeline

Description automatically generated

plotcp(class.cart)

Chart, box and whisker chart

Description automatically generated

# Check CP table, when size of tree =4, the nsplit =3 and CP = 0.02631579 Prune the tree at nsplit =3 defined by the complexity parameter

prune.class.trees <- prune(class.cart, cp=cp.class.param[3,'CP'])

prp(prune.class.trees)

Diagram, radar chart

Description automatically generated

# Calculate confusion table and accuracy

conf.mat.tree <- table(heart.test$AHD, predict(prune.class.trees, type = 'class', newdata = heart.test))

conf.mat.tree

No Yes

No 32 16

Yes 6 36

acc <- sum(diag(conf.mat.tree))/sum(conf.mat.tree)

acc

[1] 0.7555556

# Print pruned and unpruned train, test, and all accuracy

# Error at the maximum splits

train.acc <- 1 - train.err[6] # Unpruned train accuracy

train.acc

[1] 0.9371981

test.acc <- 1 - test.err[6] # Unpruned test accuracy

test.acc

[1] 0.7444444

cv.acc <- 1 - cv.err[6] # Unpruned all accuracy

cv.acc

[1] 0.9265374

# accuracy at nsplit =3, and cp = 0.02631579

train.prune.acc <- 1 - train.err[3] # Pruned train accuracy

train.prune.acc

[1] 0.8309179

test.prune.acc <- 1 - test.err[3] # Pruned test accuracy

test.prune.acc

[1] 0.7555556

cv.prune.acc <- 1 - cv.err[3] # Pruned all accuracy

cv.prune.acc

[1] 0.8371191

The table below compares the unpruned performance to the pruned performance for the regression tree. Note that here we display accuracy which is 1 – error with error representing the performance as measured by the **rpart** package.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Unpruned Tree | Pruned Tree |
| Tree Size | | 18 | 4 |
| Accuracy | Train | 0.937 | 0.831 |
| Test | 0.744 | 0.756 |
| All (Cross-validated) | 0.927 | 0.837 |