# Rpart worksheet solutions

## Single Decision Tree

### rpart package in R

Aim: Create decision tree to predict incomes in ‘income’ data set. Assess accuracy of predictions

1. Set your working directory (where you have stored your datafile) using either setwd() or the 'Session' menu

setwd("C:/Users/Tim/OneDrive/SSA\_bigdata\_course/trees")

1. Install relevant packages: rpart, rpart.plot

install.packages('rpart')  
install.packages('rpart.plot')  
install.packages('ggplot2')  
install.packages('GGally')

1. Load packages:
2. Load dataset and save as 'a'

a <- read.csv('income.csv')[,-1]

1. Explore dataset using commands such as head(), summary(), str(), ggplot(), or boxplot()
2. Split data into test and training sets

set.seed(100)  
sample<- sample(1:nrow(a), 2/3\*nrow(a), replace=F)  
train<- a[sample,]  
test<- a[-sample,]  
  
dim(train)

## [1] 18336 10

dim(test)

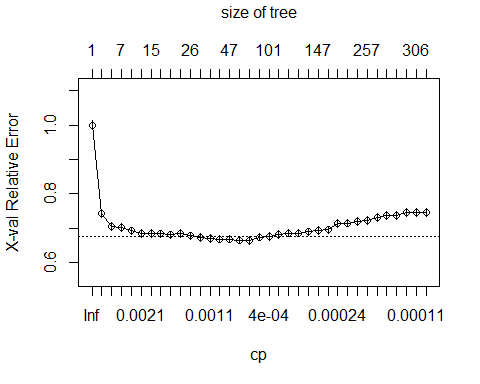
## [1] 9168 10

1. Create first tree, then prune using cp value as control.

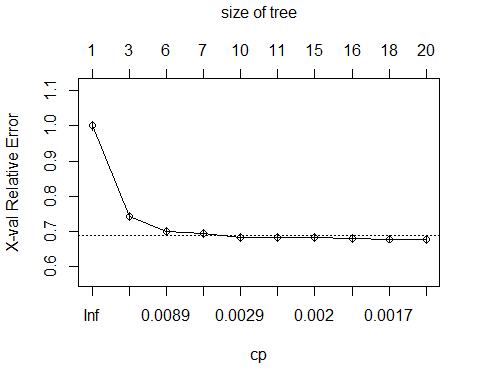
fit<- rpart(class~., data= train, control = rpart.control(cp= .0001))  
# fit  
printcp(fit)

##   
## Classification tree:  
## rpart(formula = class ~ ., data = train, control = rpart.control(cp = 1e-04))  
##   
## Variables actually used in tree construction:  
## [1] age education gender hrs.per.week   
## [5] marital.status occupation race relationship   
## [9] workclass   
##   
## Root node error: 4654/18336 = 0.25382  
##   
## n= 18336   
##   
## CP nsplit rel error xerror xstd  
## 1 0.13021057 0 1.00000 1.00000 0.012662  
## 2 0.01604355 2 0.73958 0.74130 0.011372  
## 3 0.00494199 5 0.69145 0.70498 0.011152  
## 4 0.00379602 6 0.68651 0.70155 0.011131  
## 5 0.00214869 9 0.67512 0.69145 0.011068  
## 6 0.00204125 10 0.67297 0.68393 0.011020  
## 7 0.00193382 14 0.66480 0.68350 0.011017  
## 8 0.00182639 15 0.66287 0.68285 0.011013  
## 9 0.00150408 17 0.65922 0.68028 0.010997  
## 10 0.00139665 19 0.65621 0.68328 0.011016  
## 11 0.00118178 25 0.64719 0.67684 0.010975  
## 12 0.00112806 32 0.63881 0.67211 0.010944  
## 13 0.00107434 39 0.62720 0.66996 0.010930  
## 14 0.00085948 43 0.62291 0.66781 0.010916  
## 15 0.00064461 46 0.62033 0.66738 0.010914  
## 16 0.00053717 48 0.61904 0.66459 0.010895  
## 17 0.00050136 50 0.61796 0.66416 0.010893  
## 18 0.00042974 75 0.59712 0.67340 0.010953  
## 19 0.00037602 100 0.58466 0.67555 0.010967  
## 20 0.00032230 104 0.58315 0.68178 0.011006  
## 21 0.00030696 117 0.57843 0.68543 0.011030  
## 22 0.00030082 124 0.57628 0.68543 0.011030  
## 23 0.00028649 131 0.57391 0.68865 0.011050  
## 24 0.00026859 146 0.56962 0.69360 0.011081  
## 25 0.00025784 154 0.56747 0.69467 0.011088  
## 26 0.00021487 170 0.56296 0.71251 0.011199  
## 27 0.00019338 239 0.54641 0.71444 0.011210  
## 28 0.00017906 250 0.54426 0.71831 0.011234  
## 29 0.00017190 256 0.54319 0.72260 0.011260  
## 30 0.00014325 261 0.54233 0.73077 0.011309  
## 31 0.00012278 268 0.54104 0.73700 0.011346  
## 32 0.00011937 285 0.53868 0.73700 0.011346  
## 33 0.00011720 294 0.53760 0.74474 0.011392  
## 34 0.00010743 305 0.53631 0.74474 0.011392  
## 35 0.00010000 329 0.53373 0.74452 0.011391

plotcp(fit)



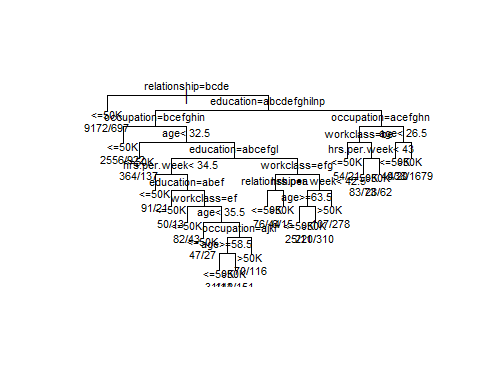
#prune to cp greater than \_\_\_\_\_\_\_\_\_?  
  
fit<- rpart(class~., data=train, control = rpart.control(cp= 0.0014))  
# fit  
plotcp(fit)



* What other options in rpart package under rpart.control?
* Why does it skip more than one split in each step?

1. Plot the tree

plot(fit, margin=0.05, uniform = T)  
text(fit, use.n=T, cex=0.7)



Toy with the values in the functions

What does 'cex' do?

cex sets the text size of the labels on the rpart plot.

What does 'uniform' do?

Uniform makes the branches the same (vertical) length. With non-uniform (uniform = F) branch lengths are proportional to the error in the fit.

1. Run train set through fitted tree. Exclude 'class' column

res.fit<- predict(fit, train[,-10])  
head(res.fit)

## <=50K >50K  
## 8465 0.7349051 0.26509488  
## 7087 0.9293748 0.07062519  
## 15190 0.7349051 0.26509488  
## 1551 0.9293748 0.07062519  
## 12886 0.7349051 0.26509488  
## 13304 0.9293748 0.07062519

res.fit2<- predict(fit, train[,-10], type = 'class')  
head(res.fit2)

## 8465 7087 15190 1551 12886 13304   
## <=50K <=50K <=50K <=50K <=50K <=50K   
## Levels: <=50K >50K

1. Create data frame with train actual class 'answers' and train 'predicted' classes as the two columns

head(test)

## age workclass education marital.status occupation  
## 2 50 Self-emp-not-inc Bachelors Married-civ-spouse Exec-managerial  
## 3 38 Private HS-grad Divorced Handlers-cleaners  
## 7 31 Private Masters Never-married Prof-specialty  
## 13 32 Private HS-grad Never-married Machine-op-inspct  
## 14 38 Private 11th Married-civ-spouse Sales  
## 16 40 Private Doctorate Married-civ-spouse Prof-specialty  
## relationship race gender hrs.per.week class  
## 2 Husband White Male 13 <=50K  
## 3 Not-in-family White Male 40 <=50K  
## 7 Not-in-family White Female 50 >50K  
## 13 Unmarried White Male 40 <=50K  
## 14 Husband White Male 50 <=50K  
## 16 Husband White Male 60 >50K

# Make sure row numbers match  
ans.fit<- data.frame('ans'=train$class, 'predict' = res.fit2)  
dim(ans.fit)

## [1] 18336 2

head(ans.fit)

## ans predict  
## 8465 <=50K <=50K  
## 7087 <=50K <=50K  
## 15190 <=50K <=50K  
## 1551 <=50K <=50K  
## 12886 <=50K <=50K  
## 13304 <=50K <=50K

1. Create confusion matrix for test set

table(ans.fit$ans, ans.fit$predict)

##   
## <=50K >50K  
## <=50K 12671 1011  
## >50K 2043 2611

1. Repeat steps 9 through 11 with the test set instead of train set. This will give a more realistic confusion matrix, print confusion matrix below:

res.fit\_test<- predict(fit, test [,-10], type = 'class')  
ans.fit\_test<- data.frame('ans'=test$class, 'predict' = res.fit\_test)  
dim(ans.fit\_test)

## [1] 9168 2

table(ans.fit\_test$ans, ans.fit\_test$predict)

##   
## <=50K >50K  
## <=50K 6295 532  
## >50K 1049 1292