## Trip History Data

Each quarter, they publish down loadable files of Capital Bikeshare trip data. The data includes:

* Duration – Duration of trip
* Start Date – Includes start date and time
* End Date – Includes end date and time
* Start Station – Includes starting station name and number
* End Station – Includes ending station name and number
* Bike Number – Includes ID number of bike used for the trip
* Member Type – Indicates whether user was a "registered" member (Annual Member, 30-Day Member or Day Key Member) or a "casual" rider (Single Trip, 24-Hour Pass, 3-Day Pass or 5-Day Pass)

library(e1071)

library(caTools)

mydata<-read.csv(file="/Users/anita/Desktop/tripdata.csv",header=TRUE,sep=",")

View(mydata)

dim(mydata)

#consider column1,4,6,9 - output class

subset\_mydata <- mydata[,c(1,4,6,9)]

temp\_field <- sample.split(subset\_mydata,SplitRatio=0.9)

train <- subset(subset\_mydata, temp\_field==TRUE)

View(train)

test <- subset(subset\_mydata, temp\_field==FALSE)

View(test)

summary(train)

summary(test)

head(train)

head(test)

train(1:10,)

View(train)

tail(train)

tail(test)

dim(test)

dim(train)

library(rpart)

# rpart parameters are as : first is outcome followed by "~." it indicates the outcome to be learned against all other fields of dataset # second is data frame and third is method **"class"** for a classification tree and **"anova"** for a regression tree.

# The **rpart** algorithm works by splitting the dataset recursively, which means that the subsets that arise from a # split are further split until a predetermined termination criterion is reached. rpart() package is used to create the tree. It allows us to grow the whole tree using all the attributes present in the data.

fit <- rpart(train$Member.type~.,data=train,method="class")

plot(fit)

text(fit)

#test excluding last colm

pred<- predict(fit,newdata=test[,-4],type=("class"))

table(pred,test$Member.type,dnn=c("predicted","actual")) // Confusion matrix

actual

predicted Casual Member

Casual 12667 4746

Member 12816 63299

output <- cbind(test,pred)

View(output)

# Validation of decision tree using the ‘Complexity Parameter’ To validate the model we use the printcp . ‘CP’ stands for Complexity Parameter of the tree. Displays the cp table for fitted itree object. This function provides the optimal prunings based on the cp value. We prune the tree to avoid any overfitting of the data. The convention is to have a small tree and the one with least cross validated error given by printcp() function i.e. ‘xerror’.

printcp(fit)

Classification tree:

rpart(formula = train$Member.type ~ ., data = train, method = "class")

Variables actually used in tree construction:

[1] Duration End.station.number Start.station.number

Root node error: 76820/280587 = 0.27378

n= 280587

CP nsplit rel error xerror xstd

1 0.210544 0 1.00000 1.00000 0.0030747

2 0.030422 1 0.78946 0.78700 0.0028350

3 0.019995 2 0.75903 0.76660 0.0028080

4 0.019175 4 0.71904 0.73577 0.0027656

5 0.011013 5 0.69987 0.70461 0.0027208

6 0.010000 6 0.68886 0.69378 0.0027048

From the above mentioned list of cp values, we can select the one having the least cross-validated error and use it to prune the tree. The value of cp should be least, so that the cross-validated error rate is minimum.

To select this, you can make use of this :

opt<- which.min(fit$cptable[,"xerror"])

cp <- fit$cptable[opt,"CP"] This function returns the optimal cp value associated with the minimum error.

# Plotcp() provides a graphical representation to the cross validated error summary.

plotcp(fit)

#prune tree to create an optimal decision tree :

pruned\_model<-prune(fit,cp)

plot(fit)

text(fit)

#find proportion of correct prediction using test set

rpart\_pruned\_predict<- predict(pruned\_model,test[,-4],type="class")

rpart\_pruned\_predict