We will fit single trees and random forests to the training data and test them at the end using the test set.

1. Reset the seed for the Mersenne Twister to 8646824. Code two models: a classification tree using all defaults, followed by a second tree setting cp=0. This way, every student should have the same CV results within rpart.

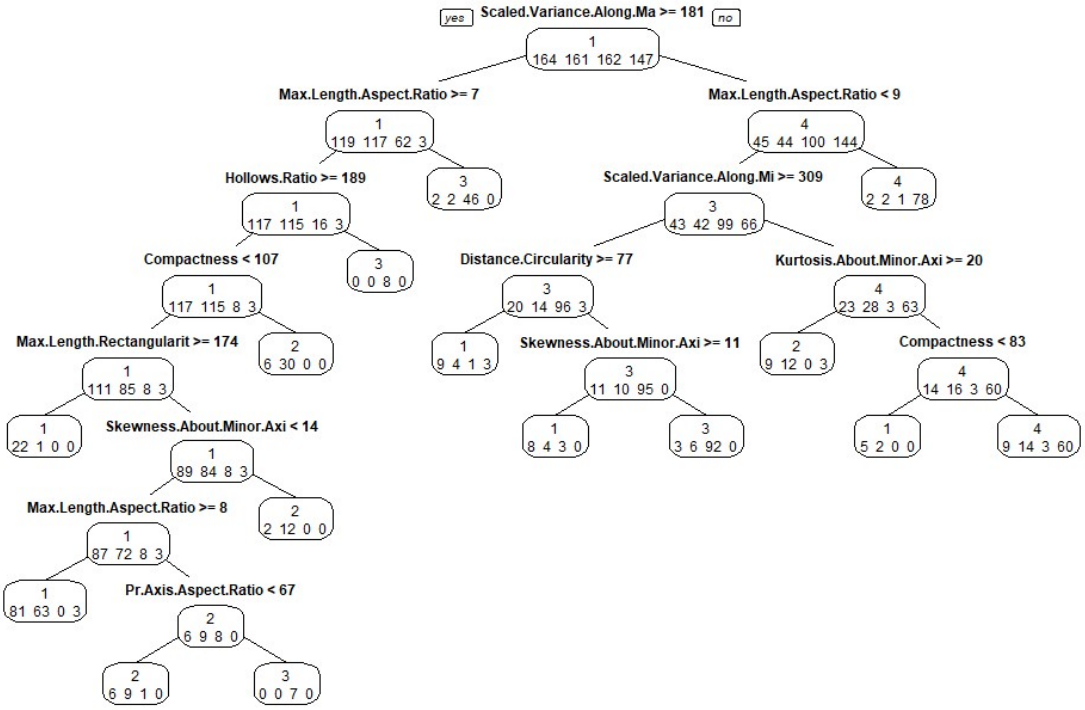
```
library(MASS)   # For ridge regression
library(glmnet) # For LASSO
library(nnet) # Fits neural net models
library(rgl)
library(FNN)
library(rpart)        # For fitting classification trees
library(rpart.plot)   # For plotting classification trees
library(randomForest) # For random forests
library(gbm)          # For boosting
source("Helper Functions.R")


# 1. Reset the seed for the Mersenne Twister to 8646824. Code two models: a classification
# tree using all defaults, followed by a second tree setting cp=0. This way, every student
# should have the same CV results within rpart.
set.seed(8646824, kind = "Mersenne-Twister")
vehdata = read.csv("vehicle.csv")

perm <- sample (x= nrow ( vehdata ))
data.train <- vehdata [ which ( perm <= 3* nrow ( vehdata )/4) , ]
data.valid <- vehdata [ which ( perm > 3* nrow ( vehdata )/4) , ]
Y.valid = data.valid[,19]

# ind.random = sample(1:n)
# data.train = data[ind.random <= n.train,]
# data.valid = data[ind.random > n.train,]
# Y.valid = data.valid[,1]
#
#
# X.train = set1[,-19]
# X.valid = set2[,-19]
# Y.train = set1[, 19]
# Y.train = as.factor(Y.train)
# Y.valid = set2[, 19]
# Y.valid = as.factor(Y.valid)

### Fitting classification trees is mostly the same as fitting regression
### trees. We use the rpart() function from the rpart package. This function
### uses model formula/data frame syntax. We can set the CP using the cp
### input; usually this is set to 0. When doing classification, we have to
### set method="class".
fit.tree.default = rpart(class ~ ., data = data.train, method = "class")
fit.tree.full = rpart(class ~ ., data = data.train, method = "class",
                      cp = 0)
```

(a) **Print out the CP table for the default tree.** Is there any evidence that larger trees are needed than what the defaults allow? **Explain.**

We will no longer use this run. All remaining questions refer to the cp=0 object.

```
prp(fit.tree.default, type = 1, extra = 1, main = "Default Tree")
```

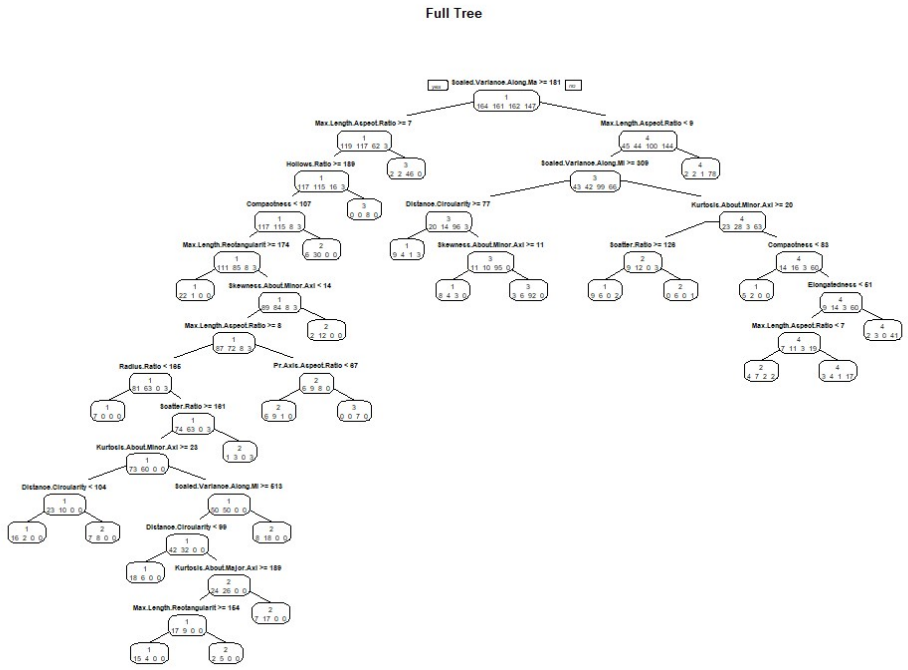
Default Tree



-> It is showing that some nodes are not separated even though they have many number of different classes. For example, the child node of compactness < 107, It has 6 class 1 and 30 class 2.

(b) **Print out the CP table for the tree with cp=0**. **How many splits are suggested as best?**

```
prp(fit.tree.full, type = 1, extra = 1, main = "Full Tree")
```



The graph is showing that 25 splits are good.

(c) Use Tom's code to find the optimal CP value for pruning using minimum CV and using the 1-SE rule. **Report these two numbers**

```
# (c) Use Tom's code to find the optimal CP value for pruning using minimum CV and
# using the 1-SE rule. Report these two numbers

### We can tune our tree using information from the CP table.
info.tree = fit.tree.full$cptable

### The actual process of tuning is the same. We can use Tom's code to do this

# Find location of minimum error
minrow <- which.min(info.tree[,4])
# Take geometric mean of cp values at min error and one step up
cplow.min <- info.tree[minrow,1]
cpup.min <- ifelse(minrow==1, yes=1, no=info.tree[minrow-1,1])
cp.min <- sqrt(cplow.min*cpup.min)

# Find smallest row where error is below +1SE
se.row <- min(which(info.tree[,4] < info.tree[minrow,4]+info.tree[minrow,5]))
# Take geometric mean of cp values at min error and one step up
cplow.1se <- info.tree[se.row,1]
cpup.1se <- ifelse(se.row==1, yes=1, no=info.tree[se.row-1,1])
cp.1se <- sqrt(cplow.1se*cpup.1se)

# Creating a pruned tree using the CV-min rule.
fit.tree.min <- prune(fit.tree.full, cp=cp.min)
# Creating a pruned tree using the CV-1se rule.
fit.tree.1se <- prune(fit.tree.full, cp=cp.1se)

### Now that we have our pruned trees, let's plot them.
prp(fit.tree.min, type = 1, extra = 1, main = "CV-min Tree")
prp(fit.tree.1se, type = 1, extra = 1, main = "CV-1se Tree")
```

```
> cp.min
        10
0.003364125

> cp.1se
[1] 0.005826836
```

(d) **Report training and test error of all three of these trees, full and two pruned.**

```
#Y.train fit.tree.full
pred.tree.full = predict(fit.tree.full, data.train, type = "class")
table(Y.train, pred.tree.full, dnn = c("Obs", "Pred"))
(mis.tree.full = mean(Y.train != pred.tree.full))

> (mis.tree.full = mean(Y.train != pred.tree.full))
[1] 0.1908517

#Y.test fit.tree.full
pred.tree.full = predict(fit.tree.full, data.valid, type = "class")
table(Y.valid, pred.tree.full, dnn = c("Obs", "Pred"))
(mis.tree.full = mean(Y.valid != pred.tree.full))

> (mis.tree.full = mean(Y.valid != pred.tree.full))
[1] 0.2924528

# Y.train fit.tree.min
pred.tree.min = predict(fit.tree.min, data.train, type = "class")
table(Y.train, pred.tree.min, dnn = c("Obs", "Pred"))
(mis.tree.min = mean(Y.train != pred.tree.min))

> (mis.tree.min = mean(Y.train != pred.tree.min))
[1] 0.192429

# Y.test fit.tree.min
pred.tree.min = predict(fit.tree.min, data.valid, type = "class")
table(Y.valid, pred.tree.min, dnn = c("Obs", "Pred"))
(mis.tree.min = mean(Y.valid != pred.tree.min))

> (mis.tree.min = mean(Y.valid != pred.tree.min))
[1] 0.2877358

# Y.train fit.tree.1se
pred.tree.1se = predict(fit.tree.1se, data.train, type = "class")
table(Y.train, pred.tree.1se, dnn = c("Obs", "Pred"))
(mis.tree.1se = mean(Y.train != pred.tree.1se))

> (mis.tree.1se = mean(Y.train != pred.tree.1se))
[1] 0.2003155
```

```
# Y.valid fit.tree.1se
pred.tree.1se = predict(fit.tree.1se, data.valid, type = "class")
table(Y.valid, pred.tree.1se, dnn = c("Obs", "Pred"))
(mis.tree.1se = mean(Y.valid != pred.tree.1se))
```

```
> (mis.tree.1se = mean(Y.valid != pred.tree.1se))
[1] 0.2924528
```

(e) **How does test error compare to other models?**

-> The test error doesn't show the lowest value (<0.2) but it shows better result than kernel density.