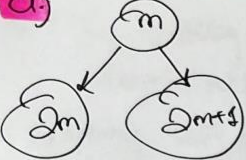


## Assignment 2

Alden Arduo - 31878039

### Question 1

a.)



$$G = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk})$$

$$\Delta(I) = I(m) - \frac{N_{2m}}{N_m} I(2m) - \frac{N_{2m+1}}{N_m} I(2m+1)$$

The best split is with the maximal impurity reduction  $\Delta(I)$ :

X1:  $\Delta(I) = 2\left(\frac{1}{3}\left(1 - \frac{1}{3}\right)\right) - \frac{3}{4}\left(2\left(\frac{2}{3} + \frac{1}{3}\right)\right) - \frac{1}{4}(0) = \frac{1}{6}$

X2:  $\Delta(I) = 2\left(\frac{1}{2}\left(1 - \frac{1}{2}\right)\right) - \frac{1}{2}\left(2\left(\frac{2}{5} + \frac{3}{5}\right)\right) - \frac{1}{2}\left(2\left(\frac{2}{5} + \frac{3}{5}\right)\right) = \frac{1}{50}$

X3:  $\Delta(I) = 2\left(\frac{1}{2}\left(1 - \frac{1}{2}\right)\right) - \frac{1}{2}\left(2\left(\frac{1}{2}\left(1 - \frac{1}{2}\right)\right)\right) - \frac{1}{2}\left(2\left(\frac{1}{2}\left(1 - \frac{1}{2}\right)\right)\right) = 0$

Therefore we split using X1 as it has the maximal  $\Delta(I)$ .

b.)

Splitting left daughter node with X2:

$$\Delta(I) = 2\left(\frac{1}{3} \times \frac{2}{3}\right) - \frac{2}{5}(0) - \frac{3}{5}\left(2\left(\frac{4}{9} \times \frac{5}{9}\right)\right) = \frac{4}{27}$$

Splitting left daughter node with X3:

$$\Delta(I) = 2\left(\frac{1}{3} \times \frac{2}{3}\right) - \frac{2}{3}\left(2\left(\frac{1}{2} \times \frac{1}{2}\right)\right) - \frac{1}{3}(0) = \frac{1}{9}$$

Splitting right daughter node with X2:

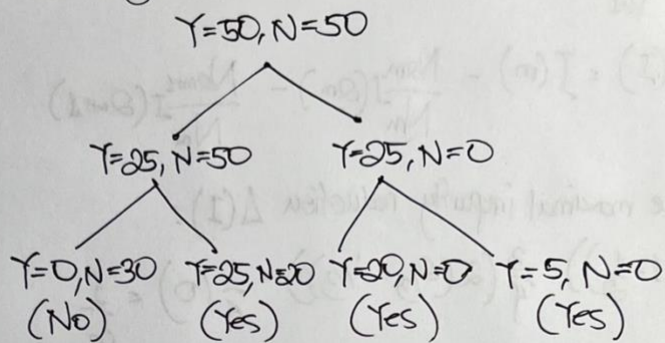
$$\Delta(I) = 0$$

Splitting right daughter node with X3:

$$\Delta(I) = 0$$

This is because right daughter node is not impure and has  $No = 0$ .

We will split both daughter nodes with  $T=25$  as it has a higher  $\Delta(I)$



c.

From tree in b, the second leaf node is classified as Yes, but there are 20 No's in this node classified incorrectly. There are 20 misclassified observations.

d.  $X_3$  splitting the root node into two daughter nodes.

Splitting left daughter node with  $X_2$  and  $X_1$ :

$$X_2: \Delta(I) = 2\left(\frac{1}{8} \times \frac{1}{8}\right) - \frac{1}{8}(0) - \frac{1}{8}(0) = \frac{1}{2} \star$$

$$X_1: \Delta(I) = 2\left(\frac{1}{8} \times \frac{1}{2}\right) - 1\left(2\left(\frac{1}{2} \times \frac{1}{8}\right)\right) - 0 = 0$$

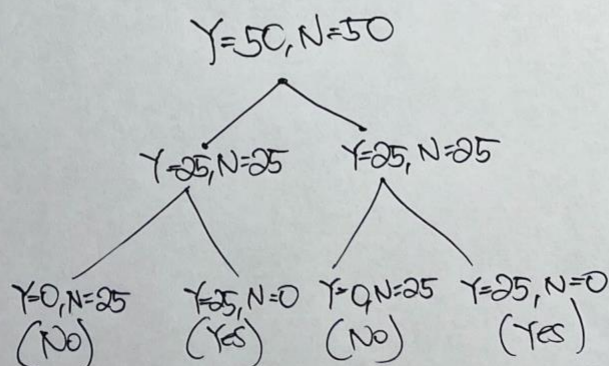
We split left daughter node with  $X_2$  as it has a higher impurity reduction  $\Delta(I)$ .

Splitting right daughter node with  $X_2$  and  $X_1$ :

$$X_2: \Delta(I) = 2\left(\frac{1}{8} \times \frac{1}{8}\right) - \frac{1}{2}\left(2\left(\frac{4}{5} \times \frac{1}{5}\right)\right) - \frac{1}{2}\left(2\left(\frac{4}{5} \times \frac{1}{5}\right)\right) = \frac{9}{50}$$

$$X_1: \Delta(I) = 2\left(\frac{1}{2} \times \frac{1}{2}\right) = \frac{1}{2} \star$$

We split right daughter node with  $X_1$  as it has a higher impurity reduction  $\Delta(I)$ .



e.

The tree in d is more accurate than tree in c as tree in d has no misclassified observations whereas tree in c has 20 misclassified observations.

This means that the greedy nature of CART is due to it choosing the locally optimal split for each iteration instead of thinking ahead and choosing the split that provides a better tree (less misclassified observations) in future steps.

For instance, the best split for the first iteration is X1 but it produces misclassified observations. The worst split for the first iteration is X3 but it produces no misclassified observations. Due to its greedy nature, CART chooses X1 instead of X3 without looking ahead.

## Question 2

(a)

```
autotrain <- read.csv("AutoTrain.csv")
autotest <- read.csv("AutoTest.csv")
autotrain = subset(autotrain, select = -c(name, X))
autotest = subset(autotest, select = -c(name, X))

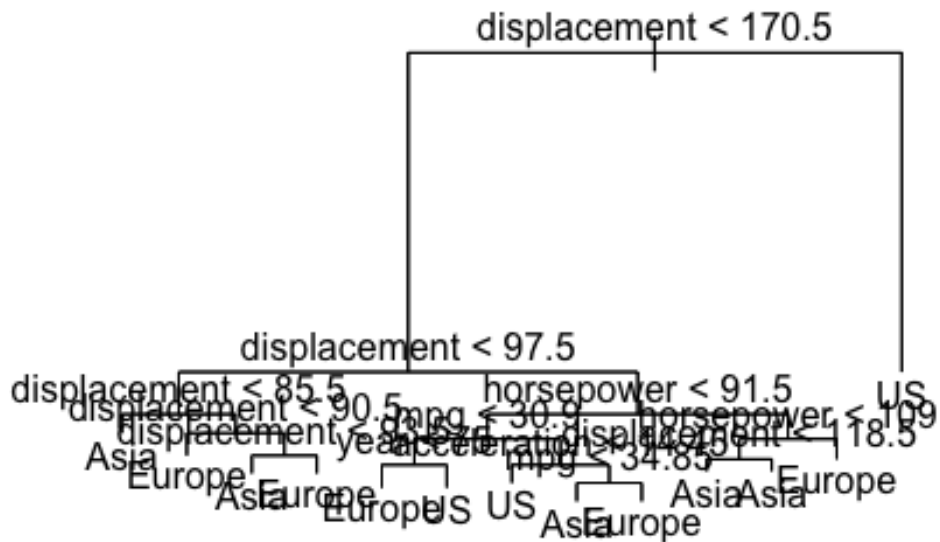
autotrain <- autotrain %>%
  mutate(origin=ifelse(origin==1,'US',origin),
           origin=ifelse(origin==2,'Europe',origin),
           origin=ifelse(origin==3,'Asia',origin))
autotrain$origin <- as.factor(autotrain$origin)

autotest <- autotest %>%
  mutate(origin=ifelse(origin==1,'US',origin),
           origin=ifelse(origin==2,'Europe',origin),
           origin=ifelse(origin==3,'Asia',origin))
autotest$origin <- as.factor(autotest$origin)

tree.autotrain = tree(origin ~ ., autotrain)
summary(tree.autotrain)

##
## Classification tree:
## tree(formula = origin ~ ., data = autotrain)
## Variables actually used in tree construction:
## [1] "displacement" "horsepower" "mpg" "year" "accelerat
ion"
## Number of terminal nodes: 13
## Residual mean deviance: 0.4966 = 90.87 / 183
## Misclassification error rate: 0.1071 = 21 / 196
```

```
plot(tree.autotrain)
text(tree.autotrain,pretty=0)
```



```
tree.autotrain
## node), split, n, deviance, yval, (yprob)
##      * denotes terminal node
##
## 1) root 196 361.600 US ( 0.20918 0.16837 0.62245 )
##    2) displacement < 170.5 97 208.000 Asia ( 0.42268 0.34021 0.23711 )
##      4) displacement < 97.5 43 70.030 Asia ( 0.60465 0.34884 0.04651 )
##        8) displacement < 85.5 16 12.060 Asia ( 0.87500 0.12500 0.00000 )
##          *
##            9) displacement > 85.5 27 48.880 Europe ( 0.44444 0.48148 0.07407 )
##              )
##                18) displacement < 90.5 10 16.040 Europe ( 0.10000 0.70000 0.2000
##                  0 ) *
##                    19) displacement > 90.5 17 22.070 Asia ( 0.64706 0.35294 0.00000 )
##                      )
##                        38) displacement < 93.5 7 0.000 Asia ( 1.00000 0.00000 0.00000 )
##                          ) *
##                            39) displacement > 93.5 10 13.460 Europe ( 0.40000 0.60000 0.00
##                              000 ) *
```



```

##      5) displacement > 97.5 54 117.600 US ( 0.27778 0.33333 0.38889 )
##      10) horsepower < 91.5 35 69.430 US ( 0.17143 0.28571 0.54286 )
##      20) mpg < 30.9 20 24.430 US ( 0.00000 0.30000 0.70000 )
##      40) year < 75 10 13.460 Europe ( 0.00000 0.60000 0.40000 ) *
##      41) year > 75 10 0.000 US ( 0.00000 0.00000 1.00000 ) *
##      21) mpg > 30.9 15 32.560 Asia ( 0.40000 0.26667 0.33333 )
##      42) acceleration < 14.45 5 5.004 US ( 0.20000 0.00000 0.80000
) *
##      43) acceleration > 14.45 10 18.870 Asia ( 0.50000 0.40000 0.100
00 )
##      86) mpg < 34.85 5 5.004 Asia ( 0.80000 0.00000 0.20000 ) *
##      87) mpg > 34.85 5 5.004 Europe ( 0.20000 0.80000 0.00000 ) *
##      11) horsepower > 91.5 19 36.290 Asia ( 0.47368 0.42105 0.10526 )
##      22) horsepower < 109 12 20.820 Asia ( 0.66667 0.16667 0.16667 )
##      44) displacement < 118.5 7 8.376 Asia ( 0.71429 0.28571 0.0000
0 ) *
##      45) displacement > 118.5 5 6.730 Asia ( 0.60000 0.00000 0.4000
0 ) *
##      23) horsepower > 109 7 5.742 Europe ( 0.14286 0.85714 0.00000 )
*
##      3) displacement > 170.5 99 0.000 US ( 0.00000 0.00000 1.00000 ) *

set.seed(1)
tree.pred.train = predict(tree.autotrain, autotrain, type="class")
tree.train.error = mean(tree.pred.train != autotrain$origin)
tree.train.error

## [1] 0.1071429

tree.pred.test = predict(tree.autotrain, autotest, type="class")
tree.test.error = mean(tree.pred.test != autotest$origin)
tree.test.error

## [1] 0.3112245

```

The classification tree has a residual mean deviance of 0.4966(4dp). There are 13 terminal nodes. Five variables are used to create this classification tree which are displacement, horsepower, mpg, year and acceleration. The most important variable is displacement.

The testing error of 0.3112245(3dp) is higher than the training error of 0.1071429(3dp) implying that there are more errors in the testing data which is bad.

(b)

```

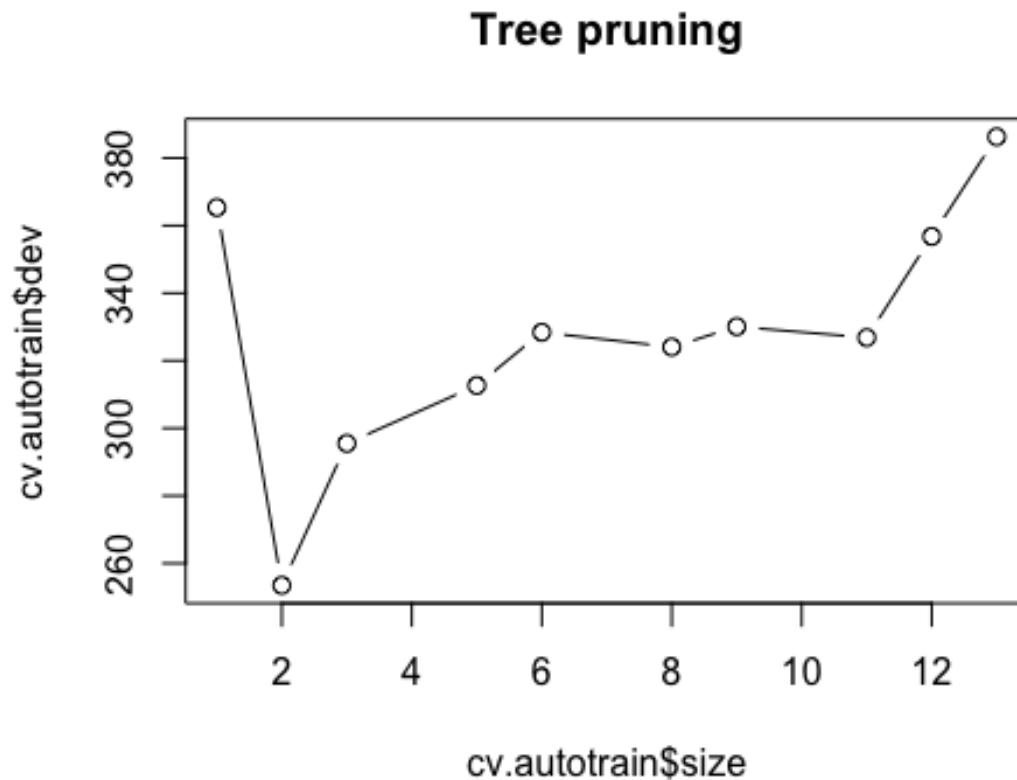
set.seed(3)
cv.autotrain = cv.tree(tree.autotrain)
cv.autotrain

## $size
## [1] 13 12 11 9 8 6 5 3 2 1
##
## $dev

```

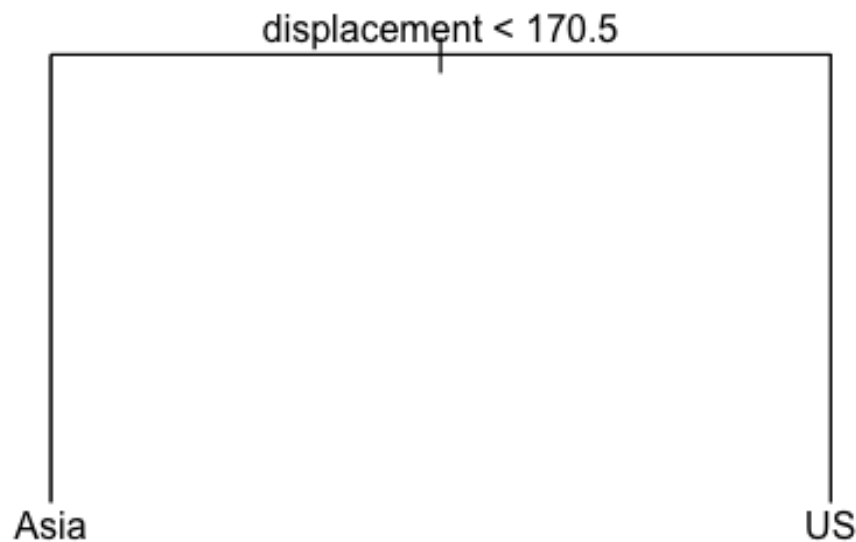
```
## [1] 386.2734 356.7785 326.8288 330.0710 324.0363 328.4066 312.6569 295.50
64
## [9] 253.4955 365.3540
##
## $k
## [1] -Inf 5.715627 8.614211 8.771793 9.731839 9.930317
## [7] 10.974339 12.180115 20.305247 153.578616
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune" "tree.sequence"

plot(cv.autotrain$size, cv.autotrain$dev, type="b", main="Tree pruning")
```



We will prune at 2 as it has the lowest deviance.

```
prune.autotrain=prune.misclass(tree.autotrain,best=2)
plot(prune.autotrain)
text(prune.autotrain,pretty=0)
```



```
set.seed(1)
tree.pred.train = predict(prune.autotrain, autotrain, type="class")
tree.train.error = mean(tree.pred.train != autotrain$origin)
tree.train.error

## [1] 0.2857143

tree.pred.test = predict(prune.autotrain, autotest, type="class")
tree.test.error = mean(tree.pred.test != autotest$origin)
tree.test.error

## [1] 0.4183673
```

The test misclassification error is 0.4183673 which is higher than that of the training error of 0.2857143 suggesting that there is more misclassification in the testing data which is bad. The pruned tree did not perform better.

(c)

```
bag.autotrain = randomForest(origin~., data=autotrain, mtry=7, importance=TRUE, ntree=1000)
bag.autotrain

##
## Call:
```



```
## randomForest(formula = origin ~ ., data = autotrain, mtry = 7,      impor
tance = TRUE, ntree = 1000)
##               Type of random forest: classification
##               Number of trees: 1000
## No. of variables tried at each split: 7
##
##               OOB estimate of  error rate: 20.41%
## Confusion matrix:
##      Asia Europe  US class.error
## Asia    27     11   3  0.3414634
## Europe   11     20   2  0.3939394
## US        5      8 109  0.1065574
```

### *#bagging*

```
set.seed(1)
tree.pred.train = predict(bag.autotrain,autotrain,type="class")
tree.train.error = mean(tree.pred.train != autotrain$origin)
tree.train.error

## [1] 0

tree.pred.test = predict(bag.autotrain,autotest,type="class")
tree.test.error = mean(tree.pred.test != autotest$origin)
tree.test.error

## [1] 0.2295918
```

The OOB estimate of error rate is 0.20(2dp). The training error rate of 0 is implying overfitting. The testing error rate is 0.23(2dp) which is reasonably close to the OOB error rate of 0.20(2dp) suggesting that bagging improves the classification.

### *#random forest*

```
set.seed(1)
rf.autotrain = randomForest(origin~.,data=autotrain,mtry=3)
rf.autotrain

##
## Call:
## randomForest(formula = origin ~ ., data = autotrain, mtry = 3)
##               Type of random forest: classification
##               Number of trees: 500
## No. of variables tried at each split: 3
##
##               OOB estimate of  error rate: 20.92%
## Confusion matrix:
##      Asia Europe  US class.error
## Asia    29      8   4  0.2926829
## Europe   11     17   5  0.4848485
## US        7      6 109  0.1065574
```

```

set.seed(1)
tree.pred.train = predict(rf.autotrain, autotrain, type="class")
tree.train.error = mean(tree.pred.train != autotrain$origin)
tree.train.error

## [1] 0

tree.pred.test = predict(rf.autotrain, autotest, type="class")
tree.test.error = mean(tree.pred.test != autotest$origin)
tree.test.error

## [1] 0.2244898

```

The OOB estimate error rate is 0.21(2dp). The training error rate is still zero implying overfitting. The testing error rate is 0.23(2dp) which is lower than that of bagging and closer to the OOB estimate error rate. Both the bagging and random forest has the same testing error rate, therefore decorrelating trees (nature of random forest) is not an effective strategy for this problem.

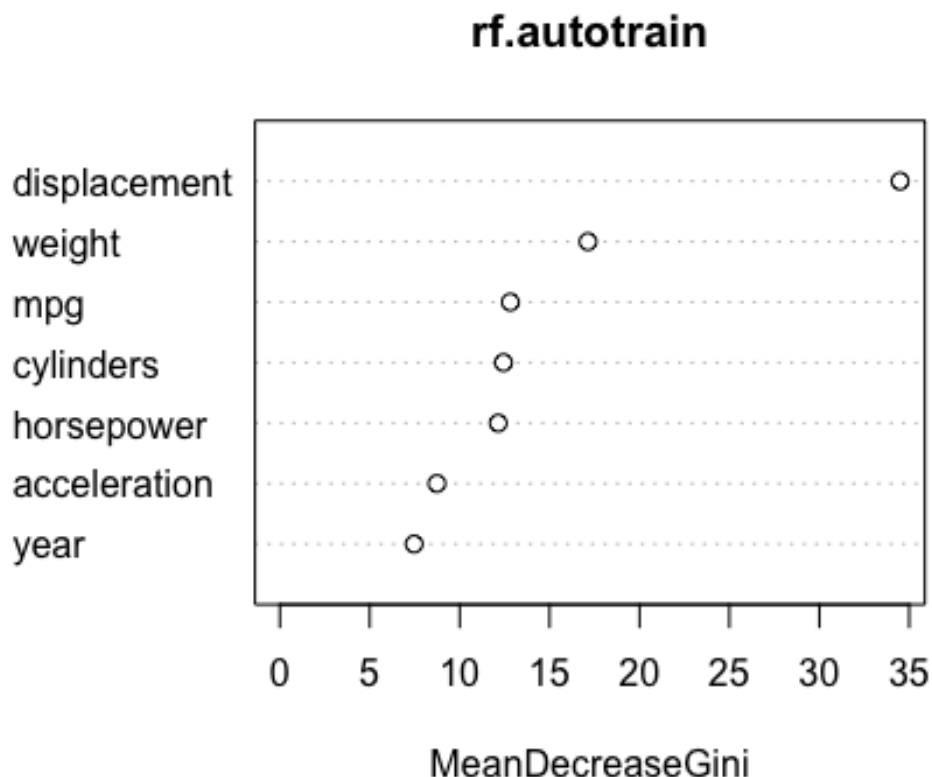
```

importance(rf.autotrain)

##           MeanDecreaseGini
## mpg                12.821163
## cylinders          12.438014
## displacement       34.492230
## horsepower         12.145256
## weight              17.136344
## acceleration        8.735524
## year                7.469548

varImpPlot(rf.autotrain)

```



The variable importances are different as shown in the plots. The variable displacement is the most important as it has highest mean decrease in the Gini index. This can be seen in the tree where displacement is placed at the top (root node). The variables mpg, cylinders and horsepower seem to have similar variable importance.

(d)

```
set.seed(1)
tree.boost.1 = gbm(origin~. , data=autotrain, distribution = "multinomial", shrinkage=0.01, interaction.depth=1, n.trees=1000)

tree.boost.2 = gbm(origin~. , data=autotrain, distribution = "multinomial", shrinkage=0.01, interaction.depth=2, n.trees=2500)

tree.boost.3 = gbm(origin~. , data=autotrain, distribution = "multinomial", shrinkage=0.001, interaction.depth=1, n.trees=5000)

tree.boost.4 = gbm(origin~. , data=autotrain, distribution = "multinomial", shrinkage=0.001, interaction.depth=2, n.trees=7500)

# tree.boost.1
tree.pred.train = predict(tree.boost.1, autotrain, type="response", n.trees=1000)
trainerror = apply(tree.pred.train, 1, which.max)
```

```

trainerror = factor(trainerror, labels=levels(autotrain$origin))
mean(trainerror != autotrain$origin)

## [1] 0.06122449

tree.pred.test = predict(tree.boost.1, autotest, type="response", n.trees=1000)
testerror = apply(tree.pred.test, 1, which.max)
testerror = factor(testerror, labels=levels(autotest$origin))
mean(testerror != autotest$origin)

## [1] 0.2244898

#tree.boost2
tree.pred.train = predict(tree.boost.2, autotrain, type="response", n.trees=2500)
trainerror = apply(tree.pred.train, 1, which.max)
trainerror = factor(trainerror, labels=levels(autotrain$origin))
mean(trainerror != autotrain$origin)

## [1] 0

tree.pred.test = predict(tree.boost.2, autotest, type="response", n.trees=2500)
testerror = apply(tree.pred.test, 1, which.max)
testerror = factor(testerror, labels=levels(autotest$origin))
mean(testerror != autotest$origin)

## [1] 0.2193878

#tree.boost.3
tree.pred.train = predict(tree.boost.3, autotrain, type="response", n.trees=5000)
trainerror = apply(tree.pred.train, 1, which.max)
trainerror = factor(trainerror, labels=levels(autotrain$origin))
mean(trainerror != autotrain$origin)

## [1] 0.1122449

tree.pred.test = predict(tree.boost.3, autotest, type="response", n.trees=5000)
testerror = apply(tree.pred.test, 1, which.max)
testerror = factor(testerror, labels=levels(autotest$origin))
mean(testerror != autotest$origin)

## [1] 0.2602041

#tree.boost4
tree.pred.train = predict(tree.boost.4, autotrain, type="response", n.trees=7500)
trainerror = apply(tree.pred.train, 1, which.max)
trainerror = factor(trainerror, labels=levels(autotrain$origin))
mean(trainerror != autotrain$origin)

## [1] 0.03061224

```

```
tree.pred.test = predict(tree.boost.4, autotest, type="response", n.trees=7500)
testerror = apply(tree.pred.test, 1, which.max)
testerror = factor(testerror, labels=levels(autotest$origin))
mean(testerror != autotest$origin)

## [1] 0.2142857
```

For tree boosted 1, training error rate is: 0.06122449 and testing error rate is: 0.2244898.

For tree boosted 2, training error rate is: 0 and testing error rate is: 0.2193878.

For tree boosted 3, training error rate is: 0.1122449 and testing error rate is: 0.2602041.

For tree boosted 4, training error rate is: 0.03061224 and testing error rate is: 0.2142857.

The errors for tree boosted 1 seems alright, but there is no telling if it is the best and perhaps a better model can be achieved. The training error for tree boosted 2 is 0 implying overfitting and this could be because of shrinkage = 0.01. By decreasing the shrinkage to 0.001 in tree boosted 3, the training error decreases, but the testing error increases, and this could be because of the smaller depth=1 and smaller number of trees = 5000. In tree boosted 4, the training error and testing error rate has improved as it has a lower shrinkage = 0.001, more depth = 2 and a greater number of trees = 7500.

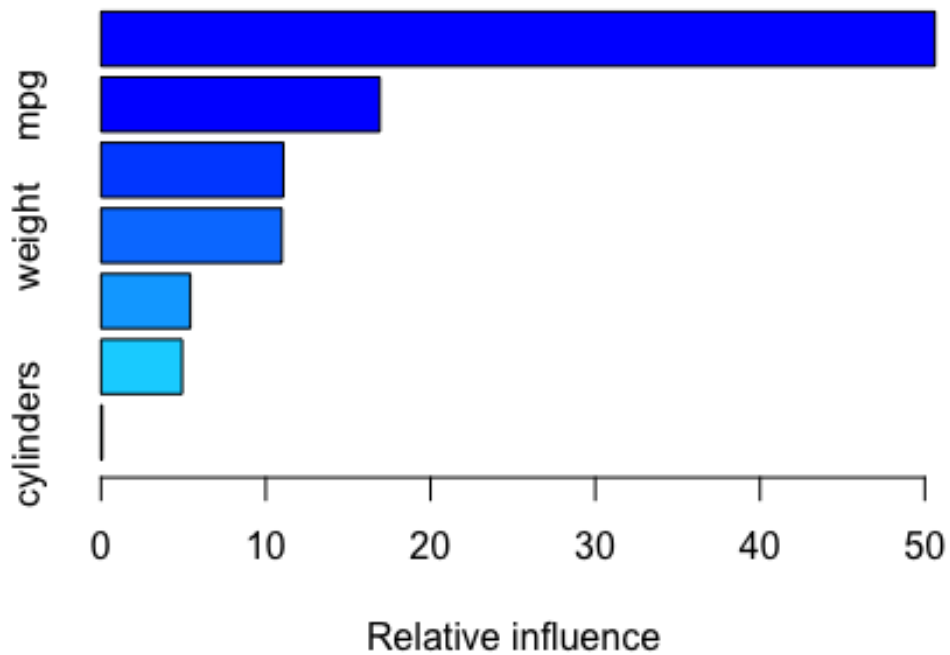
The best model is tree boosted 4 with a training error rate of 0.03061224 and testing error rate of 0.2142857.

(e)

The best boosted model from previous question has a testing error rate of 0.21(2dp) whereas the random forest model has a testing error rate of 0.23(2dp).

Out of all the models, the best boosted model performs better as it is able to classify more observations correctly due to its lower testing rate.

```
summary(tree.boost.4)
```



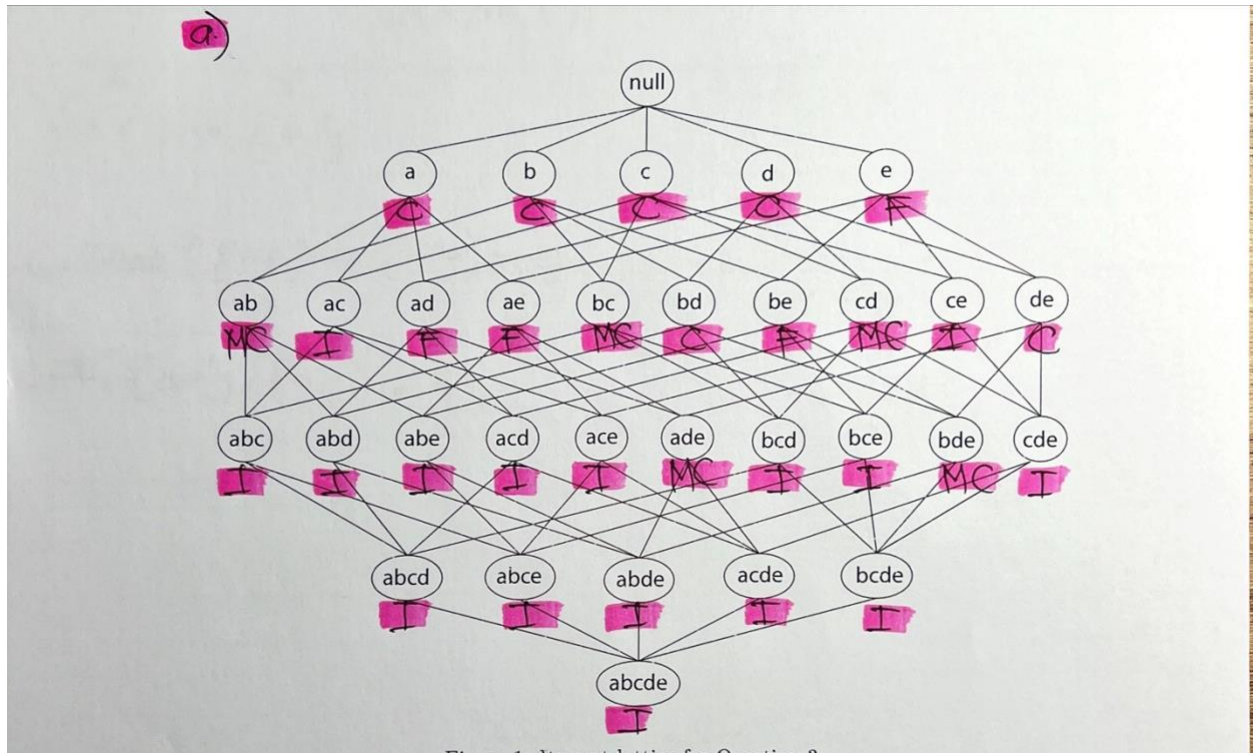
```
##           var      rel.inf
## displacement displacement 50.61290271
## mpg           mpg        16.92289806
## horsepower    horsepower 11.07077656
## weight        weight    10.98038504
## acceleration  acceleration 5.42360289
## year          year       4.92367517
## cylinders     cylinders   0.06575957
```

This best model has displacement as the most important predictor as it has the highest relative influence, followed by mpg, horsepower and weight.



### Question 3

a.



b.

b.

$$\text{Confidence} = c(X \rightarrow Y) = \frac{P(Y|X)}{P(Y)}$$

$$\text{Lift} = \text{lift}(X, Y) = \frac{P(X, Y)}{P(X)P(Y)}$$

$$\begin{matrix} X & Y \\ \{d, e\} & \rightarrow & \{a\} \end{matrix}$$

$$\text{Confidence}(\{d, e\} \rightarrow \{a\}) = \frac{2}{3}$$

$$\text{Lift}(\{d, e\}, \{a\}) = \frac{4}{3}$$

The confidence is  $\frac{2}{3}$  which means that when items d and e are bought together, then 66.7% of the time that item a is also bought.

The lift is  $\frac{4}{3}$ , and since the lift is greater than 1, then the items are positively correlated. Meaning an occurrence of items d and e, promotes the occurrence of item a. This means that a sale in items d and e could result in a 33% increase in the sale of item a.

Market owners can place items d, e and a in the same location so they can be bought together to increase sales/profit.