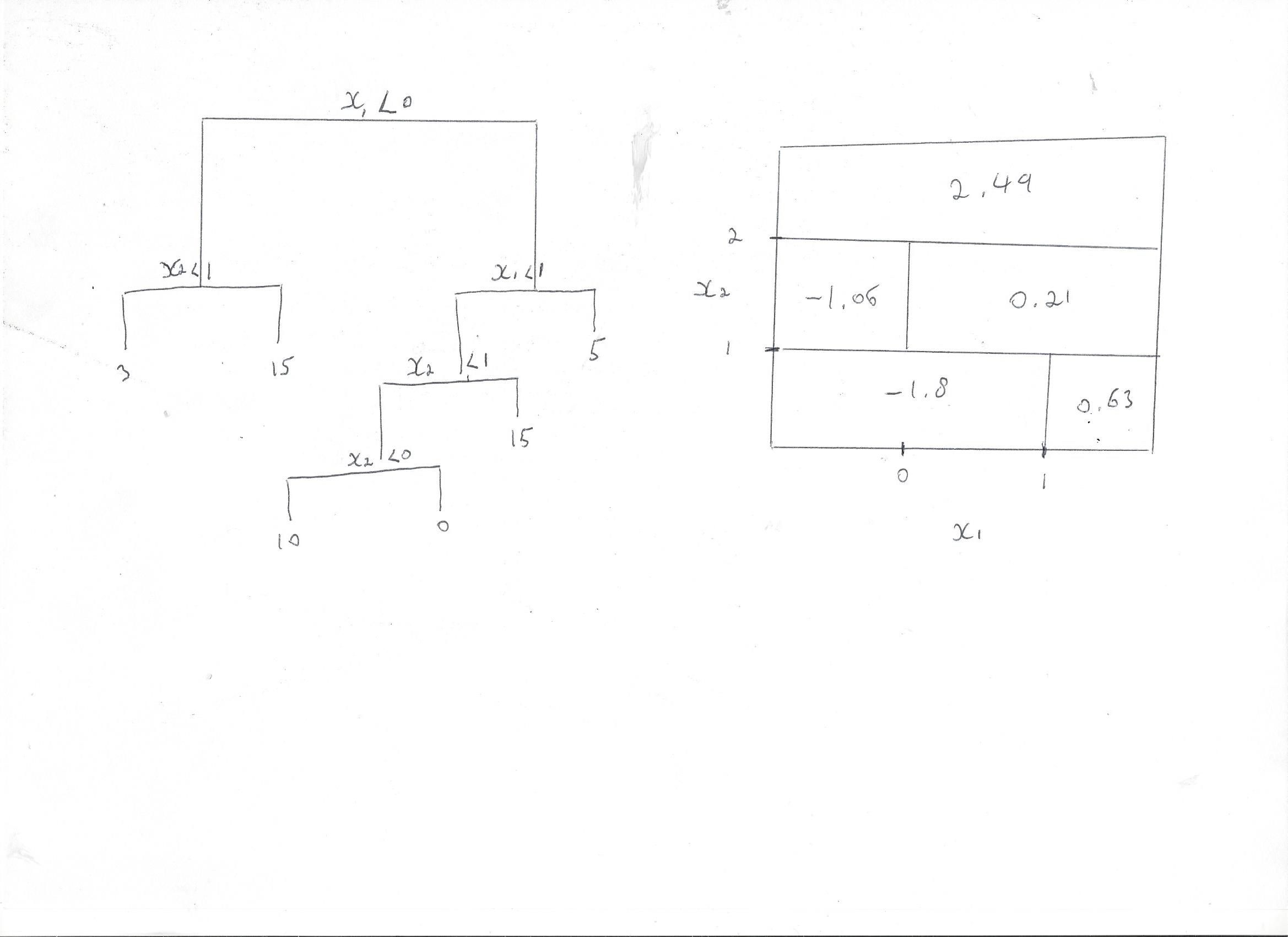
CW2\_13128128\_N\_Katz.rmd

Nuchem Katz

6 January 2019

1 Decision trees

knitr::include\_graphics('./treeScatch.jpg')



1. Regression Trees
2. and b)

library(ISLR)  
data(Carseats)   
head(Carseats)

## Sales CompPrice Income Advertising Population Price ShelveLoc Age  
## 1 9.50 138 73 11 276 120 Bad 42  
## 2 11.22 111 48 16 260 83 Good 65  
## 3 10.06 113 35 10 269 80 Medium 59  
## 4 7.40 117 100 4 466 97 Medium 55  
## 5 4.15 141 64 3 340 128 Bad 38  
## 6 10.81 124 113 13 501 72 Bad 78  
## Education Urban US  
## 1 17 Yes Yes  
## 2 10 Yes Yes  
## 3 12 Yes Yes  
## 4 14 Yes Yes  
## 5 13 Yes No  
## 6 16 No Yes

library(tree)

## Warning: package 'tree' was built under R version 3.5.2

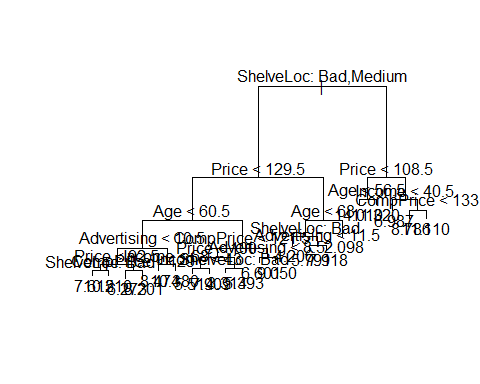
nrow(Carseats)

## [1] 400

set.seed(7)  
carseat.train<-sample(1:nrow(Carseats),200)  
carseat.test<-Carseats[-carseat.train,]  
  
#creating tree  
tree.carseat.train<- tree(Carseats$Sales ~ .,Carseats, subset=carseat.train)  
View(Carseats)  
summary(tree.carseat.train)

##   
## Regression tree:  
## tree(formula = Carseats$Sales ~ ., data = Carseats, subset = carseat.train)  
## Variables actually used in tree construction:  
## [1] "ShelveLoc" "Price" "Age" "Advertising" "CompPrice"   
## [6] "Income"   
## Number of terminal nodes: 21   
## Residual mean deviance: 1.85 = 331.2 / 179   
## Distribution of residuals:  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -3.67600 -0.85430 -0.06197 0.00000 0.87880 3.14400

plot(tree.carseat.train)  
text(tree.carseat.train, pretty=0)



#calculating MSE for unpruned tree  
carseat.yhat<-predict(tree.carseat.train, newdata = Carseats[-carseat.train,])  
carseat.test.trueSales<-Carseats[-carseat.train,"Sales"]  
print(carseat.testMSE.sales.unpruned<-mean((carseat.yhat-carseat.test.trueSales)^2))

## [1] 4.83821

sqrt(carseat.testMSE.sales.unpruned)

## [1] 2.199593

#\* The price and the quality of the shelving location where the carseats are sold   
#(which I guess mean either how they're 'visually' presented to the customer,   
#or the actual physical conditions under which they're kept/stored until sale)  
#- seem to be of biggest importance.   
#\* That if shelveLoc was good, advertising was not anymore a factor.  
#\* That the highest sales were made when shelveloc was good, the price was more than a 100,   
#community level was higher than 40.5K, but the competitors price was more than 130  
  
#21 terminal nodes. 6 predictors are used.   
#The test MSE is 4.83821 and its square root 2.199593,   
#which means that the model leads to a prediction which are within around 2.2K of the true sales of carseats at the 400 locations.   
  
  
  
  
#c) calculating cv the check wether to prune  
set.seed(8)  
cv.tree.carseat.train<-cv.tree(tree.carseat.train)  
cv.tree.carseat.train

## $size  
## [1] 21 20 19 18 17 16 15 14 12 11 10 8 7 6 5 4 3 2 1  
##   
## $dev  
## [1] 1029.463 1055.227 1048.794 1070.248 1075.000 1067.051 1074.142  
## [8] 1083.777 1115.698 1135.104 1115.885 1084.262 1068.703 1096.376  
## [15] 1104.488 1086.615 1129.119 1236.911 1549.416  
##   
## $k  
## [1] -Inf 16.69862 17.24085 18.00964 18.42218 19.12008 21.61897  
## [8] 22.49100 26.97919 32.27733 33.39821 37.74202 39.43387 60.02008  
## [15] 61.36933 85.90284 106.87019 167.30725 352.05311  
##   
## $method  
## [1] "deviance"  
##   
## attr(,"class")  
## [1] "prune" "tree.sequence"

prune.carseat.tree.train<-prune.tree(tree.carseat.train, best=21)  
summary(prune.carseat.tree.train)

##   
## Regression tree:  
## tree(formula = Carseats$Sales ~ ., data = Carseats, subset = carseat.train)  
## Variables actually used in tree construction:  
## [1] "ShelveLoc" "Price" "Age" "Advertising" "CompPrice"   
## [6] "Income"   
## Number of terminal nodes: 21   
## Residual mean deviance: 1.85 = 331.2 / 179   
## Distribution of residuals:  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -3.67600 -0.85430 -0.06197 0.00000 0.87880 3.14400

#calculating MSE for pruned tree  
pruned.carseat.yhat<-predict(prune.carseat.tree.train, newdata = Carseats[-carseat.train,])  
print(carseat.testMSE.sales.pruned<-mean((pruned.carseat.yhat-carseat.test.trueSales)^2))

## [1] 4.83821

sqrt(carseat.testMSE.sales.pruned)

## [1] 2.199593

#Pruning the tree isn't needed as the tree is already split on 21 terminal nodes, which according to the cv test produce the best MSE.  
  
#d)   
library(MASS)  
library(randomForest)

## Warning: package 'randomForest' was built under R version 3.5.2

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

set.seed(1)  
bag.sales.carseats<-randomForest(Sales~., data=Carseats, subset = carseat.train, mtry=10, importance=TRUE)  
bag.sales.carseats

##   
## Call:  
## randomForest(formula = Sales ~ ., data = Carseats, mtry = 10, importance = TRUE, subset = carseat.train)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 10  
##   
## Mean of squared residuals: 2.646289  
## % Var explained: 65.47

#calculating MSE for bagging  
yhat.carseat.bagged<-predict(bag.sales.carseats, newdata = Carseats[-carseat.train,])  
print(carseat.testMSE.sales.bagged<-mean((yhat.carseat.bagged-carseat.test.trueSales)^2))

## [1] 2.646667

sqrt(carseat.testMSE.sales.bagged)

## [1] 1.626858

importance(bag.sales.carseats)

## %IncMSE IncNodePurity  
## CompPrice 27.4390584 164.077437  
## Income 10.0957705 90.944867  
## Advertising 18.2365333 151.700051  
## Population 0.3942538 49.646314  
## Price 50.0568235 410.931673  
## ShelveLoc 53.2097815 410.482922  
## Age 19.1148237 161.216522  
## Education -0.2807402 38.677414  
## Urban -0.4987723 12.265129  
## US 3.2916492 5.267416

#We do get a better MSE as an sqrt(MSE) of 1.630 is better than 2.200. Also, the Variance explained is 65.5  
# \* The Var is best best explained when all 10 variables are considered for each split of the trees.  
# \* importance() confirms that shelveloc and price carry the most importance, and they coincide for both, bagging and the following (random forest).   
  
#d) calculate mse on mtry=10 floor 3  
rf3.bag.sales.carseats<-randomForest(Sales~., data=Carseats, subset = carseat.train, mtry=3, importance=TRUE)  
rf.yhat.carseat.bagged<-predict(rf3.bag.sales.carseats, newdata = Carseats[-carseat.train,])  
print(carseat.testMSE.sales.bagged<-mean((rf.yhat.carseat.bagged-carseat.test.trueSales)^2))

## [1] 3.159781

sqrt(carseat.testMSE.sales.bagged)

## [1] 1.777577

importance(rf3.bag.sales.carseats)

## %IncMSE IncNodePurity  
## CompPrice 14.2544522 164.46739  
## Income 5.7995929 119.38034  
## Advertising 13.5730392 152.90169  
## Population -1.2232437 89.34527  
## Price 32.9267632 326.59118  
## ShelveLoc 36.2625605 311.00154  
## Age 12.6141528 167.86985  
## Education 0.5978316 64.94182  
## Urban 1.8521890 16.38307  
## US 5.0639390 27.17933

#calculate mse on mtry=10 ceiling-division 3  
rf4.bag.sales.carseats<-randomForest(Sales~., data=Carseats, subset = carseat.train, mtry=4, importance=TRUE)  
rf.yhat.carseat.bagged<-predict(rf4.bag.sales.carseats, newdata = Carseats[-carseat.train,])  
print(rf.carseat.testMSE.sales.bagged<-mean((rf.yhat.carseat.bagged-carseat.test.trueSales)^2))

## [1] 2.913912

sqrt(rf.carseat.testMSE.sales.bagged)

## [1] 1.707018

importance(rf3.bag.sales.carseats)

## %IncMSE IncNodePurity  
## CompPrice 14.2544522 164.46739  
## Income 5.7995929 119.38034  
## Advertising 13.5730392 152.90169  
## Population -1.2232437 89.34527  
## Price 32.9267632 326.59118  
## ShelveLoc 36.2625605 311.00154  
## Age 12.6141528 167.86985  
## Education 0.5978316 64.94182  
## Urban 1.8521890 16.38307  
## US 5.0639390 27.17933

#This is not better than when all m is used, however using 4 is obviously bettern than using 3.

1. Classification trees

#a)  
library(ISLR)  
library(tree)  
data(OJ)  
head(OJ)

## Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH  
## 1 CH 237 1 1.75 1.99 0.00 0.0 0  
## 2 CH 239 1 1.75 1.99 0.00 0.3 0  
## 3 CH 245 1 1.86 2.09 0.17 0.0 0  
## 4 MM 227 1 1.69 1.69 0.00 0.0 0  
## 5 CH 228 7 1.69 1.69 0.00 0.0 0  
## 6 CH 230 7 1.69 1.99 0.00 0.0 0  
## SpecialMM LoyalCH SalePriceMM SalePriceCH PriceDiff Store7 PctDiscMM  
## 1 0 0.500000 1.99 1.75 0.24 No 0.000000  
## 2 1 0.600000 1.69 1.75 -0.06 No 0.150754  
## 3 0 0.680000 2.09 1.69 0.40 No 0.000000  
## 4 0 0.400000 1.69 1.69 0.00 No 0.000000  
## 5 0 0.956535 1.69 1.69 0.00 Yes 0.000000  
## 6 1 0.965228 1.99 1.69 0.30 Yes 0.000000  
## PctDiscCH ListPriceDiff STORE  
## 1 0.000000 0.24 1  
## 2 0.000000 0.24 1  
## 3 0.091398 0.23 1  
## 4 0.000000 0.00 1  
## 5 0.000000 0.00 0  
## 6 0.000000 0.30 0

purchase01= as.factor(OJ$Purchase)  
OJ<-data.frame(OJ, purchase01)  
nrow(OJ)

## [1] 1070

set.seed(7)  
oj.train<-sample(1:nrow(OJ),800)  
oj.test<-OJ[-oj.train,]  
nrow(oj.train)

## NULL

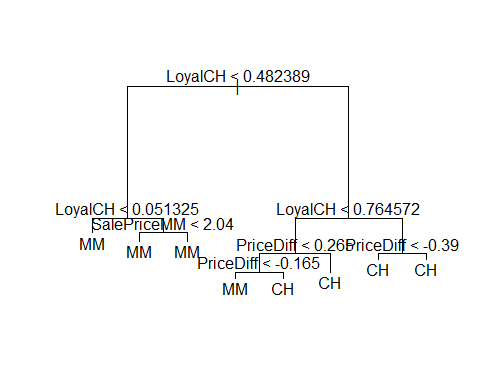
purchase01.test<-OJ$purchase01[-oj.train]  
  
#B)  
tree.oj<-tree(purchase01 ~.- Purchase, OJ, subset=oj.train)  
summary(tree.oj)

##   
## Classification tree:  
## tree(formula = purchase01 ~ . - Purchase, data = OJ, subset = oj.train)  
## Variables actually used in tree construction:  
## [1] "LoyalCH" "SalePriceMM" "PriceDiff"   
## Number of terminal nodes: 8   
## Residual mean deviance: 0.7597 = 601.6 / 792   
## Misclassification error rate: 0.1788 = 143 / 800

#The tree has 8 terminal nodes. The training error rate is 18%. The loyalty, the price of the product and the diff in price  
#between the two products are what's important.  
  
#C) and d)  
tree.oj

## node), split, n, deviance, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 800 1073.00 CH ( 0.60625 0.39375 )   
## 2) LoyalCH < 0.482389 303 334.60 MM ( 0.24092 0.75908 )   
## 4) LoyalCH < 0.051325 62 10.24 MM ( 0.01613 0.98387 ) \*  
## 5) LoyalCH > 0.051325 241 293.90 MM ( 0.29876 0.70124 )   
## 10) SalePriceMM < 2.04 137 133.10 MM ( 0.18978 0.81022 ) \*  
## 11) SalePriceMM > 2.04 104 142.80 MM ( 0.44231 0.55769 ) \*  
## 3) LoyalCH > 0.482389 497 454.80 CH ( 0.82897 0.17103 )   
## 6) LoyalCH < 0.764572 250 308.70 CH ( 0.69200 0.30800 )   
## 12) PriceDiff < 0.265 157 215.30 CH ( 0.56051 0.43949 )   
## 24) PriceDiff < -0.165 39 48.14 MM ( 0.30769 0.69231 ) \*  
## 25) PriceDiff > -0.165 118 153.60 CH ( 0.64407 0.35593 ) \*  
## 13) PriceDiff > 0.265 93 54.54 CH ( 0.91398 0.08602 ) \*  
## 7) LoyalCH > 0.764572 247 70.62 CH ( 0.96761 0.03239 )   
## 14) PriceDiff < -0.39 8 10.59 CH ( 0.62500 0.37500 ) \*  
## 15) PriceDiff > -0.39 239 48.56 CH ( 0.97908 0.02092 ) \*

plot(tree.oj)   
text(tree.oj, pretty=0)



#If loyalty to CH is more than 48%, then the only way people would go for mm was if the the price difference is less than 17 cent (line no 24, upon getting a detailed text output). Else, if their loyalty is less, they'd always go for mm, no matter the difference in price.   
# So as an example, interperting line 24, when typing the name of the tree to get detailed text output: For a loyalty score of less than 48%, if price of MM less of CH is less than 16.5 cent, based on 39 observations (with a smallest sum of squares for this node summed as 48.14 - though this has no meaning here since this is a classification and not regression tree), an 'MM purchase' is the overall prediction for this branch with a probebility of 69% .   
  
#e)  
tree.pred.oj.test<-predict(tree.oj, oj.test, type ="class")  
table(tree.pred.oj.test, purchase01.test)

## purchase01.test  
## tree.pred.oj.test CH MM  
## CH 147 18  
## MM 21 84

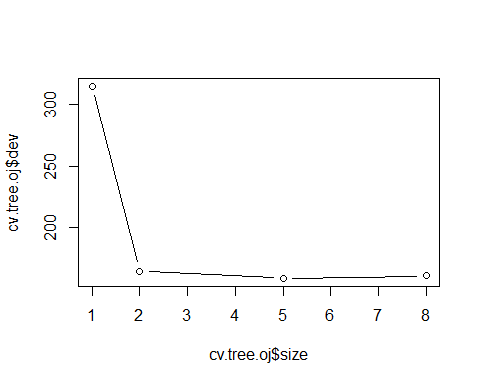
print((18+21)/270)

## [1] 0.1444444

# Test error rate is 14%, bettern than before  
  
#f), g), h) and i)  
set.seed(3)  
cv.tree.oj<- cv.tree(tree.oj, FUN=prune.misclass)  
cv.tree.oj

## $size  
## [1] 8 5 2 1  
##   
## $dev  
## [1] 161 159 165 315  
##   
## $k  
## [1] -Inf 0 5 157  
##   
## $method  
## [1] "misclass"  
##   
## attr(,"class")  
## [1] "prune" "tree.sequence"

plot(cv.tree.oj$size,cv.tree.oj$dev,type='b')



prune.tree.oj <- prune.misclass(tree.oj,best=5)  
prune.tree.pred.oj.test<-predict(prune.tree.oj, oj.test, type ="class")  
table(prune.tree.pred.oj.test, purchase01.test)

## purchase01.test  
## prune.tree.pred.oj.test CH MM  
## CH 147 18  
## MM 21 84

#cv shows that prunning the tree with 5 is best but when pruned, the error rates aren't actually make any difference   
  
#j) and k)  
summary(prune.tree.oj)

##   
## Classification tree:  
## snip.tree(tree = tree.oj, nodes = c(2L, 7L))  
## Variables actually used in tree construction:  
## [1] "LoyalCH" "PriceDiff"  
## Number of terminal nodes: 5   
## Residual mean deviance: 0.8321 = 661.5 / 795   
## Misclassification error rate: 0.1788 = 143 / 800

# the training error for both are the same and the test error for both are the same

4 SVM

#a)  
  
library(e1071)

## Warning: package 'e1071' was built under R version 3.5.2

library(ISLR)  
data(Auto)  
median(Auto$mpg)

## [1] 22.75

milAbvMedn<-c(ifelse(Auto$mpg<median(Auto$mpg),0,1))  
auto.dat<-data.frame(x=Auto, y= as.factor(milAbvMedn))  
  
#b)  
#Cost=1  
svmfit.linear.higmil.c1 <- svm(y ~ ., data=auto.dat, kernel="linear", cost=1)  
svmfit.linear.higmil.c1

##   
## Call:  
## svm(formula = y ~ ., data = auto.dat, kernel = "linear", cost = 1)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
## gamma: 0.003205128   
##   
## Number of Support Vectors: 56

summary(svmfit.linear.higmil.c1)

##   
## Call:  
## svm(formula = y ~ ., data = auto.dat, kernel = "linear", cost = 1)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
## gamma: 0.003205128   
##   
## Number of Support Vectors: 56  
##   
## ( 26 30 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

#There are 56 support vectotrs; 26 from side y=0 and 30 from side y=1.   
  
#Cost=0.01  
svmfit.linear.higmil.c001 <- svm(y ~ ., data=auto.dat, kernel="linear", cost=0.01)  
svmfit.linear.higmil.c001

##   
## Call:  
## svm(formula = y ~ ., data = auto.dat, kernel = "linear", cost = 0.01)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 0.01   
## gamma: 0.003205128   
##   
## Number of Support Vectors: 150

summary(svmfit.linear.higmil.c001)

##   
## Call:  
## svm(formula = y ~ ., data = auto.dat, kernel = "linear", cost = 0.01)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 0.01   
## gamma: 0.003205128   
##   
## Number of Support Vectors: 150  
##   
## ( 74 76 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

#There are 150 support vectotrs; 74 from side y=0 and 76 from side y=1, as smaller cost   
#means many more support vectors involved in determining the margins (or hyperplanein this case).   
  
#Cost=100,000  
svmfit.linear.higmil.c100000 <- svm(y ~ ., data=auto.dat, kernel="linear", cost=1e5)  
svmfit.linear.higmil.c100000

##   
## Call:  
## svm(formula = y ~ ., data = auto.dat, kernel = "linear", cost = 1e+05)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1e+05   
## gamma: 0.003205128   
##   
## Number of Support Vectors: 35

summary(svmfit.linear.higmil.c100000)

##   
## Call:  
## svm(formula = y ~ ., data = auto.dat, kernel = "linear", cost = 1e+05)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1e+05   
## gamma: 0.003205128   
##   
## Number of Support Vectors: 35  
##   
## ( 21 14 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

#There are 35 support vectotrs; 21 from side y=0 and 14 from side y=1.   
  
#Perform cross validation on different values of cost  
set.seed(9)  
tune.out.lin<-tune(svm, y ~., data = auto.dat, kernel="linear", ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100, 1000, 10000, 1e5)))  
summary(tune.out.lin)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 1  
##   
## - best performance: 0.01282051   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.09217949 0.05450084  
## 2 1e-02 0.07679487 0.04850079  
## 3 1e-01 0.05121795 0.03203768  
## 4 1e+00 0.01282051 0.02179068  
## 5 5e+00 0.02044872 0.02354784  
## 6 1e+01 0.02301282 0.02244393  
## 7 1e+02 0.03326923 0.03211170  
## 8 1e+03 0.03326923 0.03211170  
## 9 1e+04 0.03326923 0.03211170  
## 10 1e+05 0.03326923 0.03211170

summary(tune.out.lin$best.model)

##   
## Call:  
## best.tune(method = svm, train.x = y ~ ., data = auto.dat, ranges = list(cost = c(0.001,   
## 0.01, 0.1, 1, 5, 10, 100, 1000, 10000, 1e+05)), kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
## gamma: 0.003205128   
##   
## Number of Support Vectors: 56  
##   
## ( 26 30 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

#Best performence of the cv is a training error of 0.01282051 which is when cost = to 10^0 which is =1.  
#i.e. the best balance between having a high cost, narrow margins/fewer violations but high overfitting   
#and therefor high variance vs lower cost, wide margins/more violations and more bias but a better generlized model   
#with less variance each time it is applied.  
  
  
#c)  
set.seed(8)  
tune.out.poly<-tune(svm, y ~., data = auto.dat, kernel="polynomial", ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100, 1000, 10000, 1e5)),gamma = c(0.5,1,2,3,4), degree=c(0,1,2,3,4,5))  
summary(tune.out.poly)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 0.001  
##   
## - best performance: 0.5305769   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.5305769 0.02184519  
## 2 1e-02 0.5305769 0.02184519  
## 3 1e-01 0.5305769 0.02184519  
## 4 1e+00 0.5305769 0.02184519  
## 5 5e+00 0.5305769 0.02184519  
## 6 1e+01 0.5305769 0.02184519  
## 7 1e+02 0.5305769 0.02184519  
## 8 1e+03 0.5305769 0.02184519  
## 9 1e+04 0.5305769 0.02184519  
## 10 1e+05 0.5305769 0.02184519

summary(tune.out.poly$best.model)

##   
## Call:  
## best.tune(method = svm, train.x = y ~ ., data = auto.dat, ranges = list(cost = c(0.001,   
## 0.01, 0.1, 1, 5, 10, 100, 1000, 10000, 1e+05)), kernel = "polynomial",   
## gamma = c(0.5, 1, 2, 3, 4), degree = c(0, 1, 2, 3, 4, 5))  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: polynomial   
## cost: 0.001   
## degree: 0 1 2 3 4 5   
## gamma: 0.5 1 2 3 4   
## coef.0: 0   
##   
## Number of Support Vectors: 392  
##   
## ( 196 196 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

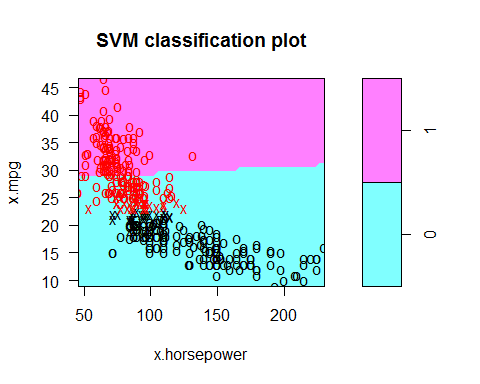
#Not good at all as training error of 0.5305769 no matter which cost so clearly not good model.   
  
  
  
set.seed(7)  
tune.out.radial <- tune(svm,y ~ .,data = auto.dat, kernel = "radial", ranges = list(cost = c(0.1,1,10,100,1000), gamma = c(0.5,1,2,3,4)))  
summary(tune.out.radial)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost gamma  
## 10 0.5  
##   
## - best performance: 0.04057692   
##   
## - Detailed performance results:  
## cost gamma error dispersion  
## 1 1e-01 0.5 0.08391026 0.04247619  
## 2 1e+00 0.5 0.04307692 0.04917112  
## 3 1e+01 0.5 0.04057692 0.05249461  
## 4 1e+02 0.5 0.04057692 0.05249461  
## 5 1e+03 0.5 0.04057692 0.05249461  
## 6 1e-01 1.0 0.56647436 0.06705049  
## 7 1e+00 1.0 0.06352564 0.04649727  
## 8 1e+01 1.0 0.05333333 0.05132281  
## 9 1e+02 1.0 0.05333333 0.05132281  
## 10 1e+03 1.0 0.05333333 0.05132281  
## 11 1e-01 2.0 0.56647436 0.06705049  
## 12 1e+00 2.0 0.12756410 0.09126708  
## 13 1e+01 2.0 0.11730769 0.08118533  
## 14 1e+02 2.0 0.11730769 0.08118533  
## 15 1e+03 2.0 0.11730769 0.08118533  
## 16 1e-01 3.0 0.56647436 0.06705049  
## 17 1e+00 3.0 0.41051282 0.16471773  
## 18 1e+01 3.0 0.39256410 0.16697298  
## 19 1e+02 3.0 0.39256410 0.16697298  
## 20 1e+03 3.0 0.39256410 0.16697298  
## 21 1e-01 4.0 0.56647436 0.06705049  
## 22 1e+00 4.0 0.49487179 0.09105724  
## 23 1e+01 4.0 0.47948718 0.10995495  
## 24 1e+02 4.0 0.47948718 0.10995495  
## 25 1e+03 4.0 0.47948718 0.10995495

summary(tune.out.radial$best.model)

##   
## Call:  
## best.tune(method = svm, train.x = y ~ ., data = auto.dat, ranges = list(cost = c(0.1,   
## 1, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4)), kernel = "radial")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 10   
## gamma: 0.5   
##   
## Number of Support Vectors: 259  
##   
## ( 127 132 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

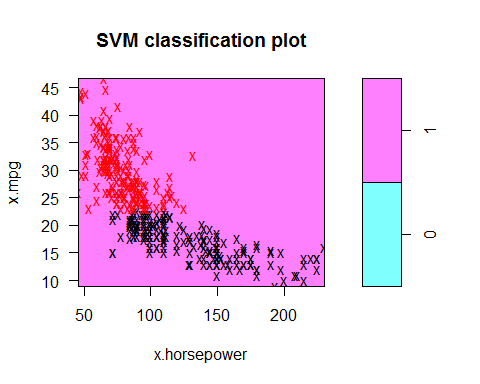
#Also not as good as linear as error is training error is still biggger, 0.04057692, cost=10 and gamma=0.5.  
  
  
#d)  
plot(tune.out.lin$best.model, auto.dat, x.mpg~x.horsepower)



tune.out.lin$best.model$index

## [1] 16 18 46 61 77 78 80 109 110 112 119 178 190 191 193 208 240  
## [18] 241 242 258 269 275 279 281 359 384 15 22 49 57 59 82 101 118  
## [35] 122 131 146 148 167 169 170 172 176 177 192 233 270 271 272 297 299  
## [52] 314 332 338 358 369

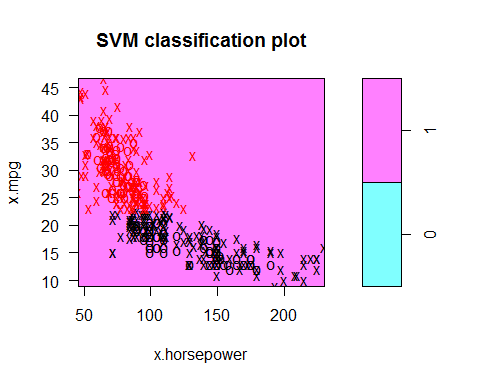
plot(tune.out.poly$best.model, auto.dat, x.mpg~x.horsepower)



tune.out.poly$best.model$index

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 16 17 18  
## [18] 25 26 27 28 29 33 34 35 36 37 38 39 40 41 42 43 44  
## [35] 45 46 47 48 60 61 62 63 64 65 66 67 68 69 70 71 72  
## [52] 73 74 75 76 77 78 80 85 86 87 88 89 90 91 92 93 94  
## [69] 95 96 97 98 99 100 103 104 105 106 107 108 109 110 111 112 113  
## [86] 115 116 119 120 121 123 124 125 126 127 132 133 134 135 136 137 138  
## [103] 139 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 168  
## [120] 173 175 178 186 187 188 189 190 191 193 198 199 200 201 206 207 208  
## [137] 209 210 211 212 213 214 220 221 222 223 224 225 226 227 228 229 230  
## [154] 231 240 241 242 248 249 250 251 252 253 255 256 257 258 259 260 261  
## [171] 262 263 264 269 273 274 275 276 279 280 281 282 283 284 285 286 287  
## [188] 288 289 290 291 315 359 361 362 384 15 19 20 21 22 23 24 30  
## [205] 31 32 49 50 51 52 53 54 55 56 57 58 59 79 81 82 83  
## [222] 84 101 102 114 117 118 122 128 129 130 131 140 141 142 143 144 145  
## [239] 146 147 148 149 150 166 167 169 170 171 172 174 176 177 179 180 181  
## [256] 182 183 184 185 192 194 195 196 197 202 203 204 205 215 216 217 218  
## [273] 219 232 233 234 235 236 237 238 239 243 244 245 246 247 254 265 266  
## [290] 267 268 270 271 272 277 278 292 293 294 295 296 297 298 299 300 301  
## [307] 302 303 304 305 306 307 308 309 310 311 312 313 314 316 317 318 319  
## [324] 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336  
## [341] 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353  
## [358] 354 355 356 357 358 360 363 364 365 366 367 368 369 370 371 372 373  
## [375] 374 375 376 377 378 379 380 381 382 383 385 386 387 388 389 390 391  
## [392] 392

plot(tune.out.radial$best.model, auto.dat, x.mpg~x.horsepower)



tune.out.radial$best.model$index

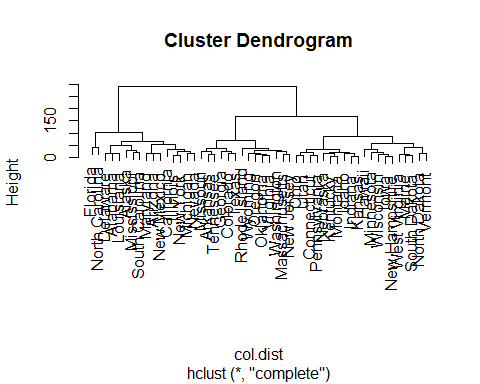
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 16 18 25  
## [18] 26 27 28 29 33 36 40 41 42 43 44 45 46 48 61 67 71  
## [35] 73 74 76 77 78 80 88 90 91 94 95 100 103 104 108 109 110  
## [52] 111 112 113 116 119 120 121 123 124 125 136 138 139 153 154 155 157  
## [69] 158 159 163 164 165 168 173 178 190 191 193 199 206 208 209 210 211  
## [86] 212 221 223 228 229 230 231 240 241 242 248 249 250 251 252 253 258  
## [103] 261 262 263 269 273 274 275 276 279 280 281 282 284 285 286 287 288  
## [120] 289 290 291 315 359 361 362 384 15 19 20 21 22 23 24 31 49  
## [137] 51 53 54 55 56 57 58 59 79 81 82 83 84 101 102 117 118  
## [154] 122 128 130 131 143 146 147 148 149 167 169 170 172 176 177 179 180  
## [171] 183 192 194 195 202 217 233 234 243 244 245 246 254 267 270 271 272  
## [188] 294 296 297 298 299 300 302 305 306 307 308 309 313 314 317 319 321  
## [205] 322 323 324 325 326 327 328 330 331 332 336 337 338 339 340 341 342  
## [222] 344 347 348 349 353 354 355 356 357 358 360 363 364 365 368 369 370  
## [239] 371 372 373 374 375 376 377 378 379 380 381 382 383 385 386 387 388  
## [256] 389 390 391 392

1. Hierarchical clustering

data("USArrests")  
#a) Calculate distance of each vector to each, using uclidean method, and cluster the the States  
col.dist<-dist(USArrests)  
clustrd.states<-hclust(col.dist,method="complete")  
  
#b)   
cutree(clustrd.states,3)

## Alabama Alaska Arizona Arkansas California   
## 1 1 1 2 1   
## Colorado Connecticut Delaware Florida Georgia   
## 2 3 1 1 2   
## Hawaii Idaho Illinois Indiana Iowa   
## 3 3 1 3 3   
## Kansas Kentucky Louisiana Maine Maryland   
## 3 3 1 3 1   
## Massachusetts Michigan Minnesota Mississippi Missouri   
## 2 1 3 1 2   
## Montana Nebraska Nevada New Hampshire New Jersey   
## 3 3 1 3 2   
## New Mexico New York North Carolina North Dakota Ohio   
## 1 1 1 3 3   
## Oklahoma Oregon Pennsylvania Rhode Island South Carolina   
## 2 2 3 2 1   
## South Dakota Tennessee Texas Utah Vermont   
## 3 2 2 3 3   
## Virginia Washington West Virginia Wisconsin Wyoming   
## 2 2 3 3 2

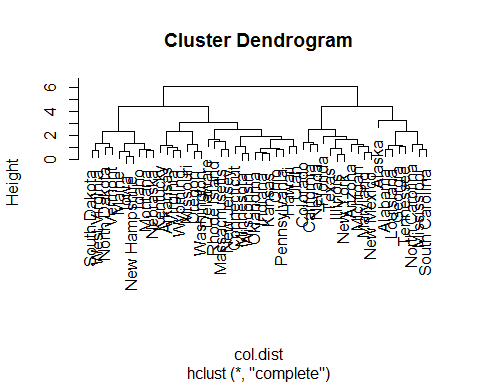
plot(clustrd.states)



#Executing cutree() gives us a list of the states with the cluster number associated to them  
  
#c)  
#?scale  
#scale(USArrests, center = TRUE, scale = TRUE)  
col.dist<-dist(scale(USArrests, center = TRUE, scale = TRUE))  
clustrd.states.scaled<-hclust(col.dist,method="complete")  
cutree(clustrd.states.scaled,3)

## Alabama Alaska Arizona Arkansas California   
## 1 1 2 3 2   
## Colorado Connecticut Delaware Florida Georgia   
## 2 3 3 2 1   
## Hawaii Idaho Illinois Indiana Iowa   
## 3 3 2 3 3   
## Kansas Kentucky Louisiana Maine Maryland   
## 3 3 1 3 2   
## Massachusetts Michigan Minnesota Mississippi Missouri   
## 3 2 3 1 3   
## Montana Nebraska Nevada New Hampshire New Jersey   
## 3 3 2 3 3   
## New Mexico New York North Carolina North Dakota Ohio   
## 2 2 1 3 3   
## Oklahoma Oregon Pennsylvania Rhode Island South Carolina   
## 3 3 3 3 1   
## South Dakota Tennessee Texas Utah Vermont   
## 3 1 2 3 3   
## Virginia Washington West Virginia Wisconsin Wyoming   
## 3 3 3 3 3

plot(clustrd.states.scaled)



#d) It divides the data into 4 to 5 clusters. My opinion is they should be scaled  
#as looking at the data, UrbanPop values are on a larger scaling than Rape and Murder respectively and the values in Assult are even larger.