MSc\_CW2\_D\_Monzon.rmd.

Delvin Monzon

08/01/2020

1a) Buy Yes | No 14/30=0.47 | 16/30=0.53

P(Student|Buy)  
Buy |Student = T |Student = F Yes |7/14 = 0.5 |7/14=0.5 No |11/16 = 0.688 |5/16 = 0.313

P(Income|Buy) Income  
Buy | High | Low Yes | 5/14=0.357 | 9/14=0.643 No | 7/16=0.438 | 9/16=0.563

P(Credit|Income,Student,Buy) |Credit Income| Student| Buy | Excellent | Fair High | T | Yes | 1/2=0.5 | 1/2=0.5 Low | T | Yes | 2/5=0.4 | 3/5=0.6 High | F | No | 2/3=0.66 | 1/3=0.33 Low | F | No | 1/2=0.5 | 1/2=0.5 High | F | Yes | 2/3=0.667 | 1/3=0.33 Low | F | Yes | 2/4=0.5 | 2/4=0.5 High | T | No | 2/4=0.5 | 2/4=0.5 Low | T | No | 2/7=0.29 | 5/7=0.71

1. Testing Instance 31: P(Income=Low|Buy=Y)*P(Student=F|Buy=Y)*P(Credit=Excellent|Income=Low,Student=F,Buy=Y)*P(Buy=Y) 0.5*0.47*0.5*0.643 = 0.076

P(Income=Low|Buy=N)*P(Student=F|Buy=N)*P(Credit=Excellent|Income=Low,Student=F,Buy=N)*P(Buy=N) 0.5*0.53*0.313*0.563 = 0.047

Most probable: Buy = Yes.

Testing Instance 32: P(Income=High|Buy=Y)*P(Student=F|Buy=Y)*P(Credit=Fair|Income= High,Student=F,Buy=Y)*P(Buy=Y) 0.33*0.47*0.5*0.357= 0.028

P(Income=High|Buy=N)*P(Student=F|Buy=N)*P(Credit=Fair|Income= High,Student=F,Buy=N)*P(Buy=N) 0.33*0.53*0.313*0.438 = 0.024

Most probable: Buy = Yes.

Buy Yes | No 14/30=0.47 | 16/30=0.53

P(Student|Buy)  
Buy |Student = T |Student = F Yes |7/14 = 0.5 |7/14=0.5 No |11/16 = 0.688 |5/16 = 0.313

P(Income|Buy) Income  
Buy | High | Low Yes | 5/14=0.357 | 9/14=0.643 No | 7/16=0.438 | 9/16=0.563

P(Credit|Buy) Credit  
Buy | Excellent | Fair Yes | 7/14=0.5 | 7/14=0.5 No | 7/16=0.4375 | 9/16=0.5625

1. Testing Instance 31: P(Income=Low|Buy=Y)*P(Student=F|Buy=Y)*P(Credit=Excellent|Buy=Y)*P(Buy=Y) 0.47*0.5*0.643*0.5 = 0.0755

P(Income=Low|Buy=N)*P(Student=F|Buy=N)*P(Credit=Excellent|Buy=N)*P(Buy=N) 0.53*0.313*0.563*0.438= 0.0409 Most probable: Buy = Yes.

Testing Instance 32: P(Income=High|Buy=Y)*P(Student=F|Buy=Y)*P(Credit=Fair|Buy=Y)*P(Buy=Y) 0.47*0.5*0.357*0.5=0.0419

P(Income=High|Buy=N)*P(Student=F|Buy=N)*P(Credit=Fair|Buy=N)*P(Buy=N) 0.53*0.313*0.438*0.563=0.0409 Most probable: Buy = Yes.

2a)

library(readr)   
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

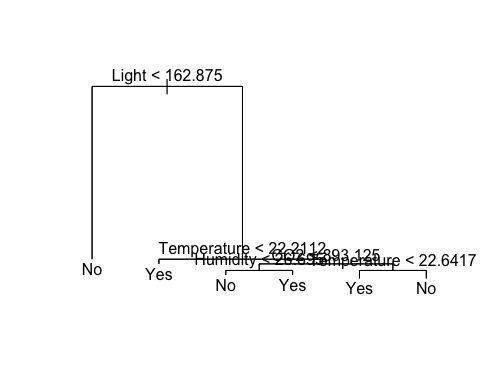
library(tree)  
  
OccTrain.data <- "RoomOccupancy\_Training.txt" %>%  
 read\_csv %>% # read in the data  
 select(Temperature, Humidity, Light, CO2, HumidityRatio, Occupancy) %>%   
 mutate(Occupancy = factor(Occupancy))

## Parsed with column specification:  
## cols(  
## Temperature = col\_double(),  
## Humidity = col\_double(),  
## Light = col\_double(),  
## CO2 = col\_double(),  
## HumidityRatio = col\_double(),  
## Occupancy = col\_character()  
## )

tree.OccTrain <- tree(Occupancy~.-Occupancy, OccTrain.data)   
summary(tree.OccTrain)

##   
## Classification tree:  
## tree(formula = Occupancy ~ . - Occupancy, data = OccTrain.data)  
## Variables actually used in tree construction:  
## [1] "Light" "Temperature" "CO2" "Humidity"   
## Number of terminal nodes: 6   
## Residual mean deviance: 0.07156 = 142.7 / 1994   
## Misclassification error rate: 0.0125 = 25 / 2000

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

 1.25% misclassification error rate on training data.

#Test data set  
Test.data <- "RoomOccupancy\_testing.txt" %>%  
 read\_csv %>% # read in the data  
 select(Temperature, Humidity, Light, CO2, HumidityRatio, Occupancy) %>%   
 mutate(Occupancy = factor(Occupancy))

## Parsed with column specification:  
## cols(  
## Temperature = col\_double(),  
## Humidity = col\_double(),  
## Light = col\_double(),  
## CO2 = col\_double(),  
## HumidityRatio = col\_double(),  
## Occupancy = col\_character()  
## )

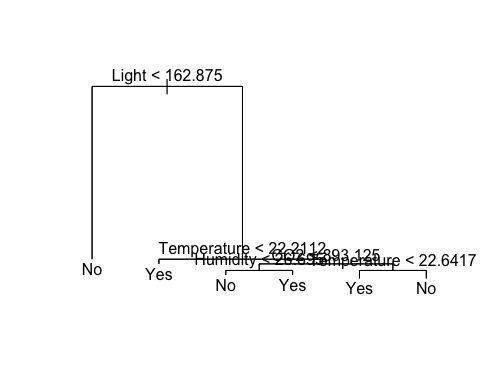
Occ.Test = Test.data$Occupancy  
summary(Occ.Test)

## No Yes   
## 240 60

#Build the tree using training set  
tree.Occ.train <- tree(Occupancy ~. -Occupancy, OccTrain.data)  
summary(tree.Occ.train)

##   
## Classification tree:  
## tree(formula = Occupancy ~ . - Occupancy, data = OccTrain.data)  
## Variables actually used in tree construction:  
## [1] "Light" "Temperature" "CO2" "Humidity"   
## Number of terminal nodes: 6   
## Residual mean deviance: 0.07156 = 142.7 / 1994   
## Misclassification error rate: 0.0125 = 25 / 2000

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



#Evaluate model with test data  
tree.Occ.pred <- predict(tree.Occ.train,Test.data,type="class")   
mean(tree.Occ.pred!= Occ.Test)

## [1] 0.2033333

table(tree.Occ.pred, Occ.Test)

## Occ.Test  
## tree.Occ.pred No Yes  
## No 195 16  
## Yes 45 44

(45+16)/(195+16+45+44)

## [1] 0.2033333

2b) Training error rate = 0.0125 = 1.25%

Test data: Using a confusion matrix (False P + False N)/total: From confusion matrix: (16+45)/(195+16+45+44) Error rate is 0.2033333 = 20.3%

Significantly higher than training data error rate but is as expected.

It is expected that the test accuracy would be higher than the training error rate, as the tree was built on trained data.

There are only 6 terminal nodes produced in this classification tree. As a result it has a higher bias, lower variance and better interpretation then a larger tree.

cv.occ = cv.tree(tree.OccTrain, FUN=prune.misclass)  
cv.occ

## $size  
## [1] 6 5 2 1  
##   
## $dev  
## [1] 32 39 65 555  
##   
## $k  
## [1] -Inf 6 11 491  
##   
## $method  
## [1] "misclass"  
##   
## attr(,"class")  
## [1] "prune" "tree.sequence"

If I were to prune the tree further, the above cv error rate ($dev) is lowest at 6 nodes and thus by pruning the tree would result in a worse test error rate. Hence, pruning is not required.

2c) y = Occupancy (yes/no) x = all features other than Occupancy

library(randomForest)

## randomForest 4.6-14

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

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':  
##   
## combine

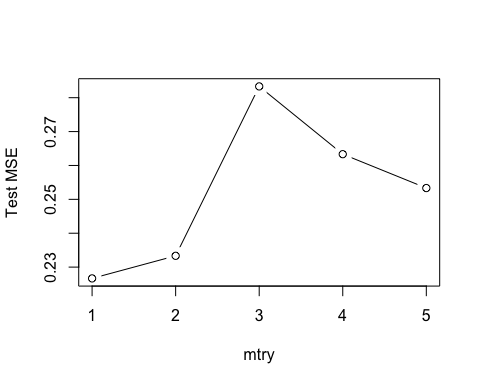
bag.occupancy = randomForest(Occupancy~. -Occupancy,   
 data = OccTrain.data,  
 mtry = 2,  
 importance = TRUE)

bag.pred = predict(bag.occupancy, newdata = Test.data, type = "class")  
print(mean(bag.pred!=Occ.Test)) #22.3%

## [1] 0.2333333

2d)

testMSE <- rep(0,5)   
for(i in 1:5){  
set.seed(4)  
rf.occ <- randomForest(Occupancy ~ ., data=OccTrain.data, mtry=i,importance=TRUE)   
yhat.rf <- predict(rf.occ, newdata=Test.data)  
testMSE[i] <- mean((yhat.rf!=Occ.Test))  
}  
plot(testMSE,type="b",xlab="mtry",ylab="Test MSE")

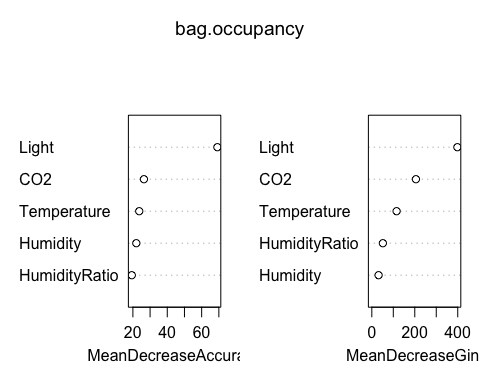
 According to the above graph to find the optimal number of features (mtry), mtry = 1, however, when running the randomforest when mtry = 2 the MSE/test error rate is the lowest. This coincides with the empirical result of mtry = squareroot(p), where p is the number of predictions. In the example above, p = 5 (minusing ‘Occupancy’) and sqrt(5) = 2.24 which is ~2.

However, the error accuracy obtained using the random forest classifier is around 21.7%. Comparatively, using a decision tree resulted in an error rate of 20.3%.

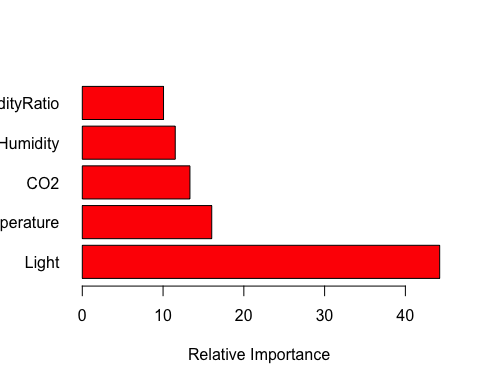
importance(bag.occupancy)

## No Yes MeanDecreaseAccuracy MeanDecreaseGini  
## Temperature 16.02591 15.35594 23.69227 115.40971  
## Humidity 11.50618 17.86059 21.98966 31.53780  
## Light 44.24628 65.73501 69.14310 398.31925  
## CO2 13.32615 24.42426 26.42940 205.42269  
## HumidityRatio 10.05628 16.21542 19.37515 51.49734

varImpPlot(bag.occupancy)

 A value close to 0 means less importance. By looking at the plots above, Light seems to be the most important variable.

barplot(sort(importance(bag.occupancy)[,1], decreasing = TRUE),   
 xlab = "Relative Importance",  
 horiz = TRUE,  
 col = "red",  
 las=1   
 )

 The larger the score given, the more influence the variable has when splitting on a particulr variable.

As seen above in the relative influence plot, the importance of features used in the random forest classifier is seen above: Light, by far, is the most important variable/feature, followed by temperature. The least important is Humidity and humidity ratio.

3a)

library(e1071)  
winedata.train = read.table("WineQuality\_training.txt", header = T, sep = ',')  
winedata.test = read.table("WineQuality\_testing.txt", header = T, sep = ',')  
  
tune.out <- tune(svm,   
 quality ~.,  
 data = winedata.train,  
 kernel = "linear",  
 ranges = list(cost = c(0.01,0.1,1,5,10),  
 gamma = c(0.01,0.03,0.1,0.5,1)))  
summary(tune.out)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost gamma  
## 0.1 0.01  
##   
## - best performance: 0.2403333   
##   
## - Detailed performance results:  
## cost gamma error dispersion  
## 1 0.01 0.01 0.2456667 0.01186862  
## 2 0.10 0.01 0.2403333 0.01773954  
## 3 1.00 0.01 0.2430000 0.01984197  
## 4 5.00 0.01 0.2420000 0.01912919  
## 5 10.00 0.01 0.2426667 0.01864549  
## 6 0.01 0.03 0.2456667 0.01186862  
## 7 0.10 0.03 0.2403333 0.01773954  
## 8 1.00 0.03 0.2430000 0.01984197  
## 9 5.00 0.03 0.2420000 0.01912919  
## 10 10.00 0.03 0.2426667 0.01864549  
## 11 0.01 0.10 0.2456667 0.01186862  
## 12 0.10 0.10 0.2403333 0.01773954  
## 13 1.00 0.10 0.2430000 0.01984197  
## 14 5.00 0.10 0.2420000 0.01912919  
## 15 10.00 0.10 0.2426667 0.01864549  
## 16 0.01 0.50 0.2456667 0.01186862  
## 17 0.10 0.50 0.2403333 0.01773954  
## 18 1.00 0.50 0.2430000 0.01984197  
## 19 5.00 0.50 0.2420000 0.01912919  
## 20 10.00 0.50 0.2426667 0.01864549  
## 21 0.01 1.00 0.2456667 0.01186862  
## 22 0.10 1.00 0.2403333 0.01773954  
## 23 1.00 1.00 0.2430000 0.01984197  
## 24 5.00 1.00 0.2420000 0.01912919  
## 25 10.00 1.00 0.2426667 0.01864549

svmWine.linear.pred = predict(tune.out$best.model, newdata = winedata.test)  
summary(tune.out$best.model)

##   
## Call:  
## best.tune(method = svm, train.x = quality ~ ., data = winedata.train,   
## ranges = list(cost = c(0.01, 0.1, 1, 5, 10), gamma = c(0.01,   
## 0.03, 0.1, 0.5, 1)), kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 0.1   
## gamma: 0.01   
##   
## Number of Support Vectors: 1752  
##   
## ( 877 875 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## Bad Good

mean(winedata.test$quality!=svmWine.linear.pred)

## [1] 0.3225

table(svmWine.linear.pred,winedata.test$quality)

##   
## svmWine.linear.pred Bad Good  
## Bad 102 89  
## Good 40 169

3c)

tune.out2 <- tune(svm,   
 quality ~.,  
 data = winedata.train,  
 kernel = "radial",  
 ranges = list(cost = c(0.01,0.1,1,5,10),  
 gamma = c(0.01,0.03,0.1,0.5,1)))  
summary(tune.out2)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost gamma  
## 5 1  
##   
## - best performance: 0.1583333   
##   
## - Detailed performance results:  
## cost gamma error dispersion  
## 1 0.01 0.01 0.2880000 0.02079886  
## 2 0.10 0.01 0.2536667 0.02208010  
## 3 1.00 0.01 0.2326667 0.02101146  
## 4 5.00 0.01 0.2116667 0.02332010  
## 5 10.00 0.01 0.2080000 0.02394438  
## 6 0.01 0.03 0.2650000 0.02050294  
## 7 0.10 0.03 0.2416667 0.02240205  
## 8 1.00 0.03 0.2086667 0.01932184  
## 9 5.00 0.03 0.2000000 0.02367475  
## 10 10.00 0.03 0.1990000 0.02233582  
## 11 0.01 0.10 0.2680000 0.02149935  
## 12 0.10 0.10 0.2096667 0.02322461  
## 13 1.00 0.10 0.1903333 0.02612647  
## 14 5.00 0.10 0.1826667 0.01967639  
## 15 10.00 0.10 0.1803333 0.02545633  
## 16 0.01 0.50 0.5210000 0.01735967  
## 17 0.10 0.50 0.2296667 0.03032540  
## 18 1.00 0.50 0.1640000 0.02164814  
## 19 5.00 0.50 0.1606667 0.02292231  
## 20 10.00 0.50 0.1653333 0.02435337  
## 21 0.01 1.00 0.5220000 0.01611686  
## 22 0.10 1.00 0.2416667 0.06228667  
## 23 1.00 1.00 0.1600000 0.01885618  
## 24 5.00 1.00 0.1583333 0.02013841  
## 25 10.00 1.00 0.1606667 0.02041544

svmWine.radial.pred<-predict(tune.out2$best.model,newdata=winedata.test)   
summary(tune.out2$best.model)

##   
## Call:  
## best.tune(method = svm, train.x = quality ~ ., data = winedata.train,   
## ranges = list(cost = c(0.01, 0.1, 1, 5, 10), gamma = c(0.01,   
## 0.03, 0.1, 0.5, 1)), kernel = "radial")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 5   
## gamma: 1   
##   
## Number of Support Vectors: 2205  
##   
## ( 1084 1121 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## Bad Good

mean(winedata.test$quality!=svmWine.radial.pred)

## [1] 0.3625

table(svmWine.radial.pred,winedata.test$quality)

##   
## svmWine.radial.pred Bad Good  
## Bad 118 121  
## Good 24 137

3e)

library(gplots)

##   
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':  
##   
## lowess

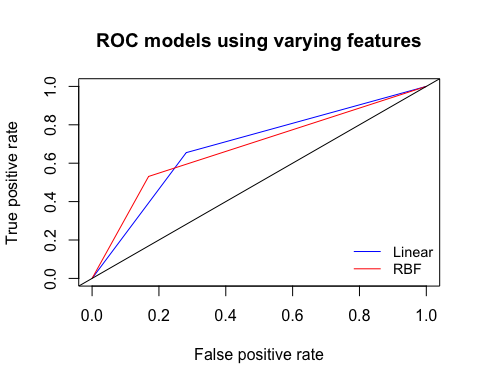
library(ROCR)  
v1 = as.numeric(svmWine.linear.pred)  
pr\_trained\_model1 <- prediction(v1, winedata.test$quality)   
auroc\_trained\_model1 <- performance(pr\_trained\_model1, measure = "auc")   
auroc\_trained\_model\_value1 <- auroc\_trained\_model1@y.values[[1]]   
print(paste("The AUROC value of the trained model using linear kernel is", auroc\_trained\_model\_value1,"."))

## [1] "The AUROC value of the trained model using linear kernel is 0.686674309422426 ."

v2 = as.numeric(svmWine.radial.pred)  
pr\_trained\_model2 <- prediction(v2, winedata.test$quality)  
auroc\_trained\_model2 <- performance(pr\_trained\_model2, measure = "auc")   
auroc\_trained\_model\_value2 <- auroc\_trained\_model2@y.values[[1]]   
print(paste("The AUROC value of the trained model using RBF kernel is", auroc\_trained\_model\_value2,"."))

## [1] "The AUROC value of the trained model using RBF kernel is 0.680996833715471 ."

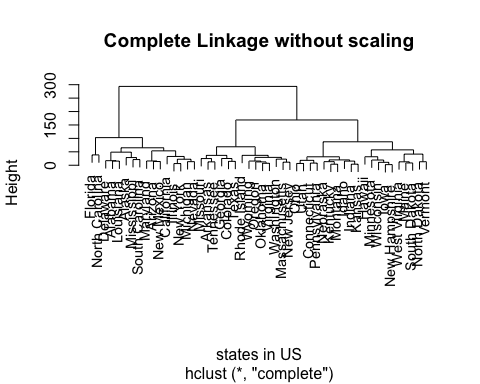
prf\_trained\_model1 <- performance(pr\_trained\_model1, measure = "tpr", x.measure = "fpr")   
prf\_trained\_model2 <- performance(pr\_trained\_model2, measure = "tpr", x.measure = "fpr")   
  
plot(prf\_trained\_model1, col= 'Blue', main = "ROC models using varying features")  
plot(prf\_trained\_model2, col= 'Red', add = TRUE)  
  
legend("bottomright",  
 legend = c('Linear','RBF'),   
 lty=1, cex=0.9, bty="n", col = c("Blue","Red")  
 )   
abline(a = 0, b = 1)



The larger area under a ROC curve indicates better accuracy, therefore using ‘RBF’ kernel obtained the best accuracy compared to ‘Linear’ (0.689,0.687 respectively), although only slightly. ‘Linear’ had a slightly higher peak TPR.

4a)

library(ISLR)  
data = USArrests  
hclust.cor.comp <- hclust(dist(data), method="complete")  
plot(hclust.cor.comp, main="Complete Linkage without scaling",xlab= "states in US", cex=0.9)

 4b)

tree = cutree(hclust.cor.comp,3)  
show(tree)

## 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

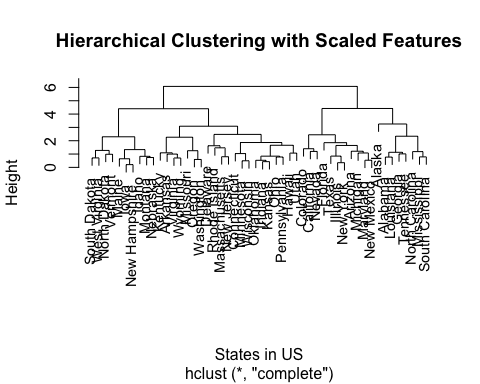
Cluster 1: Alabama, Alaska, Arizona, California, Delaware, Florida, Illinois, Louisiana, Maryland, Michigan, Misssissippi, Nevada, New Mexico, New York, North Carolina, South Carolina,

Cluster 2: Arkansas, Colorado, Georgia, Massachusetts, Missouri, New Jersey, Oklahoma, Oregon, Rhode Island, Tennessee, Texas, Virginia, Washington, Wyoming.

Cluster 3: Connecticut, Hawaii, Idaho, Indiana, Iowa, Kansas, Kentucky, Maine, Minnesota, Montana, Nebraska, New Hampshire, North Dakota, Ohio, Pennsylvania, South Dakota, Vermont, West Virginia, Wisconsin.

4c)

scaled.data <- scale(data)   
hc.scaled.data = hclust(dist(scaled.data),method = "complete")  
plot(hc.scaled.data,xlab = "States in US", main="Hierarchical Clustering with Scaled Features", cex = 0.9)



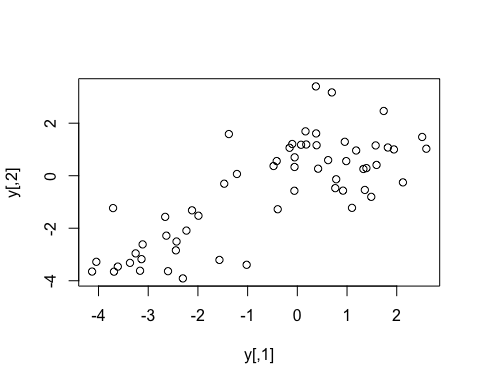
4d) The effect of scaling the variables results in a possible 4 clusters being present in the dendrogram at around height 4.5. Also, the height scale has significantly reduced.

The variables should be scaled before dissimilarities are computed because the variables are all utilising different scales and have ranging variances. By scaling them, to have mean = 0 and standard deviation = 1, each variable will be given equal importance/weighting in the clustering performed.

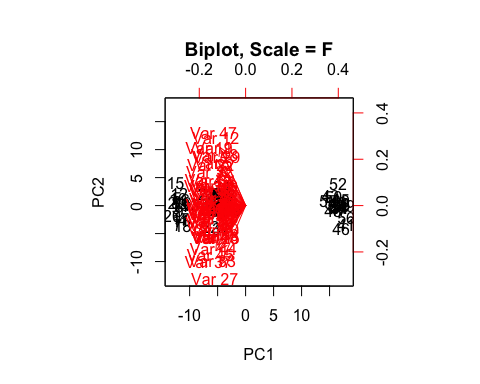
Additionally, as seen in the dendrograms above, scaling effects what states go into what clusters.

5a)

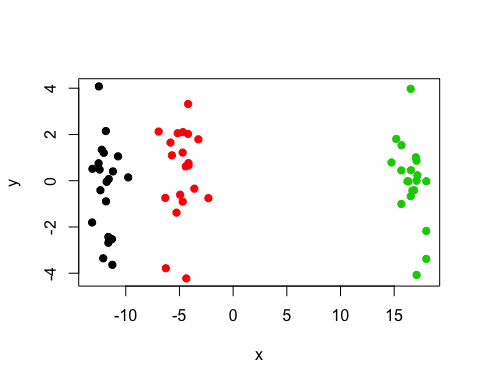
set.seed(1)  
y = matrix(rnorm(60\*50), nrow = 60, ncol = 50)  
  
y[1:20,1:50] = y[1:20,1:50] +1  
y[41:60,1:50] = y[41:60,1:50]-3  
plot(y)



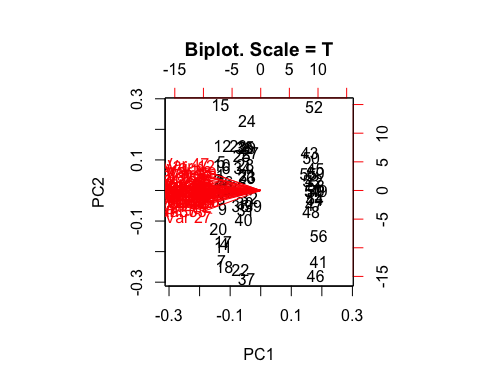
labels = c(rep(1,20),rep(2,20),rep(3,20))  
  
  
#5b)  
pc.y <- prcomp(y, scale=FALSE)  
  
#pc.y$rotation  
biplot(pc.y, scale=FALSE, main = "Biplot, Scale = F")



plot(pc.y$x[,1:2],  
 col = labels,  
 xlab="x",ylab="y",  
 pch=19)



biplot(pc.y, scale=T, main = "Biplot. Scale = T")



5c)

km.y3 = kmeans(y,3, nstart = 100)  
table(clustered = km.y3$cluster,original = labels)

## original  
## clustered 1 2 3  
## 1 0 20 0  
## 2 0 0 20  
## 3 20 0 0

The clusters obtained in K-means clustering are all correct and match the true class labels.

5d) Performing K = 2 results in the original clusters 1 and 2 being joined together to form a single cluster (clustered 2). Original 3 remained unchanged.

km.y2 = kmeans(y,2, nstart = 100)  
table(clustered = km.y2$cluster,original = labels)

## original  
## clustered 1 2 3  
## 1 0 0 20  
## 2 20 20 0

5e) Using K = 4, the original cluster 3 was split into two separate clusters. Original 1 and 2 remained the same.

km.y4 = kmeans(y,4, nstart = 100)  
table(clustered = km.y4$cluster,original = labels)

## original  
## clustered 1 2 3  
## 1 10 0 0  
## 2 10 0 0  
## 3 0 20 0  
## 4 0 0 20

5f) The clusters remain correct and match the original cluster labels.

km.y3 = kmeans(pc.y$x[,1:2],3, nstart=100)  
table(clustered =km.y3$cluster, original = labels)

## original  
## clustered 1 2 3  
## 1 0 20 0  
## 2 0 0 20  
## 3 20 0 0

5g) Scaling the data has no effect on the clustering - all are clustered correctly. Scaling is extremely useful for data that use different measuerment scales (e.g. grams vs kilograms) as increased weighting is given to those of higher variance. Seeing as all the variables are of similar scale, weighting is more evenly distributed and does not effect the results.

km.y3.scaled = kmeans(scale(y), 3, nstart=100)  
table(clustered = km.y3.scaled$cluster, original = labels)

## original  
## clustered 1 2 3  
## 1 20 0 0  
## 2 0 20 0  
## 3 0 0 20