CW2\_13154855\_A\_GUO

Anyi Guo, MSc Data Science, student id: 13154855

30/12/2018

## 1. Decision Trees (10% | 20%)

This question relates to the following figure.

1. Sketch the tree corresponding to the partition of the predictor space illustrated in the left-hand panel of the figure above. The numbers inside the boxes indicate the mean of Y within each region.
2. Create a diagram similar to the left-hand panel of the figure, using the tree illustrated in the right-hand panel of the same figure. You should divide up the predictor space into the correct regions, and indicate the mean for each region.

**Answer**: Please see attached for the sketches.

## 2. Regression Trees (15% | 20%)

In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable. Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

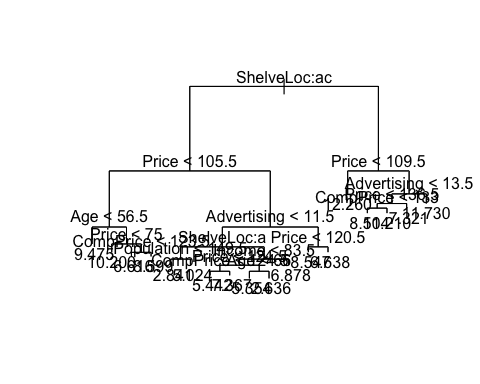
#### Split the data set into a training set and a test set.

We’ll use 70% for training set (280 observations) and 30% for testing set (120 observations)

library(ISLR)  
library(tree)  
set.seed(1)  
train<-sample(1:nrow(Carseats),280)  
training<-Carseats[train,]  
testing<-Carseats[-train,]

#### Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

tree.train<-tree(Carseats$Sales~.,Carseats,subset=train)  
plot(tree.train)  
text(tree.train)



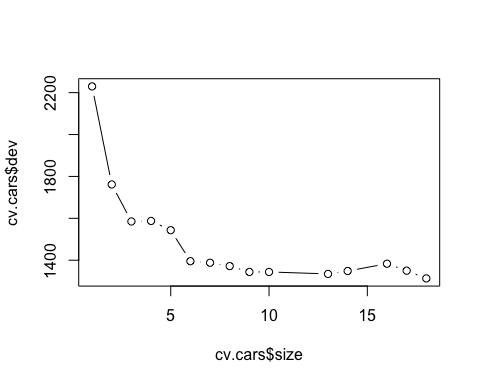
Test MSE is 5.288256.

yhat<-predict(tree.train,newdata=testing)  
mean((yhat-testing$Sales)^2)

## [1] 5.288256

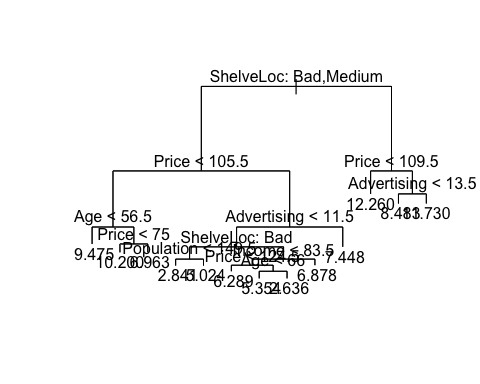
#### Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test error rate?

cv.cars<-cv.tree(tree.train)  
plot(cv.cars$size,cv.cars$dev,type="b")



CV indicated that the minimum MSE is when tree size is 13 or 10, so we’ll prune the tree to be of size 13.

prune.cars<-prune.tree(tree.train,best=13)  
plot(prune.cars)  
text(prune.cars,pretty=0)



yhat2<-predict(prune.cars,newdata=testing)  
mean((yhat2-testing$Sales)^2)

## [1] 5.162007

Test MSE after pruning is 5.162007. It is an improvement from without the pruning (which was 5.288256).

#### Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important.

library(randomForest)

## randomForest 4.6-14

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

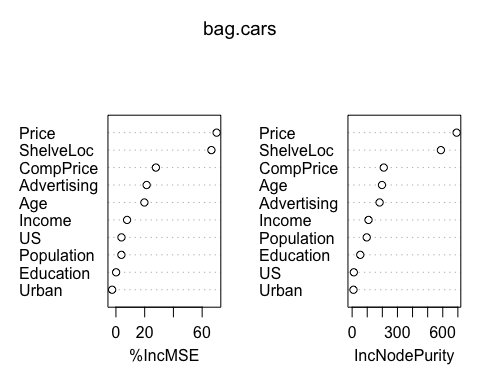
set.seed(1)  
bag.cars<-randomForest(Sales~.,data=Carseats,subset=train,mtry=10,importance=TRUE)  
  
yhat.bag<-predict(bag.cars,newdata = testing)  
mean((yhat.bag-testing$Sales)^2)

## [1] 2.363483

importance(bag.cars)

## %IncMSE IncNodePurity  
## CompPrice 27.8135565 209.349577  
## Income 7.7424034 108.501622  
## Advertising 21.3697626 181.735494  
## Population 3.8004106 96.555441  
## Price 70.0655453 691.754195  
## ShelveLoc 66.4327264 587.670575  
## Age 19.7964509 197.592670  
## Education 0.1123697 53.821669  
## Urban -2.6501254 8.846458  
## US 3.8512787 11.371161

varImpPlot(bag.cars)

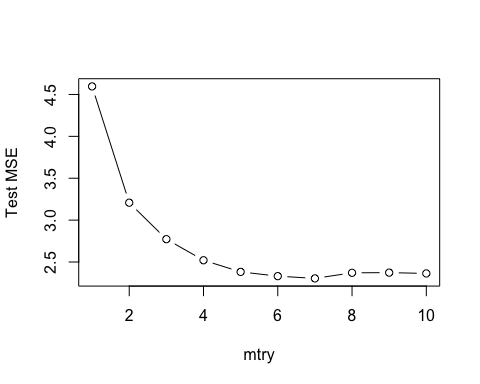
 Test MSE for the bagging method is 2.363483, which is a lot lower than without bagging.

The most important (top 3) variables are: Price, ShelveLoc, CompPrice

#### Use random forests to analyze this data. What test error rate do you obtain? Use the importance() function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the test MSE obtained.

Best value for mtry: it seems that mtry=7 would give the lowest test MSE

testMSE <- rep(0,10)   
for(i in 1:10){  
 set.seed(1)  
 rf.cars <- randomForest(Sales ~ ., data=Carseats, subset=train, mtry=i,importance=TRUE)   
 yhat.rf <- predict(rf.cars,newdata=testing)  
 testMSE[i] <- mean((yhat.rf-testing$Sales)^2)  
}  
plot(testMSE,type="b",xlab="mtry",ylab="Test MSE")



set.seed(1)  
rf.cars<-randomForest(Sales ~.,data=Carseats,subset=train,mtry=7,importance=TRUE)  
yhat.rf<-predict(rf.cars,newdata=testing)  
mean((yhat.rf-testing$Sales)^2)

## [1] 2.304234

When we use randomForest with mtry=7, the test MSE is 2.304234, which is even lower than the test MSE with bagging (which is 2.363483).

Effect of m (# of variables considered at each split):

1. Before m=7, the test MSE goes down as m increases.
2. After m=7, the test MSE goes up slightly as m increases.

## 3. Classification Trees (15% | 20%)

This problem involves the OJ data set which is part of the ISLR package.

#### Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

library(ISLR)  
data(OJ)  
set.seed(1)  
train<-sample(1:nrow(OJ),800)  
training.set<-OJ[train,]  
testing.set<-OJ[-train,]

#### Fit a tree to the training data, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

tree.OJ<-tree(Purchase~.,data=OJ,subset=train)  
summary(tree.OJ)

##   
## Classification tree:  
## tree(formula = Purchase ~ ., data = OJ, subset = train)  
## Variables actually used in tree construction:  
## [1] "LoyalCH" "PriceDiff" "SpecialCH" "ListPriceDiff"  
## Number of terminal nodes: 8   
## Residual mean deviance: 0.7305 = 578.6 / 792   
## Misclassification error rate: 0.165 = 132 / 800

Training error rate is 0.165. The tree has 8 terminal nodes.

#### Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

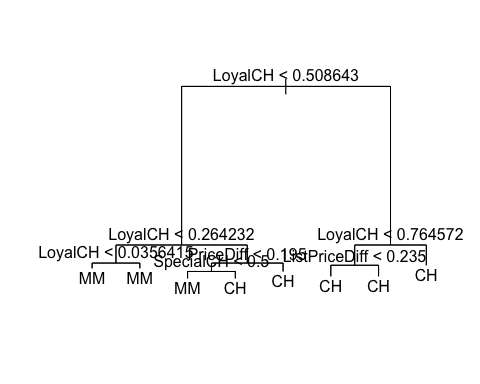
tree.OJ

## node), split, n, deviance, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 800 1064.00 CH ( 0.61750 0.38250 )   
## 2) LoyalCH < 0.508643 350 409.30 MM ( 0.27143 0.72857 )   
## 4) LoyalCH < 0.264232 166 122.10 MM ( 0.12048 0.87952 )   
## 8) LoyalCH < 0.0356415 57 10.07 MM ( 0.01754 0.98246 ) \*  
## 9) LoyalCH > 0.0356415 109 100.90 MM ( 0.17431 0.82569 ) \*  
## 5) LoyalCH > 0.264232 184 248.80 MM ( 0.40761 0.59239 )   
## 10) PriceDiff < 0.195 83 91.66 MM ( 0.24096 0.75904 )   
## 20) SpecialCH < 0.5 70 60.89 MM ( 0.15714 0.84286 ) \*  
## 21) SpecialCH > 0.5 13 16.05 CH ( 0.69231 0.30769 ) \*  
## 11) PriceDiff > 0.195 101 139.20 CH ( 0.54455 0.45545 ) \*  
## 3) LoyalCH > 0.508643 450 318.10 CH ( 0.88667 0.11333 )   
## 6) LoyalCH < 0.764572 172 188.90 CH ( 0.76163 0.23837 )   
## 12) ListPriceDiff < 0.235 70 95.61 CH ( 0.57143 0.42857 ) \*  
## 13) ListPriceDiff > 0.235 102 69.76 CH ( 0.89216 0.10784 ) \*  
## 7) LoyalCH > 0.764572 278 86.14 CH ( 0.96403 0.03597 ) \*

I choose the first node from the left (case 8)). This means that the choice would be Minute Maid MM if LoyalCH is < 0.0356415.

#### Create a plot of the tree, and interpret the results.

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



I pick the third terminal node from the left (20) in the model): it tells us that a customer is likely to buy MM under the following conditions:

1. When a customer’s brand loyalty for Citrus Hill is lower than 0.264232 (LoyalCH < 0.264232)
2. When the price difference between the two brands is less than 0.195 (MM being cheaper than CH)
3. When special on Citrus Hill is less than 0.5

#### Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

pred.OJ.unpruned<-predict(tree.OJ,testing.set,type="class")  
table(pred.OJ.unpruned,testing.set$Purchase)

##   
## pred.OJ.unpruned CH MM  
## CH 147 49  
## MM 12 62

Testing error rate = (12+49)/270=22.59%

#### Apply the cv.tree() function to the training set in order to determine the optimal tree size.

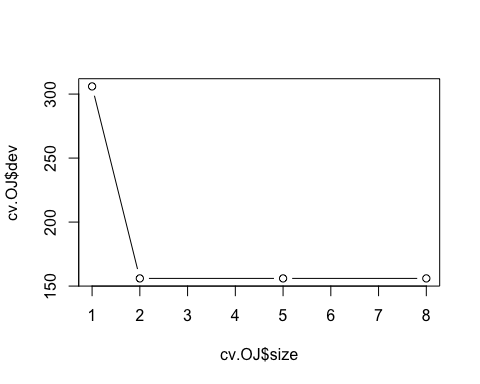
cv.OJ<-cv.tree(tree.OJ,FUN=prune.misclass)  
cv.OJ

## $size  
## [1] 8 5 2 1  
##   
## $dev  
## [1] 156 156 156 306  
##   
## $k  
## [1] -Inf 0.000000 4.666667 160.000000  
##   
## $method  
## [1] "misclass"  
##   
## attr(,"class")  
## [1] "prune" "tree.sequence"

It seems that the error rate is the same & lowest whether we use 8,5 or 2.

#### Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

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



#### Which tree size corresponds to the lowest cross-validated classification error rate?

When tree size is 8,5 or 2, they produce the same lowest CV classification error rate.

#### Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

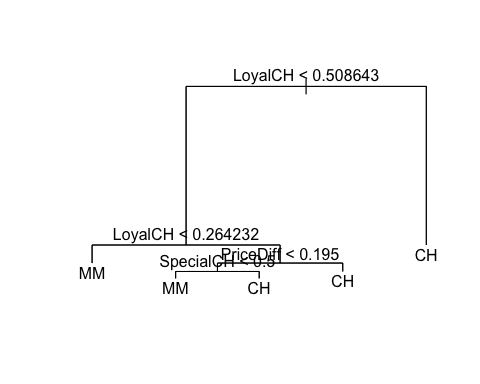
We’ll use tree size = 5 for the pruning.

prune.OJ<-prune.misclass(tree.OJ,best=5)  
tree.pred.pruned<-predict(prune.OJ,testing.set,type="class")  
table(tree.pred.pruned,testing.set$Purchase)

##   
## tree.pred.pruned CH MM  
## CH 147 49  
## MM 12 62

Test error rate for the pruned tree is (12+49)/270 = 22.59%

plot(prune.OJ)  
text(prune.OJ,pretty=0)



#### Compare the training error rates between the pruned and unpruned trees. Which is higher?

summary(tree.OJ)

##   
## Classification tree:  
## tree(formula = Purchase ~ ., data = OJ, subset = train)  
## Variables actually used in tree construction:  
## [1] "LoyalCH" "PriceDiff" "SpecialCH" "ListPriceDiff"  
## Number of terminal nodes: 8   
## Residual mean deviance: 0.7305 = 578.6 / 792   
## Misclassification error rate: 0.165 = 132 / 800

summary(prune.OJ)

##   
## Classification tree:  
## snip.tree(tree = tree.OJ, nodes = 3:4)  
## Variables actually used in tree construction:  
## [1] "LoyalCH" "PriceDiff" "SpecialCH"  
## Number of terminal nodes: 5   
## Residual mean deviance: 0.8256 = 656.4 / 795   
## Misclassification error rate: 0.165 = 132 / 800

The training error rate for unpruned and pruned tree is the same (when we select 5 as the tree size): it is 0.165

#### Compare the test error rates between the pruned and unpruned trees. Which is higher?

Test error rate is the same for the pruned vs. unpruned trees, (12+49)/270 = 22.59%

# unpruned  
table(pred.OJ.unpruned,testing.set$Purchase)

##   
## pred.OJ.unpruned CH MM  
## CH 147 49  
## MM 12 62

# pruned  
table(tree.pred.pruned,testing.set$Purchase)

##   
## tree.pred.pruned CH MM  
## CH 147 49  
## MM 12 62

## 4. SVM (15% | 20%)

In this problem, you will use support vector approaches in order to predict whether a given car gets high or low gas mileage based on the Auto data set.

#### Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

average<-median(Auto$mpg)  
High<-ifelse(Auto$mpg>=average,1,0)  
Auto <- data.frame(Auto,as.factor(High))  
colnames(Auto)[10]<-"High"

#### Fit a support vector classifier to the data with various values of cost, in order to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter. Comment on your results.

library(e1071)  
set.seed(1)  
train<-sample(1:nrow(Auto),196)  
training.set2<-Auto[train,]  
testing.set2<-Auto[-train,]  
  
tune.out<-tune(svm,High~.,data=training.set2,kernel="linear",ranges=list(cost=c(0.001,0.01,0.1,1,10,100)))  
summary(tune.out)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 1  
##   
## - best performance: 0.02026316   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.11236842 0.05264693  
## 2 1e-02 0.09184211 0.04678737  
## 3 1e-01 0.07657895 0.06021979  
## 4 1e+00 0.02026316 0.02617065  
## 5 1e+01 0.03105263 0.03648111  
## 6 1e+02 0.03105263 0.03648111

The training error rate is the smallest when cost = 1. The training error when cost=1 is 0.02026316.

best<-tune.out$best.model  
summary(best)

##   
## Call:  
## best.tune(method = svm, train.x = High ~ ., data = training.set2,   
## ranges = list(cost = c(0.001, 0.01, 0.1, 1, 10, 100)), kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
## gamma: 0.003205128   
##   
## Number of Support Vectors: 39  
##   
## ( 20 19 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

ypred<-predict(best,testing.set2)  
table(predict=ypred,truth=testing.set2$High)

## truth  
## predict 0 1  
## 0 92 2  
## 1 1 101

The testing error rate for the best model (when cost = 1) is (1+2)/196 = 0.0153

#### Now repeat (b), this time using SVMs with radial and polynomial basis kernels, with different values of gamma and degree and cost. Comment on your results.

### Radial kernel

set.seed(1)  
tune.out.radial <- tune(svm, High ~ ., data=training.set2, kernel="radial",ranges = list(cost=c(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)))  
tune.out.radial$best.model

##   
## Call:  
## best.tune(method = svm, train.x = High ~ ., data = training.set2,   
## 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: 1   
## gamma: 0.5   
##   
## Number of Support Vectors: 165

For **radial kernel**, the training error rate is the lowest when cost = 1 and gamma = 0.5

svmfit.radial <- svm(High~.,data=training.set2,kernel="radial",cost=1,gamma=0.5)  
ypred.radial <- predict(svmfit.radial,testing.set2)  
table(ypred.radial,testing.set2$High)

##   
## ypred.radial 0 1  
## 0 87 1  
## 1 6 102

Testing error rate is the lowest for the best fitting SVM radial model when cost = 1 and gamma = 0.5: (1+6)/196 = 0.0357

### Polynomial kernel:

set.seed(1)  
tune.out <- tune(svm, High ~ ., data=training.set2, kernel="polynomial",ranges = list(cost=c(0.1,1,10,100,1000),degree=c(1,2,3,4,5)))  
tune.out$best.model

##   
## Call:  
## best.tune(method = svm, train.x = High ~ ., data = training.set2,   
## ranges = list(cost = c(0.1, 1, 10, 100, 1000), degree = c(1,   
## 2, 3, 4, 5)), kernel = "polynomial")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: polynomial   
## cost: 1000   
## degree: 1   
## gamma: 0.003205128   
## coef.0: 0   
##   
## Number of Support Vectors: 27

The best performing **polynomial** SVM model based on training data is when cost = 1000 and degree = 1

# polynomial  
  
svmfit.polynomial <- svm(High~.,data=training.set2,kernel="polynomial",cost=1000,degree=1)  
ypred.polynomial <- predict(svmfit.polynomial,testing.set2)  
table(ypred.polynomial,testing.set2$High)

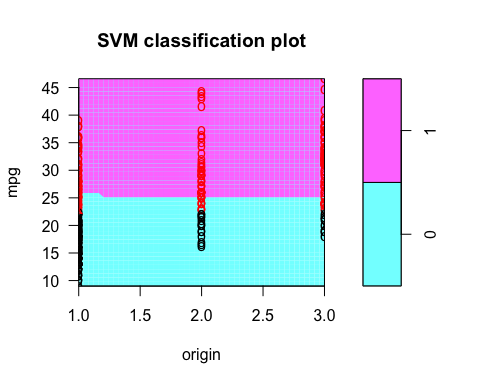
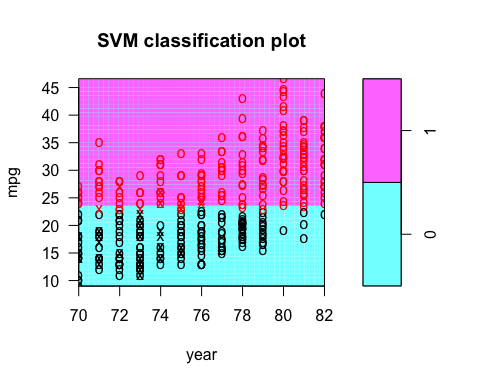
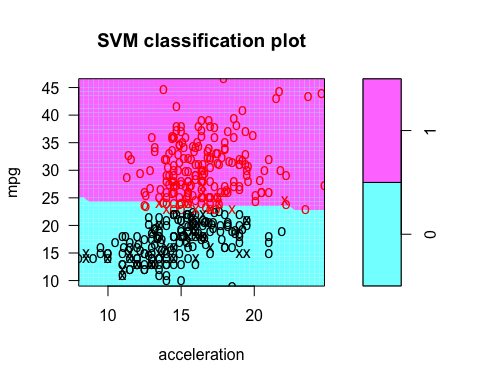
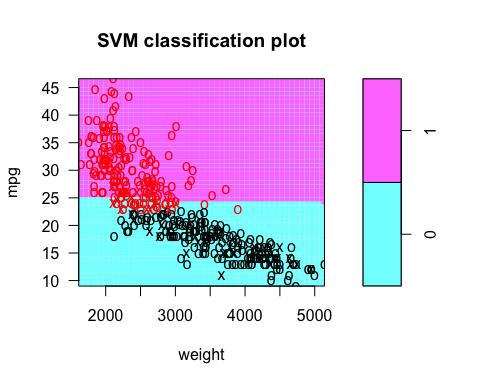
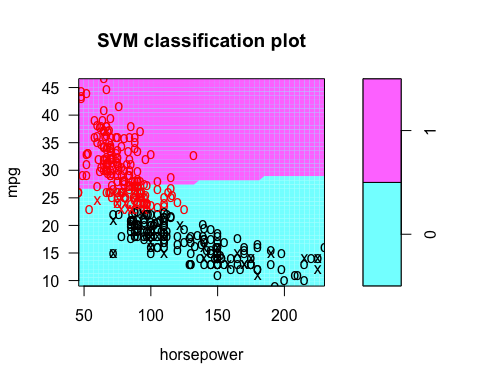
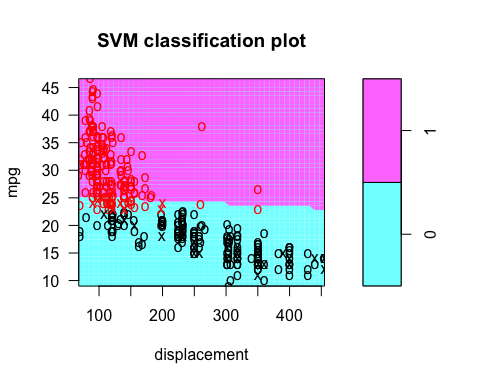
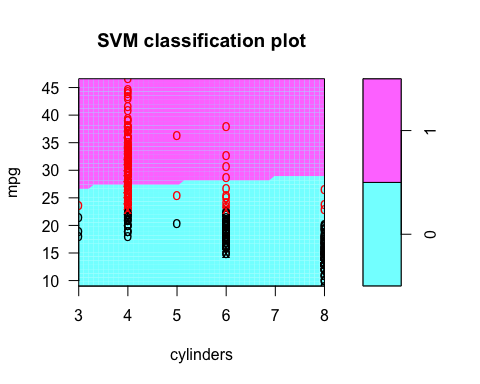
##   
## ypred.polynomial 0 1  
## 0 92 2  
## 1 1 101

The testing error rate for the best performing SVM polynomial model is when cost = 1000 and degree = 1. Testing error rate = (1+2)/196 = 0.01531

#### Make some plots to back up your assertions in (b) and (c).

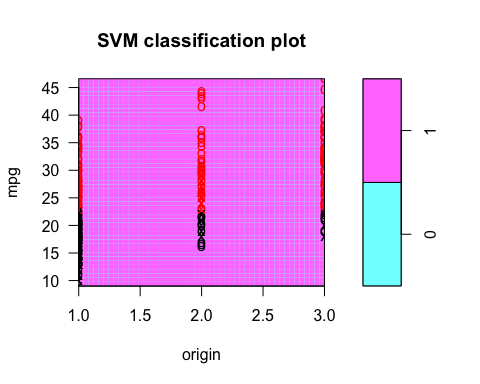
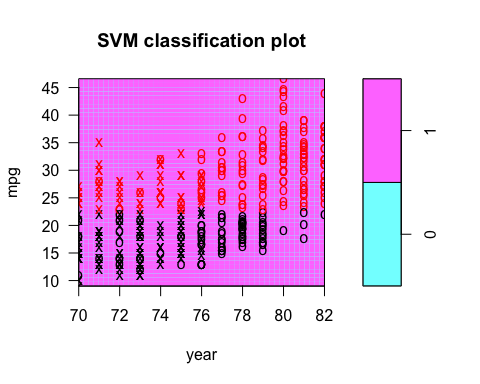
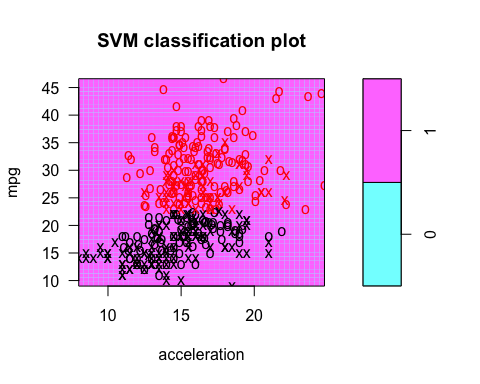
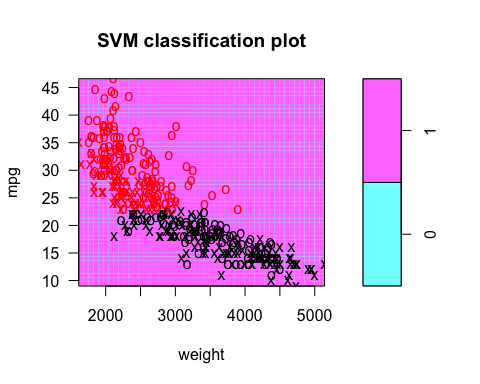
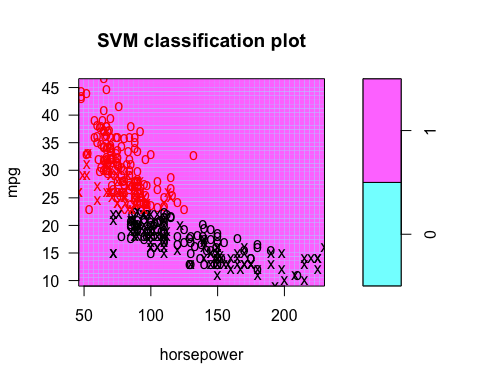
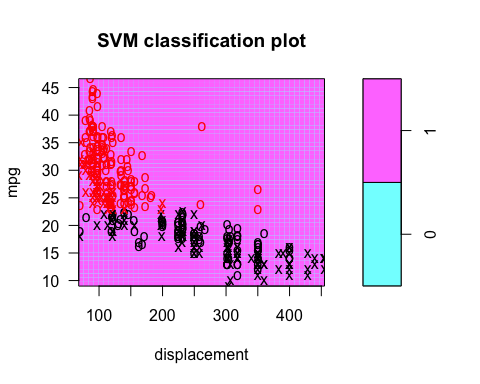
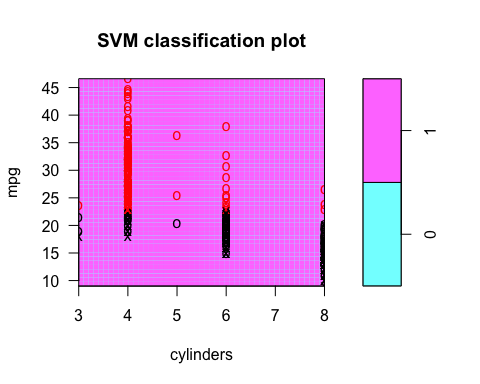
### Linear kernel

# linear  
# not plotting against the 9th, 10th and 1st column (respectively "name","High" and "mpg")  
for (i in names(Auto)[c(-9,-10,-1)]){  
 plot(best,Auto,as.formula(paste("mpg~",i,sep="")),main="linear")  
}



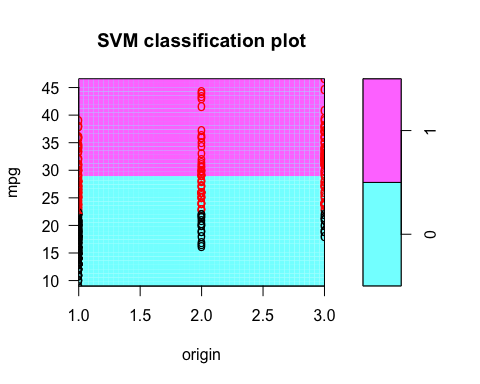
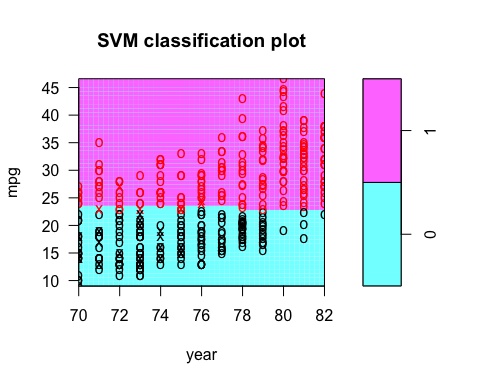
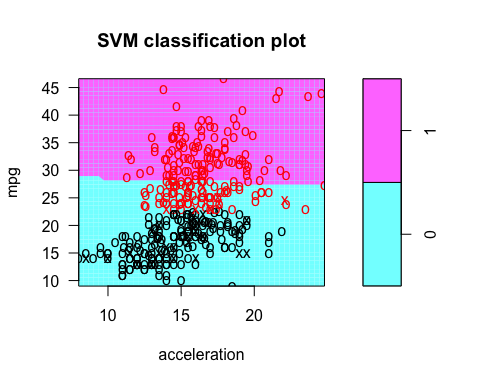
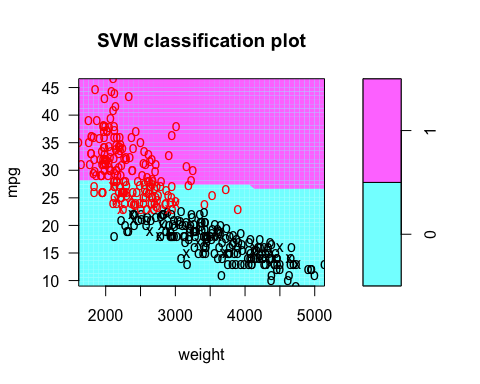
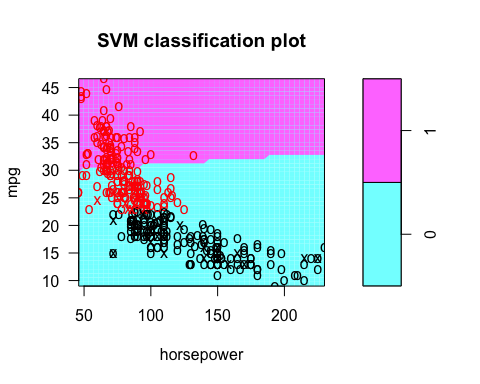
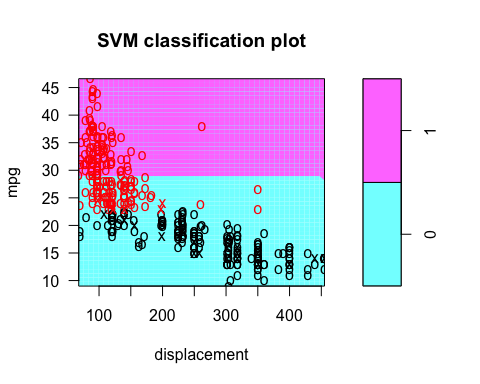
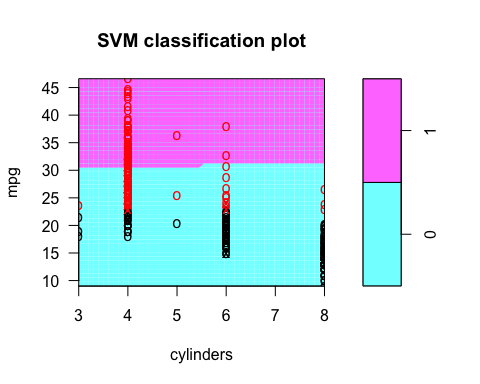
### Radial kernel

# radial  
for (i in names(Auto)[c(-9,-10,-1)]){  
 plot(svmfit.radial,Auto,as.formula(paste("mpg~",i,sep="")))  
}



### Polynomial kernel

# polynomial  
for (i in names(Auto)[c(-9,-10,-1)]){  
 plot(svmfit.polynomial,Auto,as.formula(paste("mpg~",i,sep="")))  
}

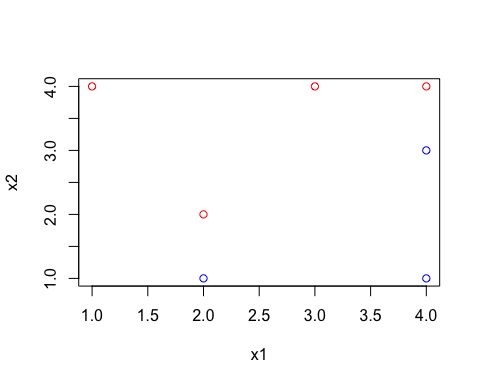


## 5. SVM (15% | 0%)

Here we explore the maximal margin classifier on a toy data set.

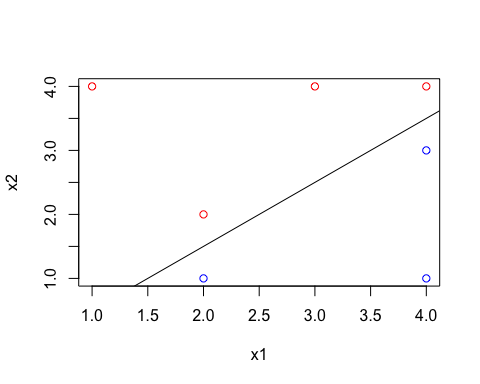
#### We are given n = 7 observations in p = 2 dimensions. For each observation, there is an associated class label. Sketch the observations.

x1<-c(3,2,4,1,2,4,4)  
x2<-c(4,2,4,4,1,3,1)  
color<-c("red","red","red","red","blue","blue","blue")  
plot(x1,x2,col=color)

 #### Sketch the optimal separating hyperplane, and provide the equation for this hyperplane of the following form. β0 + β1X1 + β2X2 = 0

The optimal separating hyperplane should go through position of (2,1.5) and (4,3.5). Therefore, we know that the intercept should be -0.5, and the slope should be 1.

plot(x1,x2,col=color)+abline(-0.5,1)



## integer(0)

The formula is y=x-0.5, which can be translated into x1-x2-0.5=0.

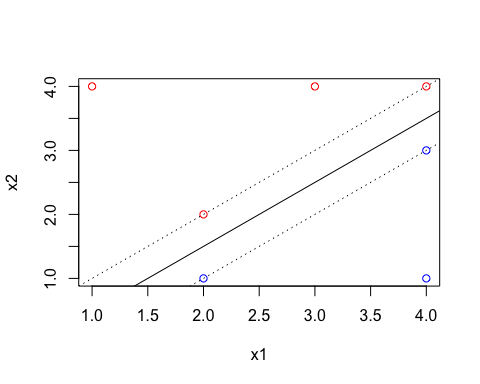
#### Describe the classification rule for the maximal margin classifier. It should be something along the lines of “Classify to Red if β0 + β1X1 + β2X2 > 0, and classify to Blue otherwise.” Provide the values for β0, β1, and β2.

* β0 =-0.5
* β1 = 1
* β2 =-1

**Classification rule**: Classify to **red** if β0 + β1X1 + β2X2 < 0. Otherwise, classify to **blue**

#### On your sketch, indicate the margin for the maximal margin hyperplane.

plot(x1,x2,col=color)+abline(-0.5,1)+  
 # margins for the maximal margin hyperplane  
 abline(-1, 1, lty = 3)+abline(0, 1, lty = 3)



## integer(0)

#### Indicate the support vectors for the maximal margin classifier.

There are 4 support vectors: 1. blue point at (2,1) 2. blue point at (4,3) 3. red point at (2,2) 4. red point at (4,4)

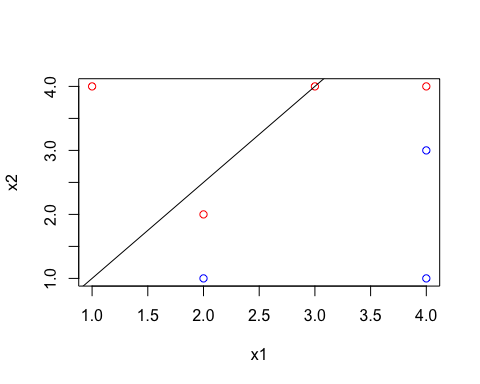
#### Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

The 7th observation is (4,1), which is neither a support vector nor a point on the margin for max margin hyperplane. Therefore, moving it would not affect the max margin hyperplane.

#### Sketch a hyperplane that is not the optimal separating hyperplane, and provide the equation for this hyperplane.

For example:

plot(x1,x2,col=color)+abline(-0.5,1.5)



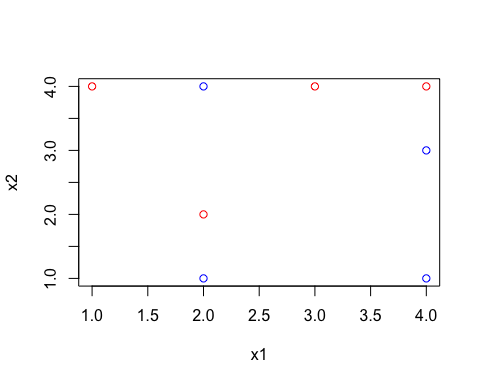
## integer(0)

The formula is 1.5x1-x2-0.5=0.

* β0 =-0.5
* β1 = 1.5
* β2 =-1

#### Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

plot(x1,x2,col=color)+  
 # draw an additional point which confuses the hyperplane  
 points(c(2), c(4), col = c("blue"))



## integer(0)

## 6. Hierarchical clustering (10% | 20%)

Consider the USArrests data. We will now perform hierarchical clustering on the states.

#### Using hierarchical clustering with complete linkage and Euclidean distance, cluster the states.

hc.complete<-hclust(dist(USArrests),method="complete")

#### Cut the dendrogram at a height that results in three distinct clusters. Which states belong to which clusters?

cutree(hc.complete,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

d<-as.data.frame(cutree(hc.complete,3))  
d<-data.frame(names=row.names(d),d)  
colnames(d)<-c("State","Group")  
  
# States in the first cluster  
State\_cluster1<-d[which(d$Group==1),]$State  
State\_cluster1

## [1] Alabama Alaska Arizona California   
## [5] Delaware Florida Illinois Louisiana   
## [9] Maryland Michigan Mississippi Nevada   
## [13] New Mexico New York North Carolina South Carolina  
## 50 Levels: Alabama Alaska Arizona Arkansas California ... Wyoming

# States in second cluster  
State\_cluster2<-d[which(d$Group==2),]$State  
State\_cluster2

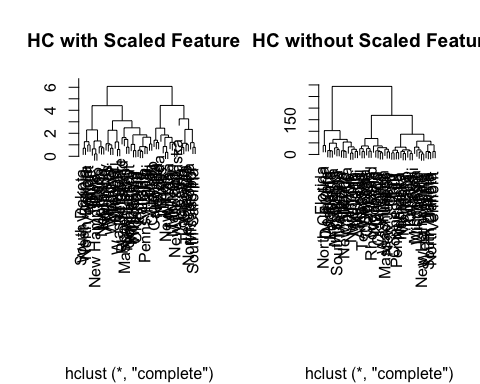
## [1] Arkansas Colorado Georgia Massachusetts Missouri   
## [6] New Jersey Oklahoma Oregon Rhode Island Tennessee   
## [11] Texas Virginia Washington Wyoming   
## 50 Levels: Alabama Alaska Arizona Arkansas California ... Wyoming

# States in third cluster  
State\_cluster3<-d[which(d$Group==3),]$State  
State\_cluster3

## [1] Connecticut Hawaii Idaho Indiana Iowa   
## [6] Kansas Kentucky Maine Minnesota Montana   
## [11] Nebraska New Hampshire North Dakota Ohio Pennsylvania   
## [16] South Dakota Utah Vermont West Virginia Wisconsin   
## 50 Levels: Alabama Alaska Arizona Arkansas California ... Wyoming

#### Hierarchically cluster the states using complete linkage and Euclidean distance, after scaling the variables to have standard deviation one.

xsc<-scale(USArrests)  
hc.complete.scaled<-hclust(dist(xsc),method="complete")  
  
par(mfrow=c(1,2))  
plot(hc.complete.scaled,main="HC with Scaled Features",xlab="",ylab="")  
plot(hc.complete,main="HC without Scaled Features",xlab="",ylab="")



#### What effect does scaling the variables have on the hierarchical clustering obtained? In your opinion, should the variables be scaled before the inter-observation dissimilarities are computed? Provide a justification for your answer.

Scaling the variables gives equal weights to the different crimes in the dataset. For example, in the dataset there are a lot more occurances of Assault than Murder, which means that not scaling the variables would give a lot more weight/importance to assault than any other crime. Scaling solves this problem.

## 7. PCA and K-Means Clustering (20% | 0%)

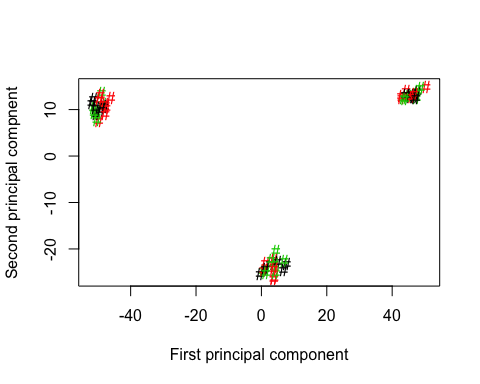
In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data.

#### Generate a simulated data set with 20 observations in each of three classes (i.e. 60 observations total), and 50 variables.

# Hint: There are a number of functions in R that you can use to generate data. One example is the rnorm() function; runif() is another option. Be sure to add a mean shift to the observations in each class so that there are three distinct classes.  
set.seed(1)  
df <- matrix(rnorm(60\*50, mean = 0, sd = 2),ncol=50)  
# Split the rows into three groups, keeping row 41-60 unchanged  
df[1:20, ] = df[1:20, ]-10  
df[21:40, ] = df[21:40, ] +10  
  
# Split same row into three different groups  
df[1:20, 11:20] = 1  
df[21:40, 1:10] = -4  
df[21:40, 21:31] = -4  
df[41:60, 11:20] = 1

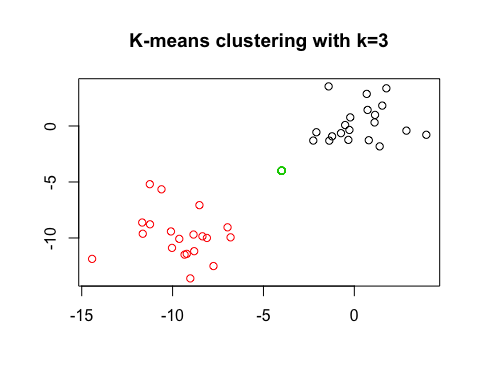
#### Perform PCA on the 60 observations and plot the first two principal components’ eigenvector. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue on to part (c). If not, then return to part (a) and modify the simulation so that there is greater separation between the three classes. Do not continue to part (c) until the three classes show at least some separation in the first two principal component eigenvectors.

pca.out = prcomp(df,scale=FALSE)  
  
# select and plot only the first two principal components  
plot(pca.out$x[,1:2], col=1:3, xlab="First principal component", ylab="Second principal compnent", pch=35)



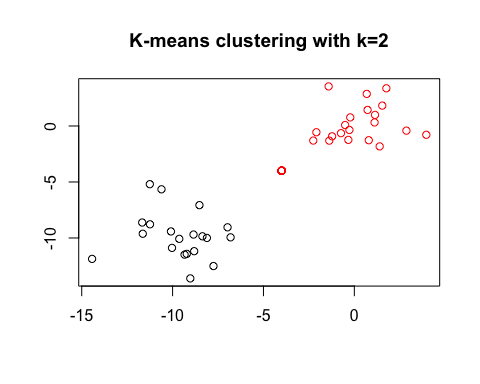
#### Perform K-means clustering of the observations with K = 3. How well do the clusters that you obtained in K-means clustering compare to the true class labels?

# Hint: You can use the table() function in R to compare the true class labels to the class labels obtained by clustering. Be careful how you interpret the results: K-means clustering will arbitrarily number the clusters, so you cannot simply check whether the true class labels and clustering labels are the same.  
  
km.out3 <- kmeans(df,3,nstart=20)  
plot(df,col=(km.out3$cluster),main="K-means clustering with k=3", xlab="",ylab="")

 The clusters match the class labels perfectly.

#### Perform K-means clustering with K = 2. Describe your results.

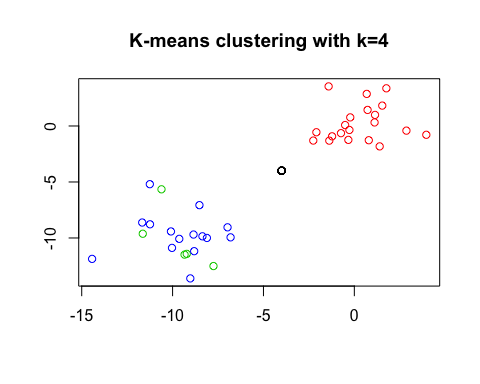
km.out2 = kmeans(df, 2, nstart=20)  
plot(df,col=(km.out2$cluster),main="K-means clustering with k=2", xlab="",ylab="")



One of the previous cluster groups has now merged with another cluster group.

#### Now perform K-means clustering with K = 4, and describe your results.

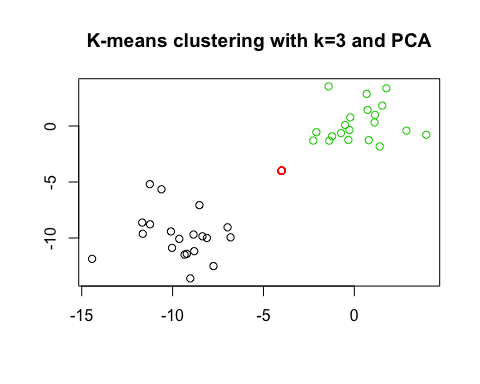
km.out4 <- kmeans(df,4,nstart=20)  
plot(df,col=(km.out4$cluster),main="K-means clustering with k=4", xlab="",ylab="")



One of the previous clusters has now split into two clusters(green and blue groups).

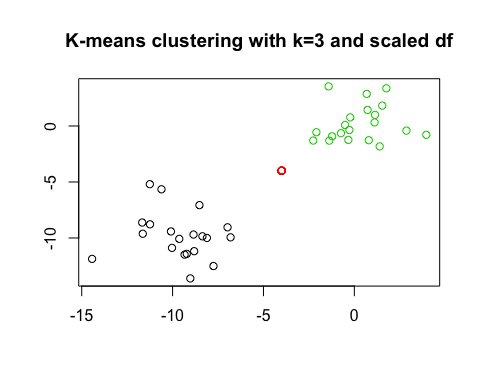
#### Now perform K-means clustering with K = 3 on the first two principal components, rather than on the raw data. That is, perform K-means clustering on the 60 × 2 matrix of which the first column is the first principal component’s corresponding eigenvector, and the second column is the second principal component’s corresponding eigenvector. Comment on the results.

km.out3.pc<-kmeans(pca.out$x[,1:2],3,nstart=20)  
plot(df,col=(km.out3.pc$cluster),main="K-means clustering with k=3 and PCA", xlab="",ylab="")

 The result is the same as (c), when k=3: the observations match well with class labels.

#### Using the scale() function, perform K-means clustering with K = 3 on the data after scaling each variable to have standard deviation one. How do these results compare to those obtained in (c)? Explain.

km.out3.pc.scale<-kmeans(scale(df),3,nstart=20)  
plot(df,col=(km.out3.pc.scale$cluster),main="K-means clustering with k=3 and scaled df", xlab="",ylab="")



The results look similar on the plot to (c), but I think scaling would have impacted the Eucliedean distance between the observations.