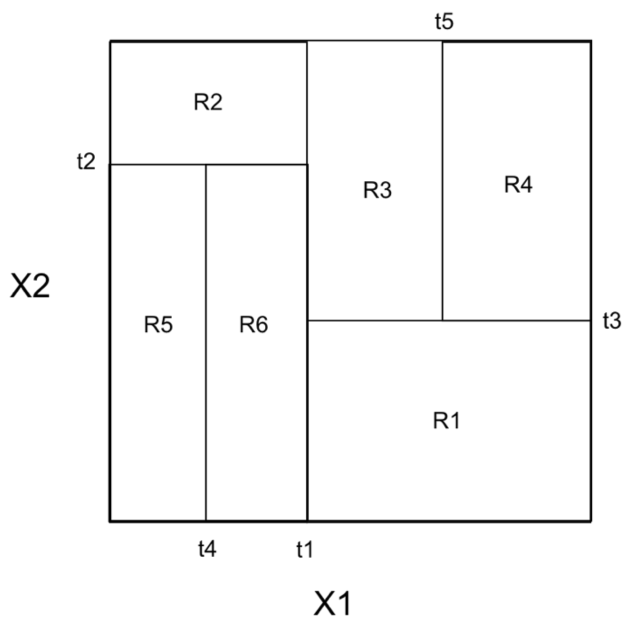
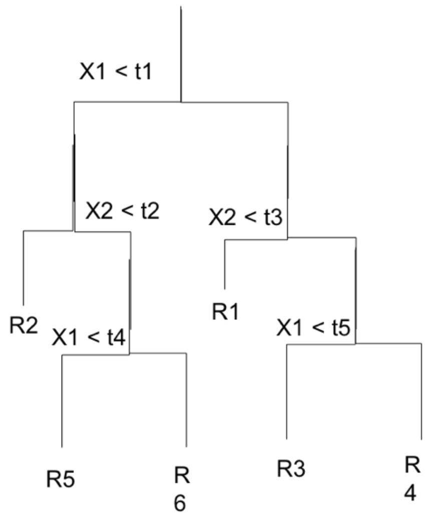
IDS 575 HW3

Scott Brewer

Ono Gantsog

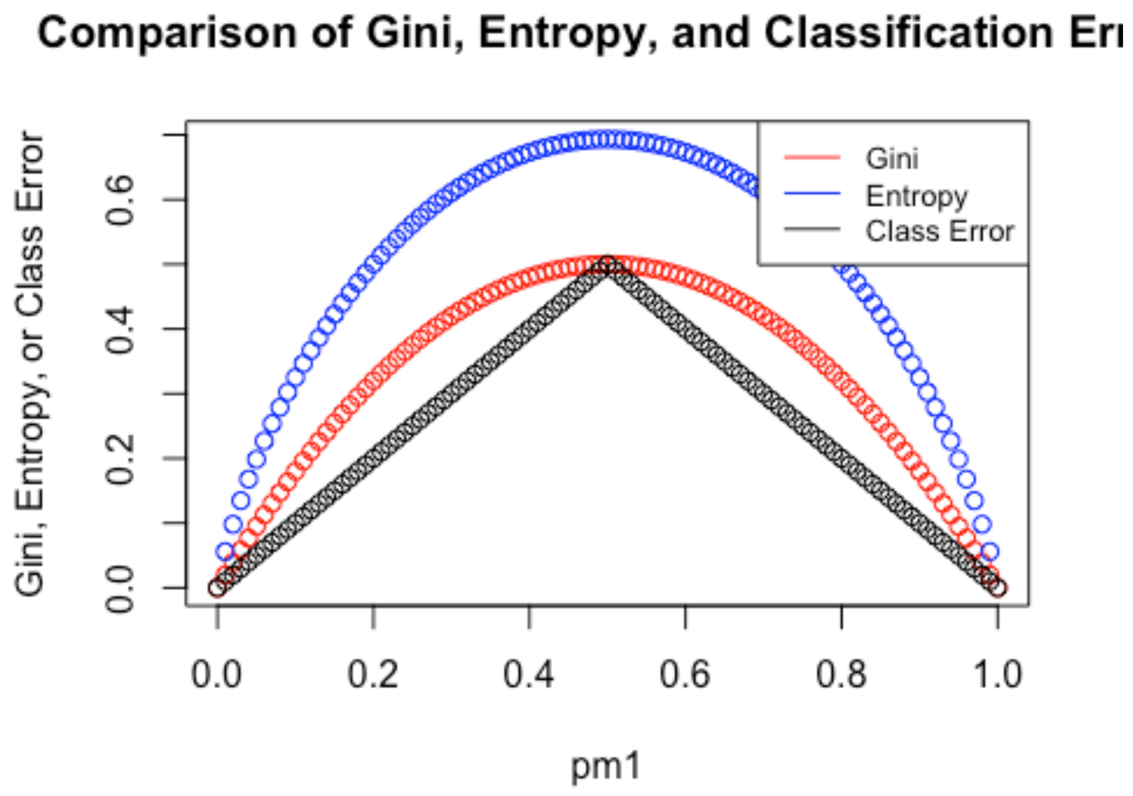
1: ISLR Ch8.1. Draw an example (of your own invention) of a partition of two-dimensional feature space that could result from recursive binary splitting. Your example should contain at least six regions. Draw a decision tree corresponding to this partition. Be sure to label all aspects of your figures, including the regions R1, R2, . . ., the cutpoints t1,t2,…, and so forth.





2: ISLR Ch8.3. Consider the Gini index, classification error, and entropy in a simple classification setting with two classes. Create a single plot that displays each of these quantities as a function of pˆm1. The x- axis should display pˆm1, ranging from 0 to 1, and the y-axis should display the value of the Gini index, classification error, and entropy.

```
pm1 = seq(0, 1, 0.01)
gini = pm1 * (1 - pm1) * 2   # Multiplied by 2 due to 2 classes
entropy = -(pm1 * log(pm1) + (1 - pm1) * log(1 - pm1))
error = 1 - pmax(pm1, 1 - pm1)
matplot(pm1, cbind(gini, entropy, error), col = c("red", "blue", "black"), pc
h = 1, ylab='Gini, Entropy, or Class Error')
title('Comparison of Gini, Entropy, and Classification Error')
legend("topright", lty=1, col=c("red", "blue", "black"),legend=c('Gini','Entr
opy','Class Error'), bty='y', cex=.8)
```



3: ISLR Ch8.8. 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.

```
library(MASS)
library(ISLR)
library(tree)

## Warning: package 'tree' was built under R version 3.4.4

attach(Carseats)
```

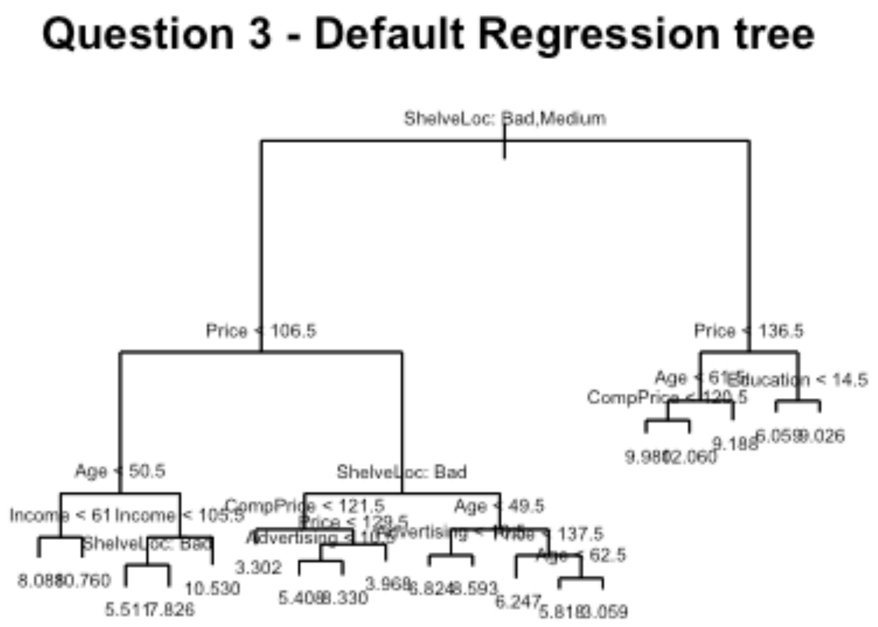
(a) Split the data set into a training set and a test set.

```
set.seed(123)
train <- sample(1:nrow(Carseats), nrow(Carseats)*7/10)
```

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

```
tree.carseats <- tree(Sales~., Carseats, subset=train)
#summary(tree.carseats)

plot(tree.carseats)
```

```r
text(tree.carseats,pretty=0, cex=.5)
title("Question 3 - Default Regression tree")
```

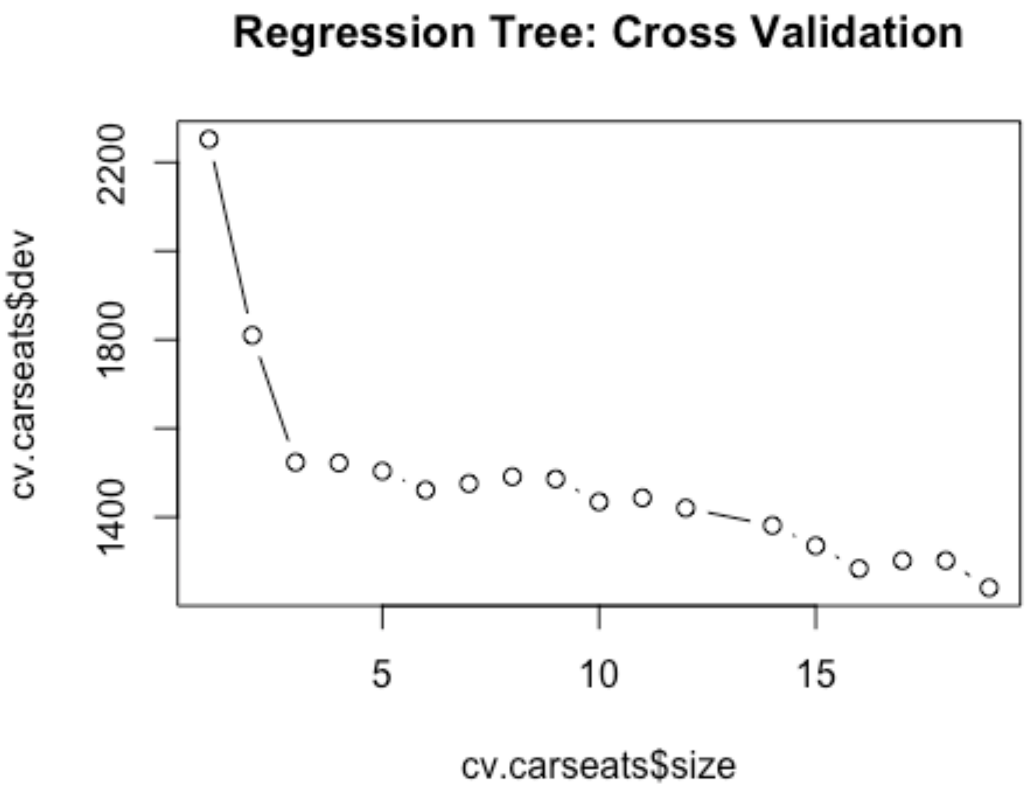


ShelveLoc and Price appear to be the highest importance variables followed by Age and Education.

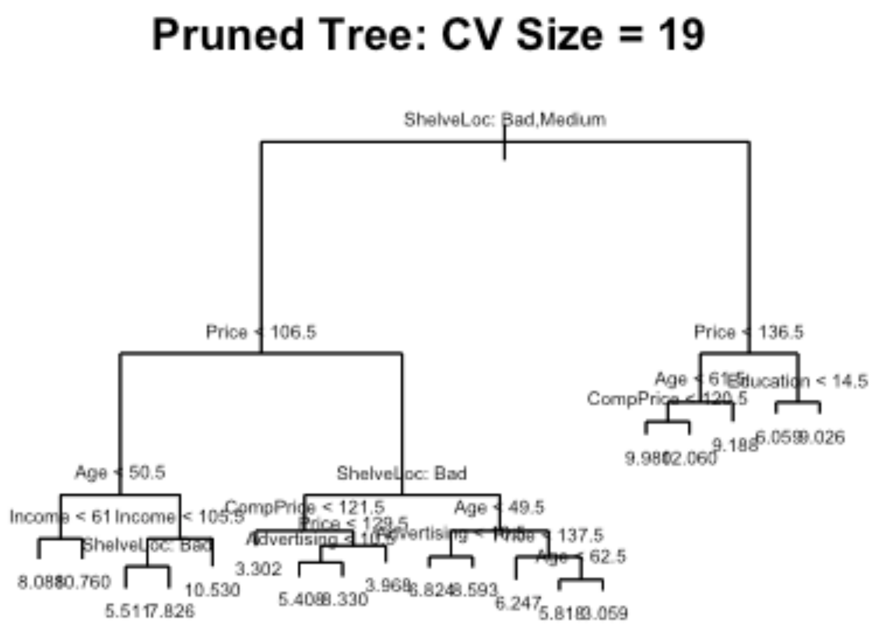
```r
Carseats.test <- Carseats[-train,]
Sales.test <- Sales[-train]

tree.pred <- predict(tree.carseats, Carseats.test, type = "vector")
error <- Sales.test - tree.pred
sumError <- sum(error)
rss <- (error)^2
mse <- mean(rss)
paste('Test MSE =',mse)

## [1] "Test MSE = 4.93842101526028"
```

(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

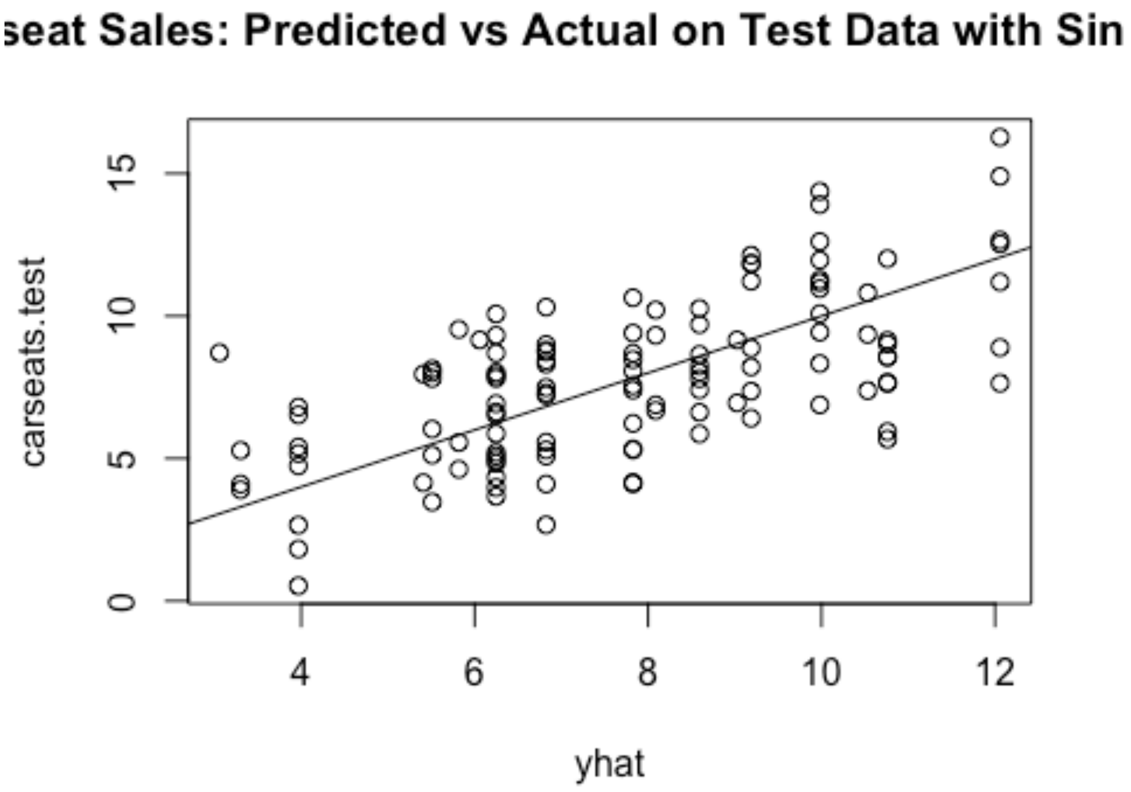
```r
set.seed(1)
cv.carseats<-cv.tree(tree.carseats)
plot(cv.carseats$size, cv.carseats$dev, type="b", main = "Regression Tree: Cr
oss Validation")
```



```
bestsize <- cv.carseats$size[which.min(cv.carseats$dev)]

prune.carseats <- prune.tree(tree.carseats, best=bestsize)
plot(prune.carseats)
title(paste("Pruned Tree: CV Size =",bestsize))
text(prune.carseats, pretty=0, cex=.5)
```



```
yhat <- predict(prune.carseats, newdata=Carseats[-train,])
carseats.test <- Carseats[-train ,"Sales"]
plot(yhat, carseats.test)
abline(0,1)
title('Carseat Sales: Predicted vs Actual on Test Data with Single Tree')
```



```
mse <- mean((yhat-carseats.test)^2) #[1] 4.710952 with best =8 #[1] 4.900733
with bes=3
paste('Test MSE =',mse,'with Best Size =',bestsize)

## [1] "Test MSE = 4.93842101526028 with Best Size = 19"
```

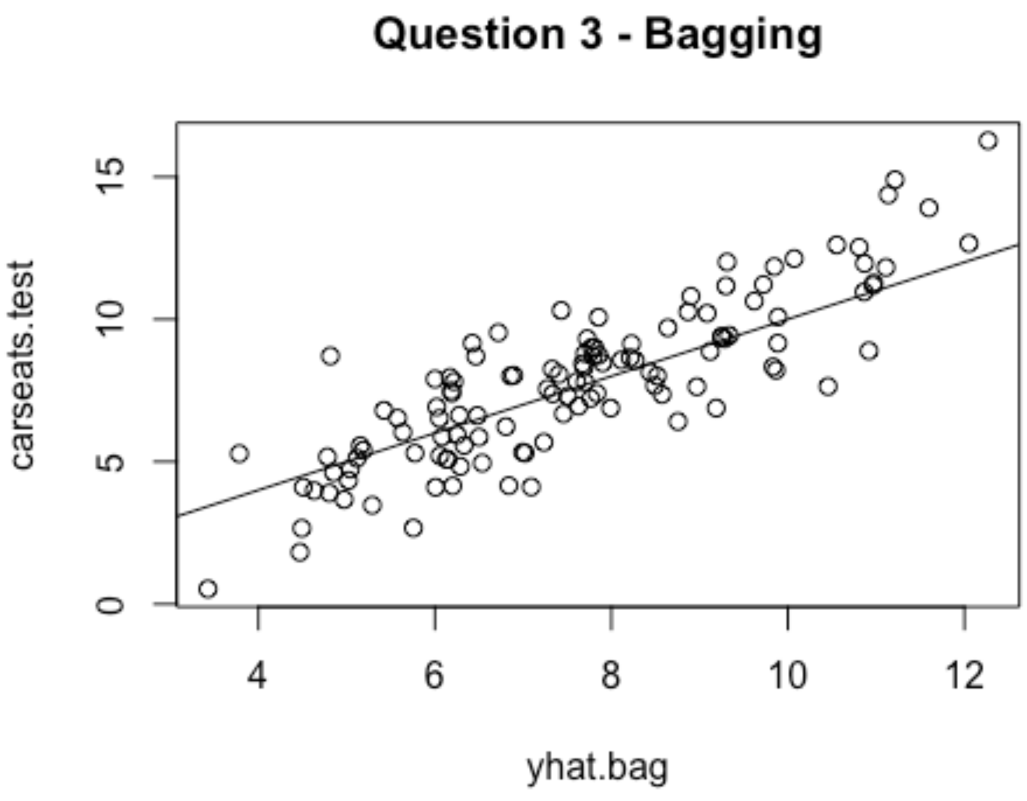
It varies by the seed, but here we see that the full tree with number of leaves = 19 is optimal via cross validation. That said, we can try pruning anyway to see how the resulting tree performs with MSE:

```
sizes <- seq(2,19,1)
mse <- vector(mode='numeric', length = length(sizes))
for(i in seq_along(sizes)){
  prune.carseats <- prune.tree(tree.carseats, best=sizes[i])
  yhat <- predict(prune.carseats, newdata=Carseats[-train,])
  carseats.test <- Carseats[-train ,"Sales"]
  mse[i] <- mean((yhat - carseats.test)^2)
}
besttest <- sizes[which.min(mse)]
print(paste('Best Size from Test Data =',besttest,', MSE =',mse[which.min(mse
)]))

## [1] "Best Size from Test Data = 8 , MSE = 4.56633165543277"
```

Here we see that pruning to 8 leaves improved MSE (4.94->4.57), but in practice the test data shouldn't be used in model development, so this process wouldn't be feasible in practice.

(d) 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 4.6-12

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

set.seed(123)
train <- sample(1:nrow(Carseats), nrow(Carseats)*7/10)
mtry <- ncol(Carseats) - 1
bag.carseats<-randomForest(Sales~., data=Carseats, subset=train, mtry=mtry, i
mportance=TRUE)

yhat.bag <- predict(bag.carseats, newdata=Carseats[-train,])
plot(yhat.bag, carseats.test, main = "Question 3 - Bagging")
abline(0,1)
```

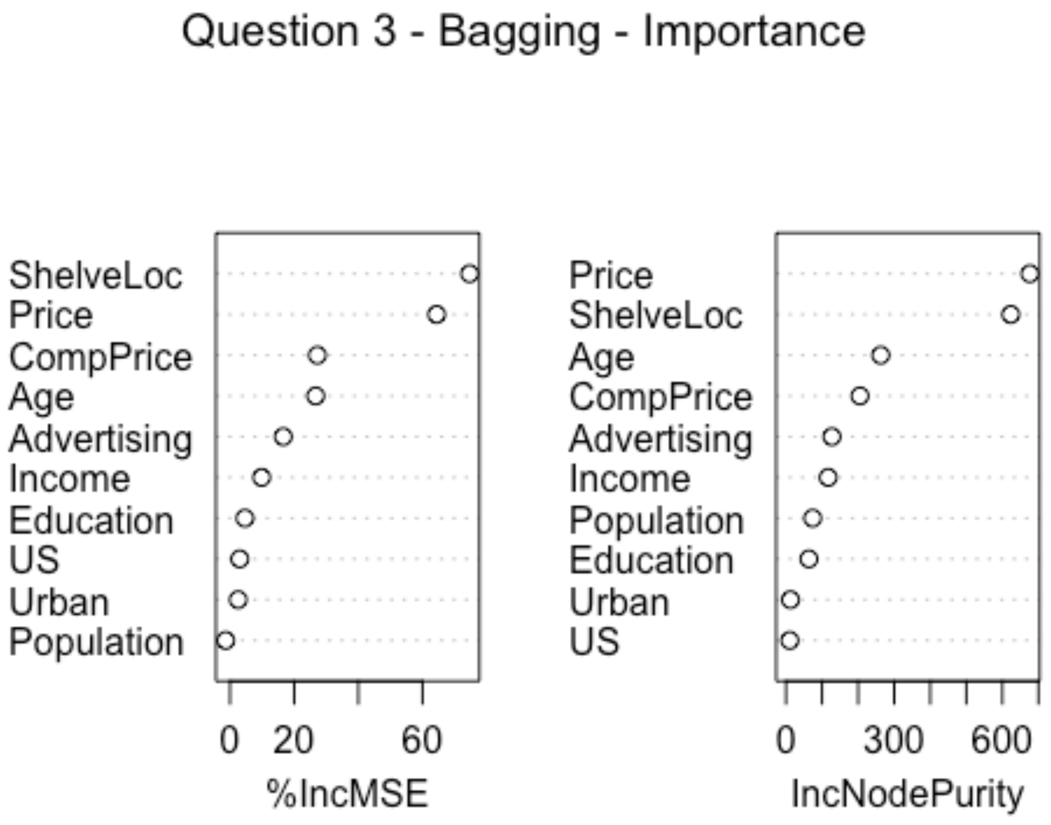


```
mse <- mean((yhat.bag-carseats.test)^2) #[1] 11.8471
paste('Test MSE = ',mse)
```

```
## [1] "Test MSE =  2.33648742577852"
```

Here we see that bagging significantly improves MSE vs a single tree.

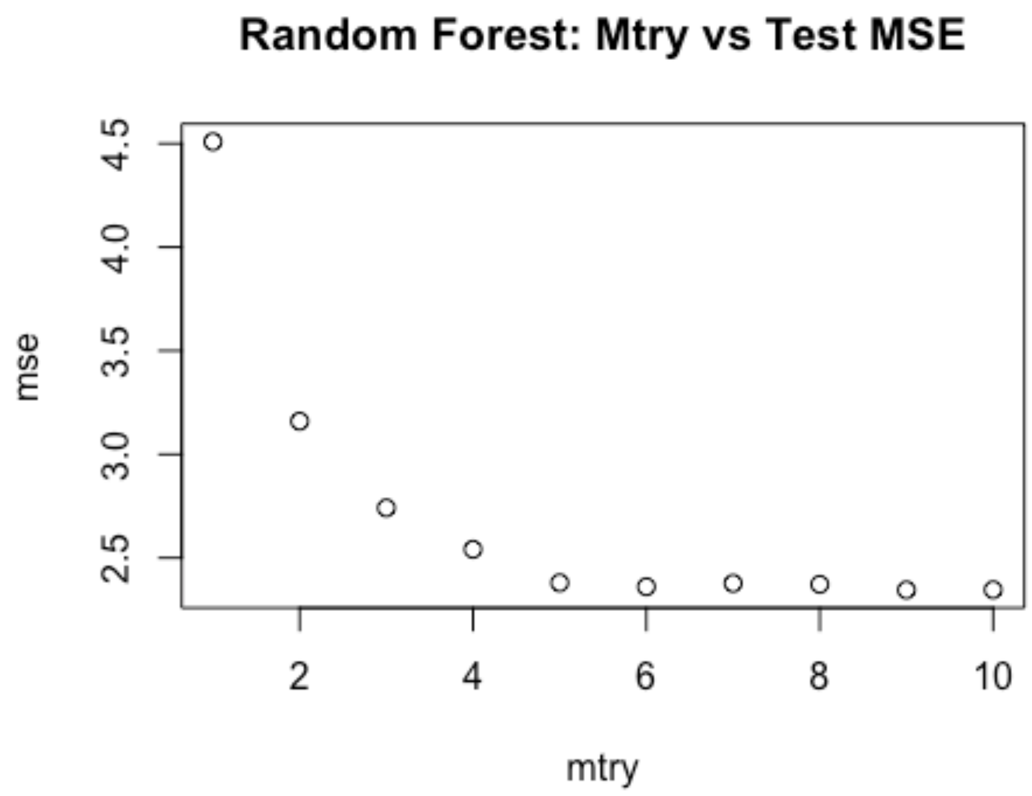
```
varImpPlot(bag.carseats, main = "Question 3 - Bagging - Importance")
```



With bagging, important variables are ShelveLoc, Price, CompPrice, and Age.

(e) Use random forests to analyze this data. What test MSE 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 error rate obtained.

```
mtry <- seq(1,ncol(Carseats)-1,1)
mse <- vector(mode='numeric', length = length(mtry))
for(i in seq_along(mtry)){
  rf.carseats <- randomForest(Sales~., data=Carseats, subset=train, mtry=mtry
[i], ntree=500)
  rf.yhat <- predict(rf.carseats, newdata=Carseats[-train,])
  mse[i] <- mean((rf.yhat - carseats.test)^2)
}
plot(mtry,mse, main = 'Random Forest: Mtry vs Test MSE')
```

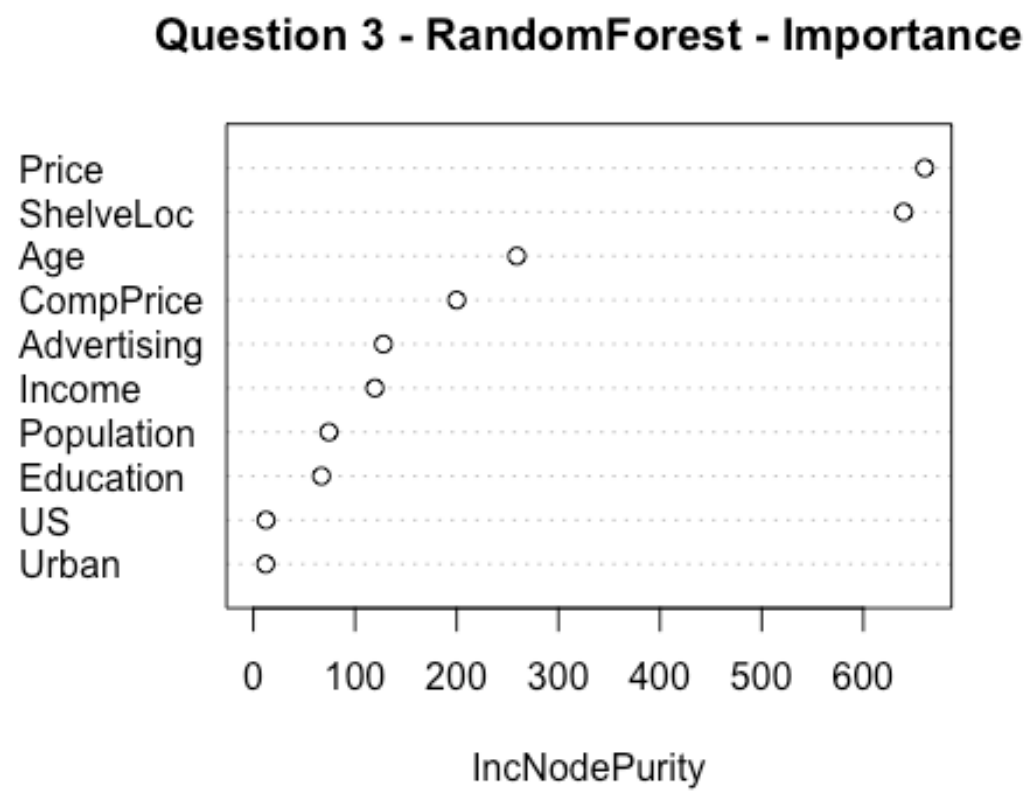


Increasing mtry generally reduces Test MSE.

```
besttest <- mtry[which.min(mse)]
print(paste('Best mtry from Test Data =',besttest,', MSE =',mse[which.min(mse
)]))

## [1] "Best mtry from Test Data = 9 , MSE = 2.34533541840536"
```

Note that test MSE for RF is slightly improved over bagging.

```
importance(rf.carseats)

##             IncNodePurity
## CompPrice       199.99601
## Income          119.53916
## Advertising     127.87404
## Population       74.29034
## Price           660.38303
## ShelveLoc       639.63307
## Age             259.13423
## Education        66.98915
## Urban            12.16129
## US               12.48686

varImpPlot(rf.carseats, main = "Question 3 - RandomForest - Importance")
```



Important variables are ShelveLoc, Price, CompPrice, and Age, which matches the bagging results.

4: ISLR Ch8.10. We now use boosting to predict Salary in the Hitters data set.

(a) Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

```
data <- Hitters
data <- data[-which(is.na(data$Salary)),]
data$Salary <- log(data$Salary)
```

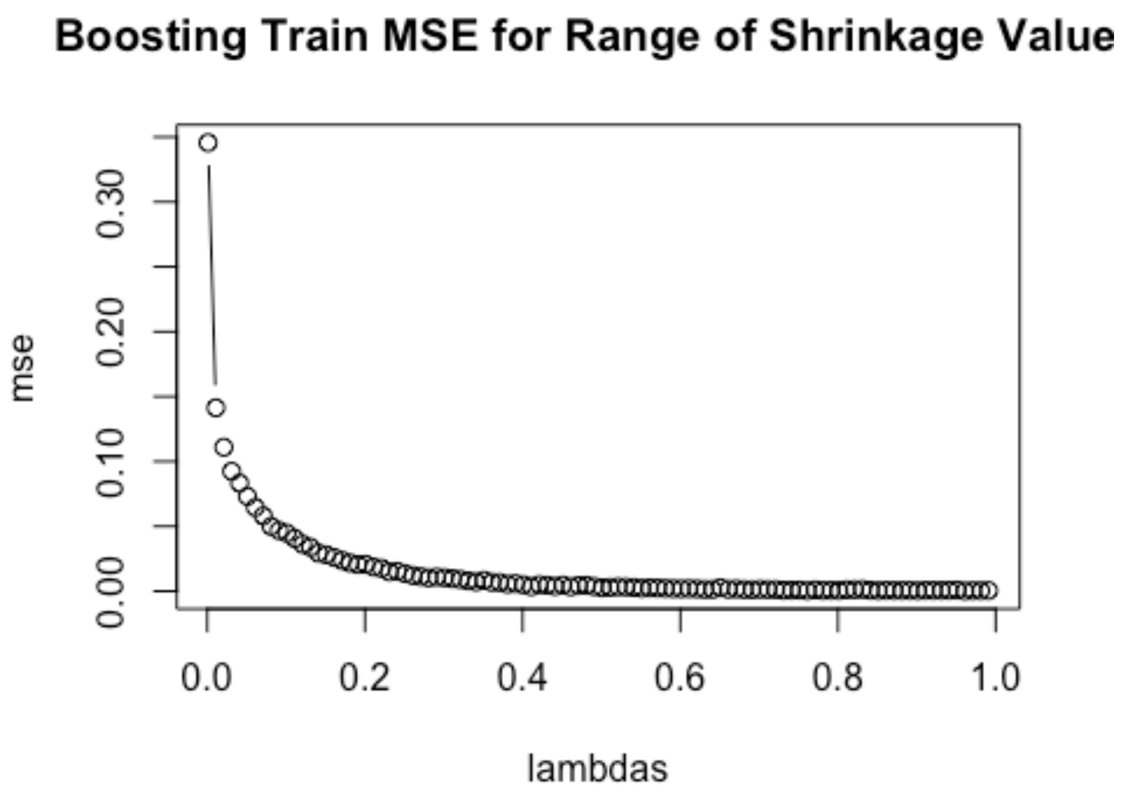
(b) Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

```
set.seed(575)
train <- sample(1:nrow(data), 200)
test <- -train
```

(c) Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter λ. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

```
library(gbm)

## Loading required package: survival
## Loading required package: lattice

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3
```

```
lambdas <- seq(0.001,1,0.01)
mse <- vector(mode='numeric',length=length(lambdas))
for (i in seq_along(lambdas)){
  boost <- gbm(Salary~., data=data[train,],
               distribution='gaussian', n.trees=1000, interaction.depth=1, sh
rinkage=lambdas[i])
  boost.pred <- predict(boost,newdata=data[train,], n.trees=1000)
  mse[i] <- mean((boost.pred - data$Salary[train])^2)
}
plot(lambdas, mse, type='b', main='Boosting Train MSE for Range of Shrinkage
Values')
```

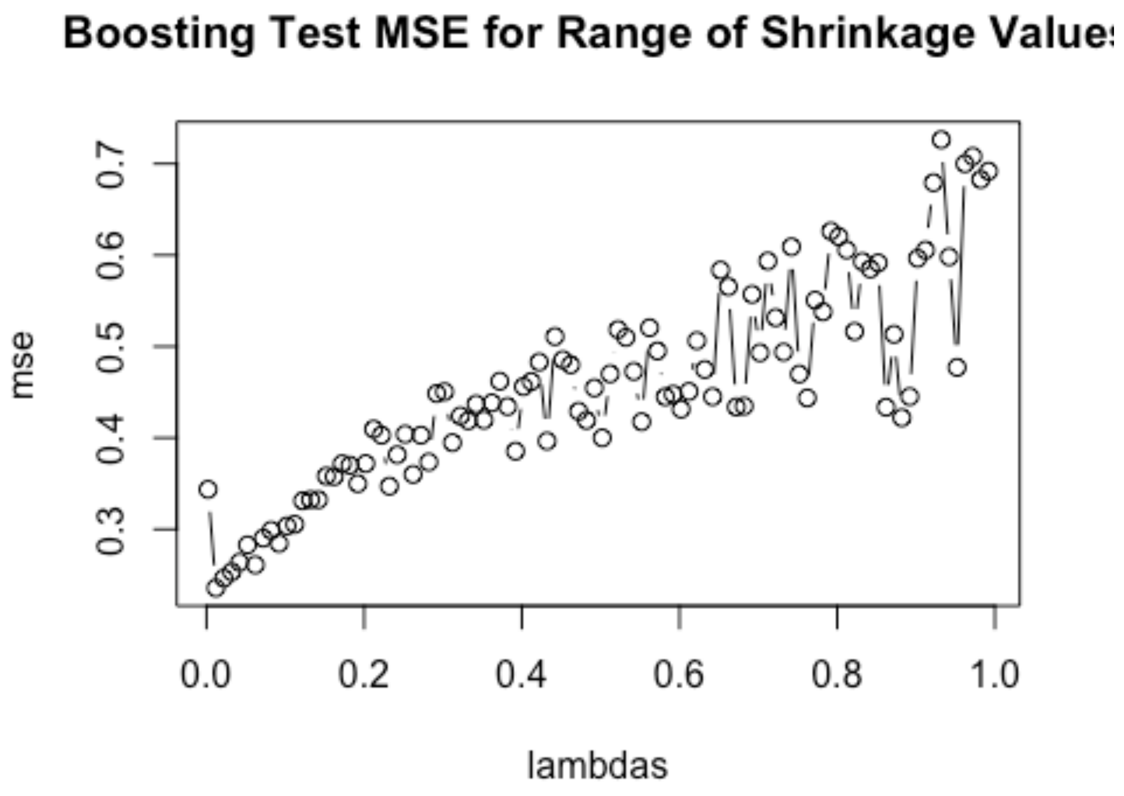


(d)

Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

```
library(gbm)
lambdas <- seq(0.002,1,0.01)
```

```
mse <- vector(mode='numeric',length=length(lambdas))
for (i in seq_along(lambdas)){
  boost <- gbm(Salary~., data=data[train,],
               distribution='gaussian', n.trees=1000, interaction.depth=1, sh
rinkage=lambdas[i])
  boost.pred <- predict(boost,newdata=data[test,], n.trees=1000)
  mse[i] <- mean((boost.pred - data$Salary[test])^2)
}
plot(lambdas, mse, type='b', main='Boosting Test MSE for Range of Shrinkage V
alues')
```



```
boost.mse <- min(mse)
boost.lambda <- lambdas[which(boost.mse==mse)]
```

(e) Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

```
glm.fit <- glm(Salary~.,data=data[train,])
glm.pred <- predict(glm.fit,newdata=data[-train,])
mse <- mean((glm.pred - data$Salary[-train])^2)
paste('Regression Test MSE:',mse)

## [1] "Regression Test MSE: 0.449714269372374"

library(glmnet)
```

```
x <- model.matrix(Salary~., data)[,-1]
y <- data$Salary
set.seed(575)
cv.out=cv.glmnet(x[train,],y[train],alpha=0)
bestlam=cv.out$lambda.min
ridge.fit <- glmnet(x[train,], y[train], alpha=0)
ridge.pred <- predict(ridge.fit, s=bestlam, newx=x[-train,])
mse <- mean((ridge.pred - data$Salary[-train])^2)
paste('Ridge Test Regression MSE:',mse)

## [1] "Ridge Test Regression MSE: 0.402422927327405"

set.seed(575)
cv.out <- cv.glmnet(x[train,],y[train],alpha=1)
bestlam <- cv.out$lambda.min
lasso.fit <- glmnet(x[train,], y[train], alpha=1)
lasso.pred <- predict(lasso.fit, s=bestlam, newx=x[-train,])
mse <- mean((lasso.pred - data$Salary[-train])^2)
paste('Lasso Test Regression MSE:',mse)

## [1] "Lasso Test Regression MSE: 0.415610482020619"

paste('Boosting Test MSE:',boost.mse)

## [1] "Boosting Test MSE: 0.235763998015269"
```

Boosting appears to have the best MSE compared to the regression methods above, which would indicate that realtionships in the data may not be linear.

(f) Which variables appear to be the most important predictors in the boosted model?

```
boost <- gbm(Salary~., data=data[train,],
               distribution='gaussian', n.trees=1000, interaction.depth=1, sh
rinkage=boost.lambda)
boost.pred <- predict(boost,newdata=data[-train,], n.trees=1000)
mse <- mean((boost.pred - data$Salary[-train])^2)
print("Top 5 relevant predictors:")

## [1] "Top 5 relevant predictors:"

print(summary(boost,plotit=F)$var[1:5])

## [1] CRuns  CRBI   CHits  CAtBat Years
## 19 Levels: Assists AtBat CAtBat CHits CHmRun CRBI CRuns ... Years
```

(g) Now apply bagging to the training set. What is the test set MSE for this approach?

```
library(randomForest)
mtry <- ncol(data) - 1
bag.fit <- randomForest(Salary~., data=data[train,], mtry=mtry, ntree=500)
bag.pred <- predict(bag.fit, newdata=data[-train,])
mse <- mean((bag.pred - data$Salary[-train])^2)
paste('Bagging Test MSE:', mse)
```

```
## [1] "Bagging Test MSE: 0.2278283172653"
```

```r
mtry <- sqrt(ncol(data) - 1)
rf.fit <- randomForest(Salary~., data=data[train,], mtry=mtry, ntree=500)
rf.pred <- predict(rf.fit, newdata=data[-train,])
mse <- mean((rf.pred - data$Salary[-train])^2)
paste('Random Forest Test MSE (sqrt(p)):', mse)
```

```
## [1] "Random Forest Test MSE (sqrt(p)): 0.21274848075952"
```

```r
mtry <- (ncol(data) - 1)/3
rf.fit <- randomForest(Salary~., data=data[train,], mtry=mtry, ntree=500)
rf.pred <- predict(rf.fit, newdata=data[-train,])
mse <- mean((rf.pred - data$Salary[-train])^2)
paste('Random Forest Test MSE (p/3):', mse)
```

```
## [1] "Random Forest Test MSE (p/3): 0.215791223103635"
```

Since we were already comparing various models, I also included MSE for random forest (with the two common mtry calculations) for my own edification. Bagging MSE is better than Boosting MSE, but RF beats both.

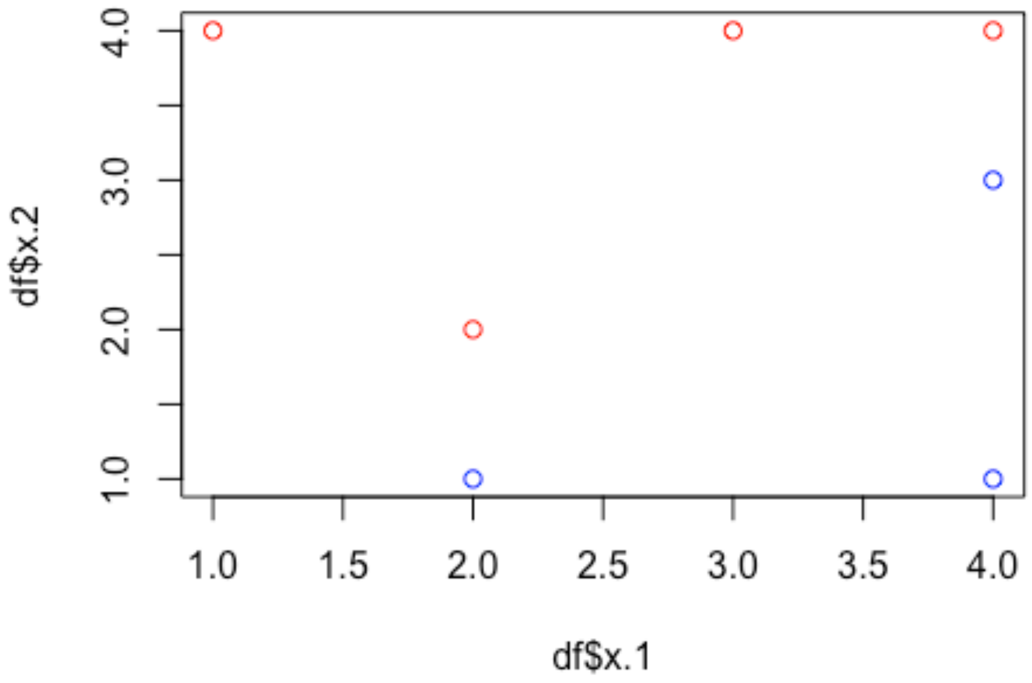
5: ISLR Ch9.3. Here we explore the maximal margin classifier on a toy data set.

(a) We are given n = 7 observations in p = 2 dimensions. For each observation, there is an associated class label.

Obs. X1 X2 Y
1 3 4 Red
2 2 2 Red
3 4 4 Red
4 1 4 Red
5 2 1 Blue
6 4 3 Blue
7 4 1 Blue

Sketch the observations.

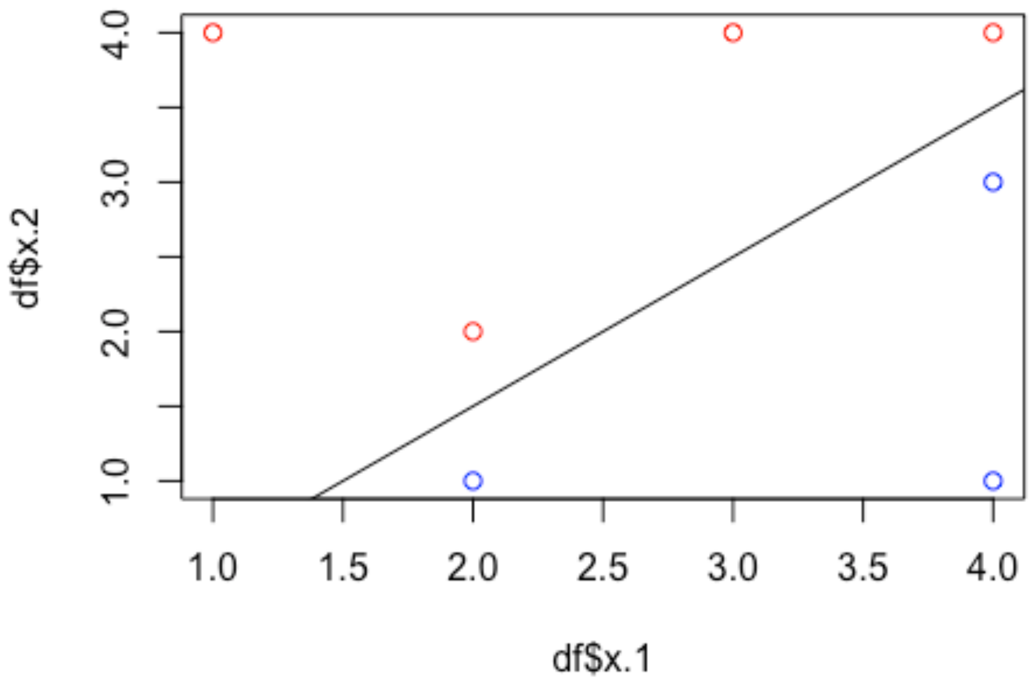
```r
x <- matrix(c(3,2,4,1,2,4,4,4,2,4,4,1,3,1),ncol=2)
y <- c('red','red','red','red','blue','blue','blue')
df <- data.frame(x=x, y=as.factor(y))
plot(df$x.1, df$x.2, type = 'p', col=y)
```



(b)

Sketch the optimal separating hyperplane, and provide the equation for this hyperplane (of the form (9.1)).

```r
# Since the classes are lineraly separable and we can visually determine the points at the boundary, we just have to average the distance between boundary points to find points on the separation line: red(2,2) with blue(2,1) & red(4,4) with blue(4,3)
a <- c((2+2)/2,(2+1)/2)
b <- c((4+4)/2,(4+3)/2)
# Now we determine the line between points a and b:
slope <- (b[2]-a[2])/(b[1]-a[1])
intercept <- a[2]-slope*a[1]
plot(df$x.1, df$x.2, type = 'p', col=y)
abline(coef = c(intercept, slope))
```



(c)

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.

```
slope

## [1] 1

intercept

## [1] -0.5
```

Classify to Red if β0 + β1X1 + β2X2 > 0, and classify to Blue otherwise, where

β0 = .05

β1 = -1

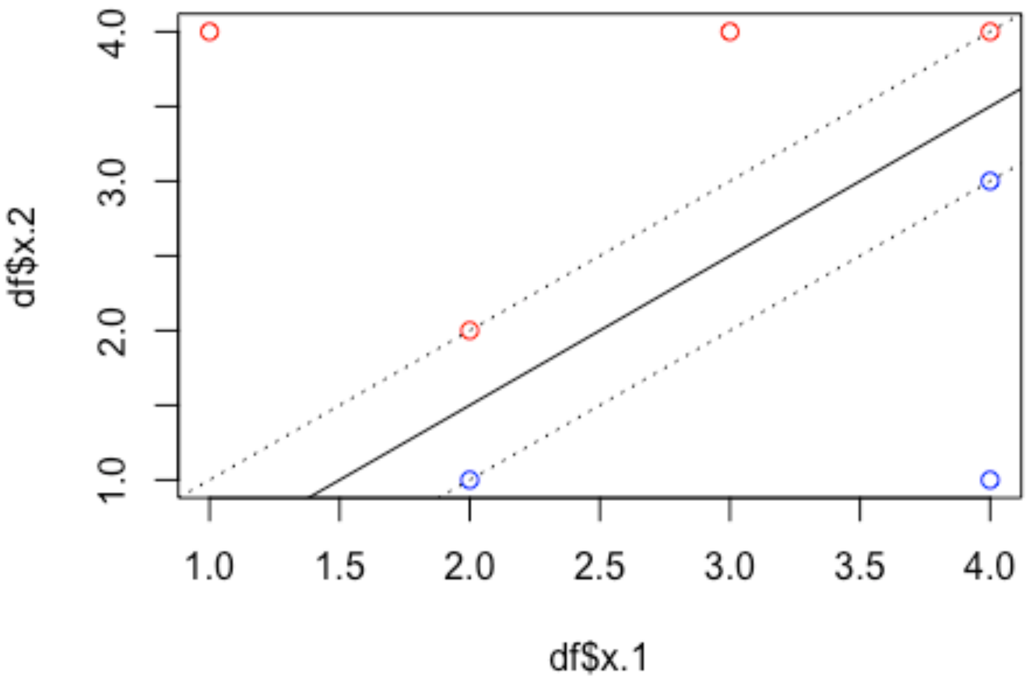
β2 = 1

ie: .05 - X1 + X2 > 0

(d) On your sketch, indicate the margin for the maximal margin hyperplane.

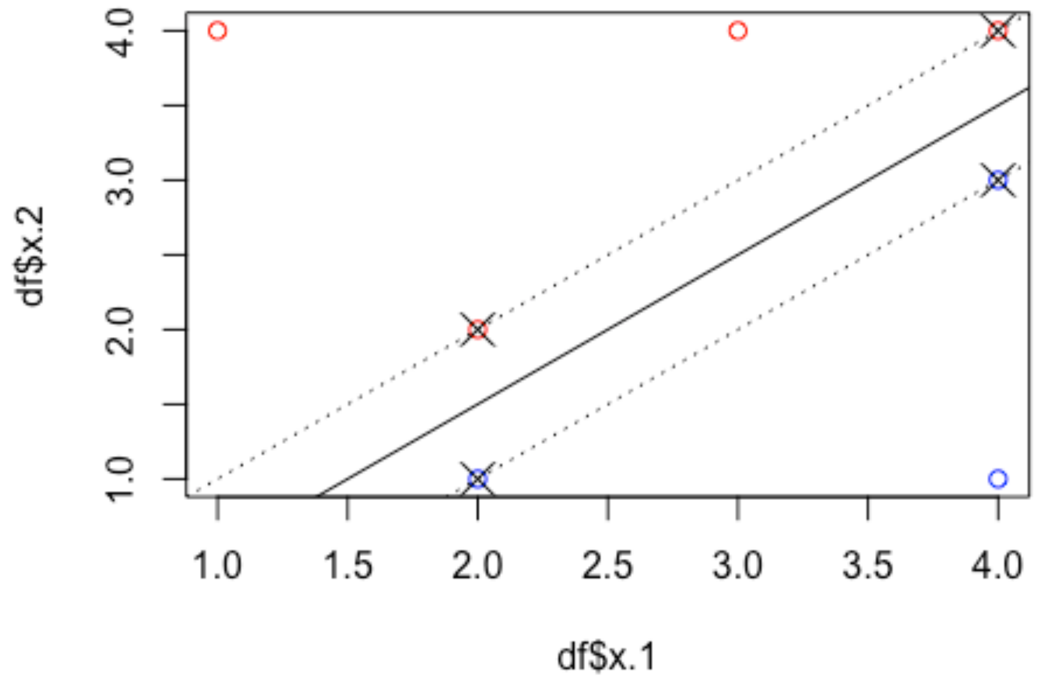
```
plot(df$x.1, df$x.2, type = 'p', col=y)
abline(coef = c(intercept, slope))
```

```r
abline(coef = c(intercept+.5, slope), lty=3)
abline(coef = c(intercept-.5, slope), lty=3)
```



(e) Indicate the support vectors for the maximal margin classifier.

```r
plot(df$x.1, df$x.2, type='p', col=y)
abline(coef = c(intercept, slope))
abline(coef = c(intercept+.5, slope), lty=3)
abline(coef = c(intercept-.5, slope), lty=3)
sv <- c(2,3,5,6)
points(df$x.1[sv], df$x.2[sv], type='p', pch=4, cex=2, col=1)
```



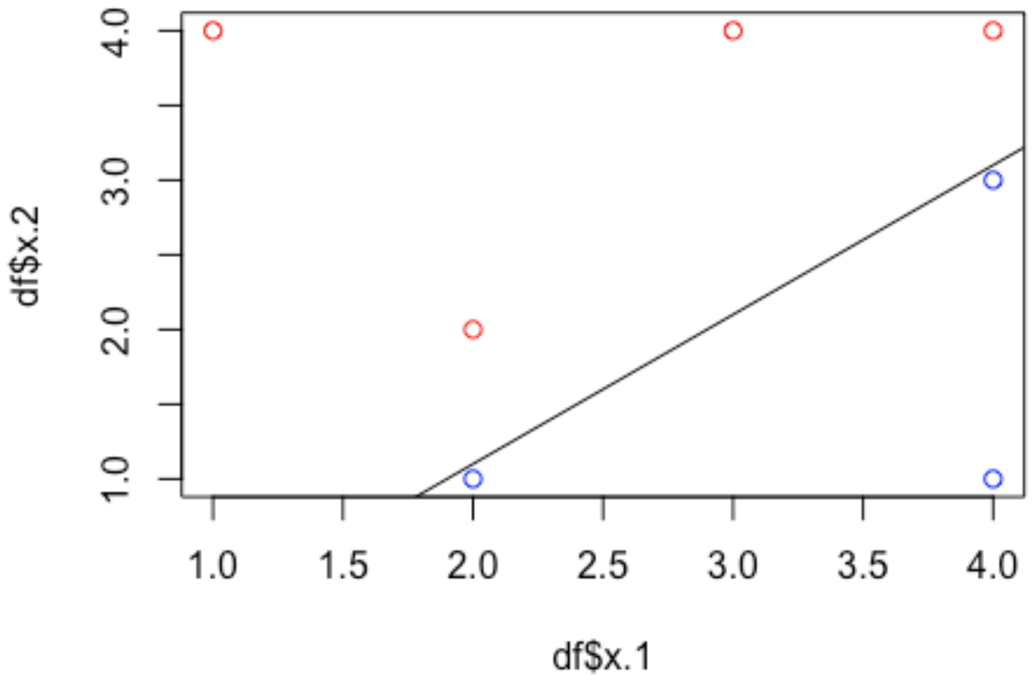
The four support vectors marked with large black 'X's.

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

Slight movement of the seventh observation blue (4,1) would not effect the hyperplane as it is not one of the support vectors which define the hyperplane. To effect the hyperplane, it would have to move significantly closer to the hyperplane, closer than the current margin.

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

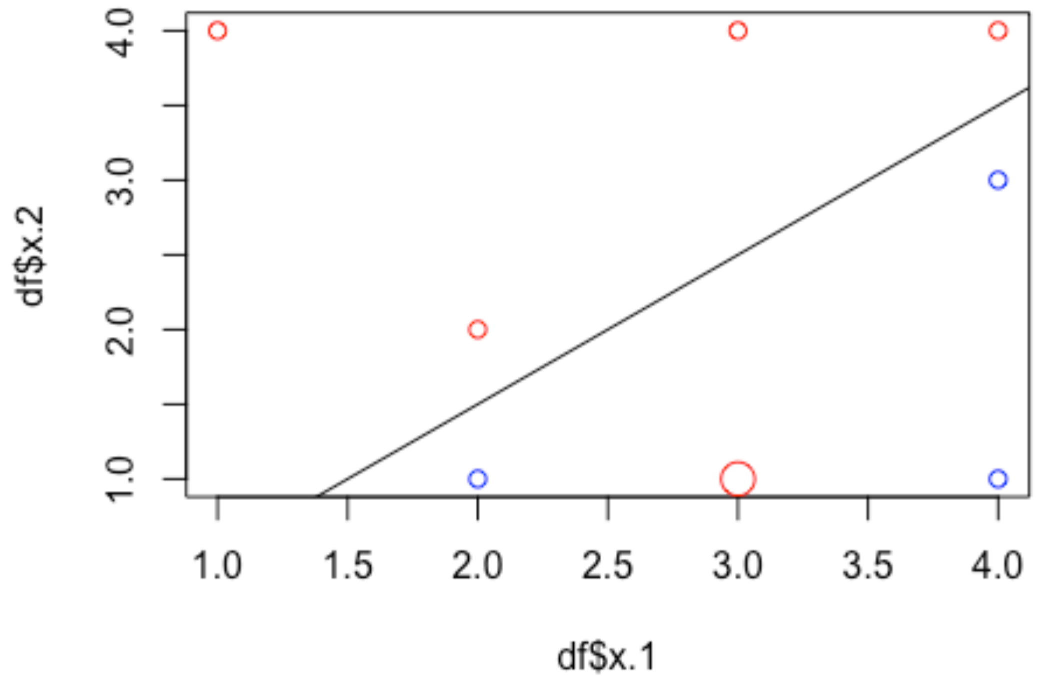
```
plot(df$x.1, df$x.2, type='p', col=y)
abline(coef = c(intercept-.4, slope))
```



By shifting the hyperplane down, we make the margin on either side not equal and thus not optimal for classifying unknown test data. The equation of this non-optimal hyperplane is: .01 - X1 + X2 > 0

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

```
plot(df$x.1, df$x.2, type='p', col=y)
abline(coef = c(intercept, slope))
points(3, 1, type='p', cex=2, col='red')
```



The new observation (large red circle) prevent separation by hyperplane.

6: ISLR Ch9.7. 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.

```
library(ISLR)
library(e1071)
```

(a) 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.

```
AutoNew<- Auto[, 1:8]
AutoNew$highmpg <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
AutoNew$highmpg <- factor(AutoNew$highmpg)
train <- sample(nrow(AutoNew), .7*nrow(AutoNew))
```

(b) 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.

```
svmfit10 <- svm(highmpg~., data=AutoNew[train,], kernel ="linear", cost=10, s
cale=FALSE)
print(paste('Number of Support Vectors = ', nrow(svmfit10$SV), ' with cost =
10'))

## [1] "Number of Support Vectors =  7  with cost = 10"
```

```r
svmfit01 <- svm(highmpg~., data=AutoNew[train,], kernel="linear", cost=0.1, s
cale=FALSE)
print(paste('Number of Support Vectors = ', nrow(svmfit01$SV), ' with cost =
0.1'))
```

```
## [1] "Number of Support Vectors =  19  with cost = 0.1"
```

```r
set.seed(1)
tune.out <- tune(svm, highmpg~., data=AutoNew[train,], kernel="linear", range
s=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))
summary(tune.out)
```

```
## 
## Parameter tuning of 'svm':
## 
## - sampling method: 10-fold cross validation 
## 
## - best parameters:
##  cost
##   100
## 
## - best performance: 0.007142857 
## 
## - Detailed performance results:
##    cost       error dispersion
## 1 1e-03 0.101587302 0.05771738
## 2 1e-02 0.072619048 0.04730139
## 3 1e-01 0.061507937 0.05316389
## 4 1e+00 0.028968254 0.03324439
## 5 5e+00 0.014417989 0.01861746
## 6 1e+01 0.010714286 0.01725164
## 7 1e+02 0.007142857 0.01505847
```

```r
bestmod.linear <- tune.out$best.model
summary(bestmod.linear)
```

```
## 
## Call:
## best.tune(method = svm, train.x = highmpg ~ ., data = AutoNew[train, 
##     ], ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), 
##     kernel = "linear")
## 
## 
## Parameters:
##    SVM-Type:  C-classification 
##  SVM-Kernel:  linear 
##        cost:  100 
##       gamma:  0.125 
## 
## Number of Support Vectors:  7
## 
```

```
##  ( 4 3 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

The best parameter for linear kernel is cost = 10.

(c) 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.

```
set.seed(1)
svm.radial.1.1 <- svm(highmpg~., data=AutoNew[train,], kernel="radial", gamma
=1, cost=10)
print(paste('Number of Support Vectors = ', nrow(svm.radial.1.1$SV), ' with c
ost = 10'))

## [1] "Number of Support Vectors =  139  with cost = 10"

svm.radial.1.1e5 <- svm(highmpg~., data=AutoNew[train,], kernel="radial", gam
ma=1, cost=.1)
print(paste('Number of Support Vectors = ', nrow(svm.radial.1.1e5$SV), ' with
cost = .1'))

## [1] "Number of Support Vectors =  251  with cost = .1"

set.seed(1)
tune.outR <- tune(svm, highmpg~., data=AutoNew[train,], kernel="radial", rang
es=list(cost=c(0.01,0.1,1,10), gamma=c(0.01,0.1,1,5,10)))
summary(tune.outR)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.1
##
## - best performance: 0.02539683
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1   0.01  0.01 0.46441799 0.14824319
## 2   0.10  0.01 0.09431217 0.04761925
## 3   1.00  0.01 0.07261905 0.04730139
## 4  10.00  0.01 0.03624339 0.04784794
## 5   0.01  0.10 0.28902116 0.15903514
## 6   0.10  0.10 0.07261905 0.04730139
## 7   1.00  0.10 0.05423280 0.05386051
## 8  10.00  0.10 0.02539683 0.02992915
## 9   0.01  1.00 0.46798942 0.13982677
## 10  0.10  1.00 0.10582011 0.08540827
## 11  1.00  1.00 0.03968254 0.05439924
## 12 10.00  1.00 0.03611111 0.04784733
## 13  0.01  5.00 0.47513228 0.12436958
## 14  0.10  5.00 0.47513228 0.12436958
## 15  1.00  5.00 0.11335979 0.07275012
## 16 10.00  5.00 0.10608466 0.06599279
## 17  0.01 10.00 0.48227513 0.11137535
## 18  0.10 10.00 0.48227513 0.11137535
## 19  1.00 10.00 0.20462963 0.12138769
## 20 10.00 10.00 0.18664021 0.11301154
```

```
bestmod.radial <- tune.outR$best.model
summary(bestmod.radial)
```

```
##
## Call:
## best.tune(method = svm, train.x = highmpg ~ ., data = AutoNew[train,
##     ], ranges = list(cost = c(0.01, 0.1, 1, 10), gamma = c(0.01,
##     0.1, 1, 5, 10)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.1
##
## Number of Support Vectors:  38
##
##  ( 17 21 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

The best parameters for radial kernel are cost = 10 and gamma = 0.1.

```
train <- sample(nrow(Auto), 300)
svm.pol.4.1 <- svm(highmpg~., data=AutoNew[train,], kernel="polynomial", degr
ee=4, cost=1)
print(paste('Number of Support Vectors = ', nrow(svm.pol.4.1$SV), ' with cost
= 1 and degree = 4'))
```

```
## [1] "Number of Support Vectors =  185  with cost = 1 and degree = 4"
```

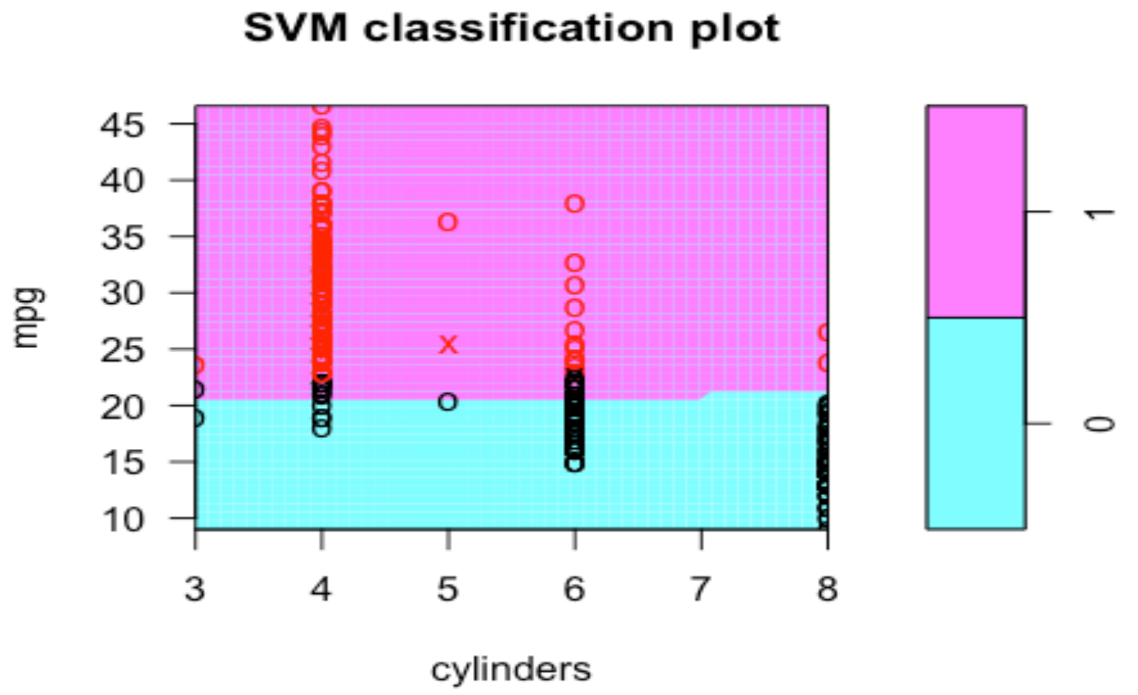
```
svm.pol.4.1e5 <- svm(highmpg~., data=AutoNew[train,], kernel="polynomial", de
gree=4, cost=1e5)
print(paste('Number of Support Vectors = ', nrow(svm.pol.4.1e5$SV), ' with co
st = 1e5 and degree = 4'))

## [1] "Number of Support Vectors =  67  with cost = 1e5 and degree = 4"

set.seed(1)
tune.outP <- tune(svm, highmpg~., data=AutoNew[train,], kernel="polynomial",
ranges=list(cost=c(0.1,1,10,100,1000), degree=c(1,2,3,4)))

summary(tune.outP)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##  1000      1
##
## - best performance: 0
##
## - Detailed performance results:
##     cost degree       error dispersion
## 1  1e-01      1 0.066666667 0.04157397
## 2  1e+00      1 0.050000000 0.03928371
## 3  1e+01      1 0.010000000 0.01610153
## 4  1e+02      1 0.003333333 0.01054093
## 5  1e+03      1 0.000000000 0.00000000
## 6  1e-01      2 0.283333333 0.05719795
## 7  1e+00      2 0.270000000 0.10237911
## 8  1e+01      2 0.193333333 0.07503086
## 9  1e+02      2 0.170000000 0.05762801
## 10 1e+03      2 0.186666667 0.07403703
## 11 1e-01      3 0.213333333 0.08344437
## 12 1e+00      3 0.070000000 0.03668350
## 13 1e+01      3 0.046666667 0.04216370
## 14 1e+02      3 0.040000000 0.03442652
## 15 1e+03      3 0.040000000 0.04388537
## 16 1e-01      4 0.286666667 0.05488484
## 17 1e+00      4 0.256666667 0.08020037
## 18 1e+01      4 0.176666667 0.05454639
## 19 1e+02      4 0.133333333 0.08012336
## 20 1e+03      4 0.123333333 0.05889937

bestmod.poly <- tune.outP$best.model
summary(bestmod.poly)
```

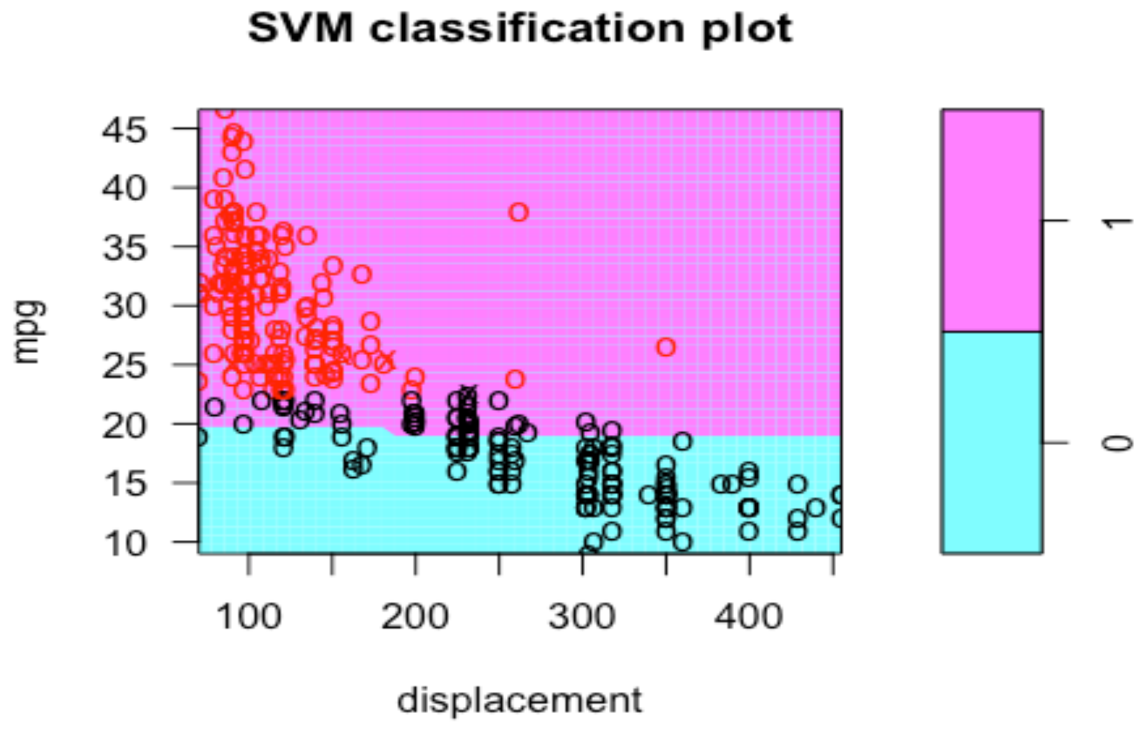
```
##
## Call:
## best.tune(method = svm, train.x = highmpg ~ ., data = AutoNew[train,
##     ], ranges = list(cost = c(0.1, 1, 10, 100, 1000), degree = c(1,
##     2, 3, 4)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1000
##      degree:  1
##       gamma:  0.125
##      coef.0:  0
##
## Number of Support Vectors:  9
##
##  ( 4 5 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

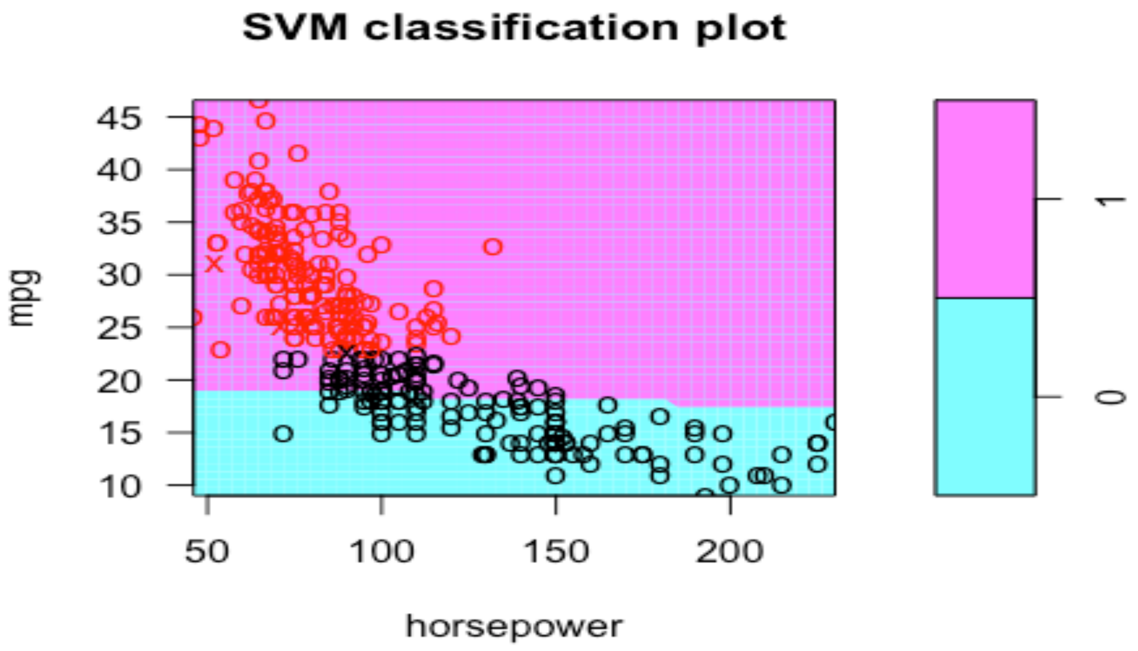
The best parameters for polynomial kernel are cost = 1000 and degree = 1.

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

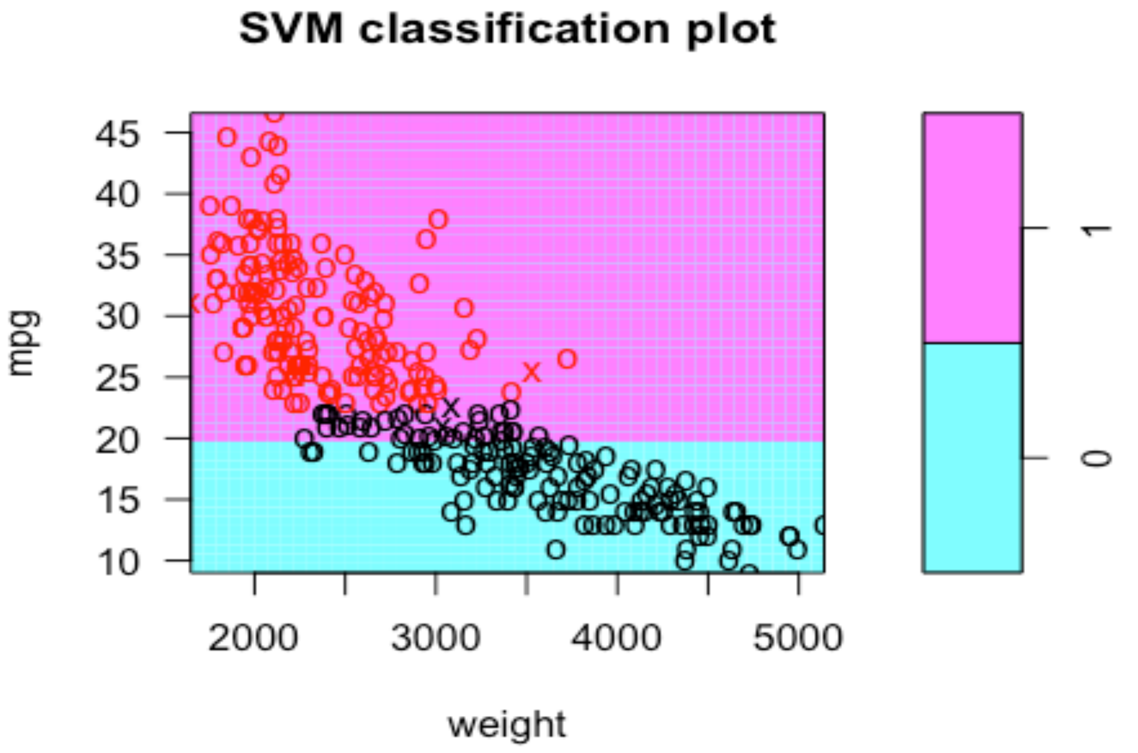
```
names <- names(AutoNew)[2:8]

for (name in names) {
  plot(bestmod.linear, AutoNew[train,], as.formula(paste0("mpg~", name)))
}
```


SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0

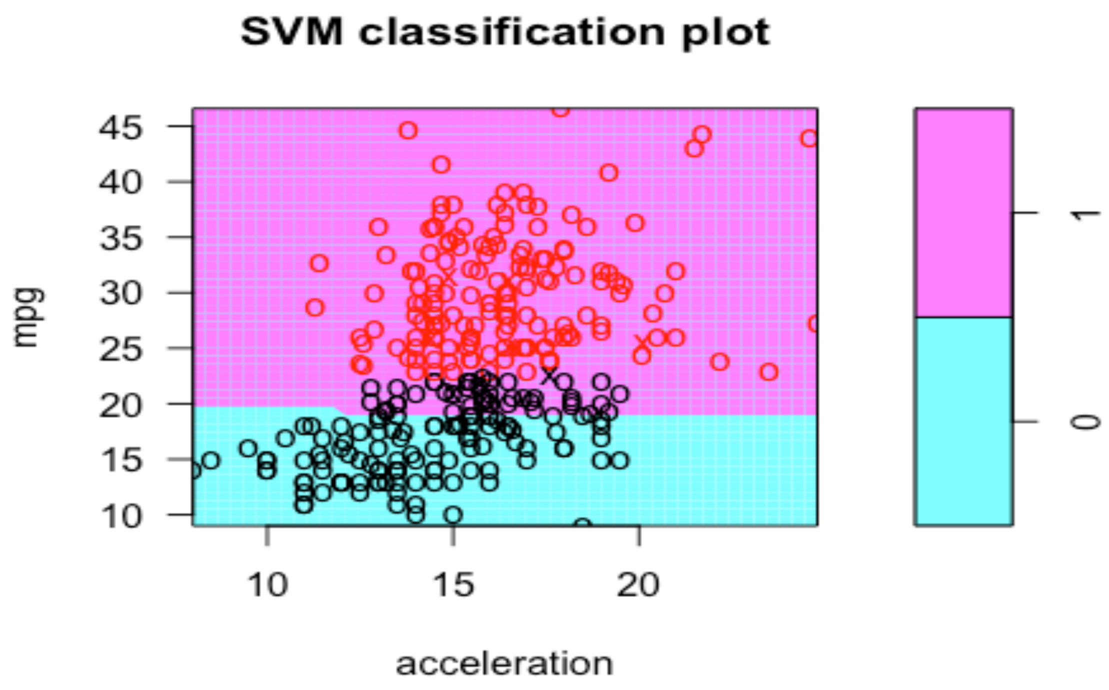

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
200
300
400
1
0

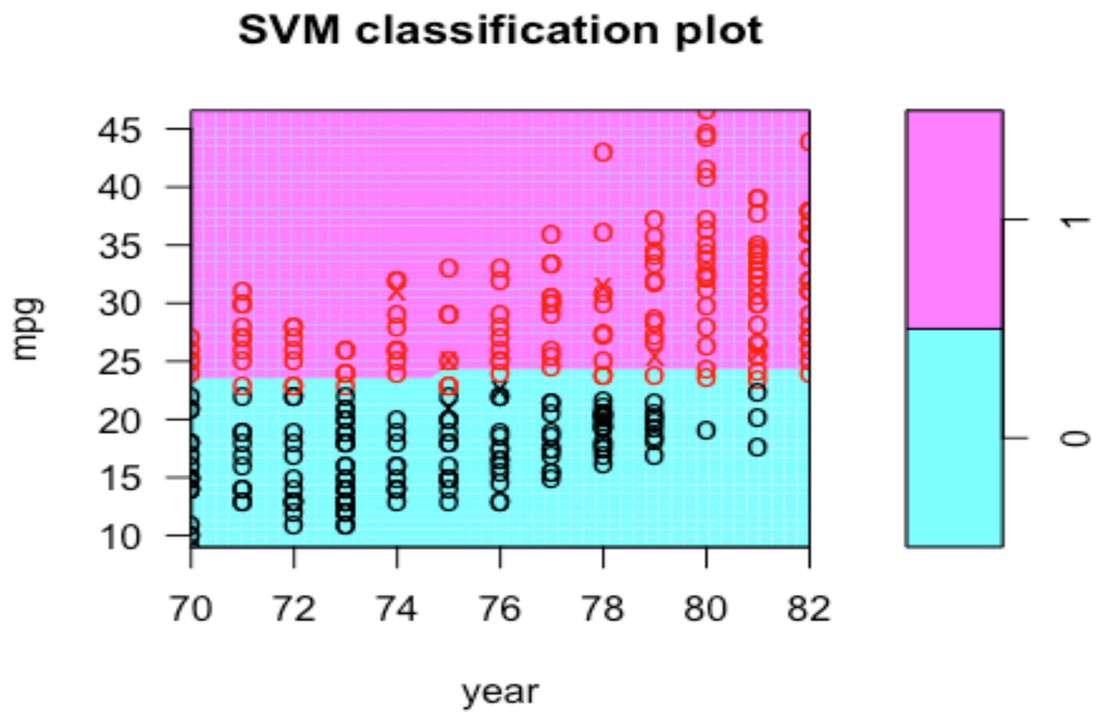


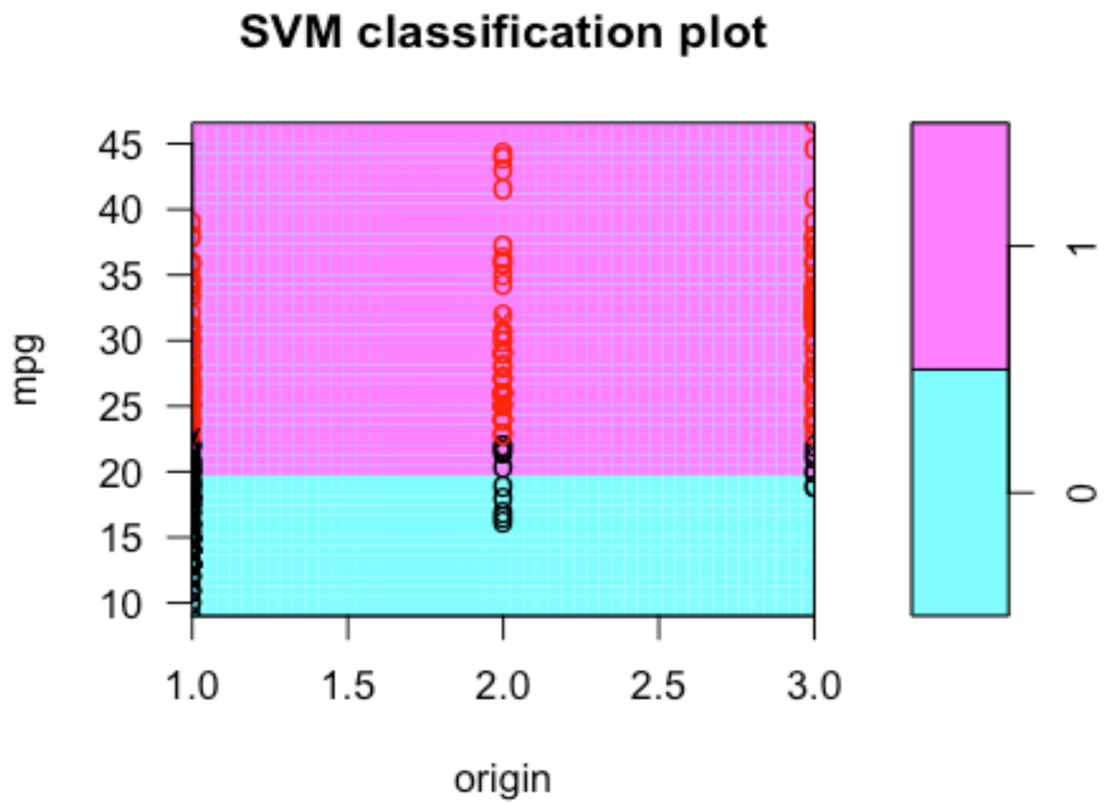
SVM classification plot
mpg
horsepower
45
40
35
30
25
20
15
10
50
100
150
200
1
0



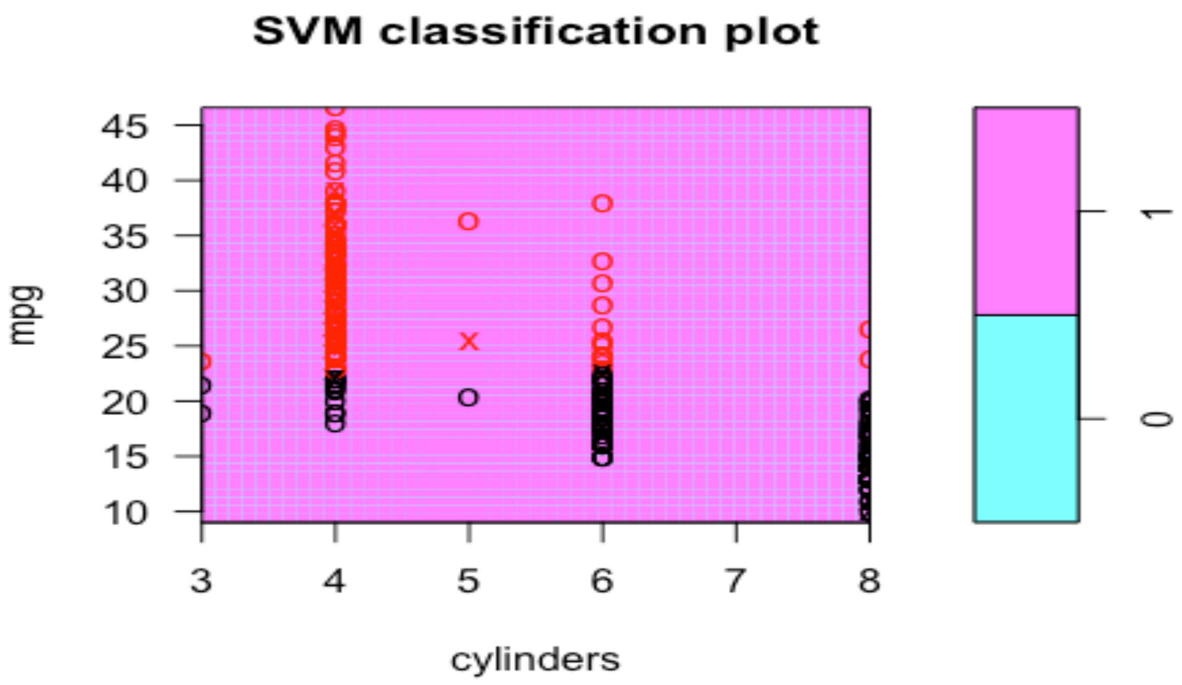
SVM classification plot
mpg
weight
45
40
35
30
25
20
15
10
2000
3000
4000
5000
1
0

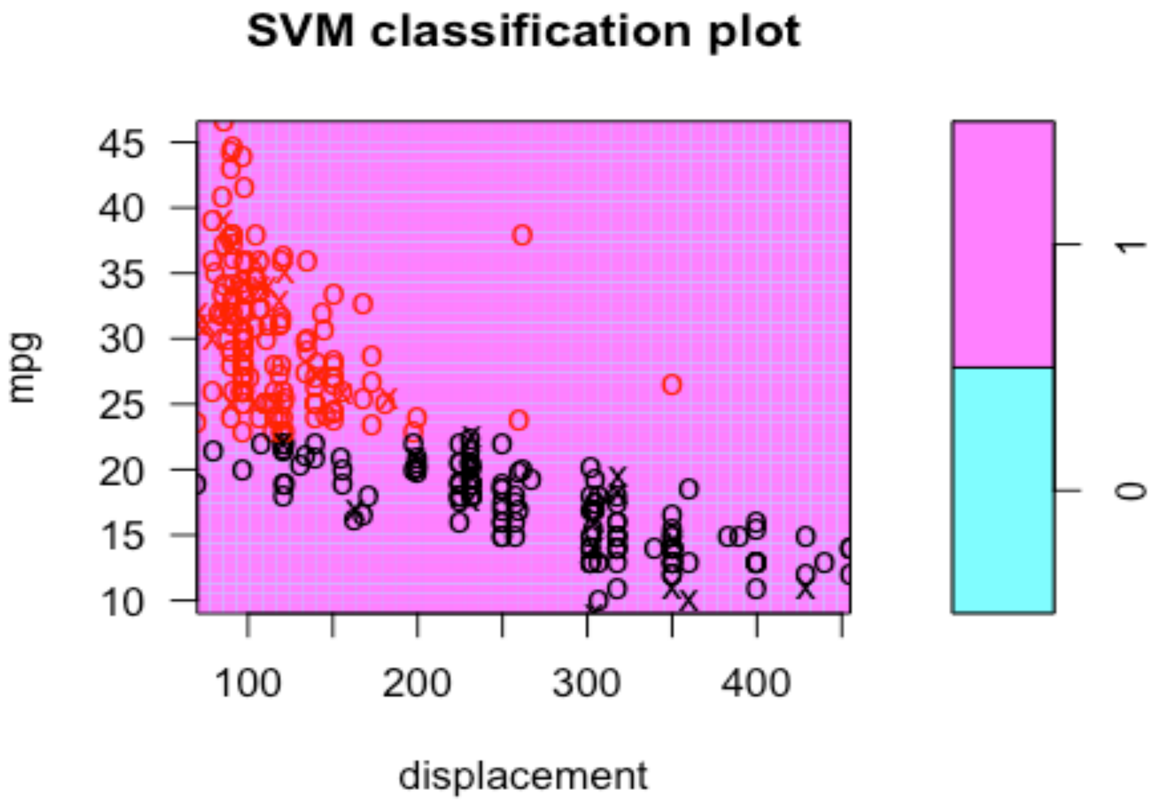

SVM classification plot
mpg
acceleration
45
40
35
30
25
20
15
10
10
15
20
1
0

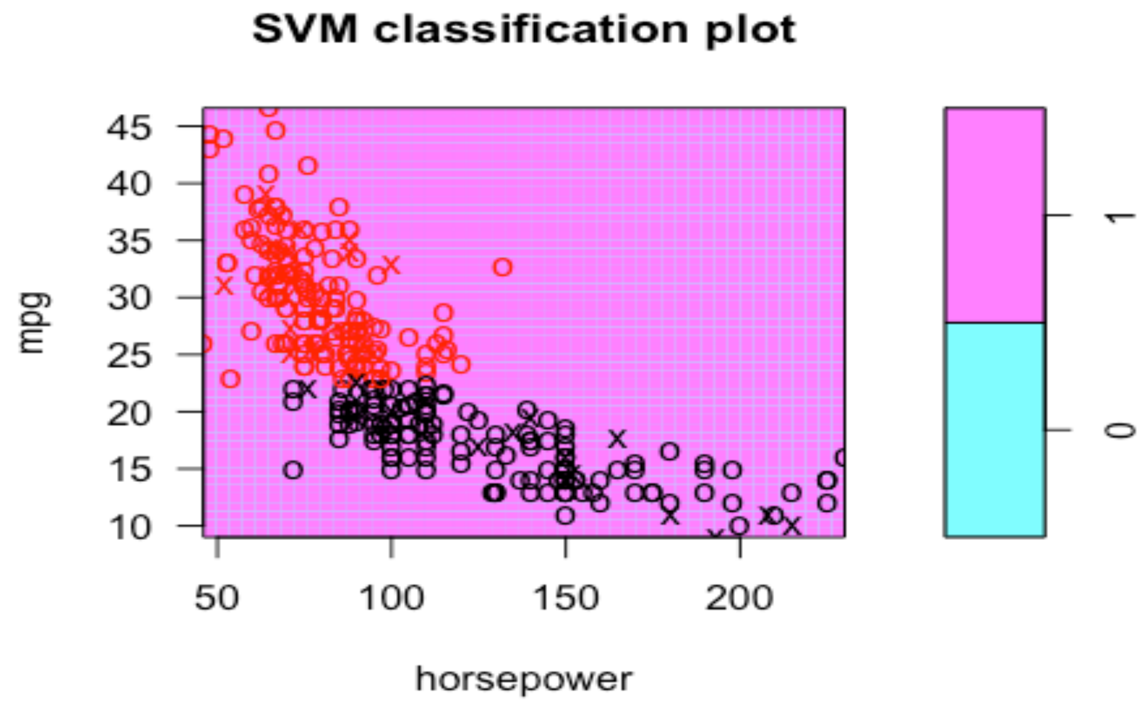

SVM classification plot
mpg
year
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
1
0

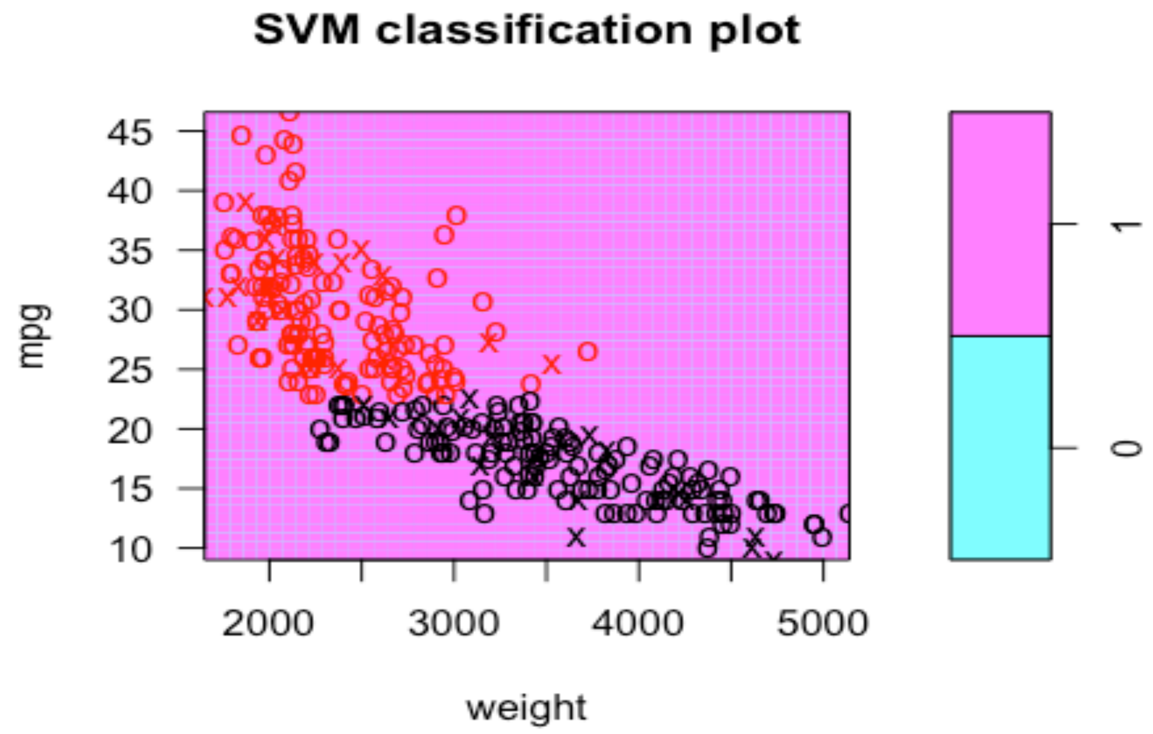


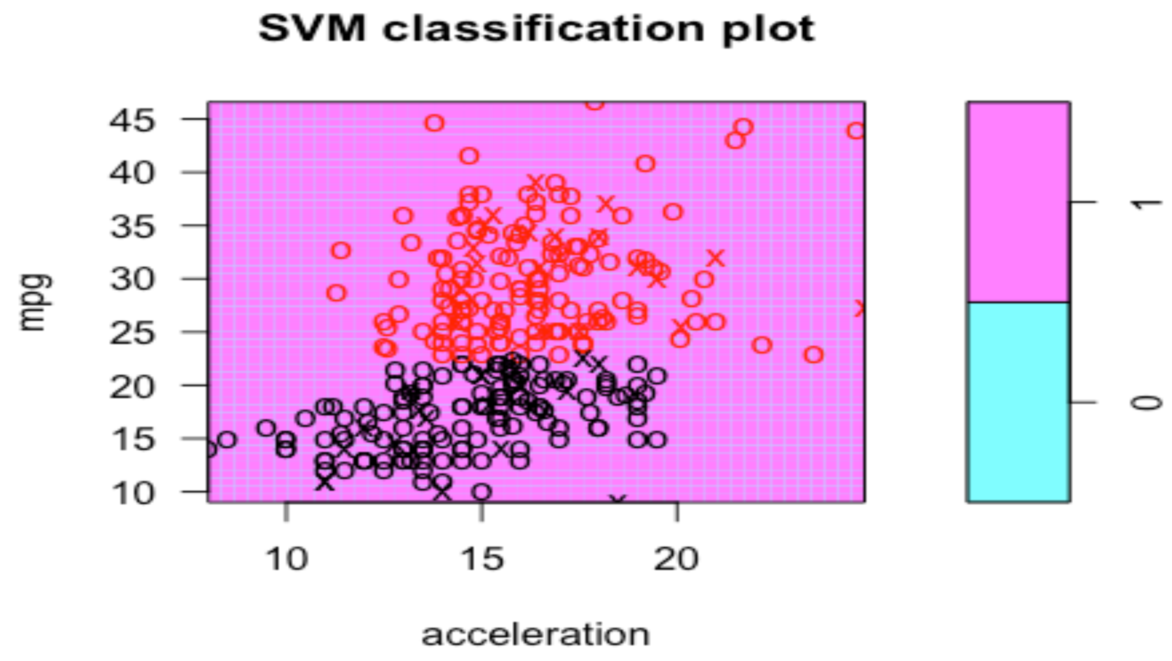
```r
for (name in names) {
  plot(bestmod.radial, AutoNew[train,], as.formula(paste0("mpg~", name)))
}
```

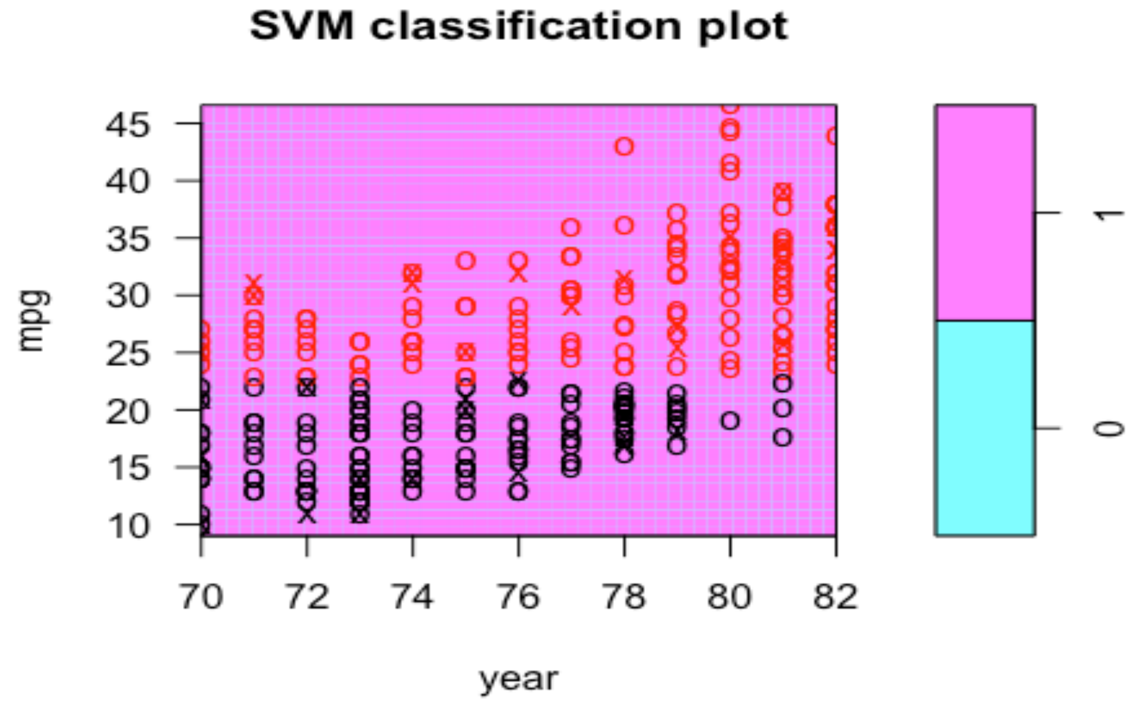

SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0

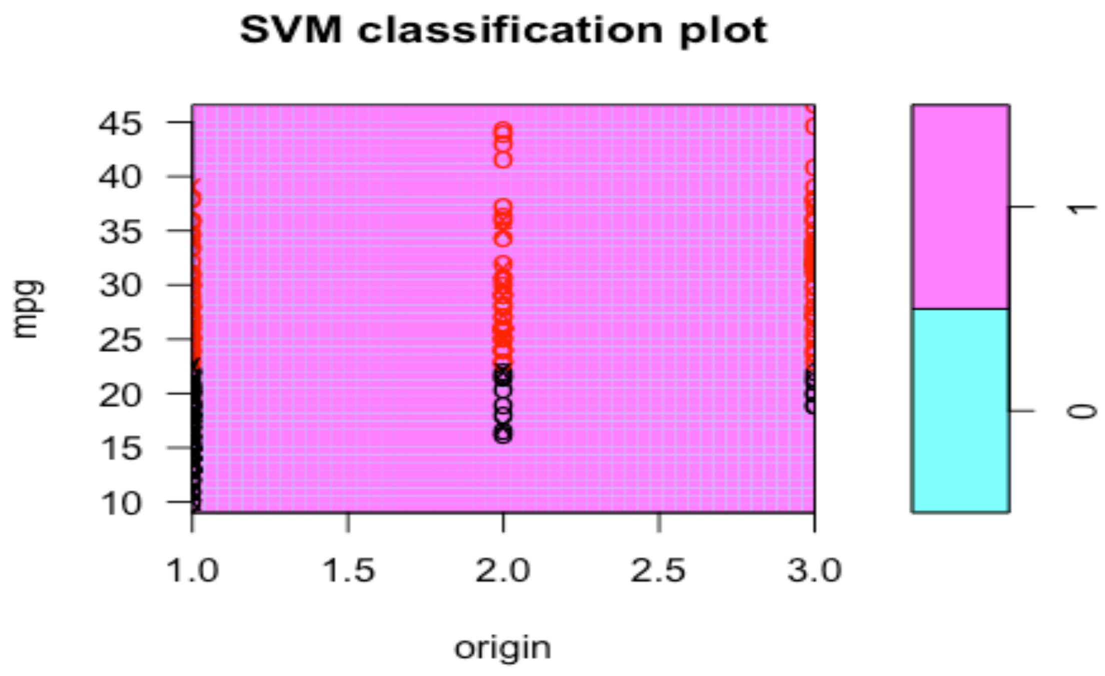

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
200
300
400
1
0


SVM classification plot
mpg
horsepower
45
40
35
30
25
20
15
10
50
100
150
200
1
0

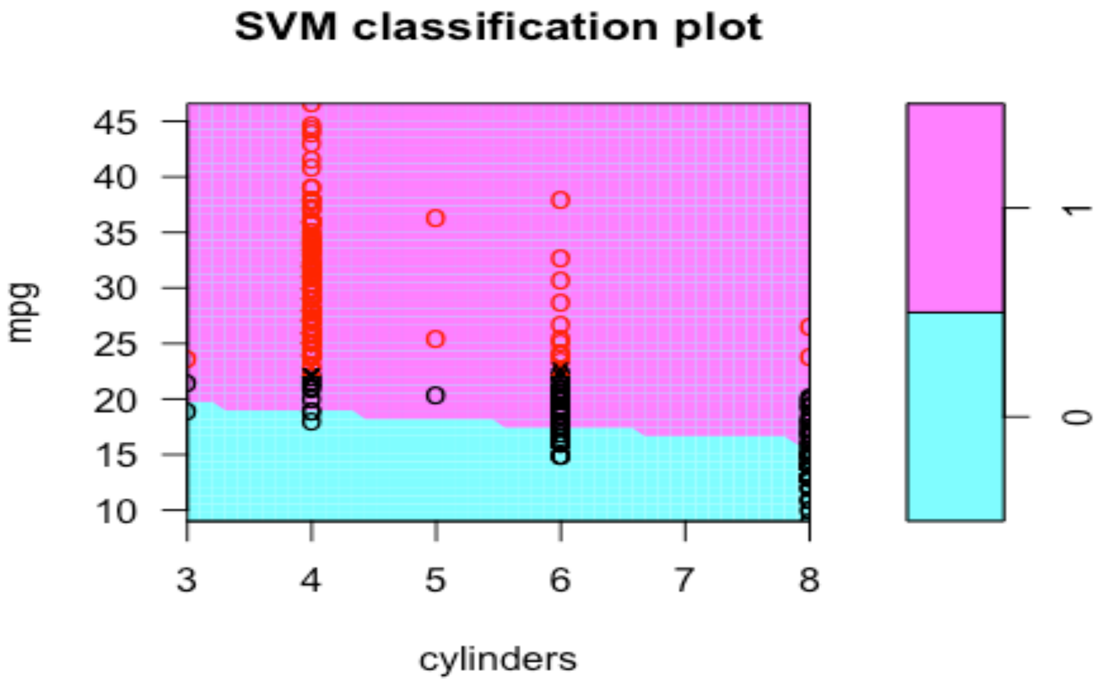

SVM classification plot
mpg
weight
45
40
35
30
25
20
15
10
2000
3000
4000
5000
1
0

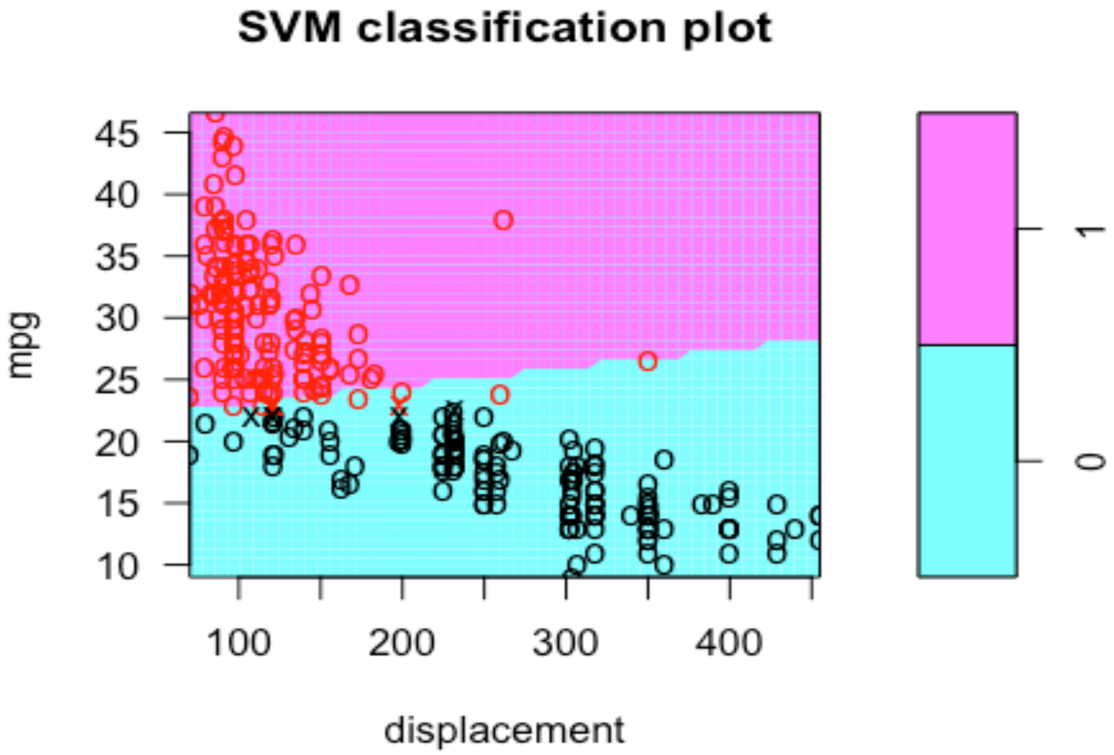

SVM classification plot
mpg
acceleration
1
0

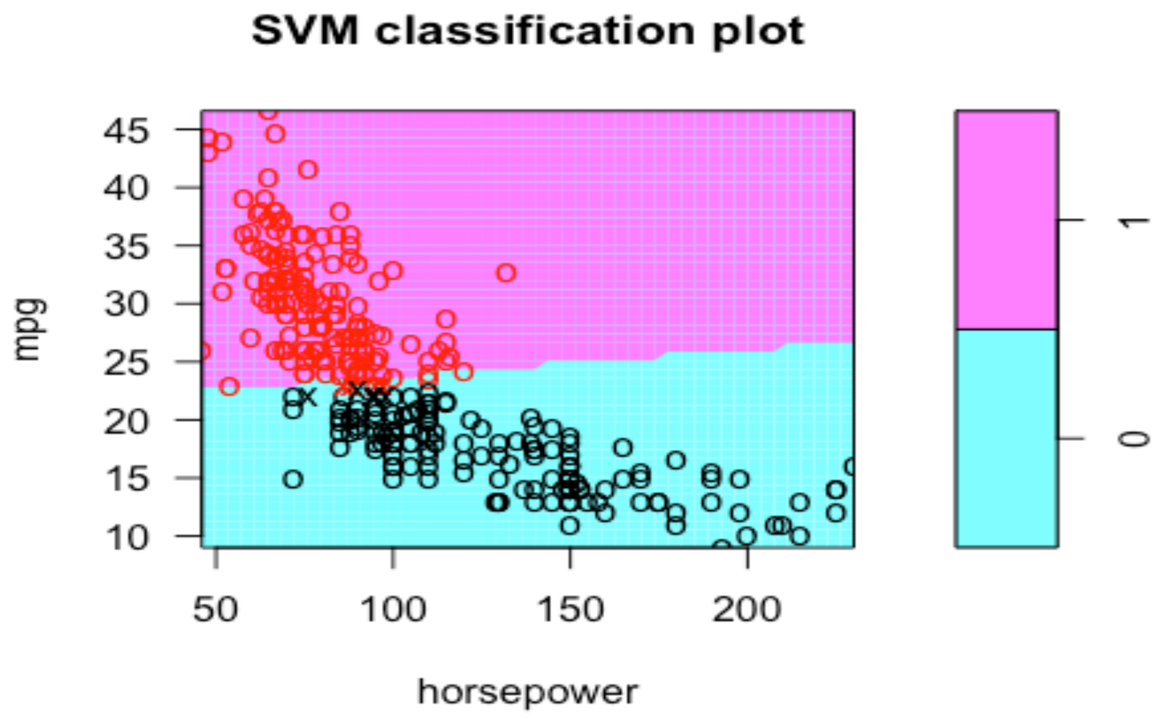

SVM classification plot
mpg
year
1
0

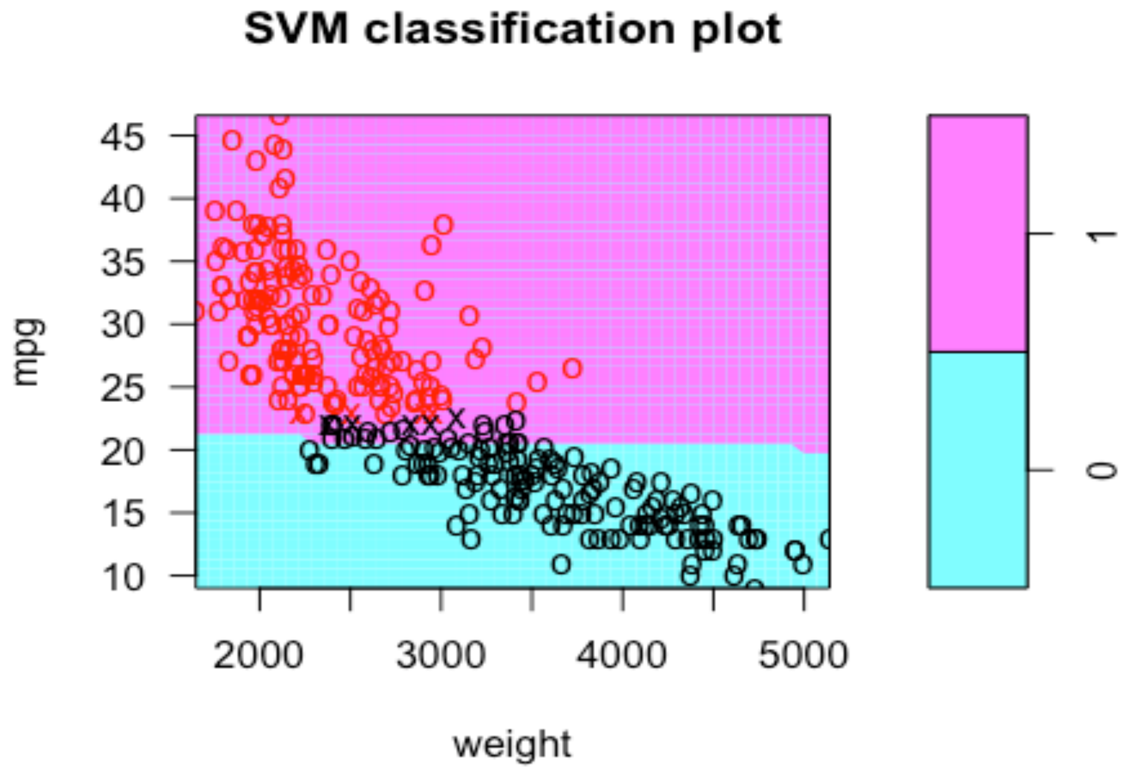


```
for (name in names) {
  plot(bestmod.poly, AutoNew[train,], as.formula(paste0("mpg~", name)))
}
```

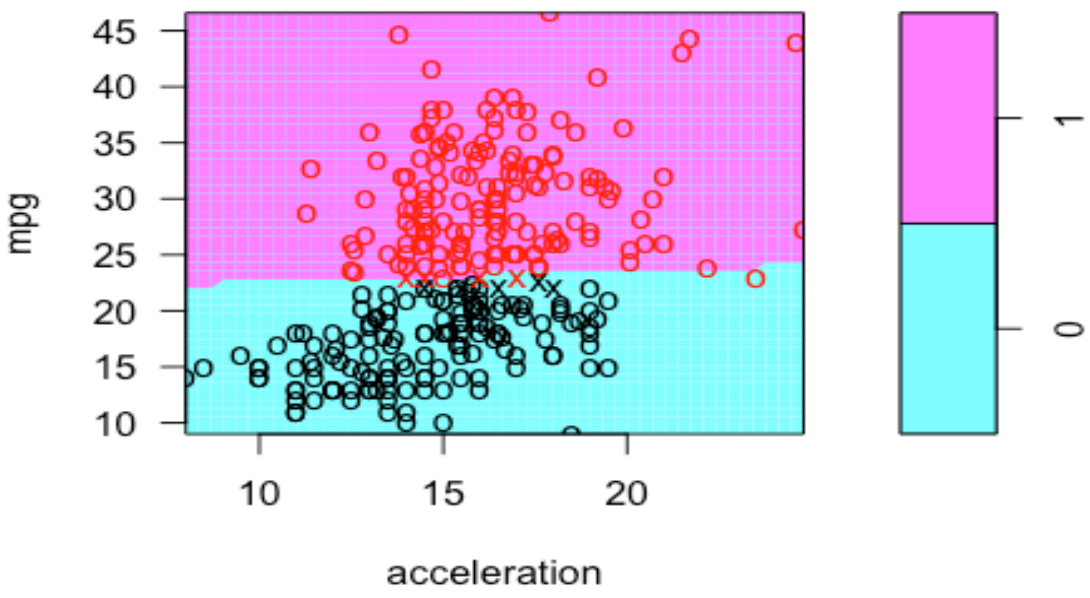

SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0


SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
200
300
400
1
0

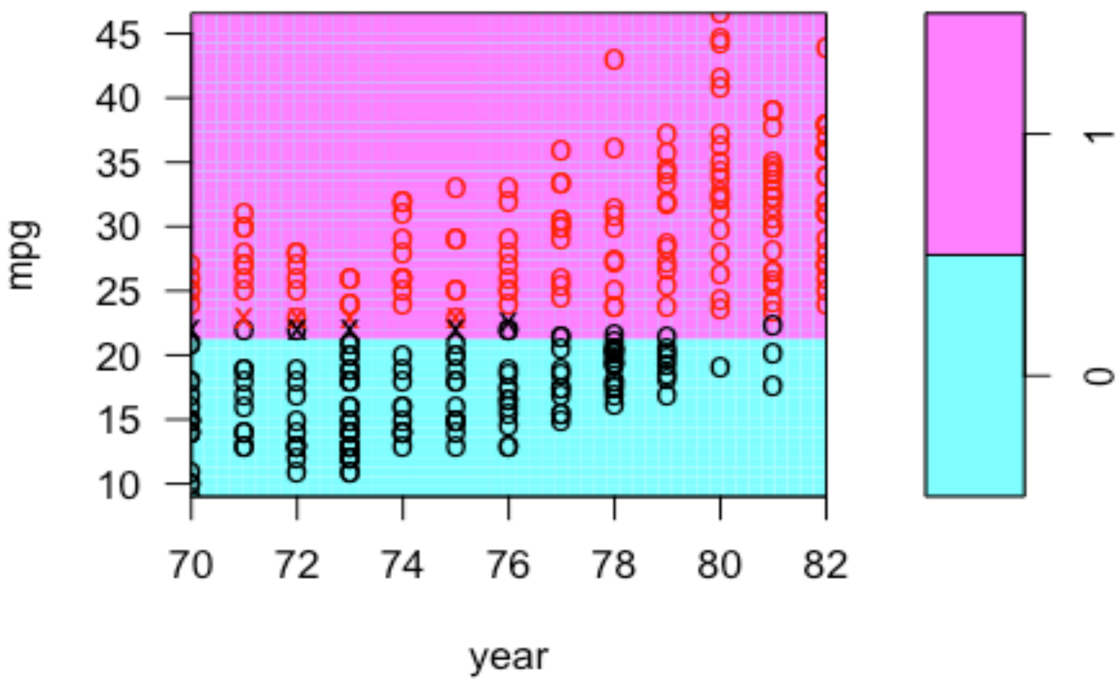

SVM classification plot
mpg
horsepower
45
40
35
30
25
20
15
10
50
100
150
200
1
0

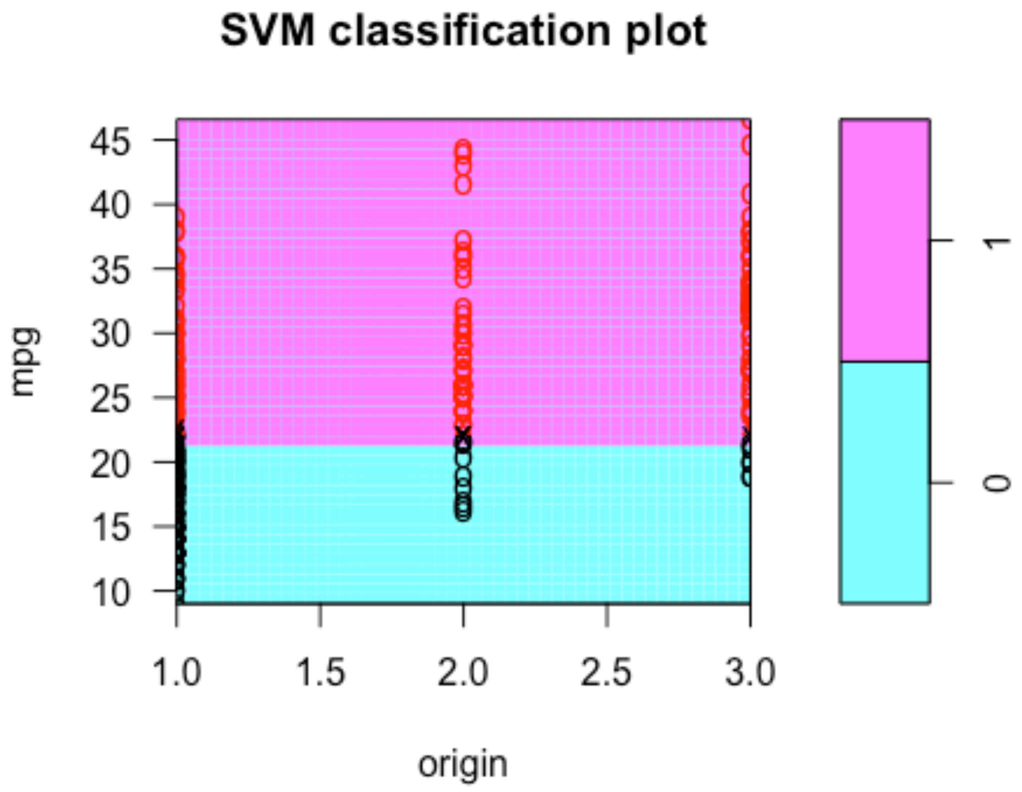

SVM classification plot
mpg
weight
45
40
35
30
25
20
15
10
2000
3000
4000
5000
1
0

SVM classification plot



SVM classification plot

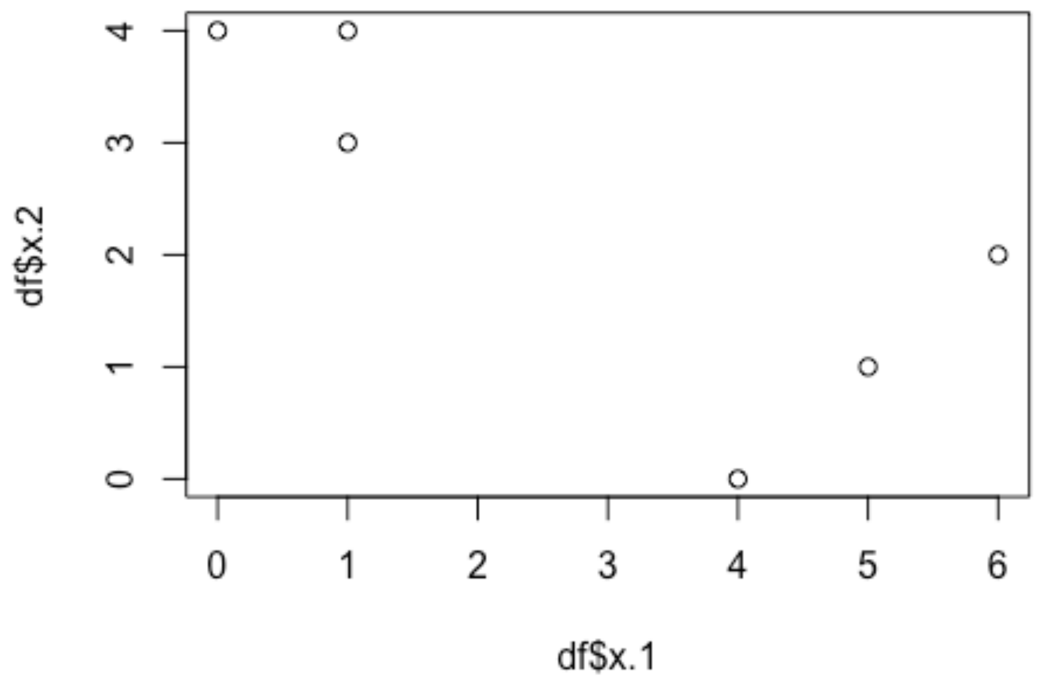




7: ISLR Ch10.3. In this problem, you will perform K-means clustering manually, with K = 2, on a small example with n = 6 observations and p = 2 features. The observations are as follows.

Obs. X1 X2
1 1 4
2 1 3
3 0 4
4 5 1
5 6 2
6 4 0
(a) Plot the observations.

```
x <- matrix(c(1,1,0,5,6,4,4,3,4,1,2,0),ncol=2)
df <- data.frame(x=x)
plot(df$x.1, df$x.2, type = 'p')
```



(b) Randomly assign a cluster label to each observation. You can use the sample() command in R to do this. Report the cluster labels for each observation.

```
cluster <- sample(2,6, replace = T)
df$cluster <- cluster
df

##   x.1 x.2 cluster
## 1   1   4       2
## 2   1   3       1
## 3   0   4       1
## 4   5   1       1
## 5   6   2       1
## 6   4   0       2
```

(c) Compute the centroid for each cluster.

```
c1 <- c(mean(df[df$cluster==1,]$x.1), mean(df[df$cluster==1,]$x.2))
c2 <- c(mean(df[df$cluster==2,]$x.1), mean(df[df$cluster==2,]$x.2))
paste0('Centroid 1 = (', c1[1], ',', c1[2],')')

## [1] "Centroid 1 = (3,2.5)"

paste0('Centroid 2 = (', c2[1], ',', c2[2],')')

## [1] "Centroid 2 = (2.5,2)"
```

(d) Assign each observation to the centroid to which it is closest, in terms of Euclidean distance. Report the cluster labels for each observation.

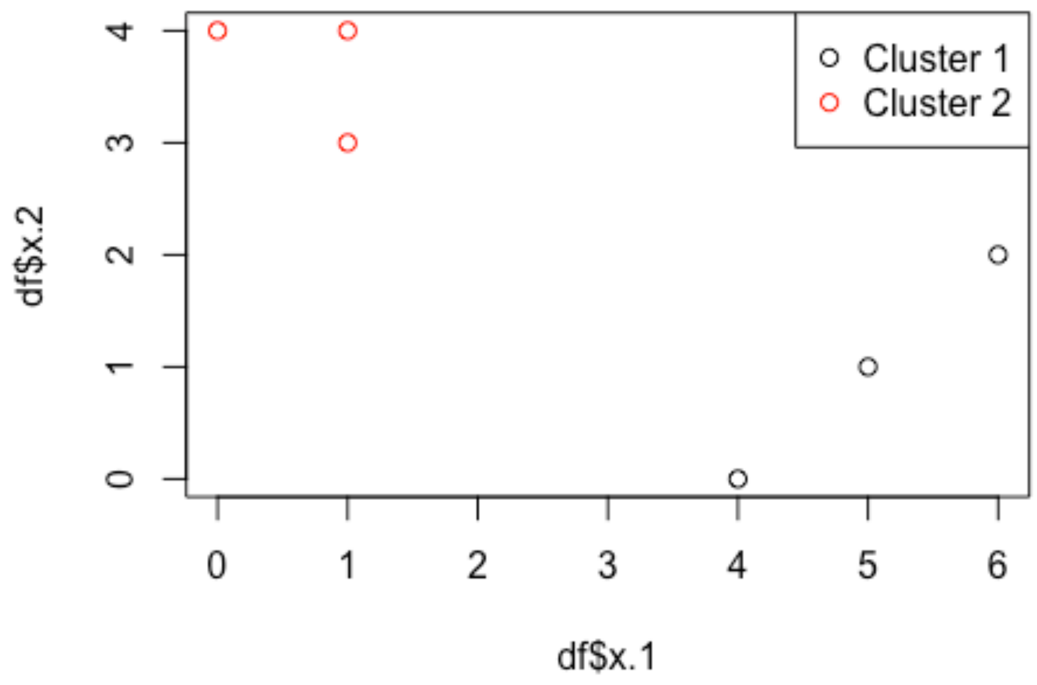
```
dist <- cbind(dist(rbind(c1,df[,1:2]))[1:6],dist(rbind(c2,df[,1:2]))[1:6])
for (i in 1:nrow(df)){
  if(dist[i,1] <= dist[i,2]){
    df$cluster[i] <- 1
  }else{
    df$cluster[i] <- 2
  }
}
df

##   x.1 x.2 cluster
## 1   1   4       1
## 2   1   3       2
## 3   0   4       2
## 4   5   1       1
## 5   6   2       1
## 6   4   0       2
```

(e) Repeat (c) and (d) until the answers obtained stop changing.

```
c1 <- c(mean(df[df$cluster==1,]$x.1), mean(df[df$cluster==1,]$x.2))
c2 <- c(mean(df[df$cluster==2,]$x.1), mean(df[df$cluster==2,]$x.2))
dist <- cbind(dist(rbind(c1,df[,1:2]))[1:6],dist(rbind(c2,df[,1:2]))[1:6])
for (i in 1:nrow(df)){
  if(dist[i,1] <= dist[i,2]){
    df$cluster[i] <- 1
  }else{
    df$cluster[i] <- 2
  }
}
df

##   x.1 x.2 cluster
## 1   1   4       2
## 2   1   3       2
## 3   0   4       2
## 4   5   1       1
## 5   6   2       1
## 6   4   0       1
```

(f) In your plot from (a), color the observations according to the cluster labels obtained.

```
plot(df$x.1, df$x.2, type = 'p', col=df$cluster)
legend('topright', c('Cluster 1', 'Cluster 2'), col = c(1,2), pch = 1)
```



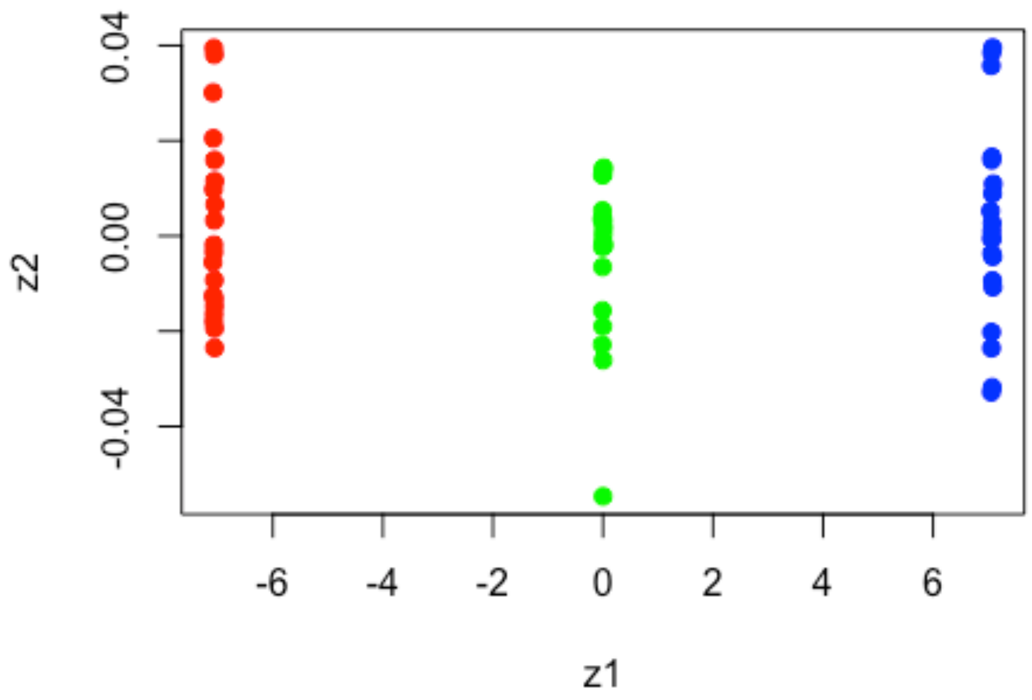
8: ISLR Ch10.10. In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data.

(a) 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.

```
y <- c(rep(1, 20), rep(2, 20), rep(3, 20))
#y <- sample(c(1, 2, 3), 60, replace = TRUE)
set.seed(556677)
#x<- matrix(rnorm(3000), ncol=50, nrow = 60 )
x1 <- matrix(rnorm(3000, mean=0, sd=.01), ncol=50, nrow = 20)
x2<- matrix(rnorm(2000, mean=1, sd=.01), ncol=50, nrow = 20)
x3<- matrix(rnorm(1000, mean=2, sd=.01), ncol=50, nrow = 20)
x<- rbind(x1, x2, x3)
gen_data <- cbind(y,x)
```

(b) Perform PCA on the 60 observations and plot the first two principal component score vectors. 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 score vectors.

```
cols <- function(vec) {
  cols <- rainbow(length(unique(vec)))
  return(cols[as.numeric(as.factor(vec))])
}

pc <- prcomp(x, scale = FALSE)
pc1<- pc$x[,1] #first principal component
plot(pc$x[, 1:2], col = cols(y), pch=19, xlab='z1', ylab='z2')
```



(c) 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?

```
set.seed(4)
km.out <- kmeans(x,3, nstart=20)
km.out$cluster

##  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

km.out$withinss

## [1] 0.09407445 0.09206628 0.09129947
```

```r
table(km.out$cluster, c(rep(3, 20), rep(1, 20), rep(2, 20)))
```

```
##
##       1  2  3
##    1 20  0  0
##    2  0 20  0
##    3  0  0 20
```

Worked out well, with all clustering accurately. (d) Perform K-means clustering with K = 2. Describe your results.

```r
set.seed(4)
km.out <- kmeans(x,2, nstart = 20)
km.out$cluster
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```r
km.out$withinss
```

```
## [1] 499.60952304   0.09206628
```

The first two groups have merged, with the last staying together.

(e) Now perform K-means clustering with K = 4, and describe your results.

```r
set.seed(4)
km.out <- kmeans(x,4, nstart = 20)
km.out$cluster
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [36] 4 4 4 4 4 1 3 3 3 1 1 3 3 1 1 1 3 3 3 3 1 1 1 3 1
```

```r
km.out$withinss
```

```
## [1] 0.03929640 0.09129947 0.04294630 0.09407445
```

Now, the first 2 groups stay together and the last group gets clustered separately. (f) Now perform K-means clustering with K = 3 on the first two principal component score vectors, 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 score vector, and the second column is the second principal component score vector. Comment on the results.

```r
set.seed(4)
km.out.pc <- kmeans(pc$x[, 1:2],3, nstart = 20)
km.out.pc$cluster
```

```
##  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```r
km.out.pc$withinss
```

```
## [1] 0.006970915 0.010851717 0.009011890
```

```
table(km.out.pc$cluster, c(rep(3, 20), rep(1, 20), rep(2, 20)))
```

```
##
##      1  2  3
##   1 20  0  0
##   2  0 20  0
##   3  0  0 20
```

As expected from earlier plot, the clustering performs well on PC1 vs PC2 (g) 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 (b)? Explain.

```
x.scaled <- scale(x)
set.seed(4)
km <- kmeans(x.scaled, centers = 3, nstart=20)
km$cluster
```

```
##  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
km$withinss
```

```
## [1] 0.1388377 0.1358489 0.1347006
```

```
table(km$cluster, c(rep(3, 20), rep(1, 20), rep(2, 20)))
```

```
##
##      1  2  3
##   1 20  0  0
##   2  0 20  0
##   3  0  0 20
```

They compare well as both accurately predicted all groupings. That said, if the scaling had been very different between the variables, then I'd expect the scaled version to perform better than unscaled.