Coursework 2

- Programme: Msc Data Science
- Name: Marco Catania
- Student ID: 13129252

1. Decision Trees

See "Q1-CW2.jpg"" file

2. Regression Trees

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

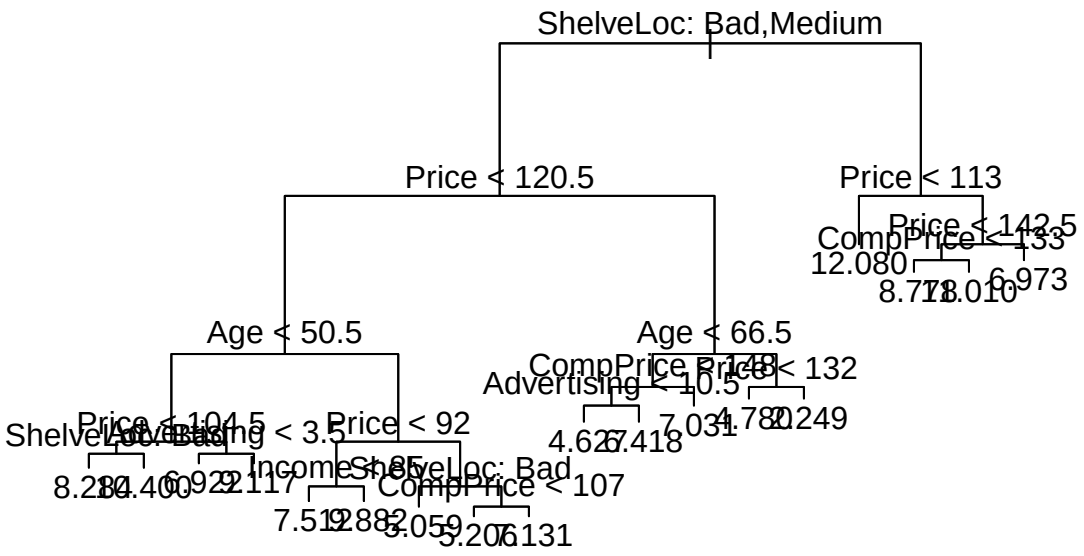
```
library(ISLR)
set.seed(1)
train<-sample(1:nrow(Carseats), nrow(Carseats)/2)
Carseats.train<-Carseats[train, ]
Carseats.test<-Carseats[-train, ]
```

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

```
library(tree)
tree.carseats<-tree(Sales~., data = Carseats.train)
summary(tree.carseats)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats.train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Advertising" "Income"
## [6] "CompPrice"
## Number of terminal nodes:  18
## Residual mean deviance:  2.36 = 429.5 / 182
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -4.2570 -1.0360  0.1024  0.0000  0.9301  3.9130
```

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



The tree plot shows that the most important indicator of Sales appears to be Shelving Location, since the first branch differenciates good Shelving Location and Bad,Medium.

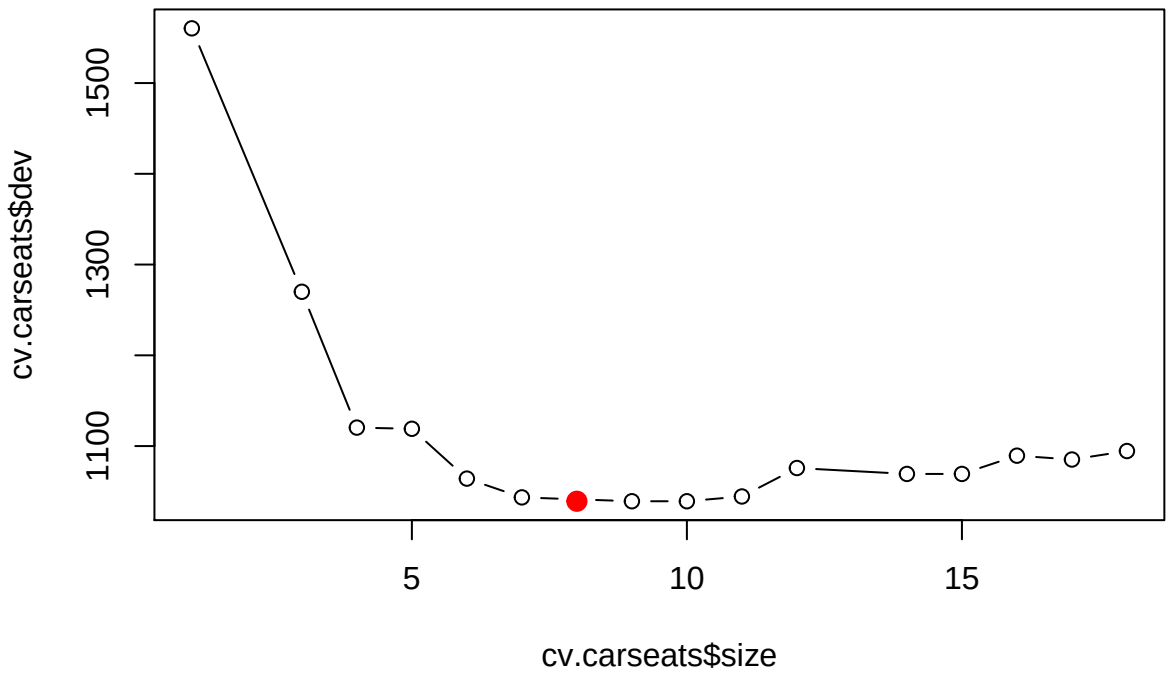
```
yhat <- predict(tree.carseats, newdata = Carseats.test)
mean((yhat - Carseats.test$Sales)^2)
```

```
## [1] 4.148897
```

The test MSE is about 4.15

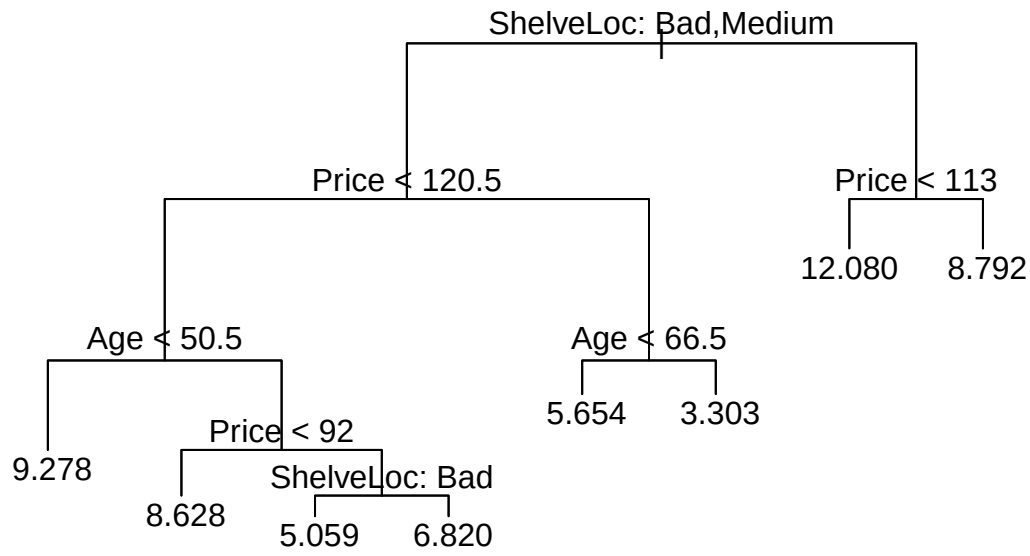
(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

```
cv.carseats <- cv.tree(tree.carseats)
plot(cv.carseats$size, cv.carseats$dev, type="b")
tree.min<- which.min(cv.carseats$dev)
points(tree.min, cv.carseats$dev[tree.min], col="red", cex=2, pch=20)
```



Size 8 is selected by cross-validation as the optimal level of tree complexity.

```
prune.carseats <- prune.tree(tree.carseats, best=tree.min)
plot(prune.carseats)
text(prune.carseats, pretty=0)
```




```
yhat <- predict(prune.carseats, newdata=Carseats.test)
mean((yhat-Carseats.test$Sales)^2)
```

```
## [1] 5.09085
```

Pruning the tree increases the test MSE to about 5.1. It does not improve it.

(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 4.6-14
```

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

```
bag.carseats <- randomForest(Sales~., data= Carseats.train, mtry=10, importance=TRUE)
yhat.bag <- predict(bag.carseats, newdata=Carseats.test)
mean((yhat.bag-Carseats.test$Sales)^2)
```

```
## [1] 2.633915
```

Bagging decreases the test MSE to 2.6.

```
importance(bag.carseats)
```

```
##                %IncMSE IncNodePurity
```

```
## CompPrice   16.9874366    126.852848
## Income       3.8985402     78.314126
## Advertising 16.5698586    123.702901
## Population   0.6487058     62.328851
## Price       55.3976775    514.654890
## ShelveLoc   42.7849818    319.133777
## Age         20.5135255    185.582077
## Education    3.4615211     42.253410
## Urban       -2.5125087      8.700009
## US           7.3586645     18.180651
```

Price and ShelveLoc are the most important variables.

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

```
rf.carseats <- randomForest(Sales~., data= Carseats.train, mtry=3, importance=TRUE)
yhat.rf <- predict(rf.carseats, newdata=Carseats.test)
mean((yhat.rf-Carseats.test$Sales)^2)
```

```
## [1] 3.321154
```

Random Forests gives test MSE 3.3. For m= square root (p), test MSE increases compared to bagging that uses m=p.

```
importance(rf.carseats)
```

```
##               %IncMSE IncNodePurity
## CompPrice    7.443405     130.87552
## Income       3.227858     127.18662
## Advertising 13.388259     139.53499
## Population  -1.031306     102.32154
## Price       36.616911     369.59534
## ShelveLoc   31.284175     233.49549
## Age         17.622273     206.09959
## Education    1.454555      70.41374
## Urban       -1.864781      15.13225
## US           6.193082      35.74746
```

Price and ShelveLoc are here too, the most important variables.

3. Classification Trees

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

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

(b) 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, ans describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

```
library(tree)
tree.oj <- tree(Purchase~., data = OJ.train)
summary(tree.oj)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ.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
```

The fitted tree has a training error rate of 0.165 and 8 terminal nodes.

(c) 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.

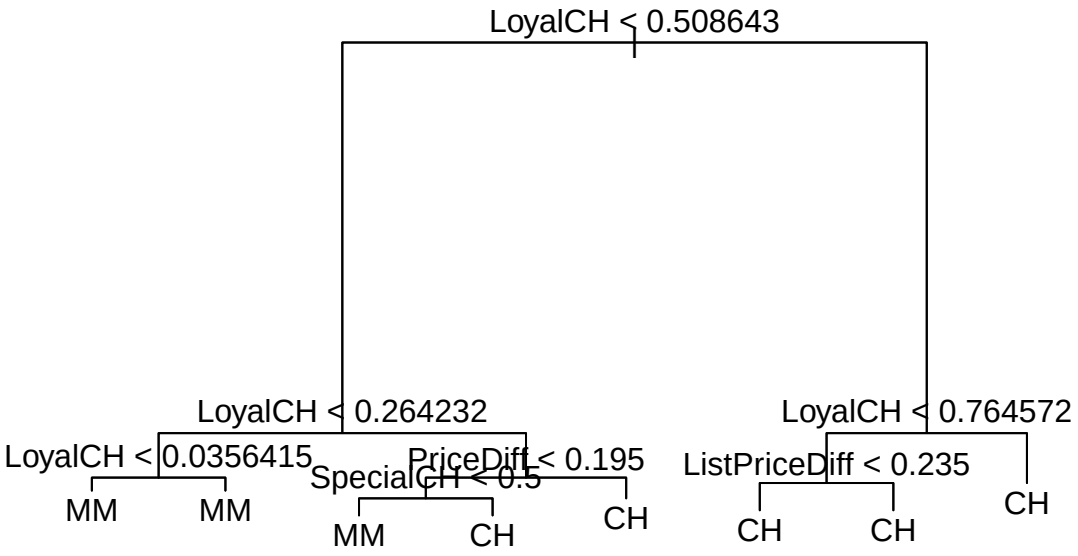
```
tree.oj
```

```
## 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 ) *
```

We pick the terminal node with label 20. The split criterion is SpecialCH < 0.5 The number of observations in that branch is 70. and the deviance is 60.89. The overall prediction in that branch is MM. The fraction of observations in that branch that takes on value of MM is about 84%. The 16% left take on value of CH.

(d) Create a plot of the tree, and interpret the results.

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



The most important indicator of "Purchase" is "LoyalCH", since the first branch differenciates customer brand loyalty level to CH.

(e) 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 ?

```
tree.pred <- predict(tree.oj, OJ.test, type="class")
table(tree.pred, OJ.test$Purchase)
```

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

```
(12+49)/270
```

```
## [1] 0.2259259
```

The test error rate is about 22%

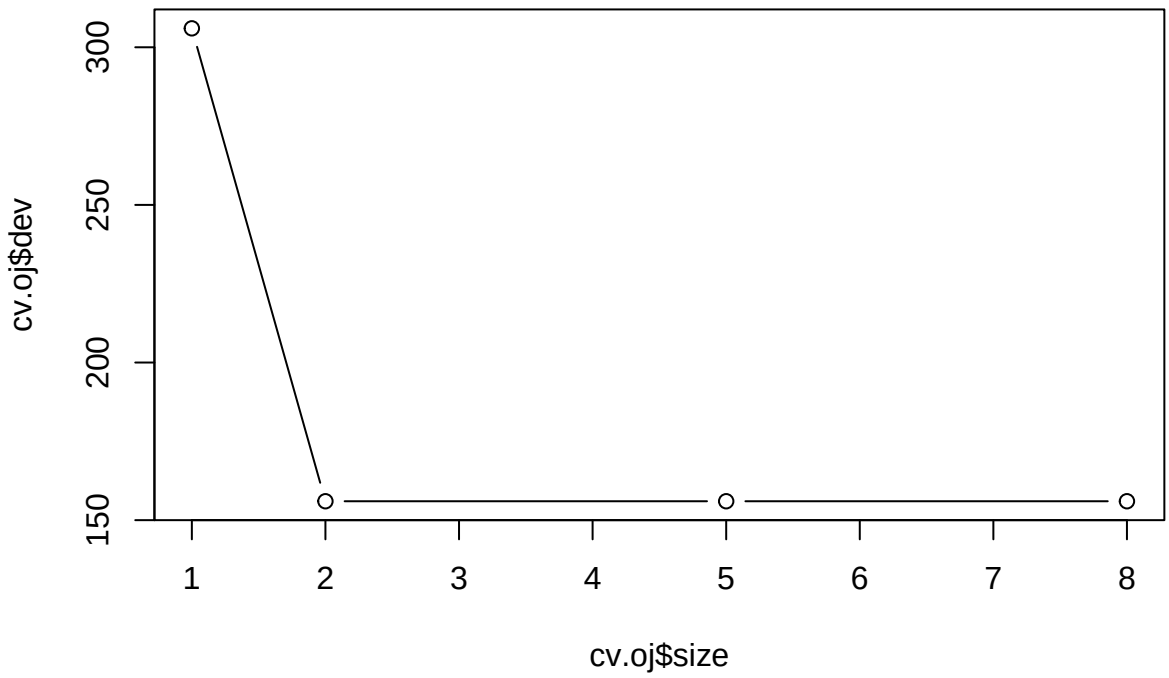
(f) Apply the cv.tree() function to the training set in order to determine the optimal tree size.

```
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"
```

(g) 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")
```

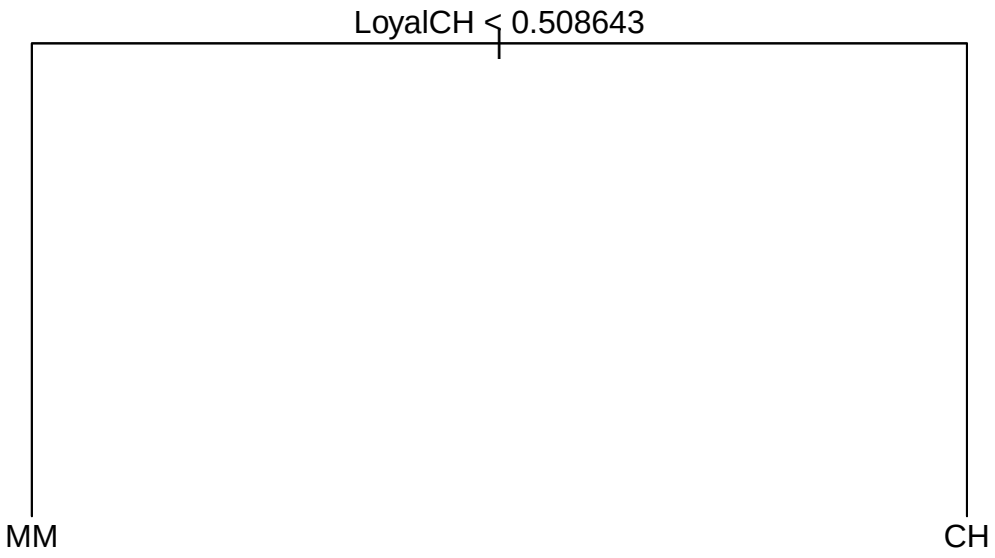


(h) Which tree size corresponds to the lowest cross-validated classification error rate?

All trees's size have the same lowest cv error rate: 156. But the smallest tree is the 2-node tree: therefore it is the best for prediction on test data.

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

```
prune.oj <- prune.misclass(tree.oj, best=2)
plot(prune.oj)
text(prune.oj, pretty=0)
```



(j) Compare the training error rates between the pruned and unpruned trees. Which is higher?

```
summary(tree.oj)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ.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 = 3:2)
## Variables actually used in tree construction:
## [1] "LoyalCH"
## Number of terminal nodes:  2
## Residual mean deviance:  0.9115 = 727.4 / 798
## Misclassification error rate: 0.1825 = 146 / 800
```

Misclassification error rate unpruned tree: 0.165 Misclassification error rate pruned tree: 0.1825

The misclassification error rate is slighly higher for the pruned tree.

(k) Compare the test error rates between the pruned and unpruned trees. Which is higher?

Misclassification error rate unpruned tree: 0.165

```
prune.predict <- predict(prune.oj, OJ.test, type="class")
table(prune.predict, OJ.test$Purchase)
```

```
##
## prune.predict  CH  MM
##            CH 119  30
##            MM  40  81
```

```
(40+30)/270
```

```
## [1] 0.2592593
```

The pruning increased the test error rate to about 0.259, but produced a more interpretable tree regarding customer loyalty and purchase.

4. SVM

(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.

```
library(ISLR)
var <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
Auto$mpglevel <- as.factor(var)
```

(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.

```
library(e1071)
set.seed(1)
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "linear", ranges = list(cost = c(0.001, 0.01,
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.01275641
```

```
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.09442308 0.03837365
## 2 1e-02 0.07403846 0.05471525
## 3 1e-01 0.03826923 0.05148114
## 4 1e+00 0.01275641 0.01344780
## 5 5e+00 0.01782051 0.01229997
## 6 1e+01 0.02038462 0.01074682
## 7 1e+02 0.03820513 0.01773427
```

Cost 1 seems to perform best, with the lowest cross validation error rate.

(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)
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "polynomial", ranges = list(cost = c(0.001, 0
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      2
##
## - best performance: 0.3013462
##
## - Detailed performance results:
##     cost degree     error dispersion
## 1  1e-03      2 0.5611538 0.04344202
## 2  1e-02      2 0.5611538 0.04344202
## 3  1e-01      2 0.5611538 0.04344202
## 4  1e+00      2 0.5611538 0.04344202
## 5  5e+00      2 0.5611538 0.04344202
## 6  1e+01      2 0.5382051 0.05829238
## 7  1e+02      2 0.3013462 0.09040277
## 8  1e-03      3 0.5611538 0.04344202
## 9  1e-02      3 0.5611538 0.04344202
## 10 1e-01      3 0.5611538 0.04344202
## 11 1e+00      3 0.5611538 0.04344202
## 12 5e+00      3 0.5611538 0.04344202
## 13 1e+01      3 0.5611538 0.04344202
## 14 1e+02      3 0.3322436 0.11140578
## 15 1e-03      4 0.5611538 0.04344202
## 16 1e-02      4 0.5611538 0.04344202
## 17 1e-01      4 0.5611538 0.04344202
## 18 1e+00      4 0.5611538 0.04344202
## 19 5e+00      4 0.5611538 0.04344202
```

```
## 20 1e+01      4 0.5611538 0.04344202
## 21 1e+02      4 0.5611538 0.04344202
```

For a polynomial kernel, Cost 100 and degree 2 seem to perform the best, with the lowest cross validation error rate.

```
set.seed(1)
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "radial", ranges = list(cost = c(0.001, 0.01,
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100  0.01
##
## - best performance: 0.01532051
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-03 1e-03 0.56115385 0.04344202
## 2  1e-02 1e-03 0.56115385 0.04344202
## 3  1e-01 1e-03 0.53551282 0.06253584
## 4  1e+00 1e-03 0.09442308 0.03837365
## 5  5e+00 1e-03 0.07403846 0.05471525
## 6  1e+01 1e-03 0.07147436 0.05103685
## 7  1e+02 1e-03 0.02294872 0.02807826
## 8  1e-03 1e-02 0.56115385 0.04344202
## 9  1e-02 1e-02 0.56115385 0.04344202
## 10 1e-01 1e-02 0.09185897 0.03862507
## 11 1e+00 1e-02 0.07147436 0.05103685
## 12 5e+00 1e-02 0.04326923 0.04975032
## 13 1e+01 1e-02 0.02551282 0.03812986
## 14 1e+02 1e-02 0.01532051 0.01788871
## 15 1e-03 1e-01 0.56115385 0.04344202
## 16 1e-02 1e-01 0.19153846 0.07612945
## 17 1e-01 1e-01 0.07916667 0.05201159
## 18 1e+00 1e-01 0.05608974 0.05092939
## 19 5e+00 1e-01 0.03064103 0.02637448
## 20 1e+01 1e-01 0.02551282 0.02076457
## 21 1e+02 1e-01 0.02807692 0.01458261
## 22 1e-03 1e+00 0.56115385 0.04344202
## 23 1e-02 1e+00 0.56115385 0.04344202
## 24 1e-01 1e+00 0.56115385 0.04344202
## 25 1e+00 1e+00 0.06634615 0.06187383
## 26 5e+00 1e+00 0.06128205 0.06186124
## 27 1e+01 1e+00 0.06128205 0.06186124
## 28 1e+02 1e+00 0.06128205 0.06186124
## 29 1e-03 5e+00 0.56115385 0.04344202
## 30 1e-02 5e+00 0.56115385 0.04344202
## 31 1e-01 5e+00 0.56115385 0.04344202
## 32 1e+00 5e+00 0.49224359 0.03806832
```

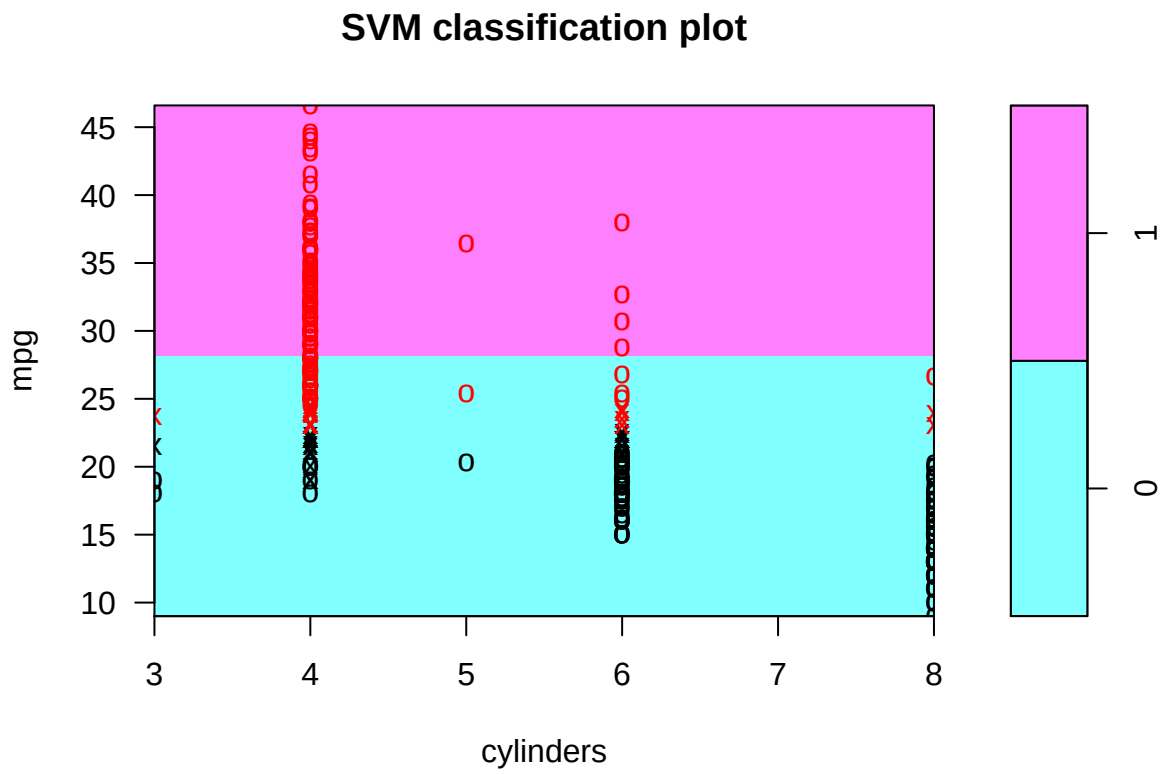
```
## 33 5e+00 5e+00 0.48967949 0.03738577
## 34 1e+01 5e+00 0.48967949 0.03738577
## 35 1e+02 5e+00 0.48967949 0.03738577
## 36 1e-03 1e+01 0.56115385 0.04344202
## 37 1e-02 1e+01 0.56115385 0.04344202
## 38 1e-01 1e+01 0.56115385 0.04344202
## 39 1e+00 1e+01 0.51775641 0.04471079
## 40 5e+00 1e+01 0.51012821 0.03817175
## 41 1e+01 1e+01 0.51012821 0.03817175
## 42 1e+02 1e+01 0.51012821 0.03817175
## 43 1e-03 1e+02 0.56115385 0.04344202
## 44 1e-02 1e+02 0.56115385 0.04344202
## 45 1e-01 1e+02 0.56115385 0.04344202
## 46 1e+00 1e+02 0.56115385 0.04344202
## 47 5e+00 1e+02 0.56115385 0.04344202
## 48 1e+01 1e+02 0.56115385 0.04344202
## 49 1e+02 1e+02 0.56115385 0.04344202
```

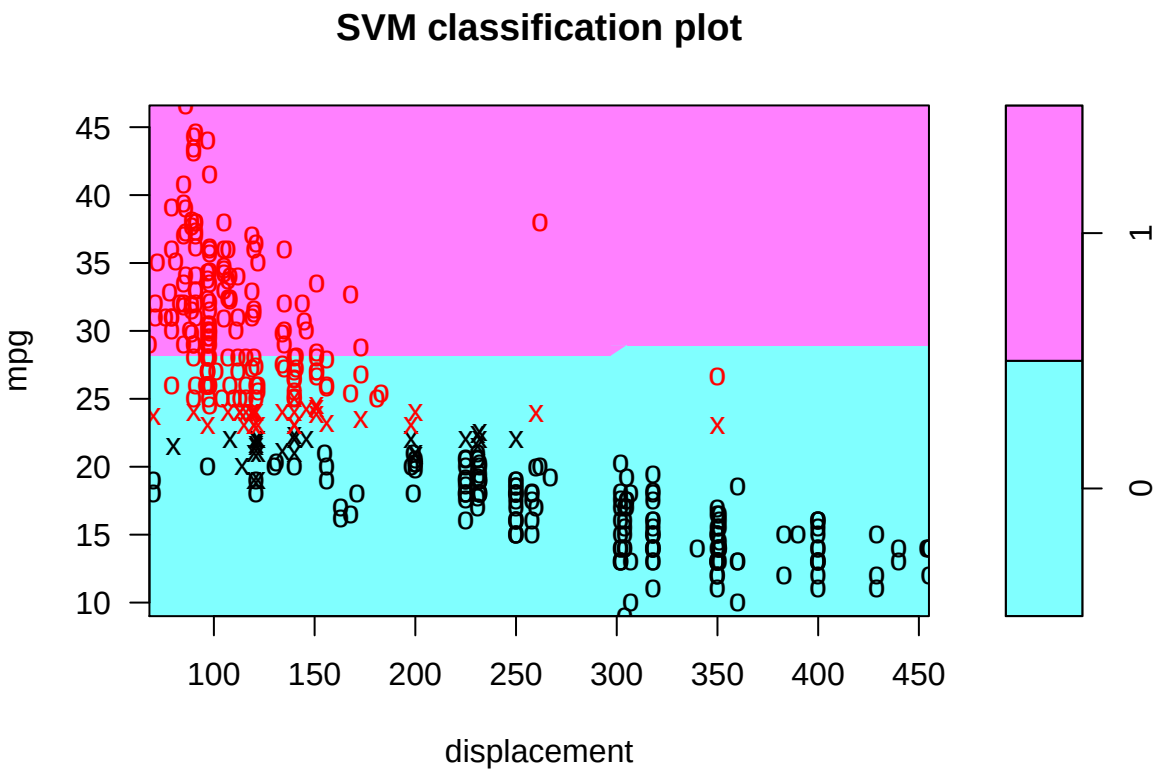
For a radial kernel, Cost 100 and radial 0.01 seem to perform the best, with the lowest cross validation error rate.

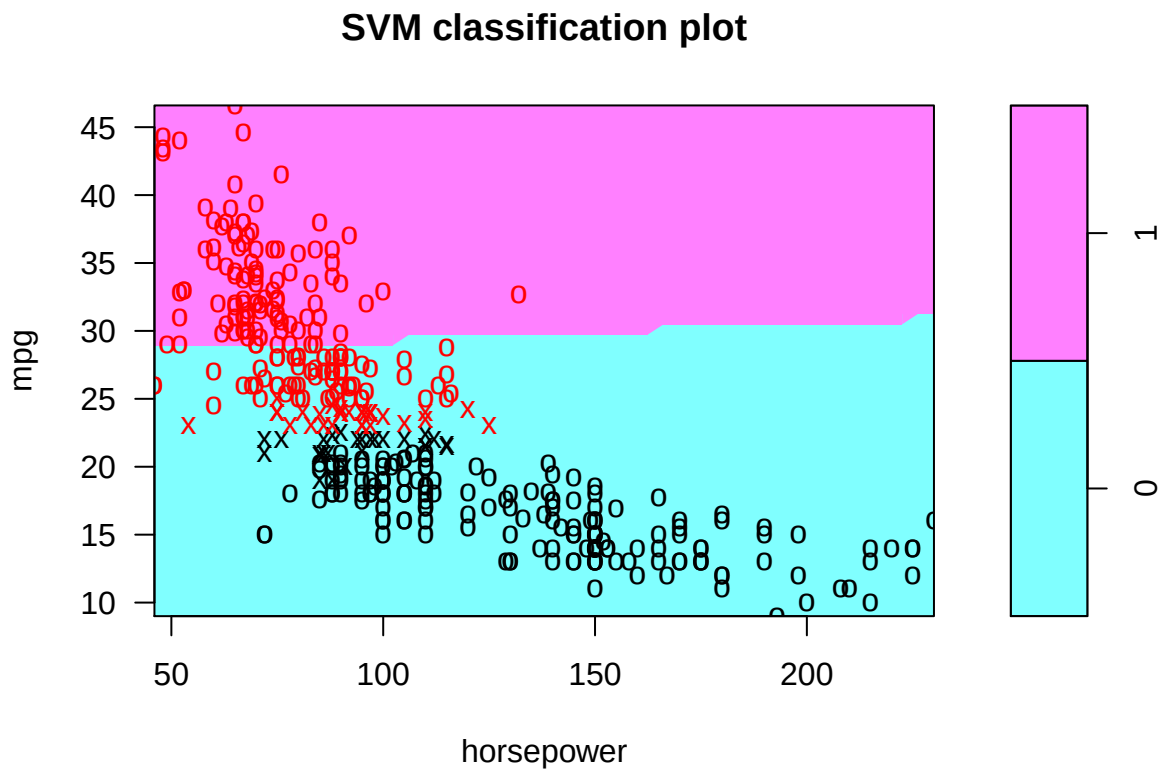
(d) Make some plots to back up your assertions in (b) and (c)

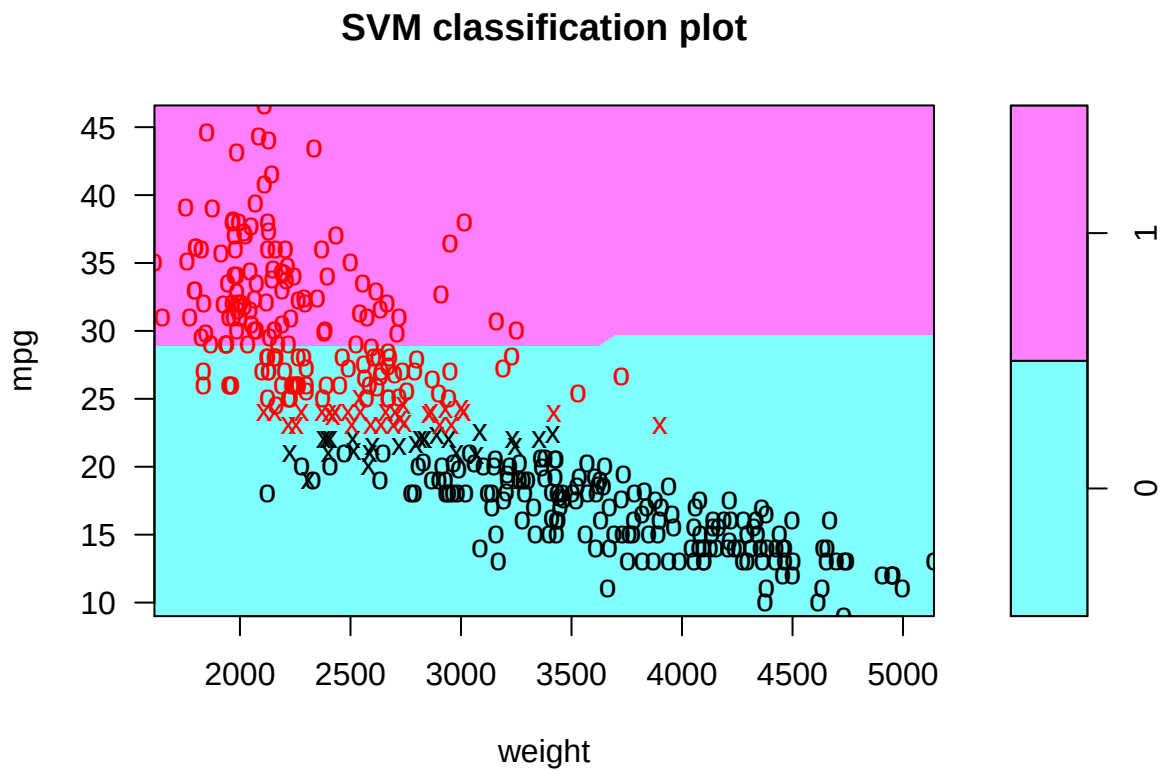
```
svm.linear <- svm(mpglevel ~ ., data = Auto, kernel = "linear", cost = 1)
svm.poly <- svm(mpglevel ~ ., data = Auto, kernel = "polynomial", cost = 100, degree = 2)
svm.radial <- svm(mpglevel ~ ., data = Auto, kernel = "radial", cost = 100, gamma = 0.01)
```

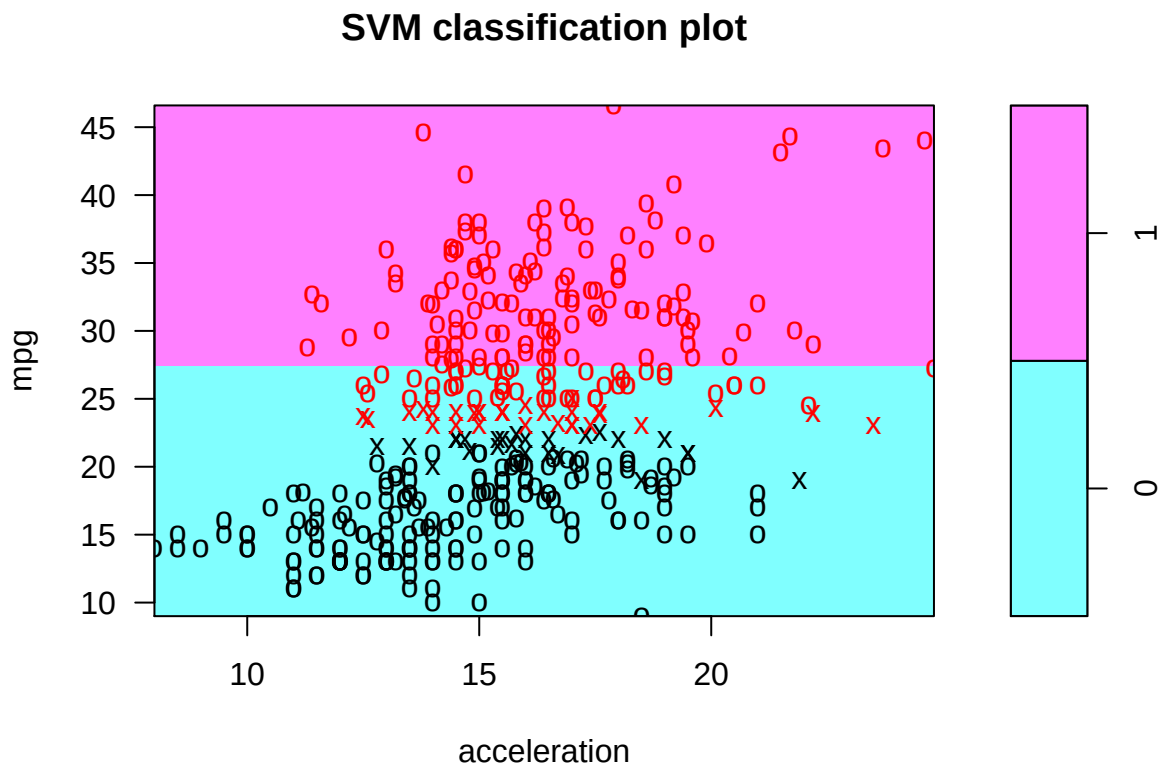
```
for (name in names(Auto)[!(names(Auto) %in% c("mpg", "mpglevel", "name"))])
{
plot(svm.linear, Auto, as.formula(paste("mpg~", name, sep="")))
}
```

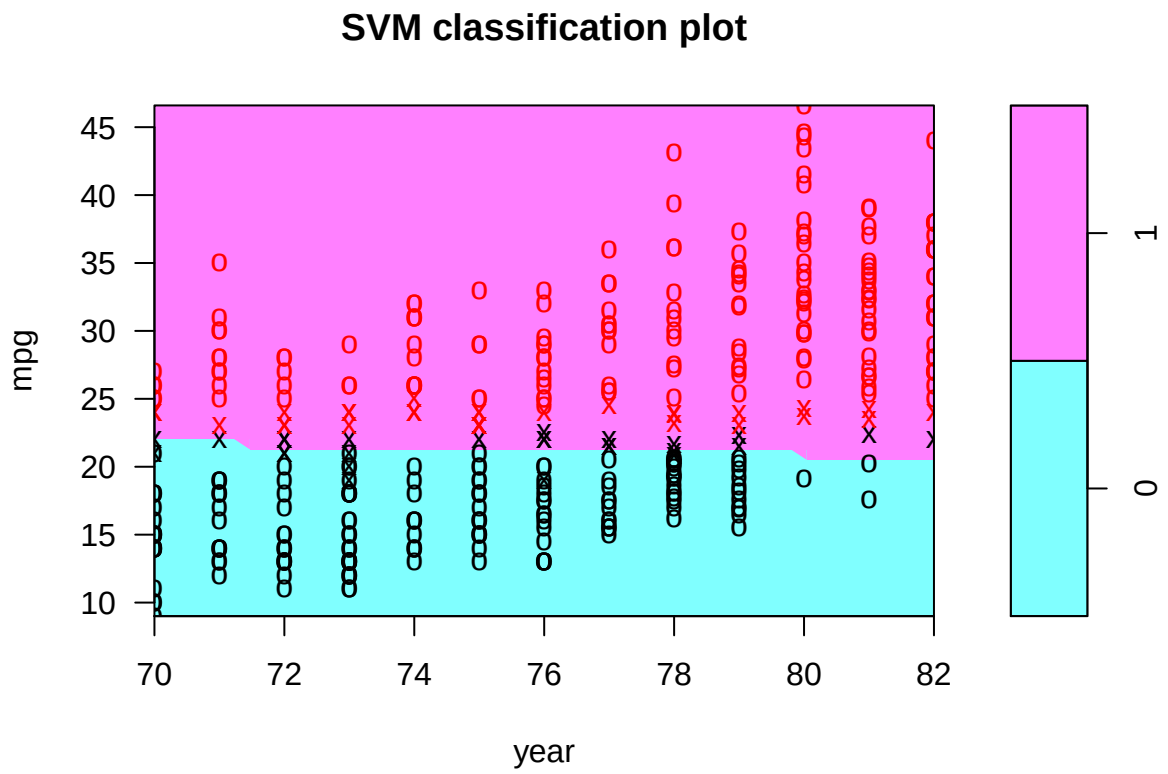

SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0

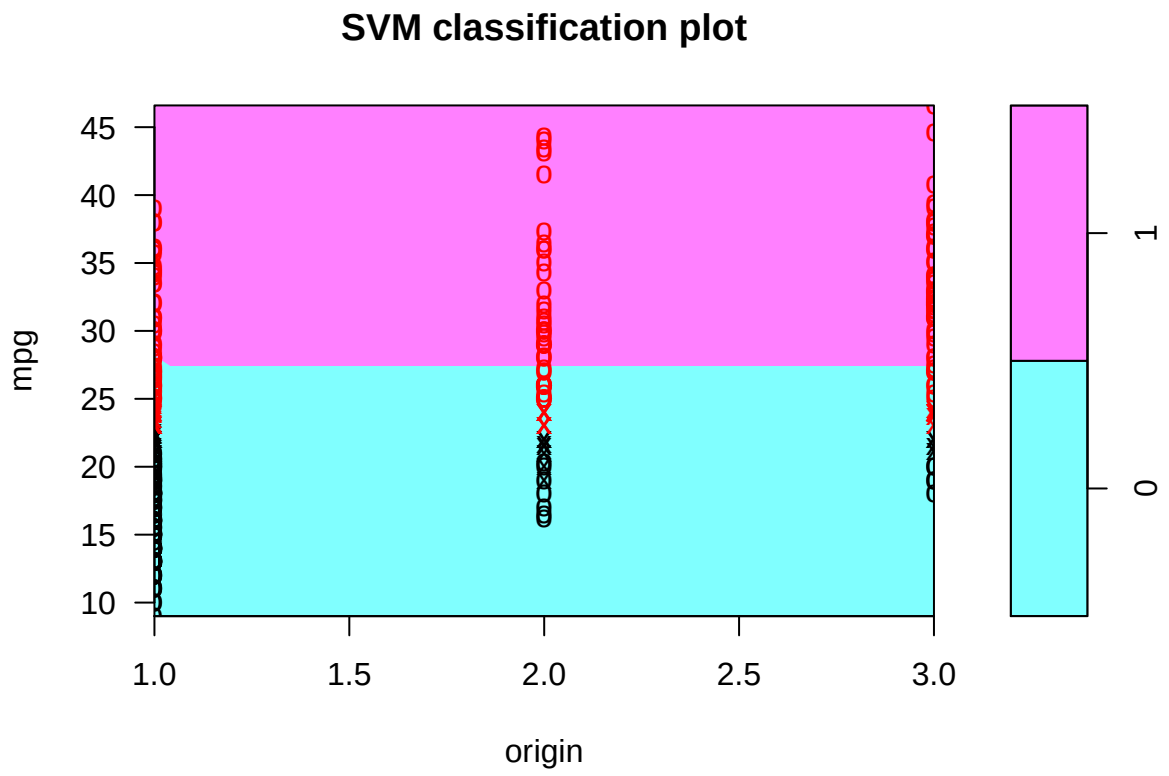

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0


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

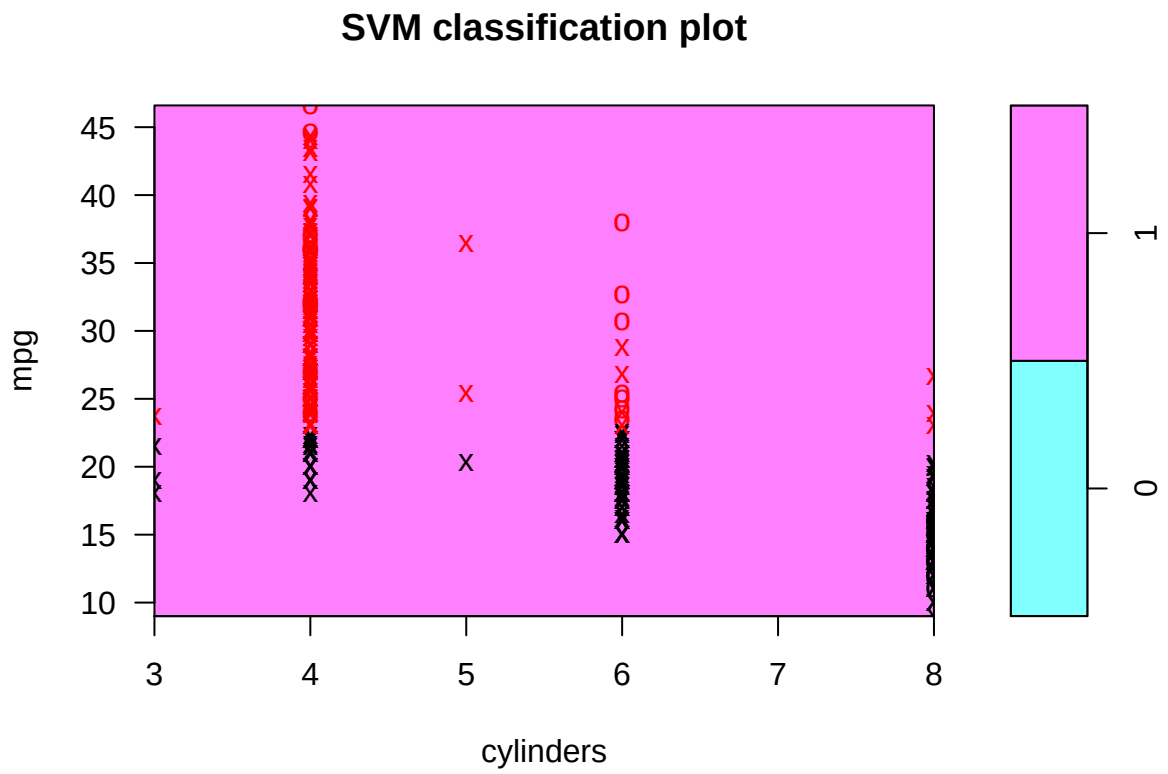

SVM classification plot
mpg
weight
45
40
35
30
25
20
15
10
2000
2500
3000
3500
4000
4500
5000
1
0

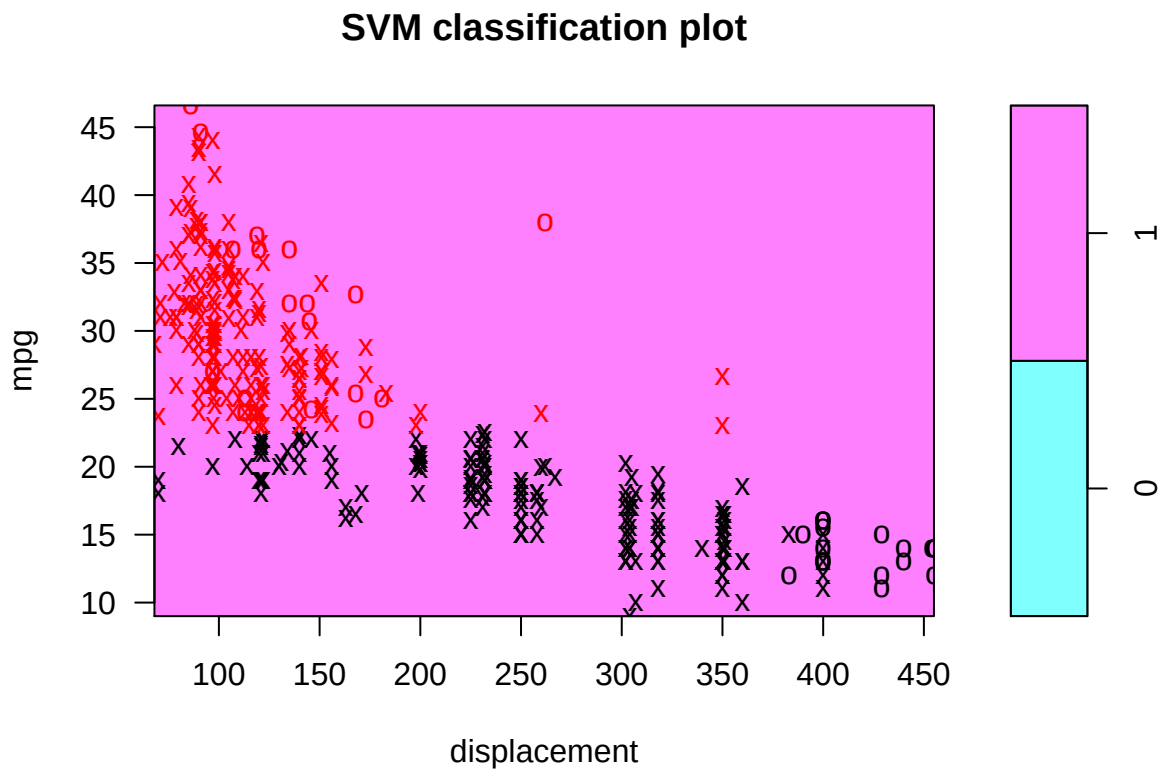

SVM classification plot
mpg
acceleration
45
40
35
30
25
20
15
10
10
15
20
1
0

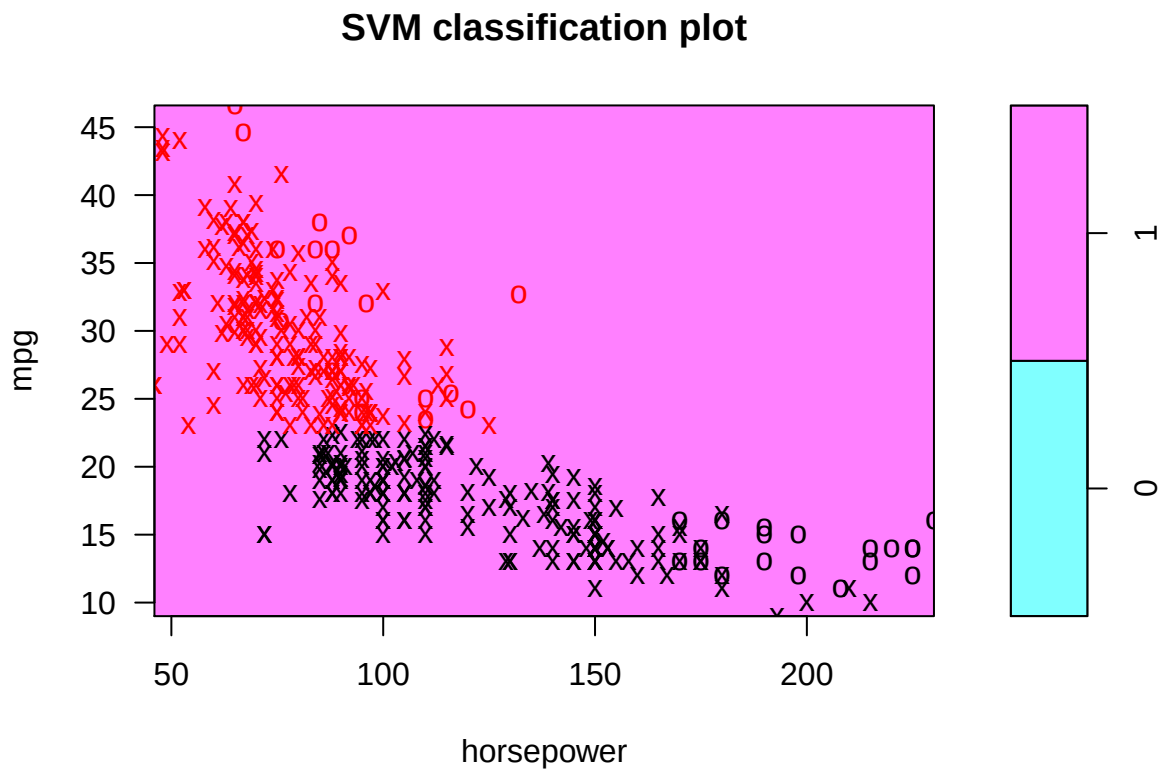

SVM classification plot
mpg
year
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
1
0

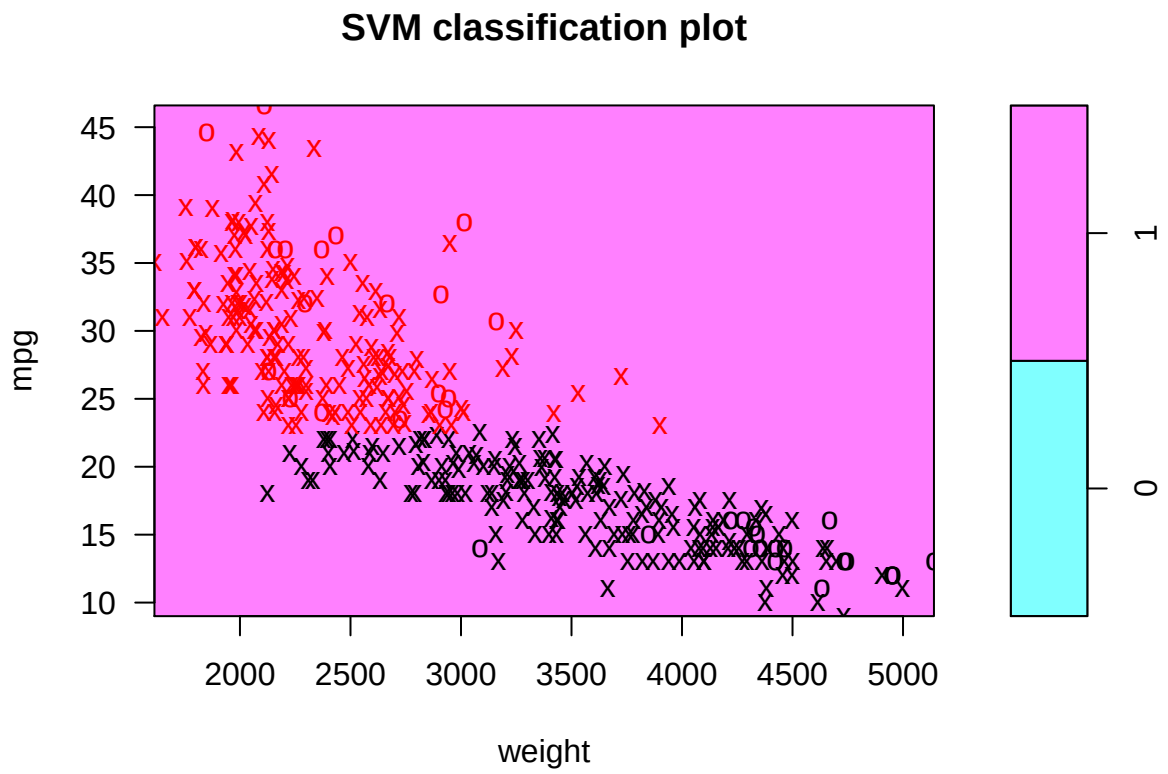


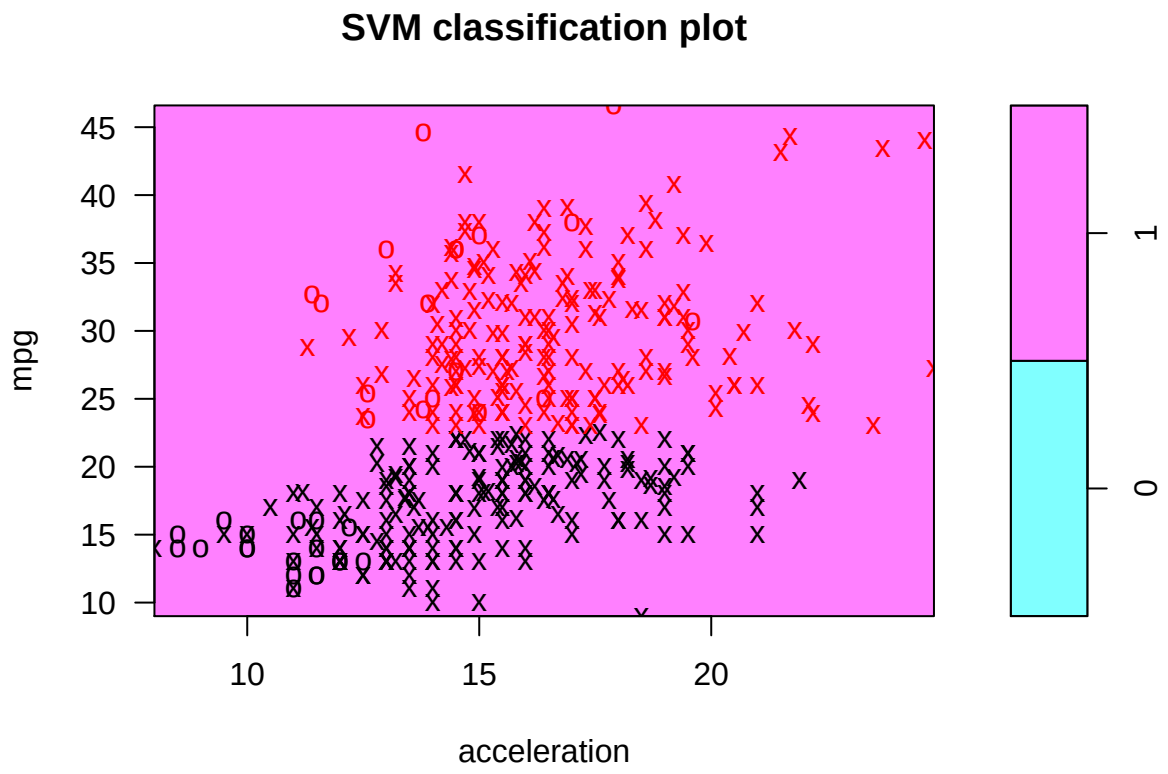

```
for (name in names(Auto)[!(names(Auto) %in% c("mpg", "mpglevel", "name"))])
{
plot(svm.poly, Auto, as.formula(paste("mpg~", name, sep="")))
}
```


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

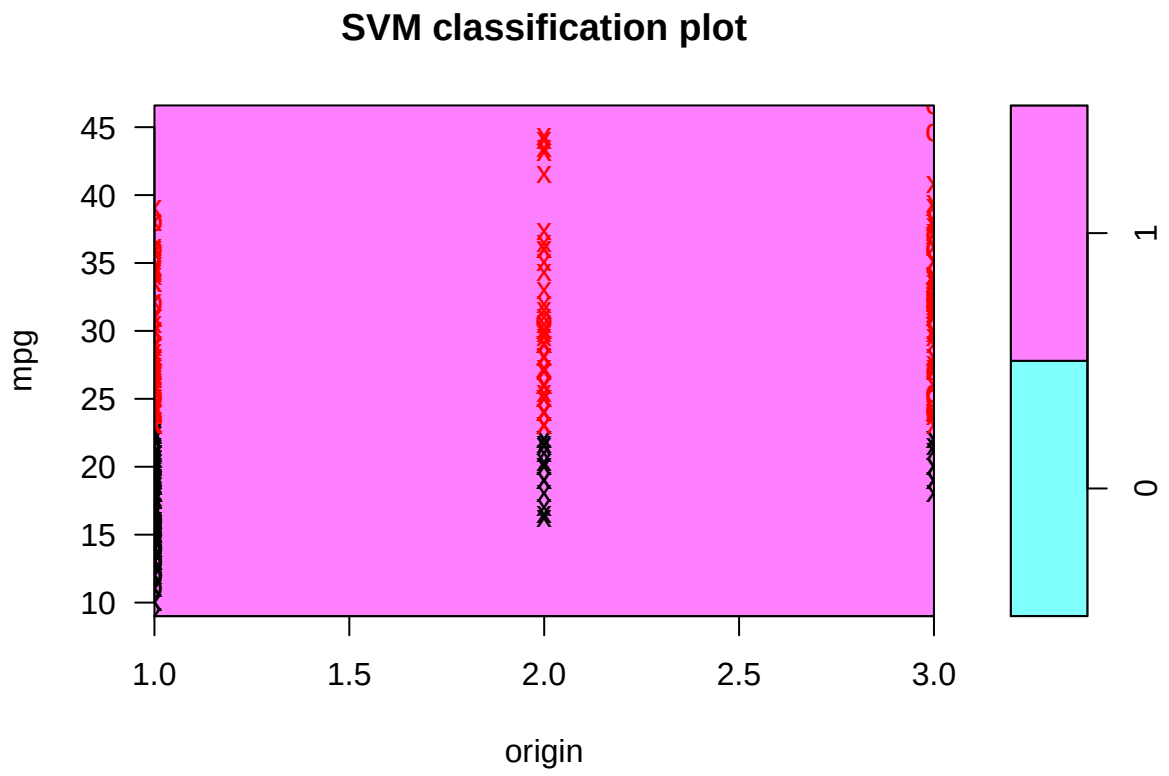

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0


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

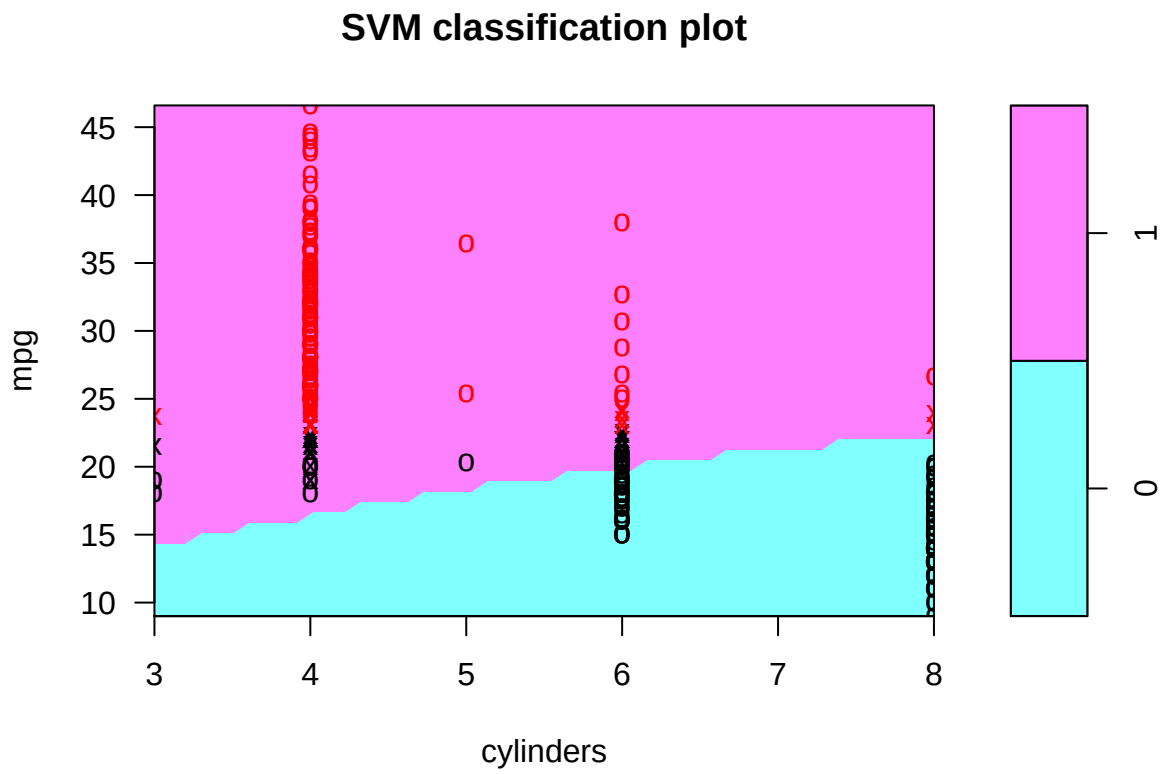

SVM classification plot
mpg
weight
45
40
35
30
25
20
15
10
2000
2500
3000
3500
4000
4500
5000
1
0

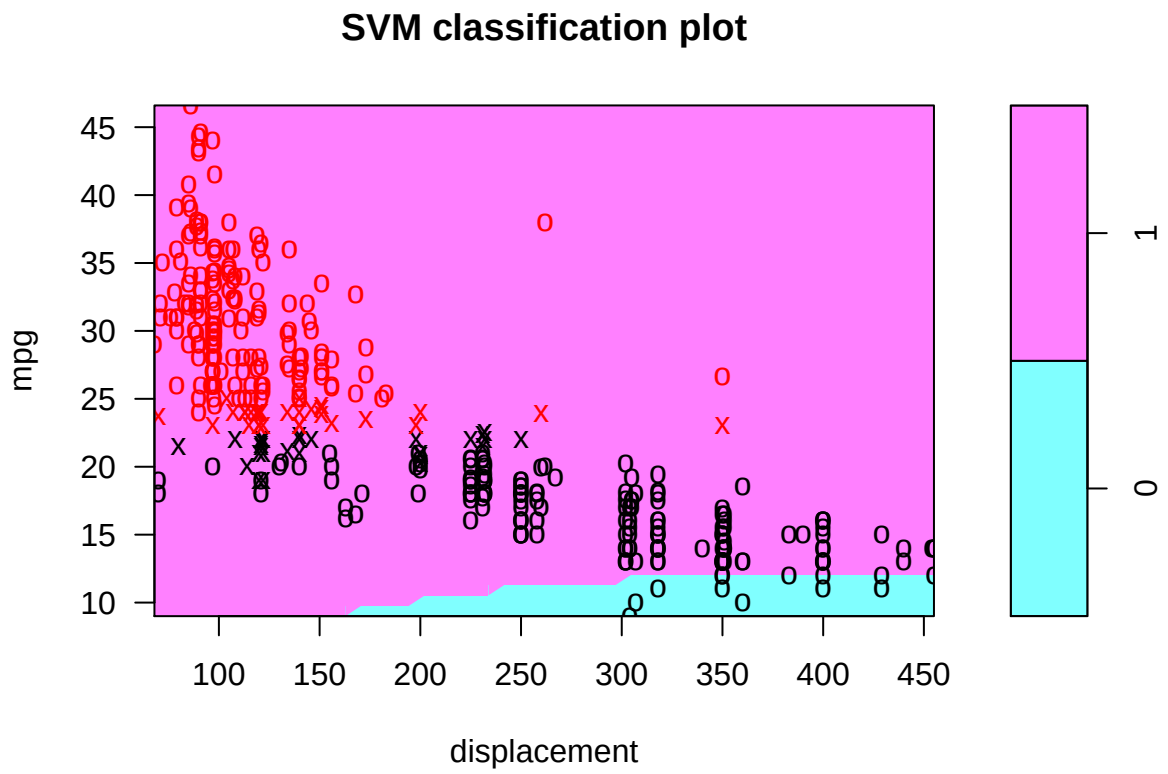

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0

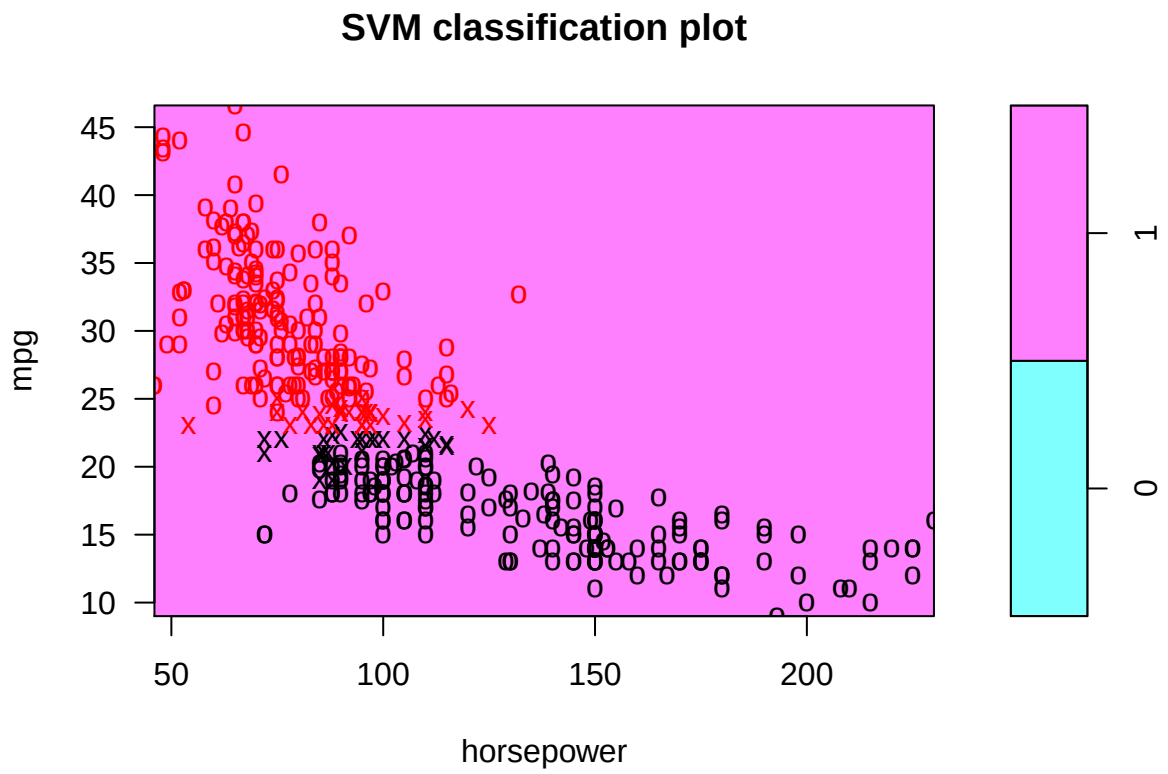

SVM classification plot
mpg
year
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
1
0

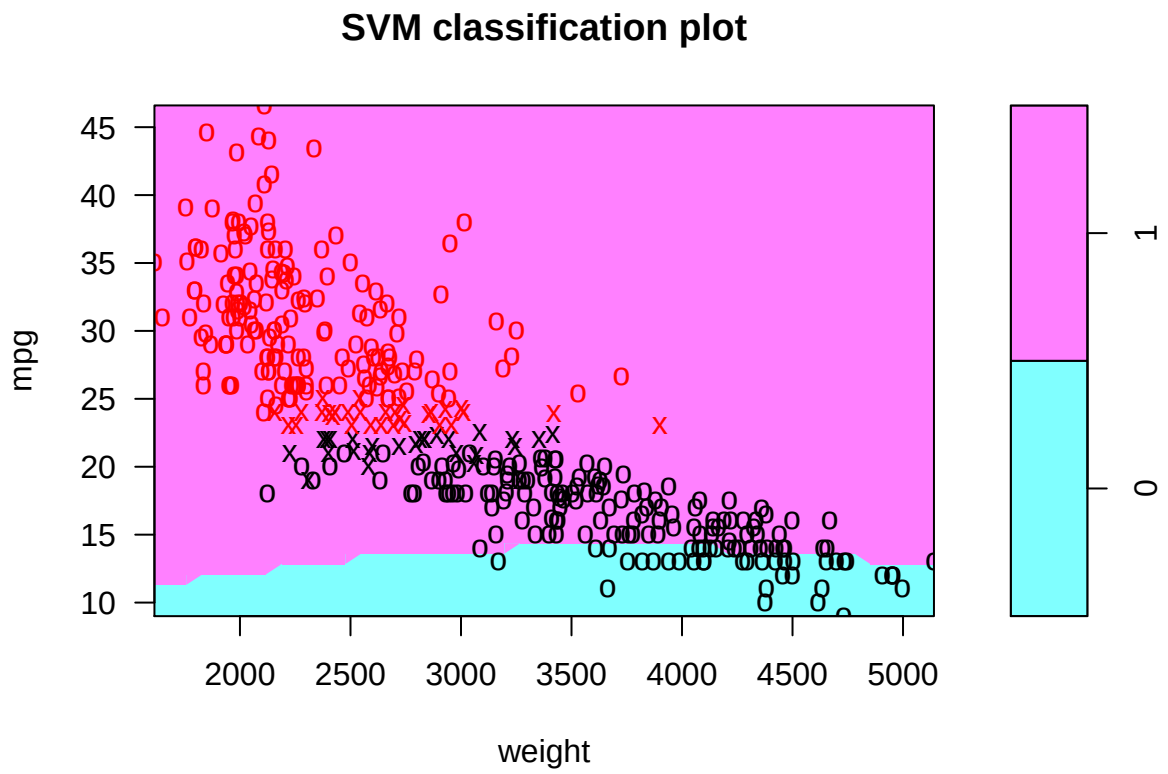


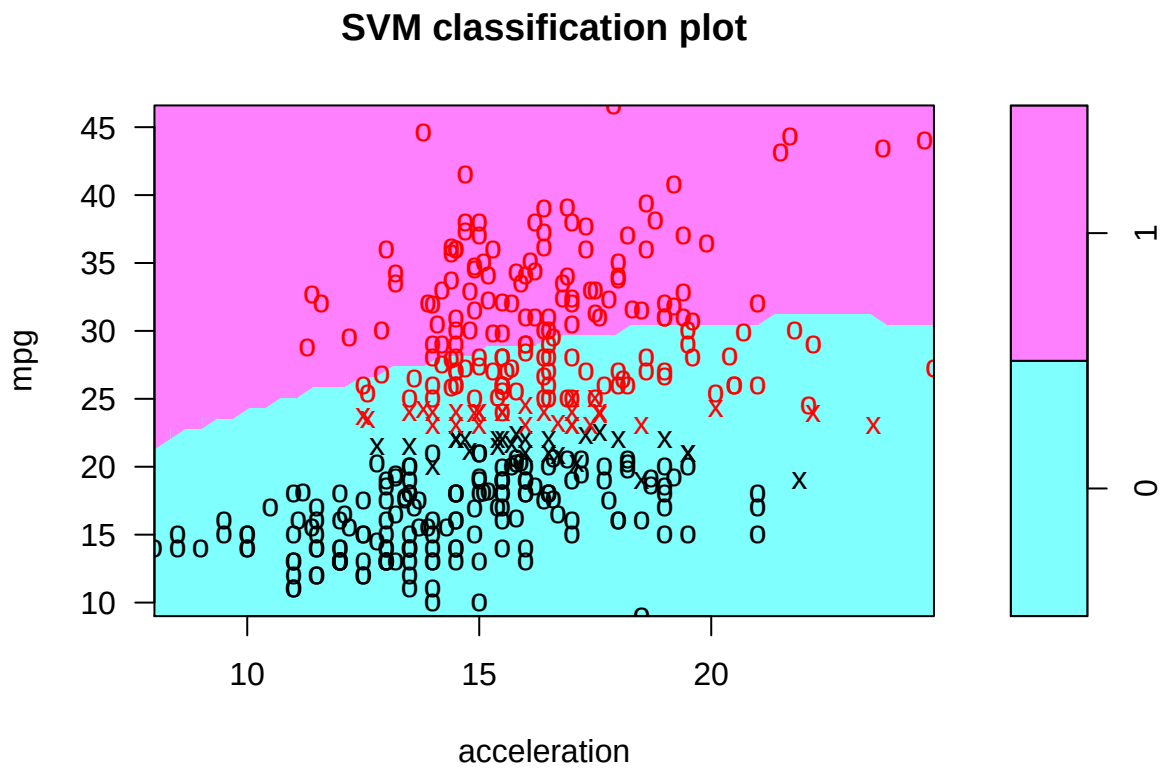

```
for (name in names(Auto)[!(names(Auto) %in% c("mpg", "mpglevel", "name"))])
{
plot(svm.radial, Auto, as.formula(paste("mpg~", name, sep="")))
}
```

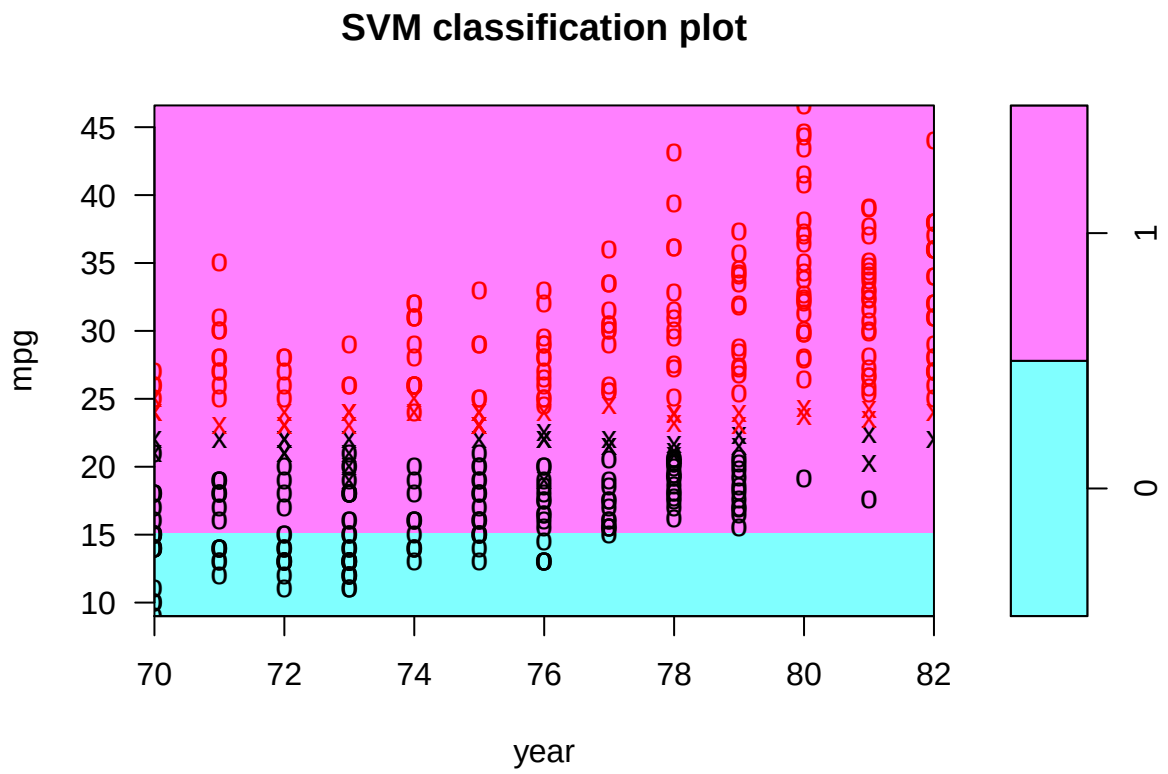

SVM classification plot
mpg
45
40
35
30
25
20
15
10
3
4
5
6
7
8
cylinders
1
0

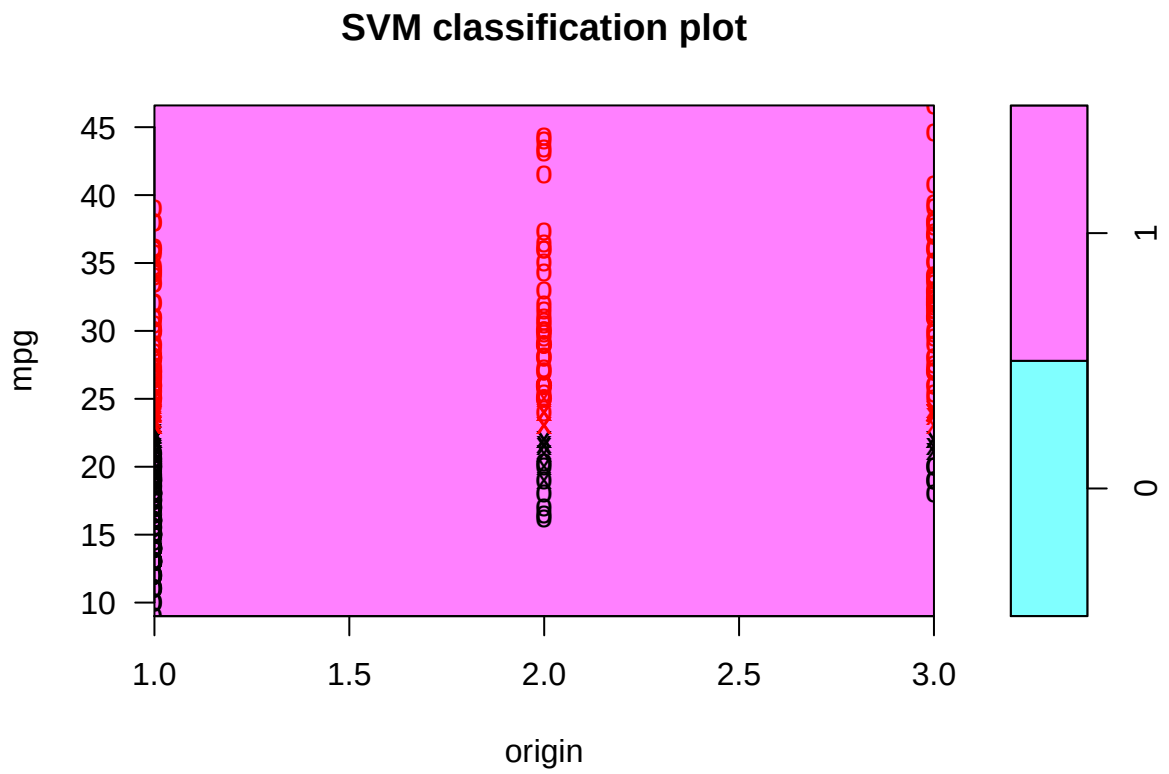

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0


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


SVM classification plot
mpg
45
40
35
30
25
20
15
10
2000
2500
3000
3500
4000
4500
5000
weight
1
0


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

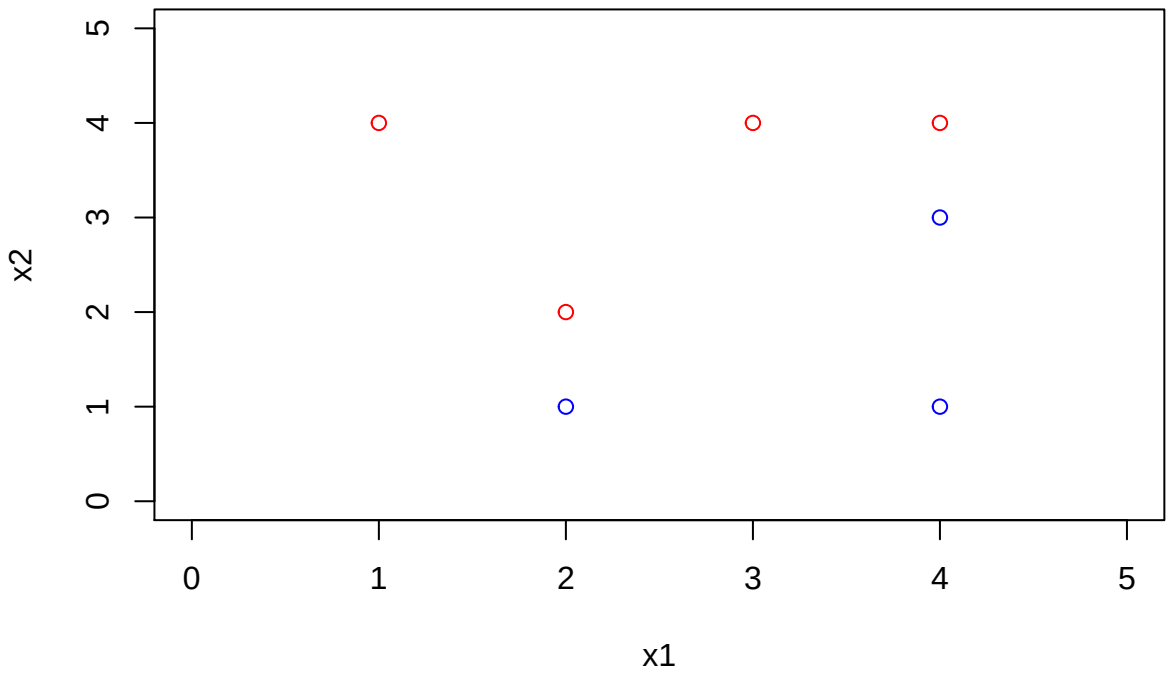

SVM classification plot
mpg
year
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
1
0



5. SVM

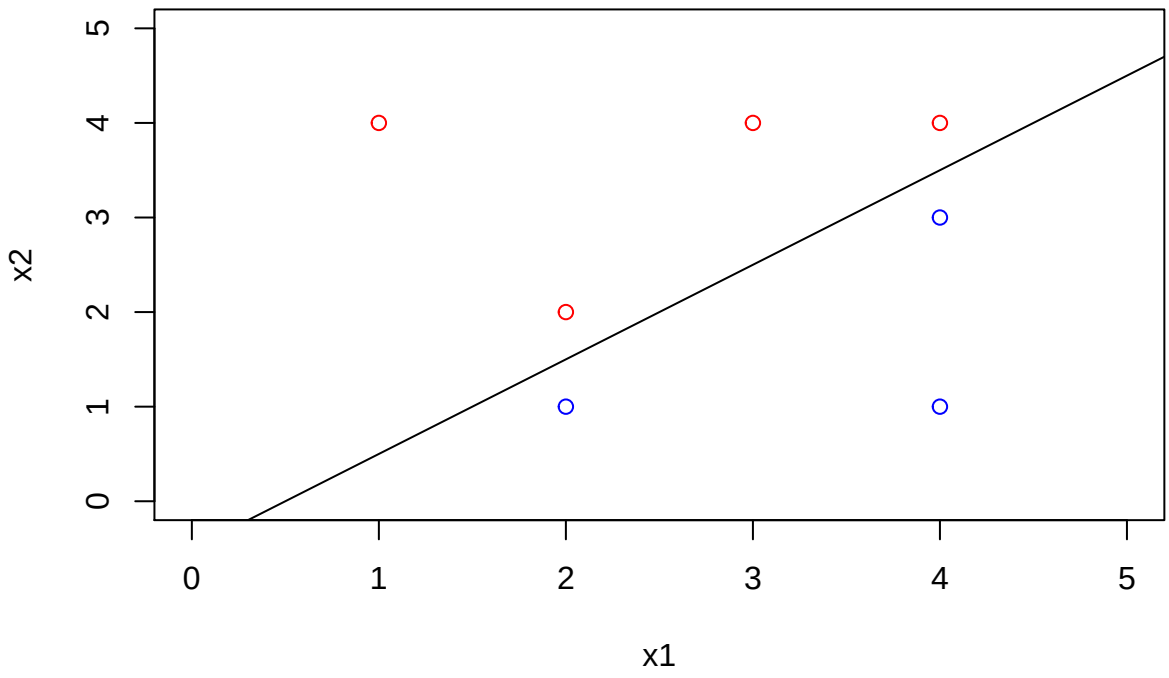
(a) Sketch the observations.

```
x1 = c(3, 2, 4, 1, 2, 4, 4)
x2 = c(4, 2, 4, 4, 1, 3, 1)
colors = c("red", "red", "red", "red", "blue", "blue", "blue")
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
```



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

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
```



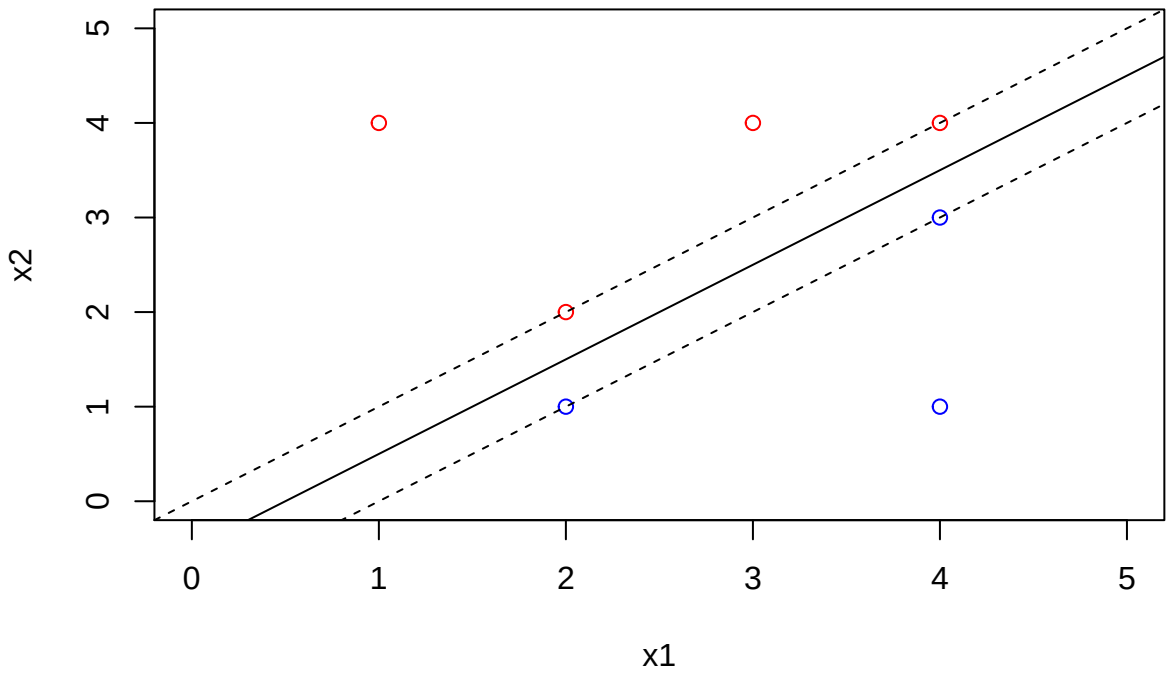
The optimal separating hyperplane is between observations (2,1) (2,2) and observations (4,3) (4,4). The line passes through the points (2,1.5) and (4,3.5) with intercept -0.5 and slope 1. The equation for this hyperplane is -0.5 + X1 - X2 = 0.

(c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if beta0+beta1X1+beta2X2>0, and classify to Blue otherwise." Provide the values for beta0, beta1 and beta2.

The classification rule is "Classify to Red if -0.5 + X1 - X2 $<$ 0, and classify to Blue otherwise."

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

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
abline(-1, 1, lty = 2)
abline(0, 1, lty = 2)
```



The margin is 0.25

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

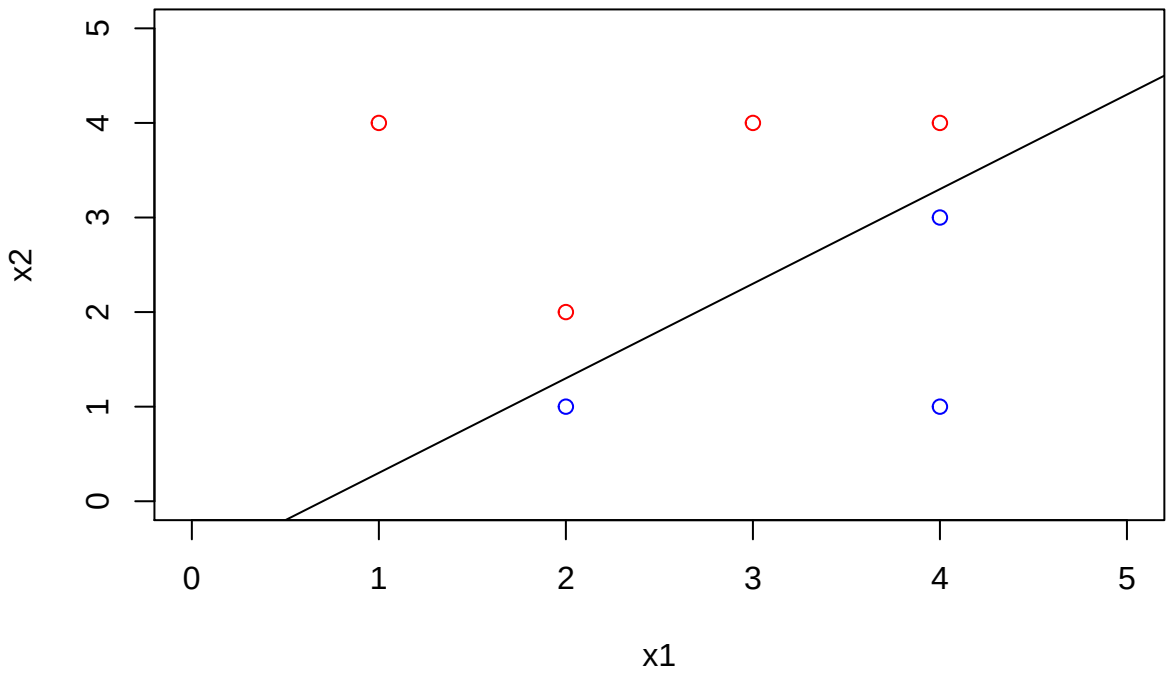
The support vectors for the maximalmargin classifier are the observations (2,1) (2,2) (4,3) (4,4)

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

Moving even slightly the seventh observation (4,1) would not affect the maximal margin hyperplane because it is not a support vector.

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

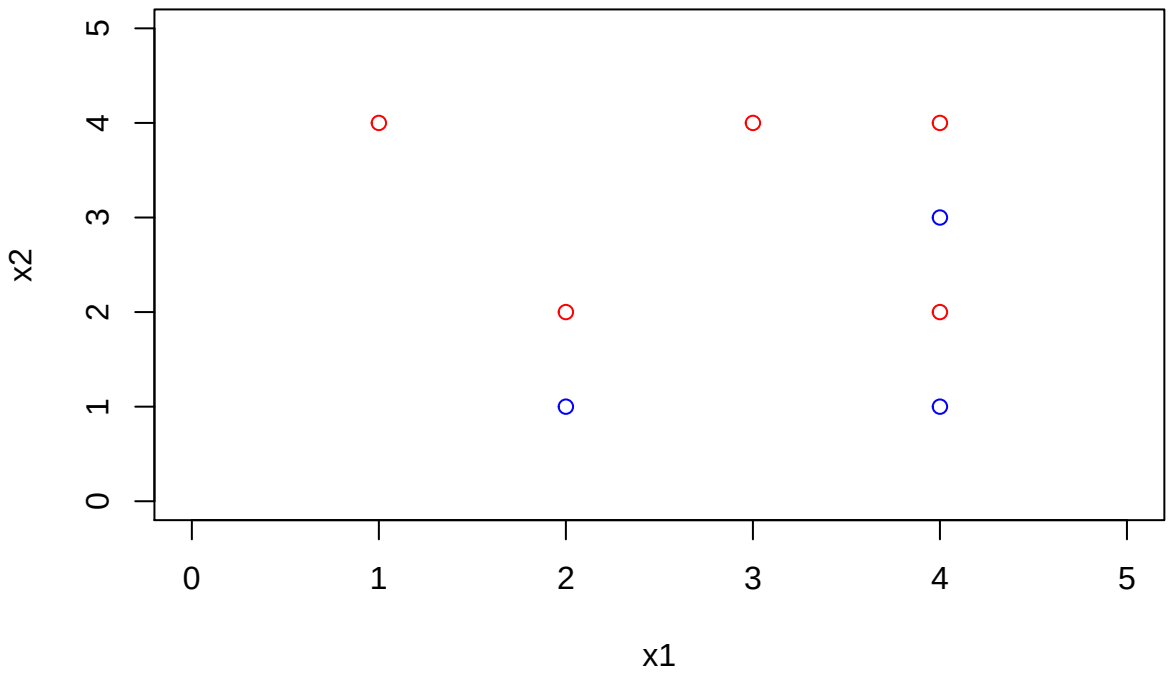
```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.7, 1)
```



The hyperplane which equation is -0.7 + X1 - X2 = 0 is not the optimal separating hyperplane.

(h) Draw an additional observation on the plot so that the two classes are not longer separable by a hyperplane.

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
points(c(4), c(2), col = c("red"))
```

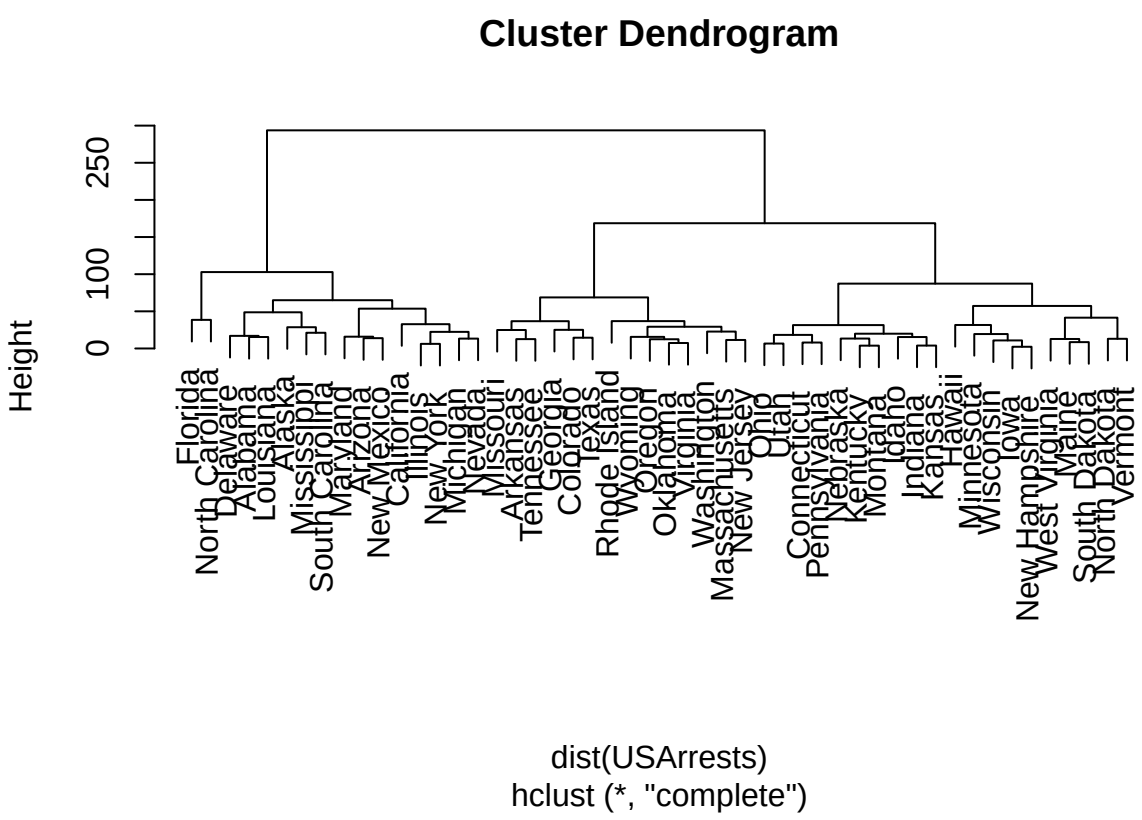


The point (4,2) is added to the plot. The plot shows that the two classes are no longer separable by an hyperplane.

6. Hierarchical clustering

(a) Using hierarchical clustering with complete linkage and Euclidian distance, cluster the states.

```
set.seed(1)
hc.complete <- hclust(dist(USArrests), method = "complete")
plot(hc.complete)
```



(b) Cut the dendrogram at a height that results in three distinct clusters. Which states belong to the clusters?

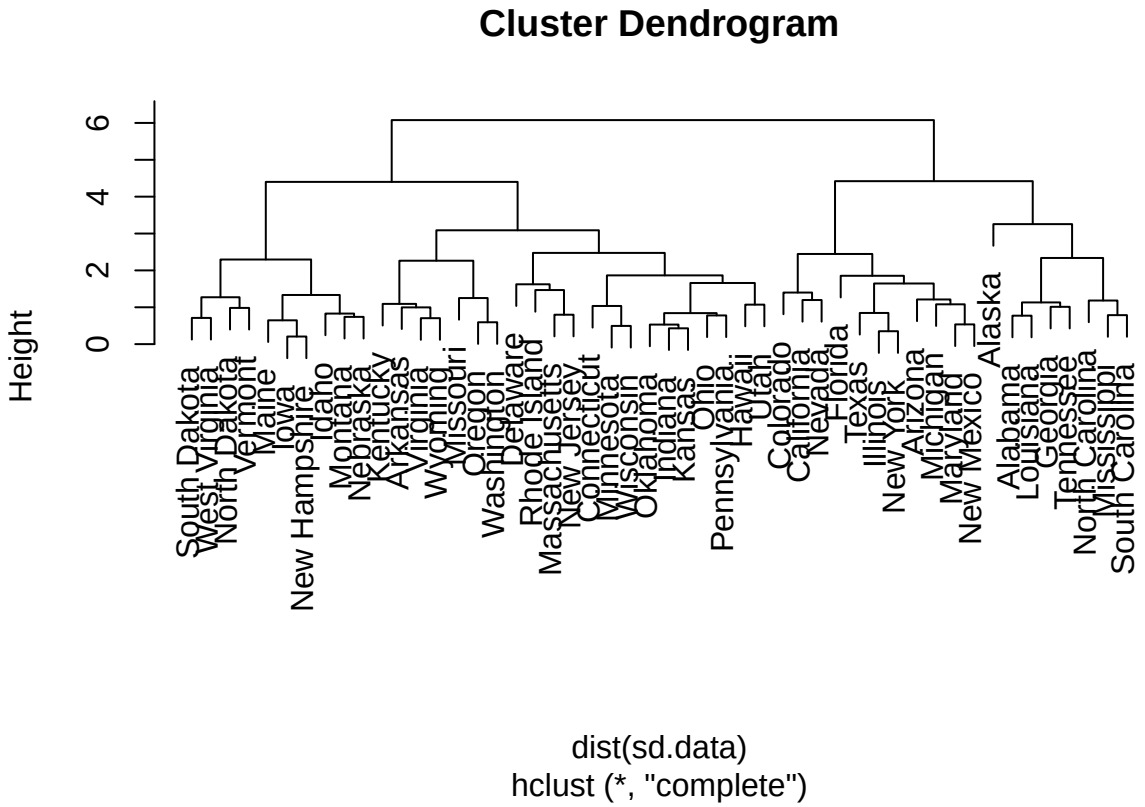
```
cutree(hc.complete, 3)
```

```
##        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
```

(c) Hierachically cluster the states using complete linkage and Euclidean distance, after scaling the variables to have standard deviation one.

```
sd.data <- scale(USArrests)
hc.complete.sd <- hclust(dist(sd.data), method = "complete")
plot(hc.complete.sd)
```



(d) 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.

```
table(cutree(hc.complete, 3), cutree(hc.complete.sd, 3))
```

```
##
##      1  2  3
##   1  6  9  1
##   2  2  2 10
##   3  0  0 20
```

Scaling the variables affect the clusters obtained, because the variables have different units. Therefore to get a more accurate result, scaling should be done before clustering.

7. PCA and K-Mean Clustering

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

```
set.seed(2)
x <- matrix(rnorm(20 * 3 * 50, mean = 0, sd = 0.001), ncol = 50)
x[1:20, 1] <- 1
x[21:40, 2] <- 2
x[21:40, 1] <- 2
x[41:60, 2] <- 1
```

(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, the 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.

```
pr.out <- prcomp(x)
summary(pr.out)
```

```
## Importance of components:
##                           PC1    PC2      PC3      PC4      PC5      PC6
## Standard deviation     1.008 0.5822 0.001731 0.001673 0.001648 0.001582
## Proportion of Variance 0.750 0.2500 0.000000 0.000000 0.000000 0.000000
## Cumulative Proportion  0.750 1.0000 0.999970 0.999970 0.999970 0.999970
##                            PC7      PC8      PC9     PC10     PC11
## Standard deviation     0.001543 0.001497 0.001474 0.001411 0.001393
## Proportion of Variance 0.000000 0.000000 0.000000 0.000000 0.000000
## Cumulative Proportion  0.999980 0.999980 0.999980 0.999980 0.999980
##                           PC12     PC13     PC14     PC15     PC16
## Standard deviation     0.001335 0.001297 0.001257 0.001244 0.001226
## Proportion of Variance 0.000000 0.000000 0.000000 0.000000 0.000000
## Cumulative Proportion  0.999980 0.999980 0.999990 0.999990 0.999990
##                           PC17     PC18     PC19     PC20     PC21
## Standard deviation     0.00116 0.001118 0.001091 0.001021 0.001012
## Proportion of Variance 0.00000 0.000000 0.000000 0.000000 0.000000
## Cumulative Proportion  0.99999 0.999990 0.999990 0.999990 0.999990
##                           PC22      PC23      PC24      PC25      PC26
## Standard deviation     0.0009849 0.0009379 0.0009316 0.0009081 0.0008668
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## Cumulative Proportion  0.9999900 0.9999900 0.9999900 0.9999900 1.0000000
##                           PC27     PC28      PC29      PC30      PC31
## Standard deviation     0.0008228 0.000801 0.0007485 0.0007124 0.0006966
```

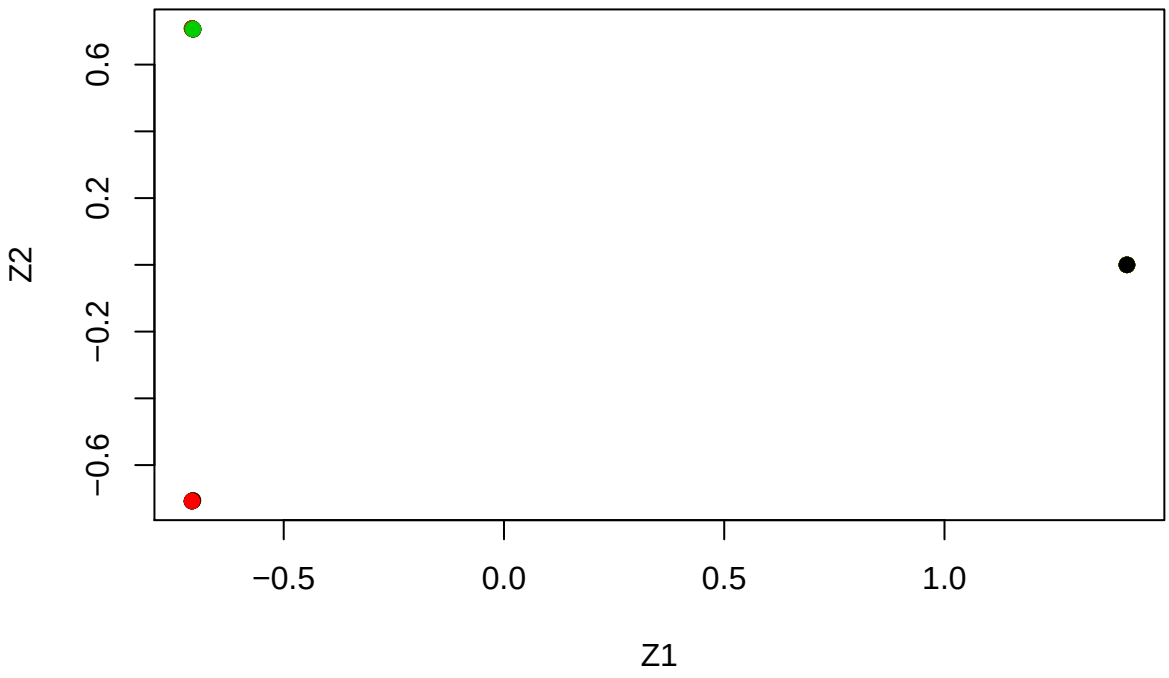
```
## Proportion of Variance 0.0000000 0.000000 0.0000000 0.0000000 0.0000000
## Cumulative Proportion  1.0000000 1.000000 1.0000000 1.0000000 1.0000000
##                            PC32      PC33      PC34      PC35     PC36
## Standard deviation     0.0006732 0.0006323 0.0005909 0.0005654 0.000538
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.0000000 0.000000
## Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.0000000 1.000000
##                            PC37      PC38      PC39      PC40      PC41
## Standard deviation     0.0005325 0.0004756 0.0004475 0.0004261 0.0003913
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##                            PC42      PC43      PC44      PC45      PC46
## Standard deviation     0.0003774 0.0003144 0.0002964 0.0002732 0.0002495
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##                            PC47      PC48      PC49      PC50
## Standard deviation     0.0001915 0.0001466 0.0001289 7.781e-05
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.000e+00
## Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.000e+00
```

```
pr.out$x[,1:2]
```

```
##              PC1           PC2
##  [1,] -0.7084490 -0.7082144522
##  [2,] -0.7057474 -0.7055151322
##  [3,] -0.7076811 -0.7074443389
##  [4,] -0.7070712 -0.7068401340
##  [5,] -0.7068260 -0.7065881662
##  [6,] -0.7077633 -0.7075323762
##  [7,] -0.7085958 -0.7083642381
##  [8,] -0.7075217 -0.7072903431
##  [9,] -0.7071228 -0.7068891886
## [10,] -0.7078169 -0.7075824650
## [11,] -0.7078354 -0.7076019476
## [12,] -0.7069492 -0.7067184128
## [13,] -0.7072834 -0.7070501646
## [14,] -0.7068746 -0.7066429264
## [15,] -0.7072216 -0.7069894025
## [16,] -0.7078236 -0.7075923548
## [17,] -0.7062589 -0.7060279040
## [18,] -0.7066366 -0.7064059371
## [19,] -0.7064387 -0.7062068706
## [20,] -0.7081812 -0.7079458686
## [21,]  1.4142336 -0.0001261091
## [22,]  1.4142331 -0.0001241534
## [23,]  1.4142335 -0.0001283129
## [24,]  1.4142317 -0.0001245673
## [25,]  1.4142315 -0.0001256704
## [26,]  1.4142334 -0.0001250513
## [27,]  1.4142322 -0.0001296945
## [28,]  1.4142312 -0.0001243237
## [29,]  1.4142319 -0.0001281171
## [30,]  1.4142319 -0.0001257620
## [31,]  1.4142330 -0.0001251149
## [32,]  1.4142328 -0.0001258712
## [33,]  1.4142311 -0.0001253563
```

```
## [34,]  1.4142330 -0.0001252763
## [35,]  1.4142334 -0.0001281810
## [36,]  1.4142319 -0.0001213042
## [37,]  1.4142350 -0.0001280683
## [38,]  1.4142323 -0.0001244199
## [39,]  1.4142329 -0.0001226637
## [40,]  1.4142317 -0.0001247165
## [41,] -0.7072659  0.7075355117
## [42,] -0.7083815  0.7086544785
## [43,] -0.7075922  0.7078599683
## [44,] -0.7056475  0.7059183608
## [45,] -0.7065550  0.7068238660
## [46,] -0.7055892  0.7058580687
## [47,] -0.7072120  0.7074803573
## [48,] -0.7070589  0.7073305340
## [49,] -0.7071242  0.7073944056
## [50,] -0.7078417  0.7081120183
## [51,] -0.7075847  0.7078554909
## [52,] -0.7055355  0.7058028214
## [53,] -0.7073920  0.7076629485
## [54,] -0.7060945  0.7063642425
## [55,] -0.7077345  0.7080056488
## [56,] -0.7083842  0.7086545881
## [57,] -0.7072243  0.7074930429
## [58,] -0.7063334  0.7066033935
## [59,] -0.7061885  0.7064602627
## [60,] -0.7058133  0.7060853488
```

```r
plot(pr.out$x[, 1:2], col = 1:3, xlab = "Z1", ylab = "Z2", pch = 19)
```



(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 ?

```
km.out = kmeans(x, 3, nstart=20)
table(c(rep(1, 20), rep(2, 20), rep(3, 20)), km.out$cluster)
```

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

The clusters are identicals to the true labels.

(d) Perform K-means clustering with K=2. Describe your results.

```
km.out <- kmeans(x, 2, nstart = 20)
table(c(rep(1,20), rep(2,20), rep(3,20)), km.out$cluster)
```

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

```
##   3 20  0
```

All observations of one of the clusters are now observed in another cluster.

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

```
km.out <- kmeans(x, 4, nstart = 20)
table(c(rep(1,20), rep(2,20), rep(3,20)), km.out$cluster)
```

```
##
##      1  2  3  4
##   1  0  0 20  0
##   2  0 20  0  0
##   3 12  0  0  8
```

The first cluster is now splitted into two clusters.

(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 60x2 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.

```
km.out <- kmeans(pr.out$x[, 1:2], 3, nstart = 20)
table(c(rep(1,20), rep(2,20), rep(3,20)), km.out$cluster)
```

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

The clusters are identicals to the true labels.

(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.

```
km.out <- kmeans(scale(x), 3, nstart = 20)
table(c(rep(1,20), rep(2,20), rep(3,20)), km.out$cluster)
```

```
##
##      1  2  3
##   1  8  2 10
##   2  0 19  1
##   3 11  1  8
```

The results are poorer than with unscaled data as scaling affects the distance between observations.