Bruce Campbell ST-617 Homework 5

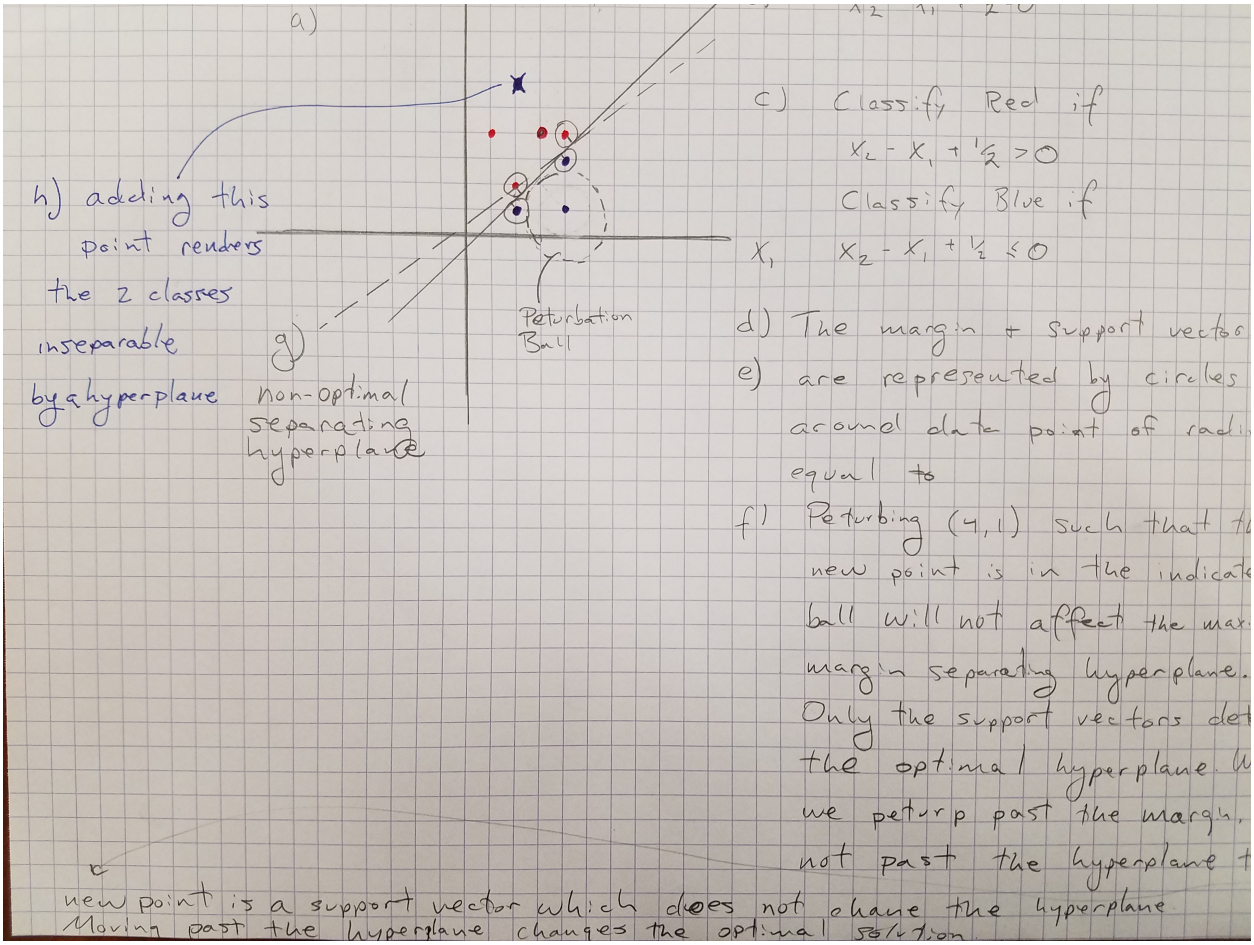
Wed Jul 27 20:23:57 2016

```
rm(list = ls())
set.seed(7)
```

Chapter 9

Problem 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.
(b) Sketch the optimal separating hyperplane, and provide the equation
for this hyperplane (of the form (9.1)).
(c) Describe the classification rule for the maximal margin classifier.
It should be something along the lines of "Classify to Red if
$\beta_0 +\beta_1 X_1 +\beta_2 X_2 > 0$, and classify to Blue otherwise." Provide
the values for $\beta_0, \beta_1, and \beta_2$.
(d) On your sketch, indicate the margin for the maximal margin
hyperplane.
(e) Indicate the support vectors for the maximal margin classifier.
(f) Argue that a slight movement of the seventh observation would
not affect the maximal margin hyperplane.
(g) Sketch a hyperplane that is not the optimal separating hyperplane,
and provide the equation for this hyperplane.
(h) Draw an additional observation on the plot so
```



Chapter 9

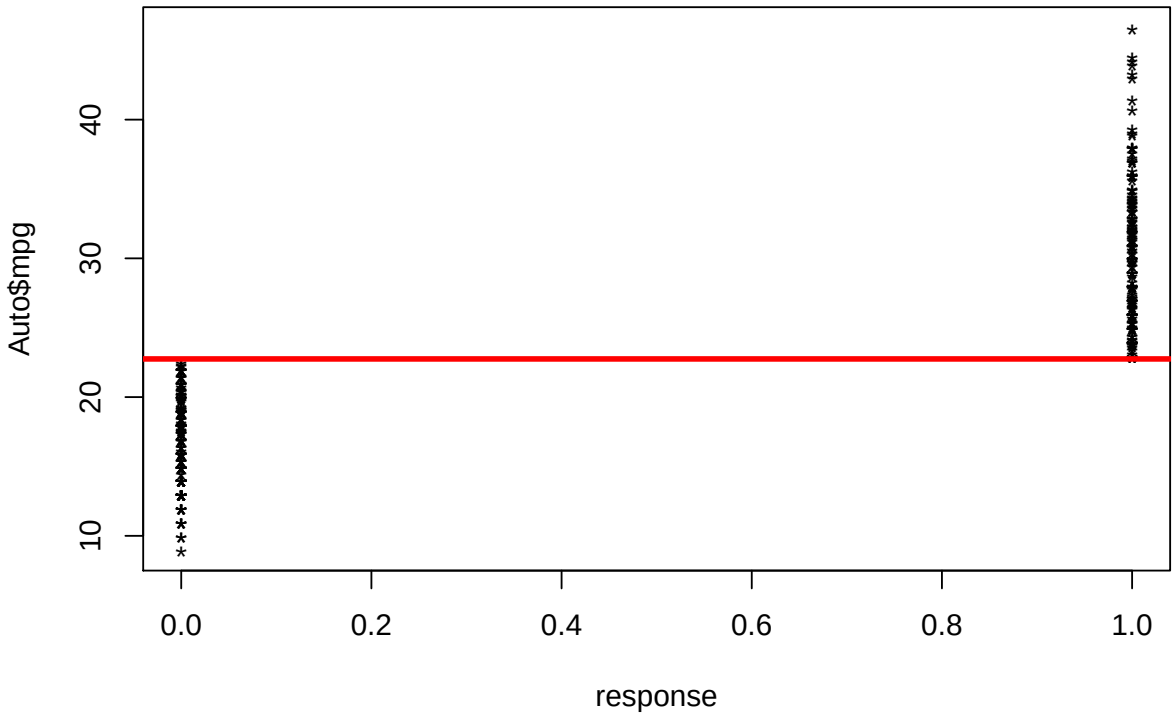
Problem 7

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.

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)
attach(Auto)

median_mpg <- median(Auto$mpg)
response <- matrix(data = 1, nrow = nrow(Auto), ncol = 1)
response[Auto$mpg < median_mpg] <- 0

plot(response, Auto$mpg, pch = "*")
abline(h = median_mpg, col = "red", lwd = 3)
```




```
DF <- Auto
DF$mpg <- NULL  #We don't want to use this as a predictor since our response is derived from it.
DF$name <- NULL
DF <- data.frame(scale(DF))  #We might run into convergence issues with the sum without standardising t
DF$response <- as.factor(response)
# In order for e1071 to train a classifier we need to encode the response as
# a factor.
```

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.

```
# We do a logistic classification to help us decide which variables we may
# want to consider looking at in the sum plot.
glm.fit <- glm(response ~ ., data = DF, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = response ~ ., family = binomial, data = DF)
##
## Deviance Residuals:
```

```
##     Min       1Q   Median       3Q      Max
## -2.4277  -0.1061   0.0080   0.2123   3.1631
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.13983    0.31057  -3.670 0.000242 ***
## cylinders    -0.27734    0.72188  -0.384 0.700835
## displacement  0.21923    1.25926   0.174 0.861789
## horsepower   -1.57886    0.91887  -1.718 0.085750 .
## weight       -3.66559    0.96860  -3.784 0.000154 ***
## acceleration  0.04432    0.39027   0.114 0.909582
## year          1.58202    0.27711   5.709 1.14e-08 ***
## origin        0.38451    0.29161   1.319 0.187314
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 543.43  on 391  degrees of freedom
## Residual deviance: 157.54  on 384  degrees of freedom
## AIC: 173.54
##
## Number of Fisher Scoring iterations: 8
```

```
library(e1071)
svmfit = svm(response ~ ., data = DF, kernel = "linear", cost = 100, scale = FALSE)
summary(svmfit)
```

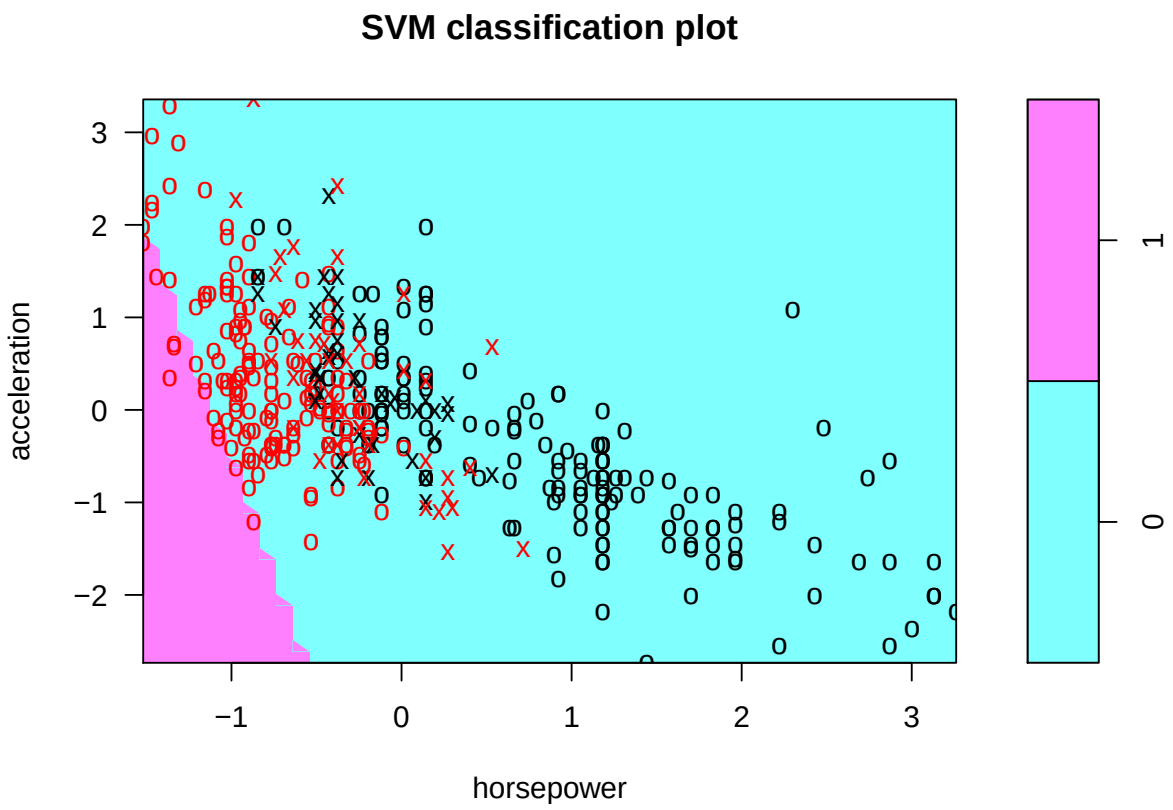
```
##
## Call:
## svm(formula = response ~ ., data = DF, kernel = "linear", cost = 100,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  100
##       gamma:  0.1428571
##
## Number of Support Vectors:  81
##
##  ( 40 41 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
svmfit = svm(response ~ ., data = DF, kernel = "linear", cost = 10, scale = FALSE)
names(DF)
```

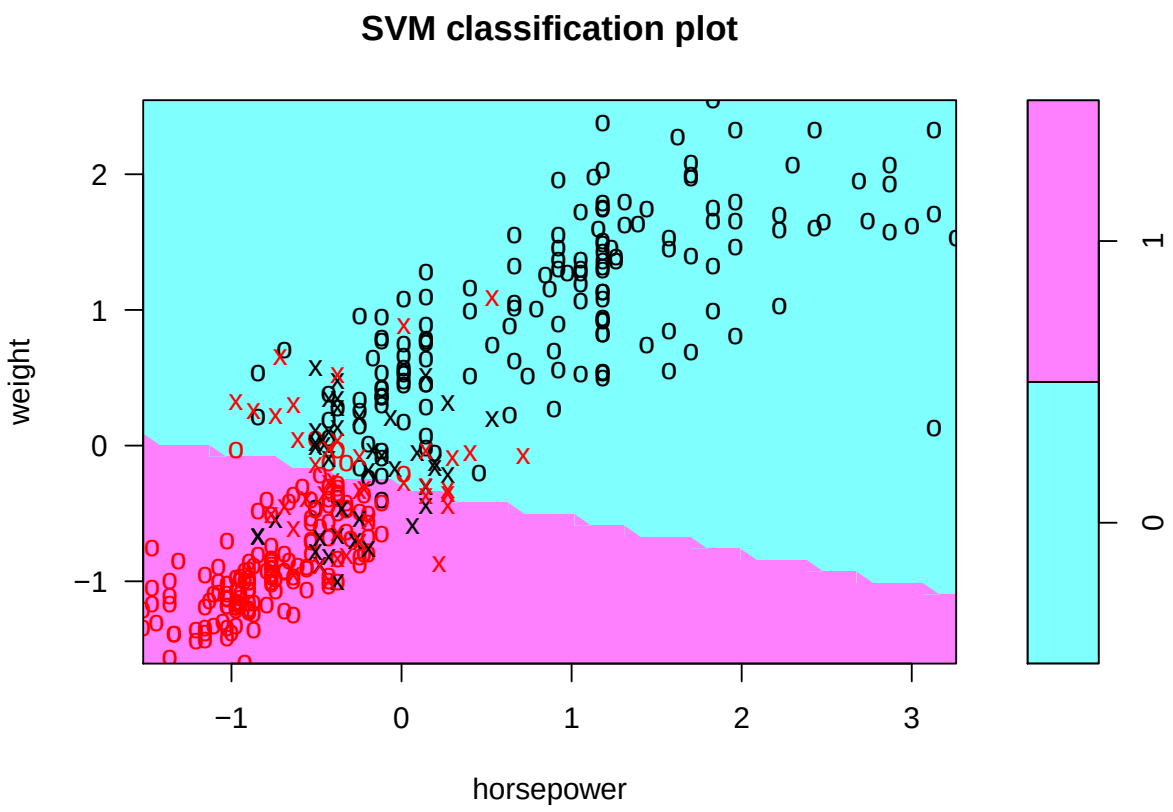
```
## [1] "cylinders"    "displacement" "horsepower"   "weight"
```

```
## [5] "acceleration" "year"         "origin"       "response"
```

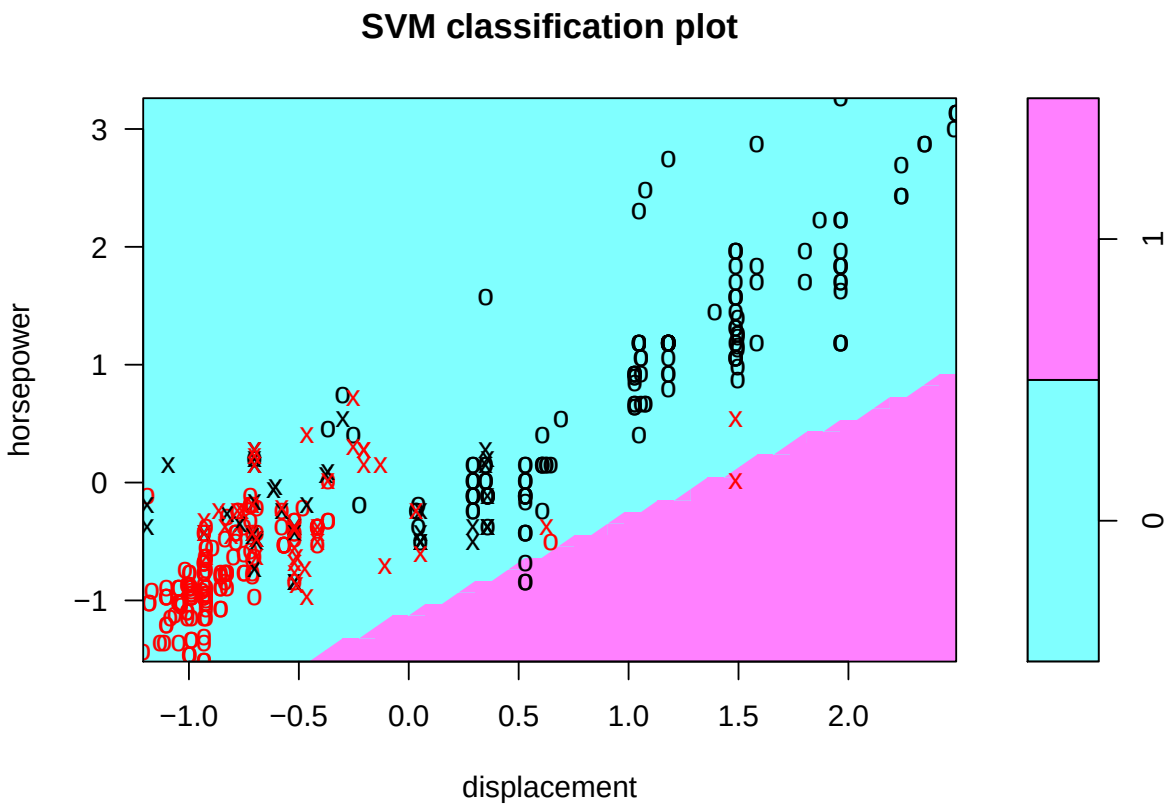
```
plot(svmfit, DF, acceleration ~ horsepower)
```




```
plot(svmfit, DF, weight ~ horsepower)
```




```
plot(svmfit, DF, horsepower ~ displacement)
```



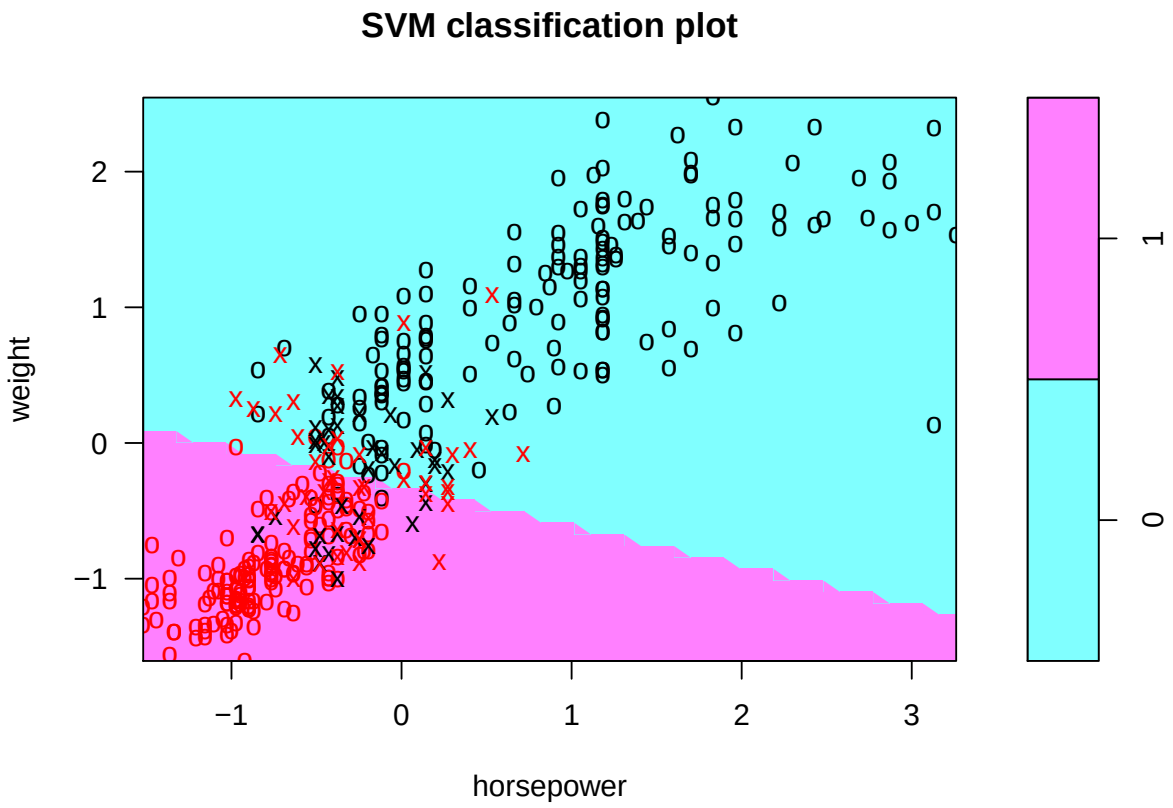

```
tune.svm = tune(svm, response ~ ., data = DF, kernel = "linear", ranges = list(cost = c(0.001,
    0.01, 0.1, 1, 5, 10, 100)))

summary(tune.svm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.08641026
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.14544872 0.07048980
## 2 1e-02 0.09160256 0.04454547
## 3 1e-01 0.09673077 0.04560697
## 4 1e+00 0.08647436 0.04103727
## 5 5e+00 0.08641026 0.04254423
## 6 1e+01 0.08641026 0.04254423
## 7 1e+02 0.09153846 0.04594544
```

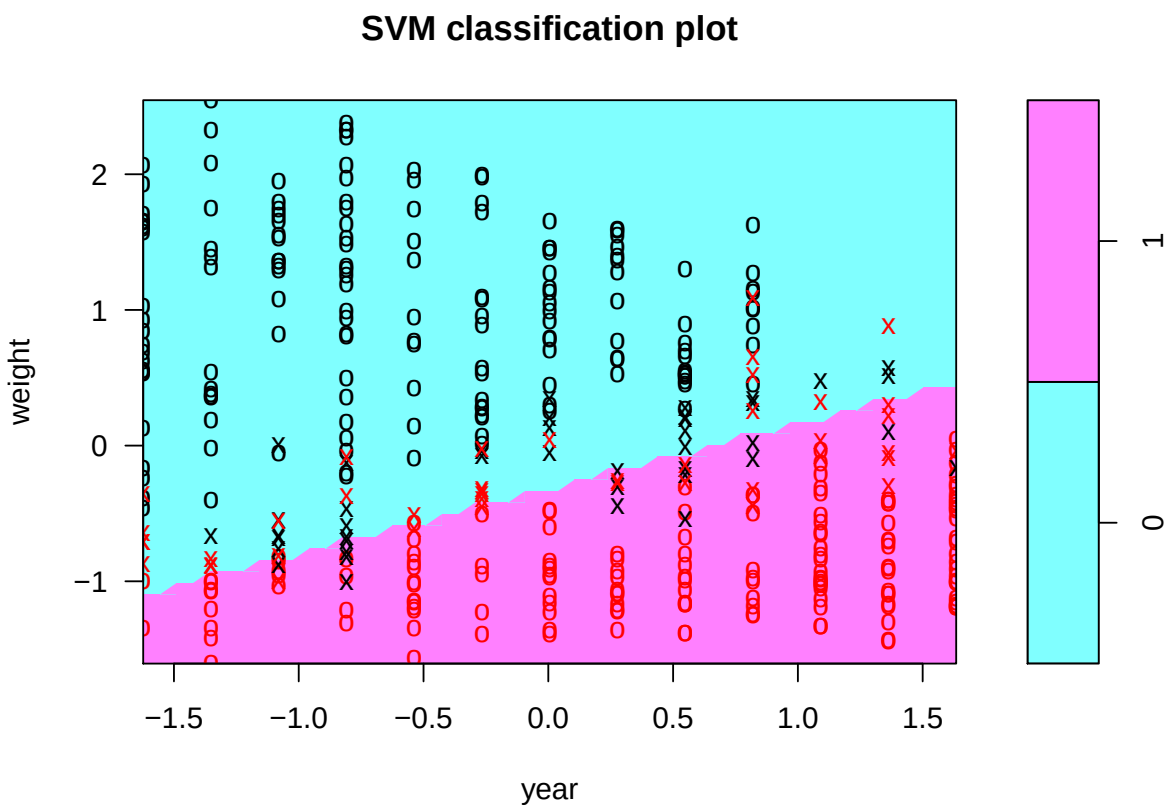
```
svmfit = svm(response ~ ., data = DF, kernel = "linear", cost = 5, scale = FALSE)
svm.pred <- predict(svmfit, DF)
TB <- table(svm.pred, DF$response)
library(pander)
pander(TB)
```

	0	1
0	174	11
1	22	185

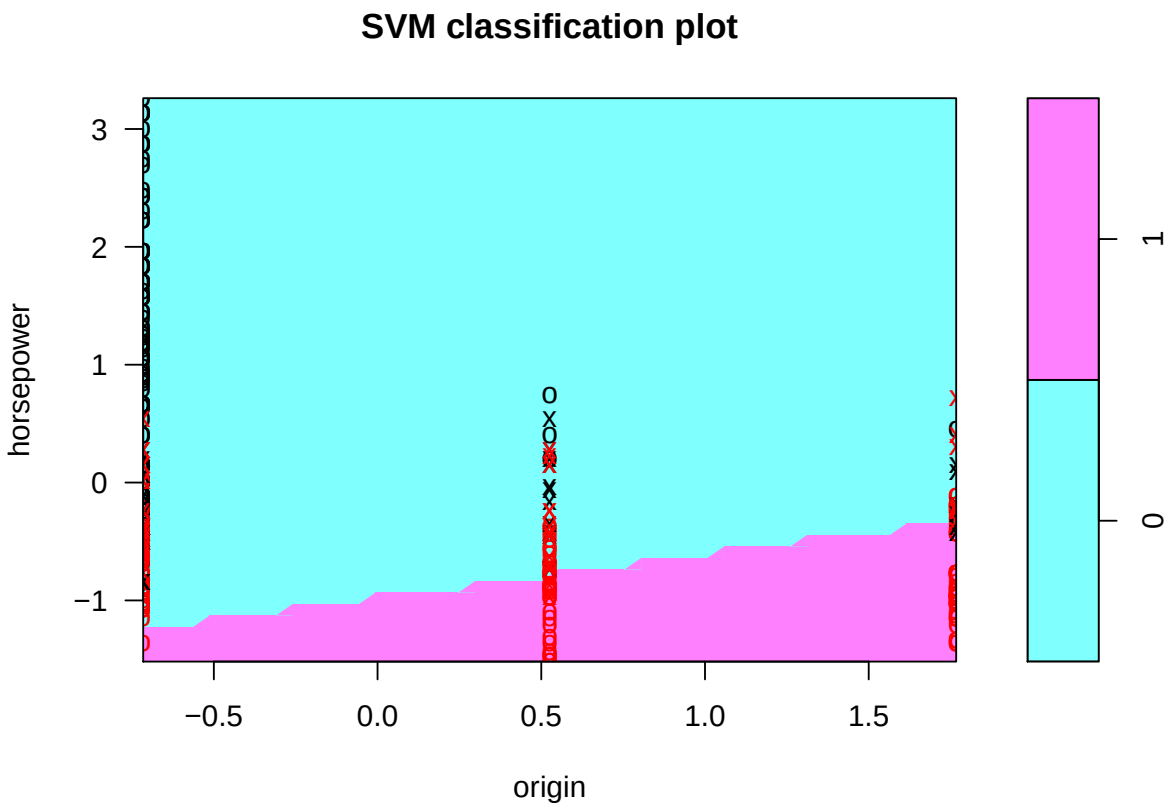
```
ACC_Linear = (TB[1] + TB[4])/length(DF$response)

plot(svmfit, DF, weight ~ horsepower)
```



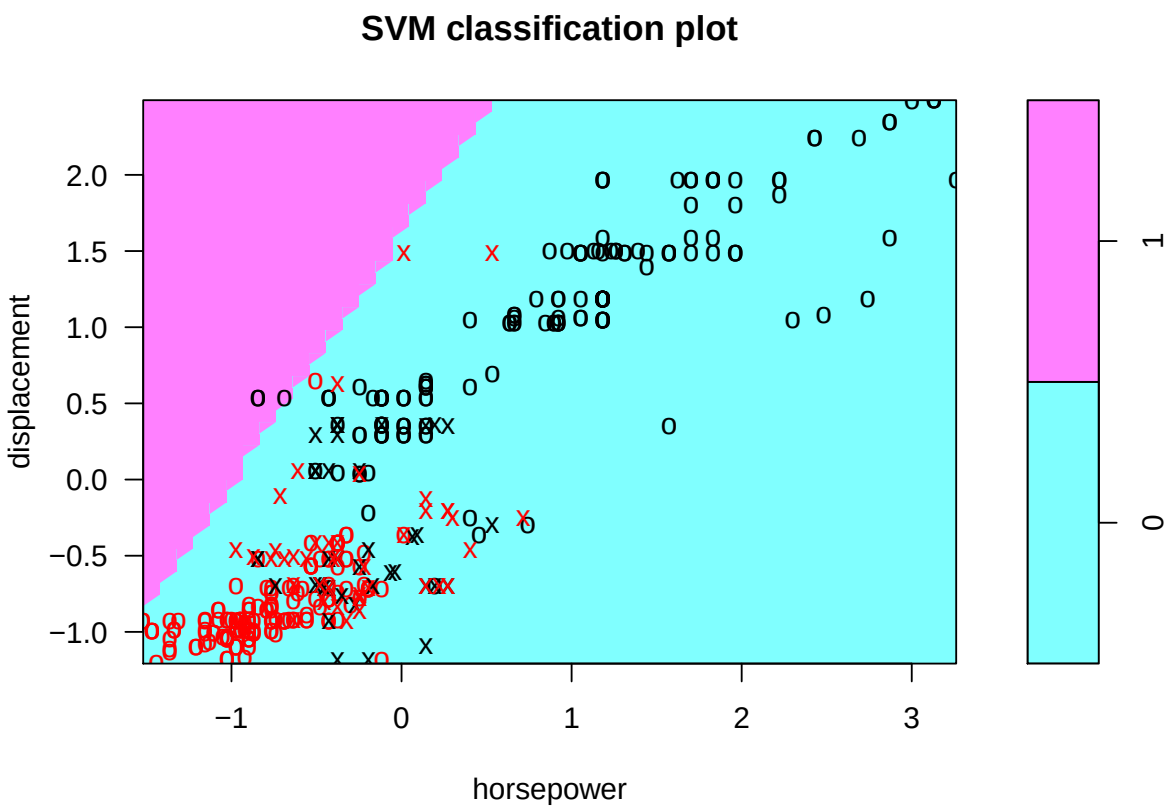

```
plot(svmfit, DF, weight ~ year)
```



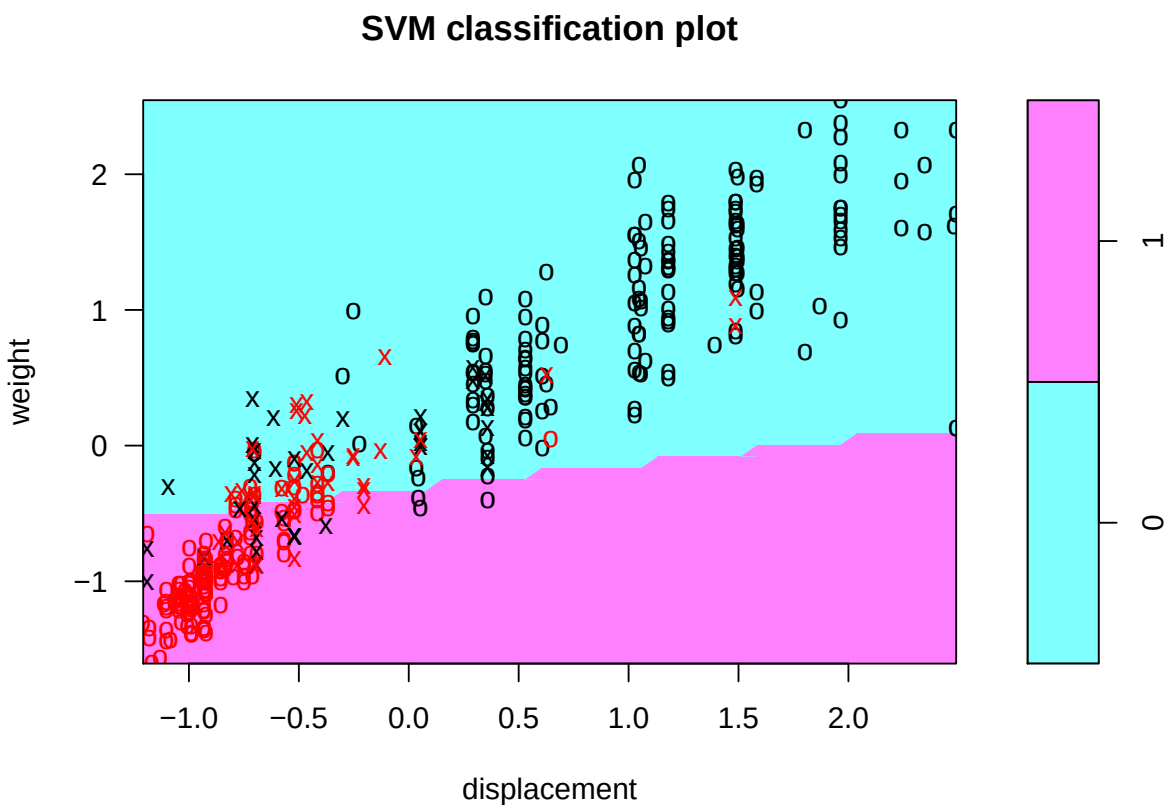

```
plot(svmfit, DF, horsepower ~ origin)
```




```
# Now looking at some pairs not suggested by logistic regression
plot(svmfit, DF, displacement ~ horsepower)
```




```
plot(svmfit, DF, weight ~ displacement)
```



In the plots we see the need to consider kernels of non linear similarity functions. The optimal cost reported by the cross validation routine is 5. The accuracy of the model with optimal cost chosen by cross validation is 0.9158163

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.

```
gamma_default <- 1/(ncol(DF) - 1)

gamma_list <- ((1:20)/10) * gamma_default
tune.svm = tune(svm, response ~ ., data = DF, kernel = "radial", ranges = list(gamma = gamma_list))
summary(tune.svm)
```

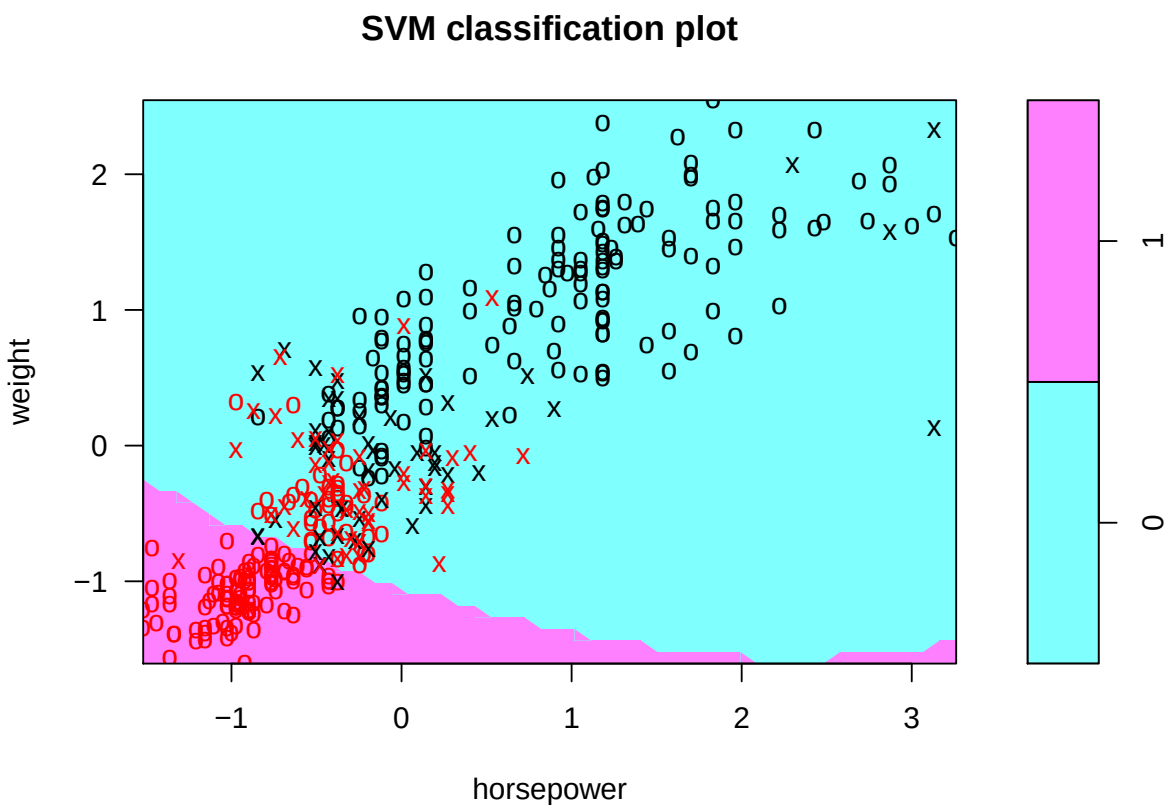
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma
##    0.1
##
```

```
## - best performance: 0.08903846
##
## - Detailed performance results:
##         gamma      error dispersion
## 1  0.01428571 0.09160256 0.03950213
## 2  0.02857143 0.08910256 0.03603454
## 3  0.04285714 0.09416667 0.03535979
## 4  0.05714286 0.09673077 0.03693115
## 5  0.07142857 0.09673077 0.03273690
## 6  0.08571429 0.09160256 0.03561197
## 7  0.10000000 0.08903846 0.03954371
## 8  0.11428571 0.09160256 0.03760738
## 9  0.12857143 0.09160256 0.03760738
## 10 0.14285714 0.09160256 0.03760738
## 11 0.15714286 0.09416667 0.03535979
## 12 0.17142857 0.09416667 0.03535979
## 13 0.18571429 0.09423077 0.03137462
## 14 0.20000000 0.09679487 0.03313000
## 15 0.21428571 0.09679487 0.03313000
## 16 0.22857143 0.09679487 0.03313000
## 17 0.24285714 0.09679487 0.03313000
## 18 0.25714286 0.09935897 0.03663767
## 19 0.27142857 0.09935897 0.03663767
## 20 0.28571429 0.09423077 0.03960780
```

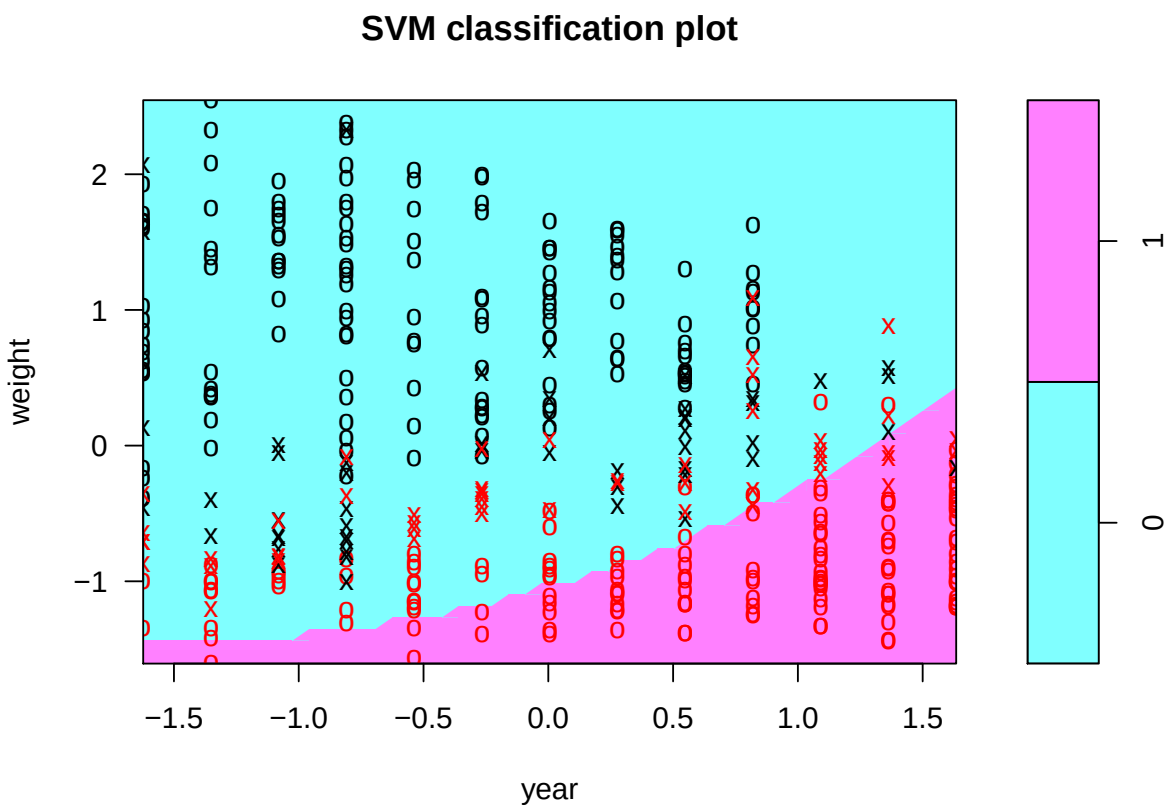
```
svm.fit <- tune.svm$best.model

svm.pred <- predict(svm.fit, DF)
TB <- table(svm.pred, DF$response)
pander(TB)
```

	0	1
0	171	8
1	25	188

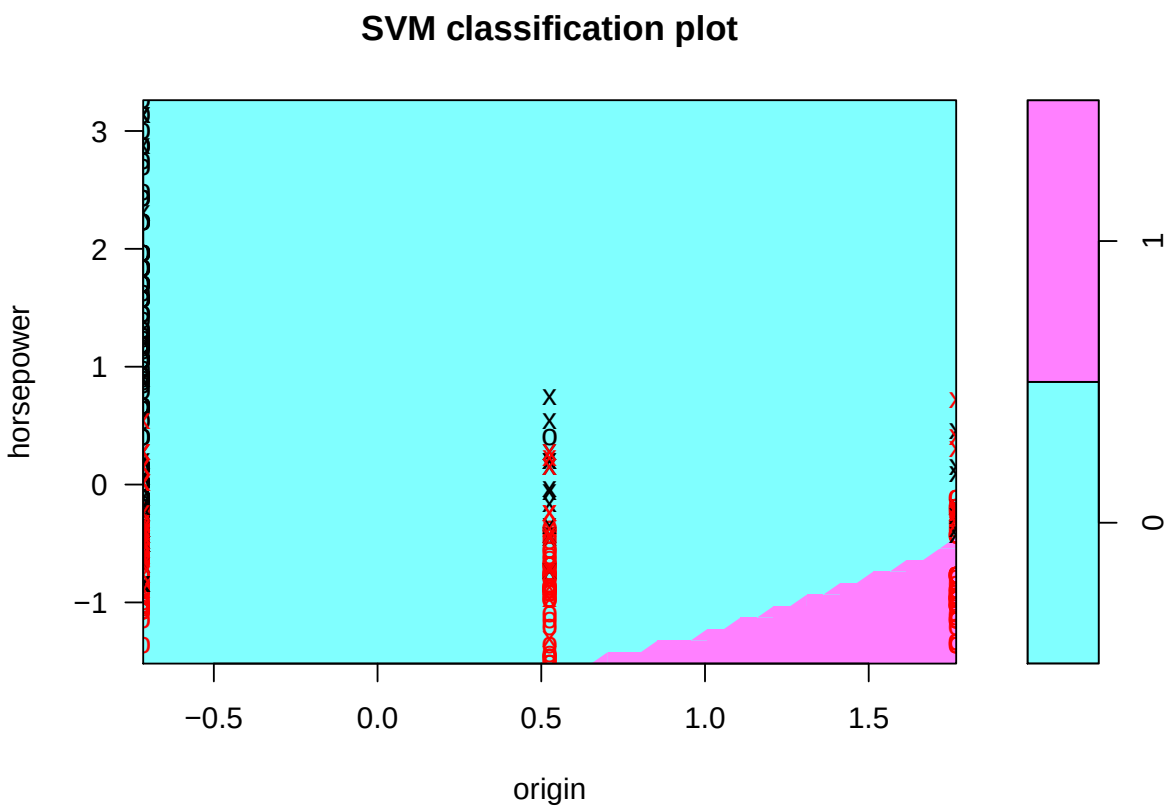
```
ACC_Radial = (TB[1] + TB[4])/length(DF$response)

plot(svm.fit, DF, weight ~ horsepower)
```



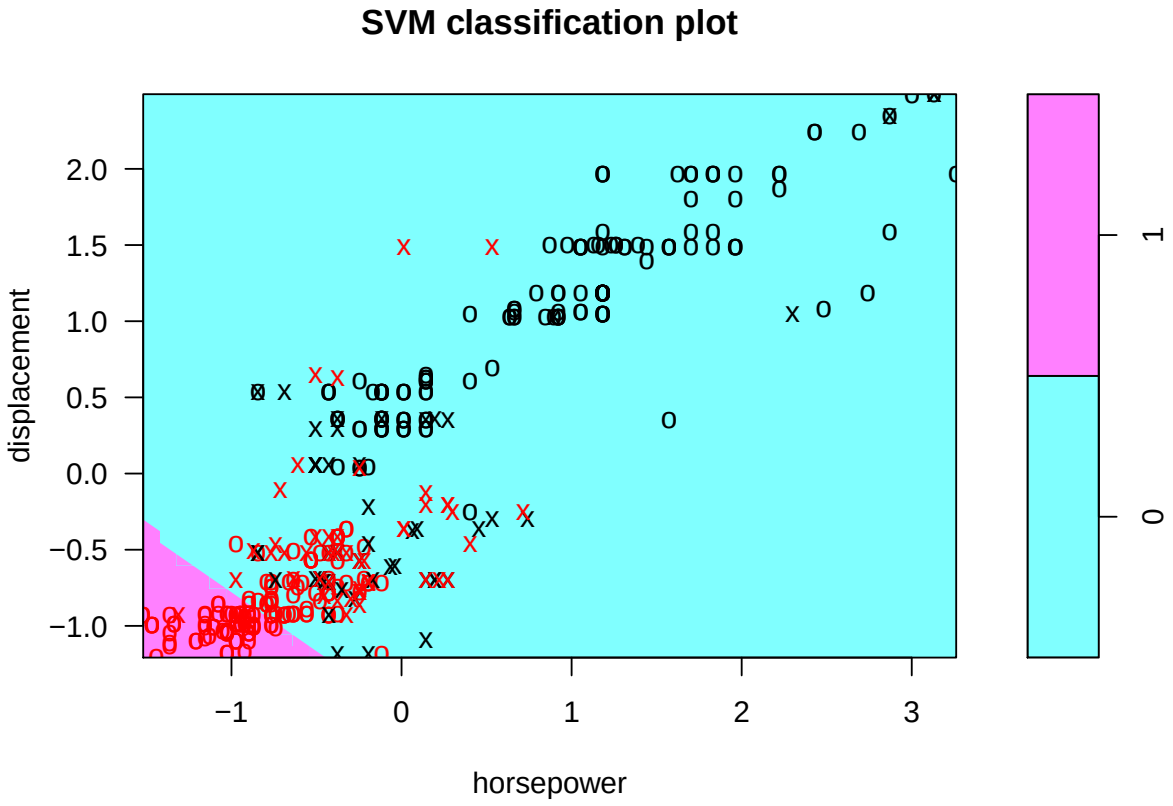

```
plot(svm.fit, DF, weight ~ year)
```



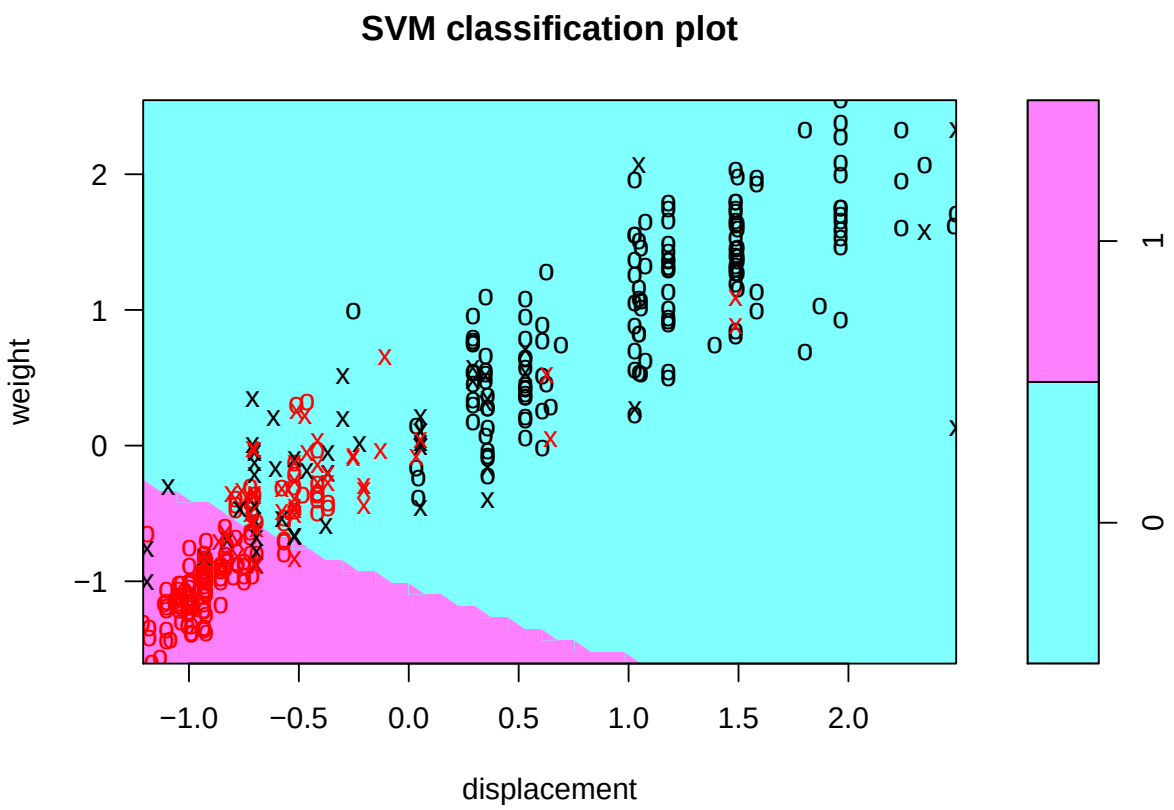

```
plot(svm.fit, DF, horsepower ~ origin)
```




```
# Now looking at some pairs not suggested by logistic regression
plot(svm.fit, DF, displacement ~ horsepower)
```




```
plot(svm.fit, DF, weight ~ displacement)
```



For the SVM trained with a kernel using the radial basis similarity function the accuracy has improved from 0.9158163 to 0.9158163. This is not a dramatic imporvement. It seems the displacement predictor is not being fit well we hope this is ameliorated by using the polynomial kernel. It's possible there is correlation between displacement and one of the other variables.

```
gamma_default <- 1/(ncol(DF) - 1)

gamma_list <- ((1:20)/10) * gamma_default
tune.svm = tune(svm, response ~ ., data = DF, kernel = "polynomial", ranges = list(gamma = gamma_list),
    degree = 4)
summary(tune.svm)
```

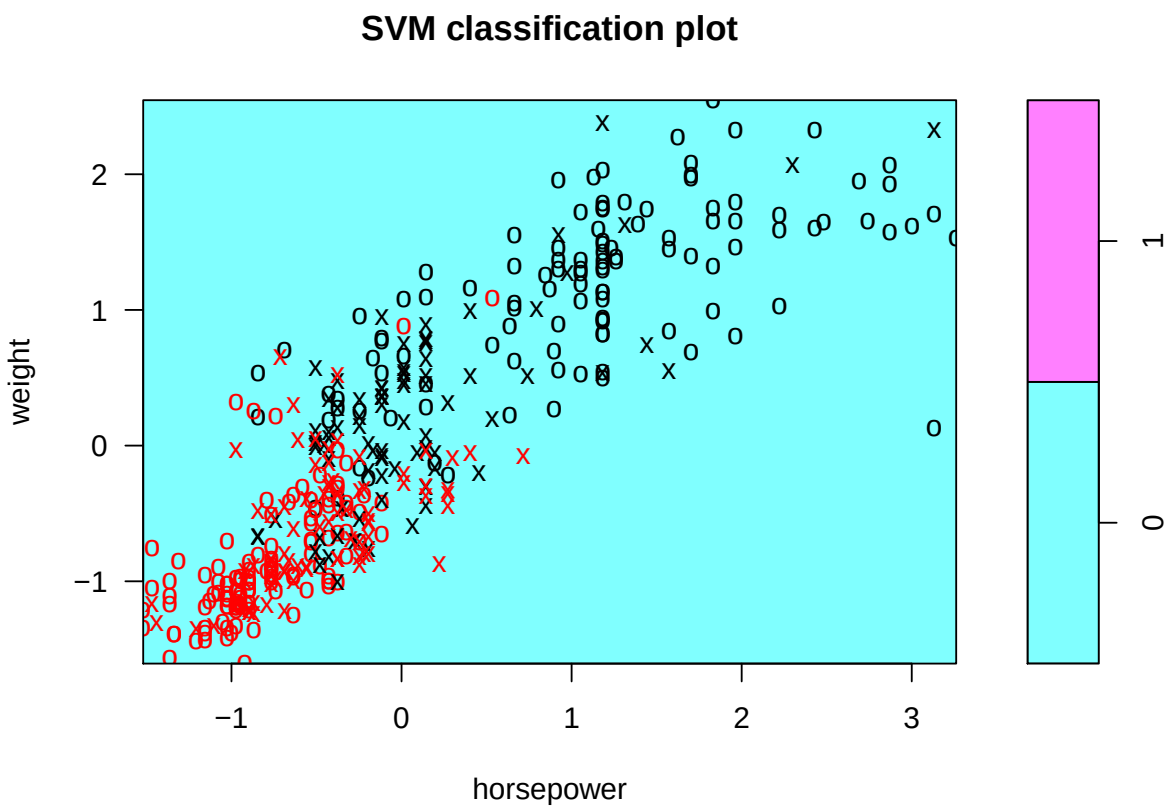
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      gamma
##  0.2714286
##
## - best performance: 0.1478846
##
## - Detailed performance results:
##         gamma     error dispersion
```

```
## 1  0.01428571 0.5714103 0.04147633
## 2  0.02857143 0.4850000 0.10284870
## 3  0.04285714 0.3932051 0.09189137
## 4  0.05714286 0.3344872 0.09568490
## 5  0.07142857 0.2808333 0.09016712
## 6  0.08571429 0.2604487 0.09000575
## 7  0.10000000 0.2655769 0.09039853
## 8  0.11428571 0.2604487 0.08919042
## 9  0.12857143 0.2451282 0.10385416
## 10 0.14285714 0.2426282 0.08019242
## 11 0.15714286 0.2145513 0.07224880
## 12 0.17142857 0.1887821 0.06184430
## 13 0.18571429 0.1964744 0.06060640
## 14 0.20000000 0.2016667 0.05635859
## 15 0.21428571 0.1939103 0.04705442
## 16 0.22857143 0.1811538 0.06671062
## 17 0.24285714 0.1708333 0.06015270
## 18 0.25714286 0.1580769 0.05205191
## 19 0.27142857 0.1478846 0.04921961
## 20 0.28571429 0.1503846 0.05544384
```

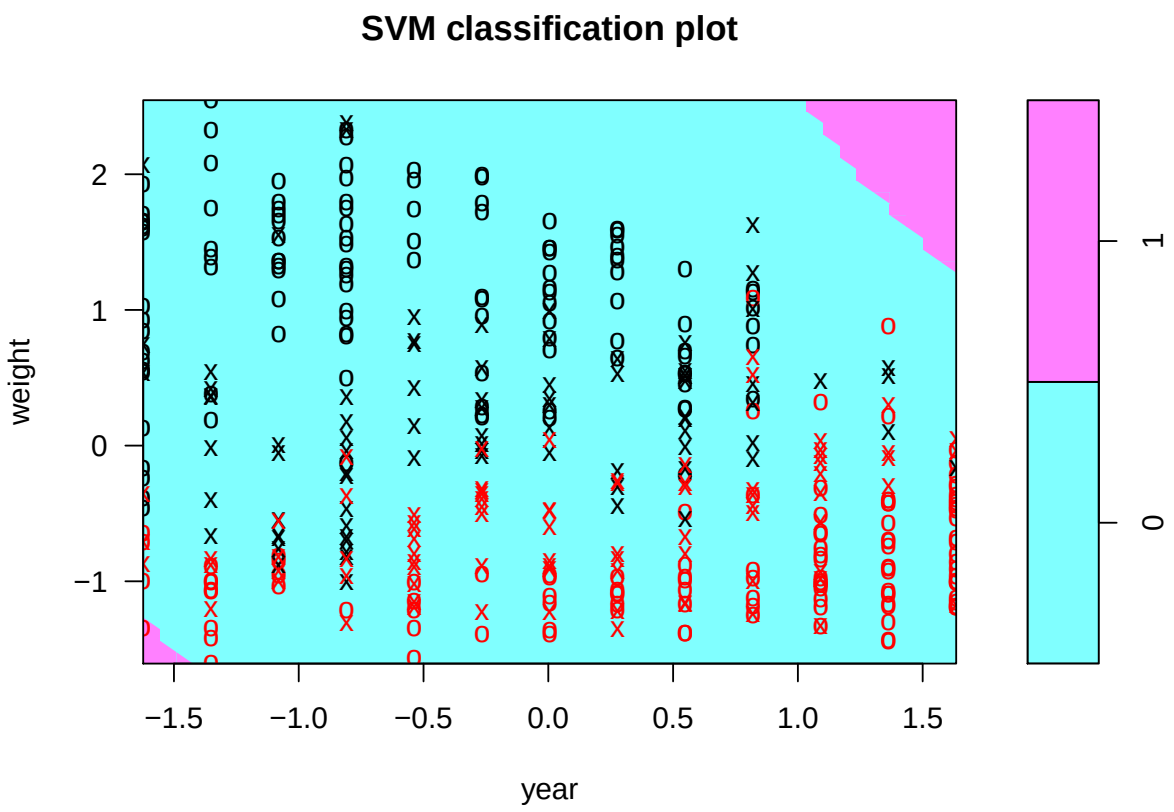
```
svm.fit <- tune.svm$best.model

svm.pred <- predict(svm.fit, DF)
TB <- table(svm.pred, DF$response)
pander(TB)
```

	0	1
0	188	31
1	8	165

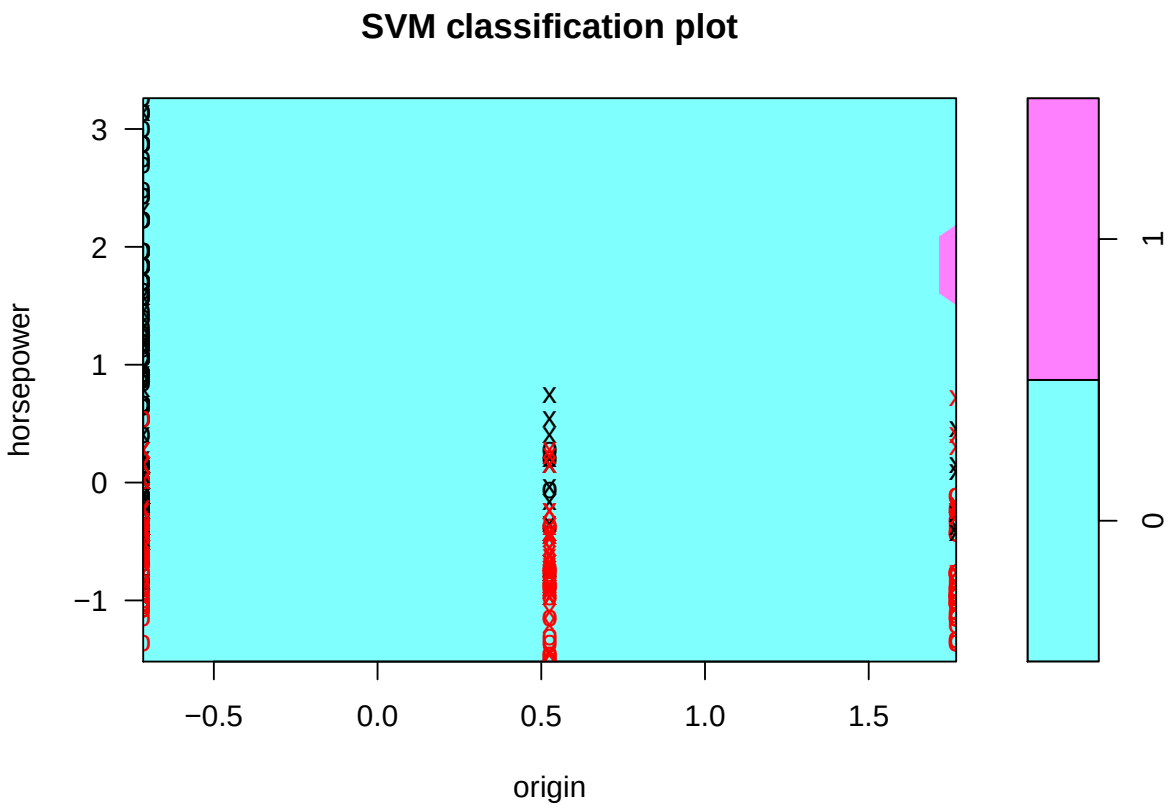
```
ACC_polynomial = (TB[1] + TB[4])/length(DF$response)

plot(svm.fit, DF, weight ~ horsepower, fill = TRUE)
```



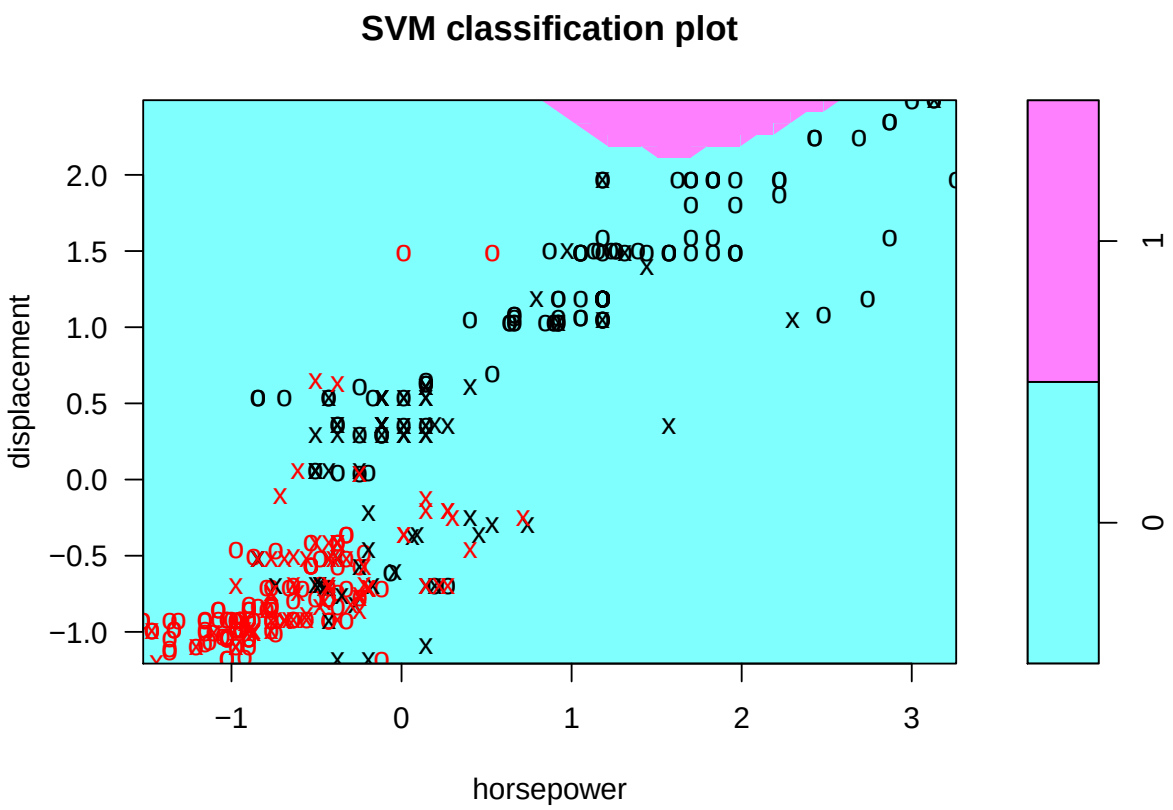

```
plot(svm.fit, DF, weight ~ year, fill = TRUE)
```



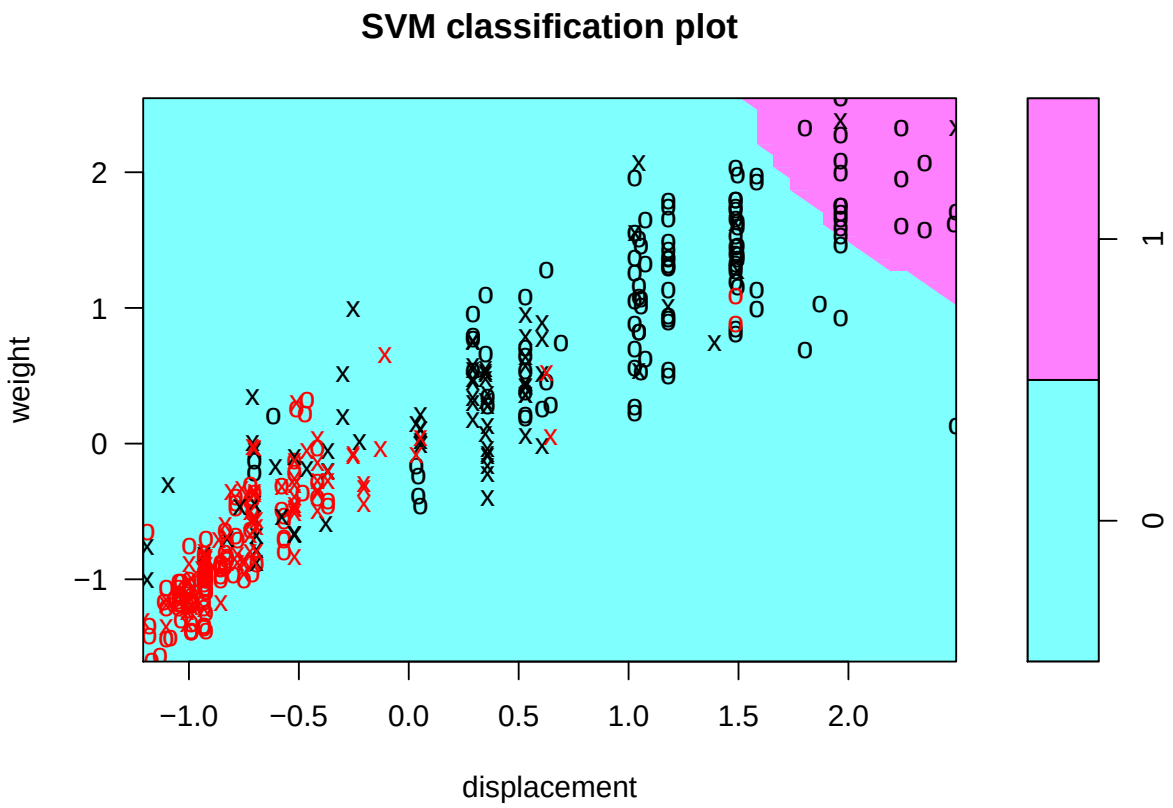

```
plot(svm.fit, DF, horsepower ~ origin, fill = TRUE)
```




```
# Now looking at some pairs not suggested by logistic regression
plot(svm.fit, DF, displacement ~ horsepower, fill = TRUE)
```




```
plot(svm.fit, DF, weight ~ displacement, fill = TRUE)
```



We're pleased to see that the accuracy has improved even further to `0.9005102`.

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

See above for svm plots. We'd like to note that the regions of classification are not plotted well for the ploynomical case. We can see the support vectors lead to resonable classification results in the plots - but we notice that the shaded regions of class 1 and class 0 do not align with the support vectors - even though the accuracy reported is quite high. This warrants further investigation. The plot function had no issues with the linear decision boundary. We tried looking into the documentation and changing some of the optnios for plot.svm but this did not resolve the issue.

A brief google session revealed this type of issue has come up before

http://grokbase.com/t/r/r-help/127s7jrj5x/r-only-one-class-shown-in-svm-plot

Chapter 10

Problem 2

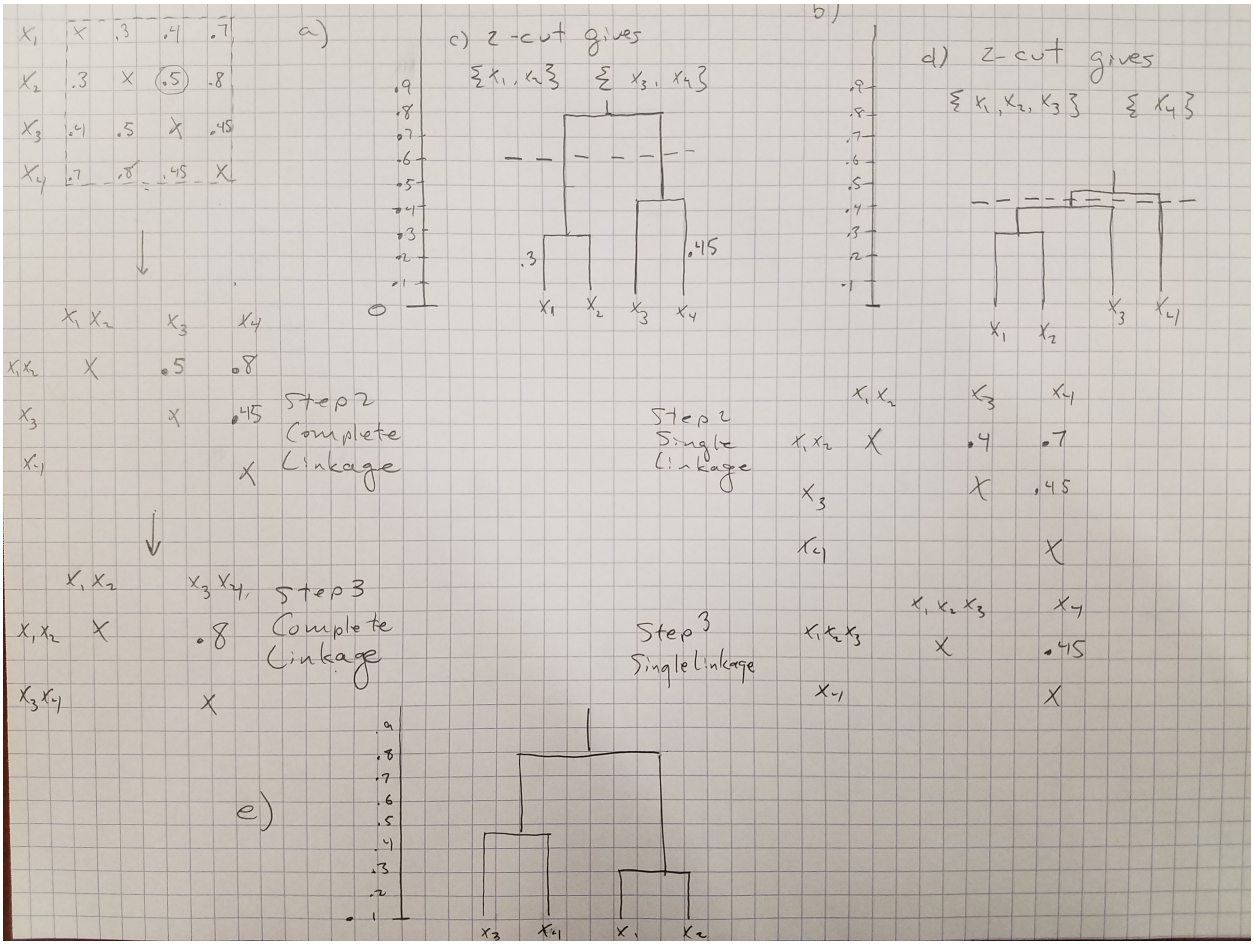
```
Suppose that we have four observations, for which we compute a
dissimilarity matrix, given by

0.3 0.4 0.7
```

0.3 0.5 0.8

0.4 0.5 0.45

0.7 0.8 0.45

For instance, the dissimilarity between the first and second observations is 0.3, and the dissimilarity between the second and fourth observations is 0.8.
(a) On the basis of this dissimilarity matrix, sketch the dendrogram that results from hierarchically clustering these four observations using complete linkage. Be sure to indicate on the plot the height at which each fusion occurs, as well as the observations corresponding to each leaf in the dendrogram.
(b) Repeat (a), this time using single linkage clustering.
(c) Suppose that we cut the dendogram obtained in (a) such that two clusters result. Which observations are in each cluster?
(d) Suppose that we cut the dendogram obtained in (b) such that two clusters result. Which observations are in each cluster?
(e) It is mentioned in the chapter that at each fusion in the dendrogram, the position of the two clusters being fused can be swapped without changing the meaning of the dendrogram. Draw a dendrogram that is equivalent to the dendrogram in (a), for which two or more of the leaves are repositioned, but for which the meaning of the dendrogram is the same.

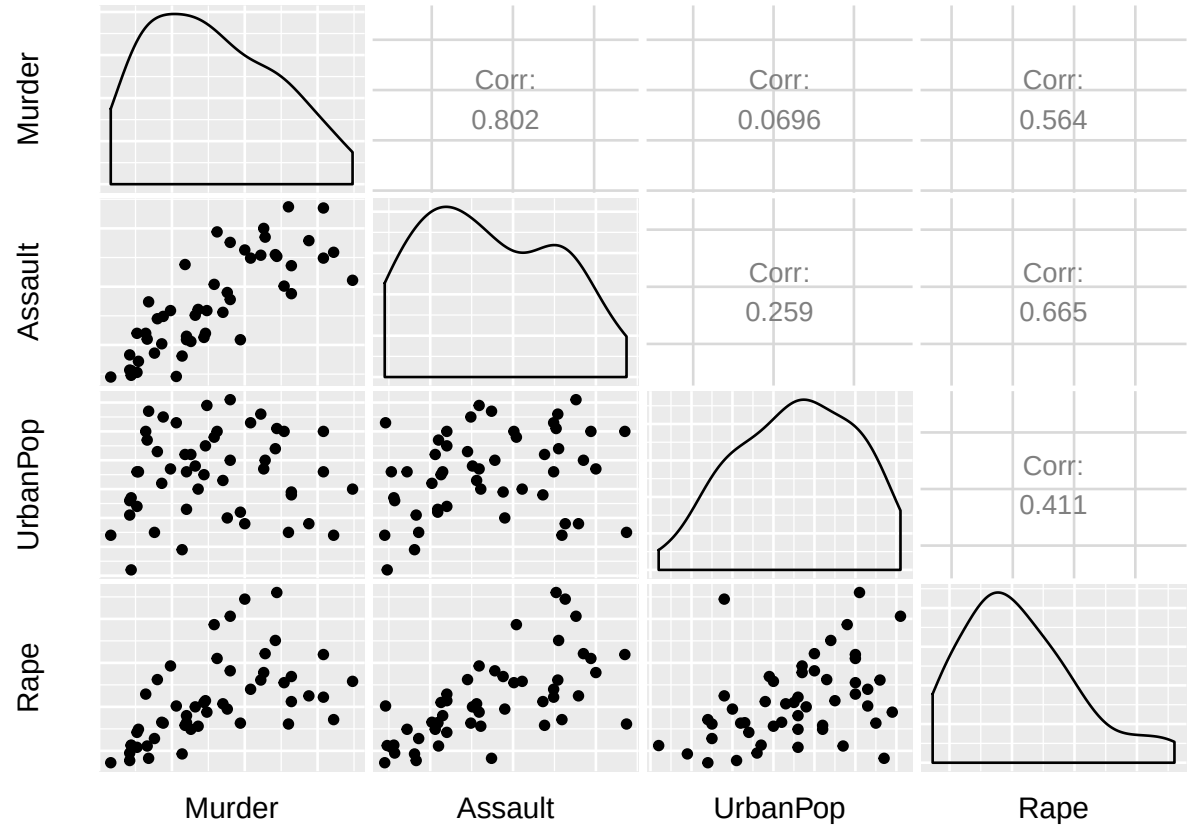


Chapter 10

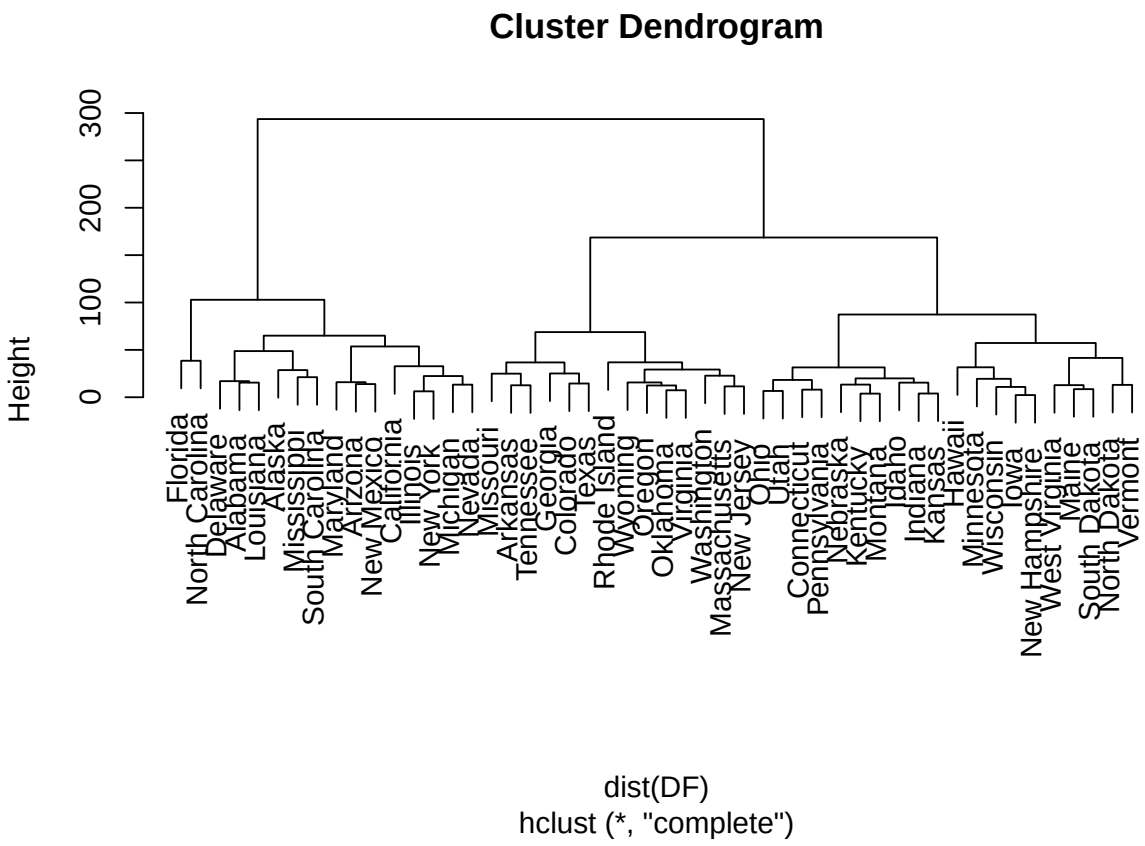
Problem 9.

Consider the USArrests data. We will now perform hierarchical clustering on the states.
a) Using hierarchical clustering with complete linkage and Euclidean distance, cluster the states.

```
attach(USArrests)
DF <- USArrests

library(ggplot2)
require(GGally)
ggpairs(DF) + theme(axis.line = element_blank(), axis.text = element_blank(),
    axis.ticks = element_blank())
```



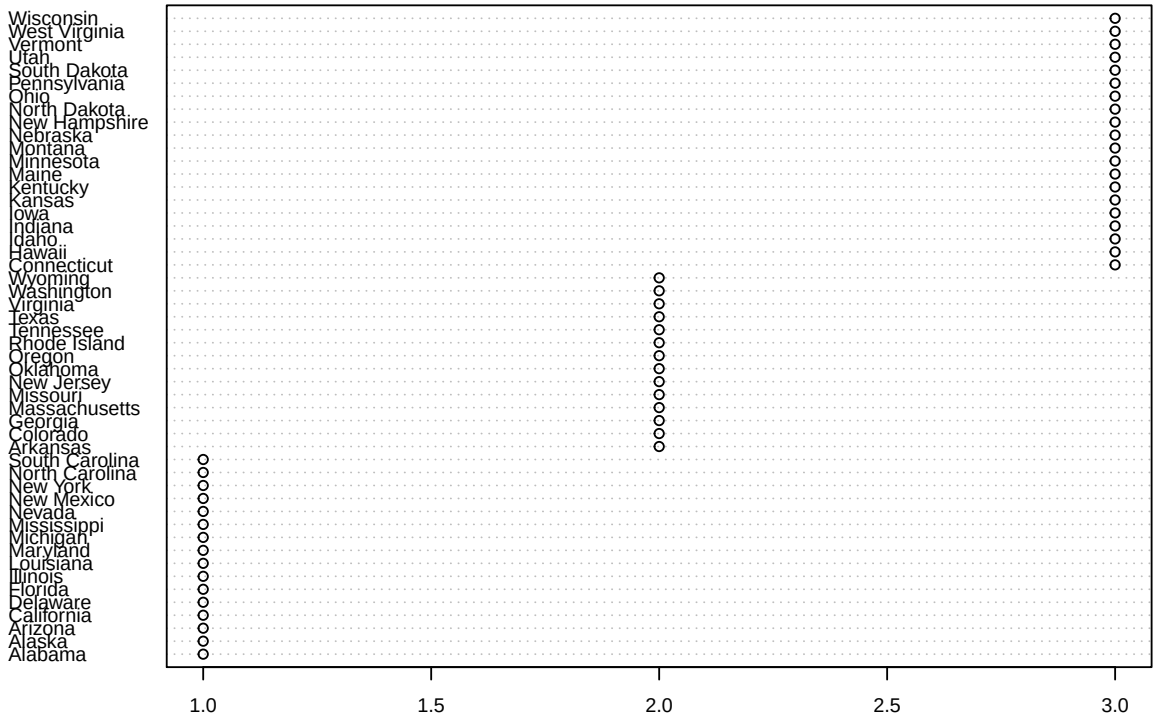

```
hc.complete = hclust(dist(DF), method = "complete")
plot(hc.complete)
```



b)

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

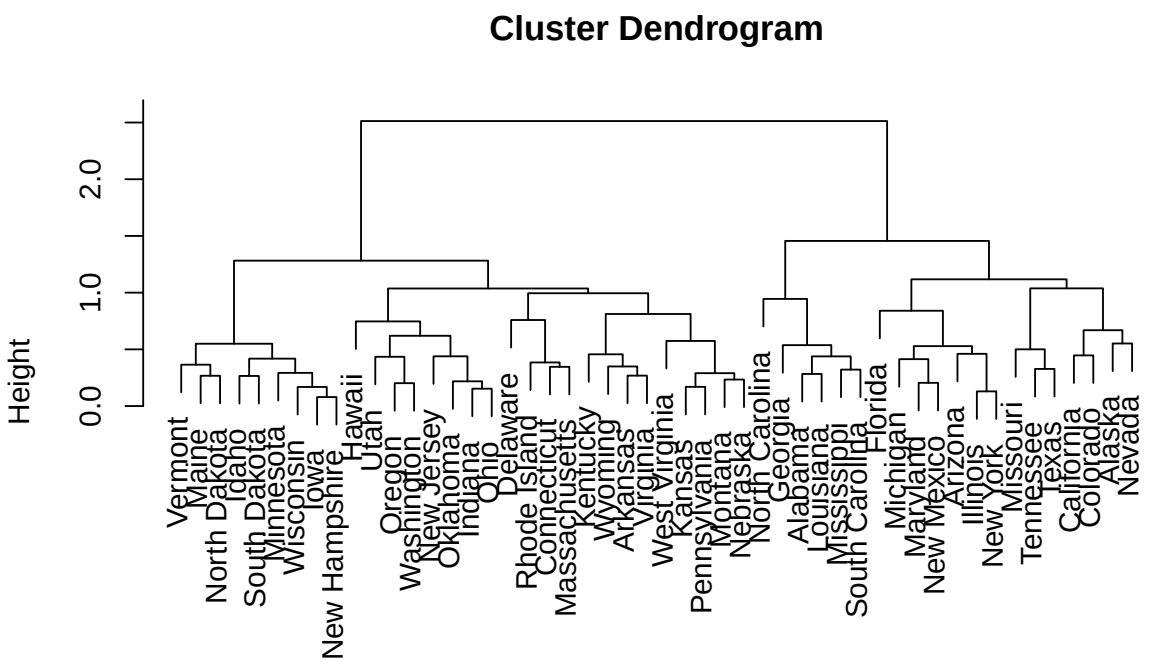
```
cut.hc <- cutree(hc.complete, 3)

results.cut <- data.frame(states = rownames(DF), cluster = cut.hc, UrbanPop = DF$UrbanPop)
results.cut <- results.cut[order(results.cut$cluster), ]
dotchart(results.cut$cluster, results.cut$states, cex = 0.7)
```



c)

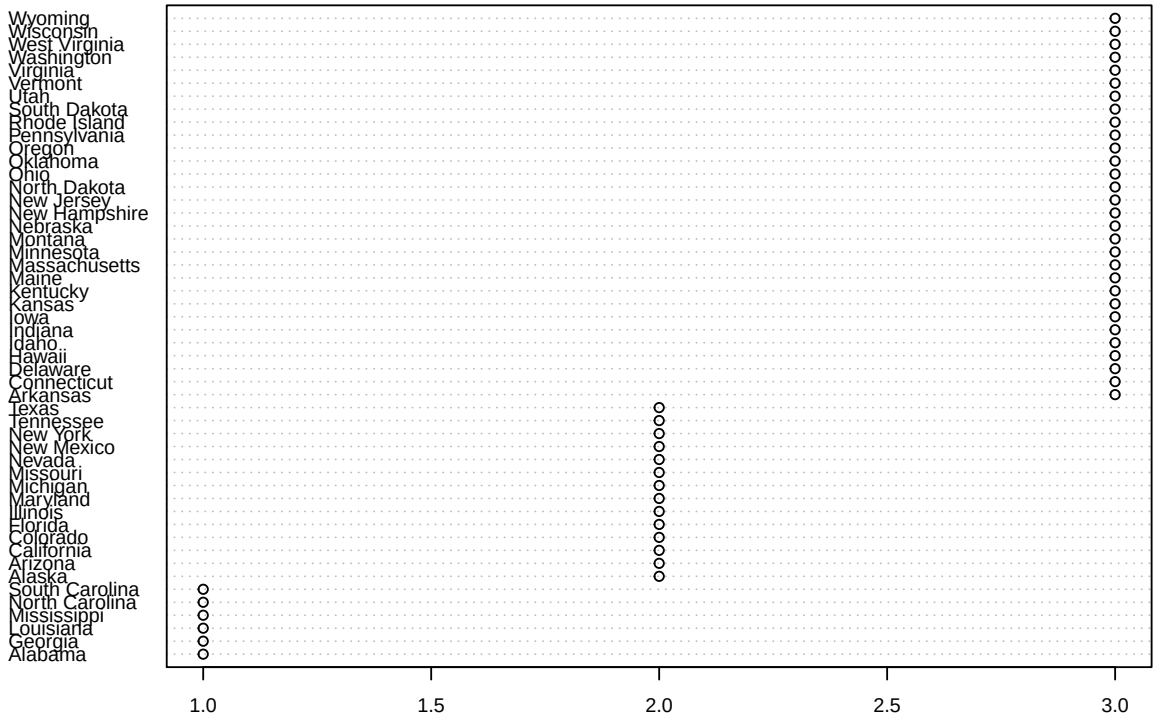
Hierarchically cluster the states using complete linkage and Euclidean distance, after scaling the variables to have standard deviation one.

```
hc.complete.scaled = hclust(dist(data.frame(scale(DF, center = FALSE, scale = TRUE))),
    method = "complete")
plot(hc.complete.scaled)
```



dist(data.frame(scale(DF, center = FALSE, scale = TRUE)))
hclust (*, "complete")

```
cut.hc.scaled <- cutree(hc.complete.scaled, 3)

results.cut.scaled <- data.frame(states = rownames(DF), cluster = cut.hc.scaled,
    UrbanPop = DF$UrbanPop)
results.cut.scaled <- results.cut.scaled[order(results.cut.scaled$cluster),
    ]
dotchart(results.cut.scaled$cluster, results.cut.scaled$states, cex = 0.7)
```



d)

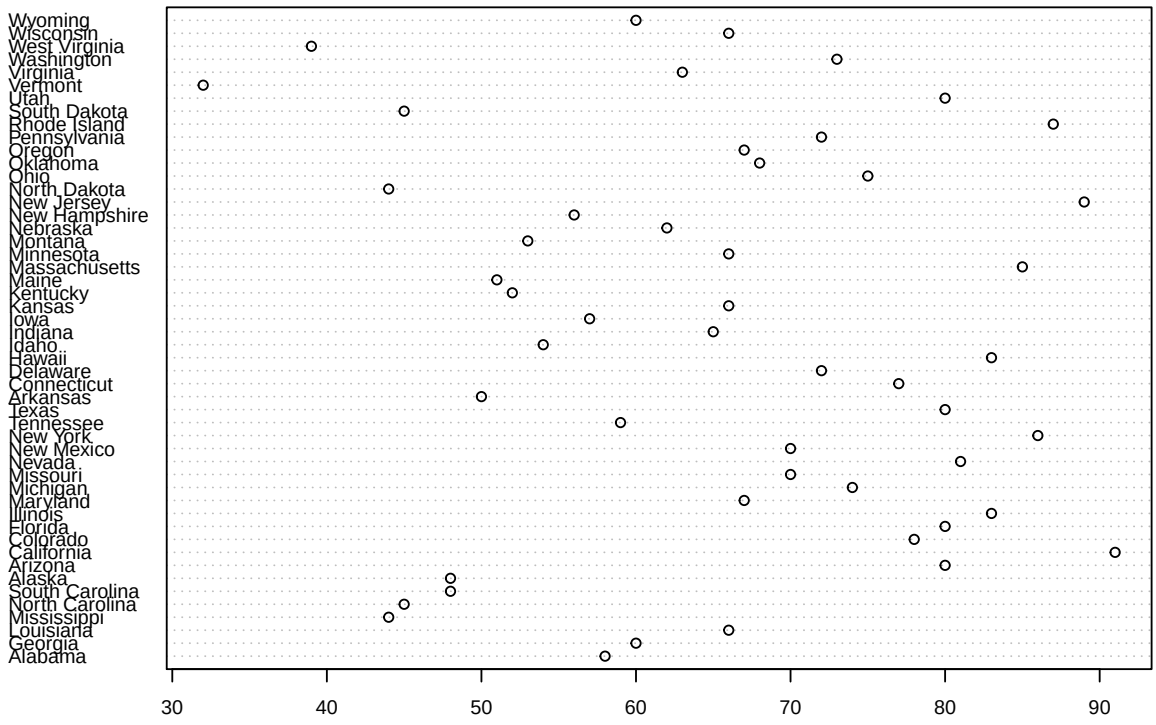
`What effect does scaling the variables have on the hierarchical clustering obtained? In your opinion, should the variables be scaled before the inter-observation dissimilarities are computed? Provide a justification for your answer.`

Scaling changes the partitioning of the states in the 3 clusters. The cluster membership count is less balanced. We surmise this is because with scaling the UrbanPop feature is given more weight and assult is given less weight in the calculation of the euclidian distance matrix between the sample points.

In general when using euclidian distance as a similarity measure variables should be scaled to give equal weights to them. We can weight variables unevenly if we know which ones are more important.

Three of the variables are in units of incidents per 100,000 population but the UrbanPop is a numeric percent. We can see that when we scale the variables the UrbanPop. Below we look at the 2 sample t-test for the differences in means between the cluster urbanpop levels for scaled and non-scaled data. We see that the p-values indicate that means are all different for the scaled data, but for the unscaled data we must accept the null hypothese that the means are the same for cluster 1 and 2.

```
dotchart(results.cut.scaled$UrbanPop, results.cut.scaled$states, cex = 0.7)
```




```r
# Non Scaled t-test for difference in urban pop between the three clusters
cluster1.urbanpop <- results.cut[results.cut$cluster == 1, "UrbanPop"]

cluster2.urbanpop <- results.cut[results.cut$cluster == 2, "UrbanPop"]

cluster3.urbanpop <- results.cut[results.cut$cluster == 3, "UrbanPop"]

t.test(cluster1.urbanpop, cluster2.urbanpop)
```

```
##
##  Welch Two Sample t-test
##
## data:  cluster1.urbanpop and cluster2.urbanpop
## t = -0.46583, df = 27.552, p-value = 0.645
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -12.58528   7.92457
## sample estimates:
## mean of x mean of y
##  68.31250  70.64286
```

```r
t.test(cluster2.urbanpop, cluster3.urbanpop)
```

```
##
```

```
##  Welch Two Sample t-test
##
## data:  cluster2.urbanpop and cluster3.urbanpop
## t = 2.4528, df = 30.671, p-value = 0.02007
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   1.831382 19.954332
## sample estimates:
## mean of x mean of y
##  70.64286  59.75000
```

```
t.test(cluster3.urbanpop, cluster1.urbanpop)
```

```
##
##  Welch Two Sample t-test
##
## data:  cluster3.urbanpop and cluster1.urbanpop
## t = -1.7232, df = 30.574, p-value = 0.09496
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -18.702732   1.577732
## sample estimates:
## mean of x mean of y
##   59.7500   68.3125
```

```
# Scaled t-test for difference in urban pop between the three clusters
cluster1.urbanpop.scaled <- results.cut.scaled[results.cut.scaled$cluster ==
    1, "UrbanPop"]

cluster2.urbanpop.scaled <- results.cut.scaled[results.cut.scaled$cluster ==
    2, "UrbanPop"]

cluster3.urbanpop.scaled <- results.cut.scaled[results.cut.scaled$cluster ==
    3, "UrbanPop"]

t.test(cluster1.urbanpop.scaled, cluster2.urbanpop.scaled)
```

```
##
##  Welch Two Sample t-test
##
## data:  cluster1.urbanpop.scaled and cluster2.urbanpop.scaled
## t = -4.4539, df = 11.847, p-value = 0.0008122
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -31.71334 -10.85809
## sample estimates:
## mean of x mean of y
##  53.50000  74.78571
```

```
t.test(cluster2.urbanpop.scaled, cluster3.urbanpop.scaled)
```

```
##
```

```
##  Welch Two Sample t-test
##
## data:  cluster2.urbanpop.scaled and cluster3.urbanpop.scaled
## t = 2.7845, df = 31.955, p-value = 0.008934
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   2.993718 19.311044
## sample estimates:
## mean of x mean of y
##  74.78571  63.63333
```

```
t.test(cluster3.urbanpop.scaled, cluster1.urbanpop.scaled)
```

```
##
##  Welch Two Sample t-test
##
## data:  cluster3.urbanpop.scaled and cluster1.urbanpop.scaled
## t = 2.2307, df = 10.84, p-value = 0.0478
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   0.1171813 20.1494854
## sample estimates:
## mean of x mean of y
##  63.63333  53.50000
```

Chapter 10

Problem 10

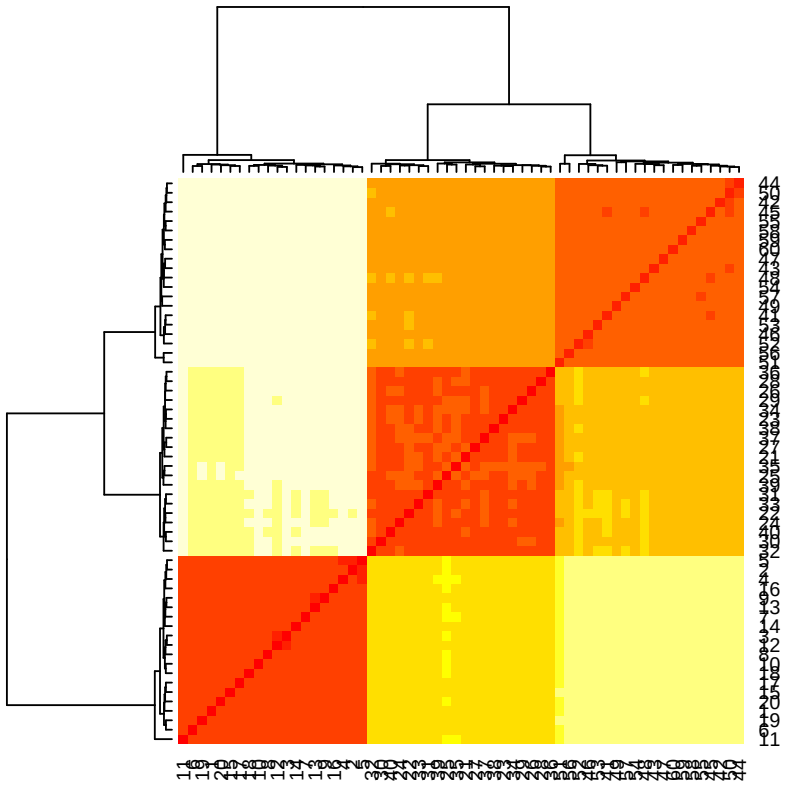
In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data.

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

```
library(MASS)
X1 <- mvrnorm(20, mu = rnorm(50, 1), Sigma = diag(runif(50)))

X2 <- mvrnorm(20, mu = rnorm(50, 5), Sigma = diag(runif(50)))

X3 <- mvrnorm(20, mu = rnorm(50, 7), Sigma = diag(runif(50)))


class.label <- rbind(matrix(data = 1, nrow = 20, ncol = 1), matrix(data = 2,
    nrow = 20, ncol = 1), matrix(data = 3, nrow = 20, ncol = 1))

DF <- rbind(X1, X2, X3)
```

```
dist.mat <- as.matrix(dist(DF, method = "euclidian"))

heatmap(dist.mat)
```



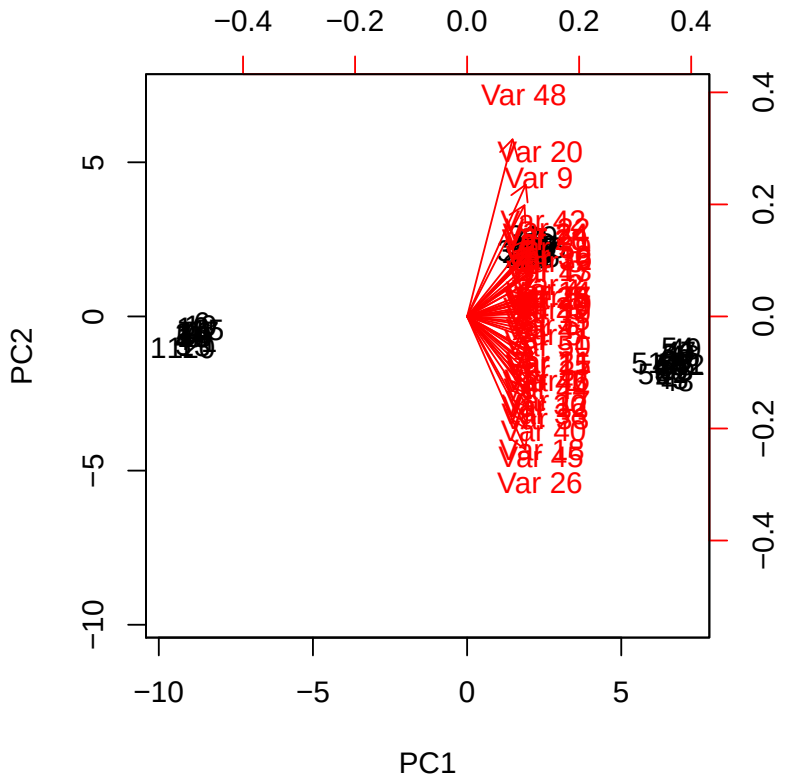
The heat map and dendrogarm show us we have adequate class separation in our simulated data set.

b)

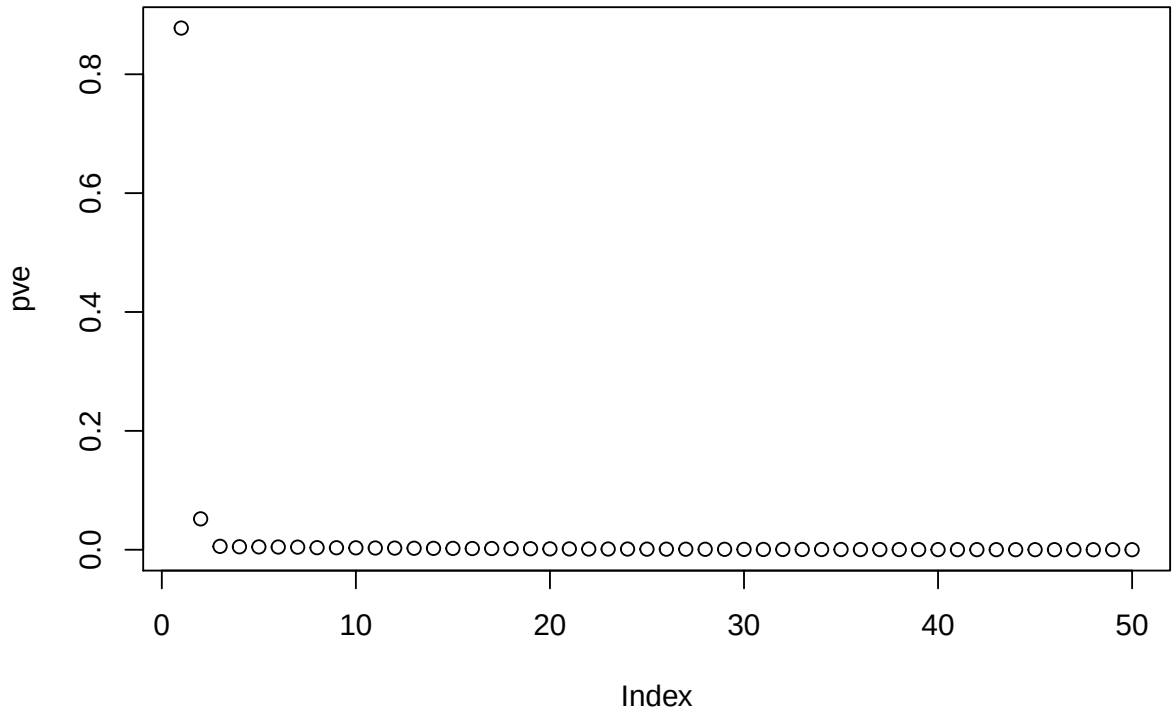
Perform PCA on the 60 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue on to part (c). If not, then return to part (a) and modify the 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.

We've ensured there is good class separation - so we expect the PCA to demonstrate this.

```
pr.out = prcomp(DF, scale = TRUE)
biplot(pr.out, scale = 0, col = c(1, 2))
```




```
pr.var = pr.out$sdev^2

pve = pr.var/sum(pr.var)

plot(pve)
```



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? 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.out = kmeans(DF, 3, nstart = 20)

cluster_kmean.label <- km.out$cluster

TB <- table(cluster_kmean.label, class.label)
library(pander)
pander(TB)
```

1	2	3
20	0	0
0	0	20
0	20	0

K-means with $k = 3$ correctly found all three clusters.

d)

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

```
km.out = kmeans(DF, 2, nstart = 20)

cluster_kmean.label <- km.out$cluster

TB <- table(cluster_kmean.label, class.label)
library(pander)
pander(TB)
```

1	2	3
20	0	0
0	20	20

We see that kmeans with $k = 2$ found cluster 1 and merged cluster 2 and cluster 3 into one cluser.

e)

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

```
km.out = kmeans(DF, 4, nstart = 20)

cluster_kmean.label <- km.out$cluster

TB <- table(cluster_kmean.label, class.label)
library(pander)
pander(TB)
```

1	2	3
0	0	9
0	0	11
20	0	0
0	20	0

K-means with $k = 4$ correctly identified clusters 2 and 3, and split cluster 1 into two clusters - one of size 7 and one of size 13.

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.`

```
pr.1and2 <- pr.out$x[, 1:2]

km.out = kmeans(pr.1and2, 3, nstart = 20)

cluster_kmean.label <- km.out$cluster

TB <- table(cluster_kmean.label, class.label)
library(pander)
pander(TB)
```

1	2	3
20	0	0
0	0	20
0	20	0

We see that performing K-means on the first two principal components correctly identifies all the clusters. This is expected - even if we had chosen our mean vectors in the sumulated data set such that there was more class overlap.

```
sum(pve[1:2])
```

```
## [1] 0.9296832
```

We see that most of the variance in this data set is captured by the first two principal components.

g)

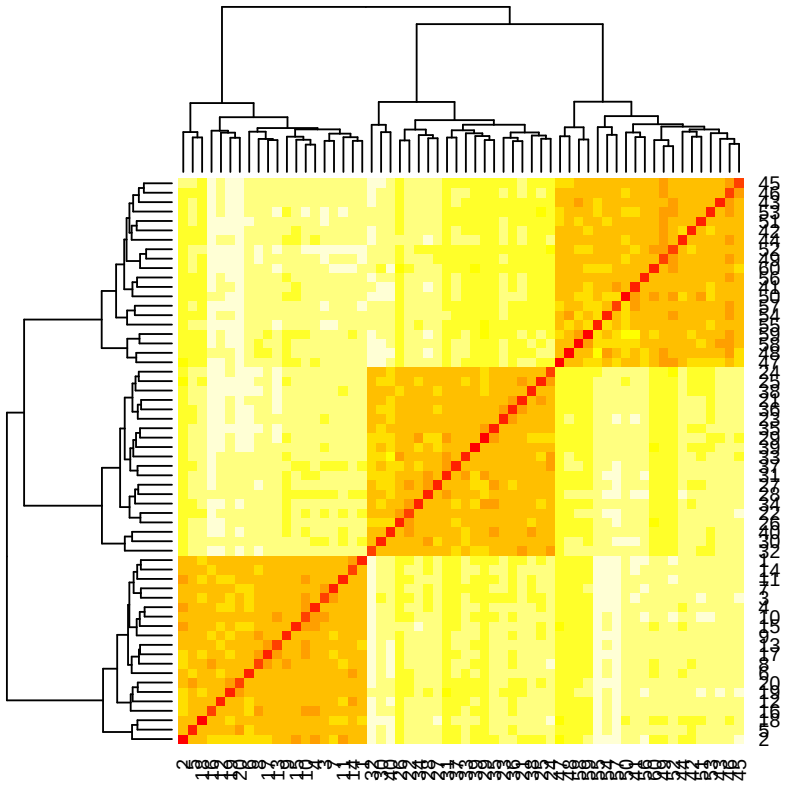
Using the scale() function, perform K-means clustering with K = 3 on the data after scaling each variable to have standard deviation one. How do these results compare to those obtained in (b)? Explain.

```
km.out = kmeans(scale(DF, scale = TRUE), 3, nstart = 20)

cluster_kmean.label <- km.out$cluster

TB <- table(cluster_kmean.label, class.label)
library(pander)
pander(TB)
```

1	2	3
20	0	0
0	20	0
0	0	20

Again, we see perfect class identification. If the classes were generated with more overlap - we would expect that scaling would improve the results. Below we experiment with finding a data set that better demonstates the principles the exercise is trying to show.

```
X1 <- mvrnorm(20, mu = rnorm(50, 1), Sigma = diag(runif(50)))
X2 <- mvrnorm(20, mu = rnorm(50, 1), Sigma = diag(runif(50)))
X3 <- mvrnorm(20, mu = rnorm(50, 1), Sigma = diag(runif(50)))

class.label <- rbind(matrix(data = 1, nrow = 20, ncol = 1), matrix(data = 2,
    nrow = 20, ncol = 1), matrix(data = 3, nrow = 20, ncol = 1))

DF <- rbind(X1, X2, X3)

dist.mat <- as.matrix(dist(DF, method = "euclidian"))

heatmap(dist.mat)
```

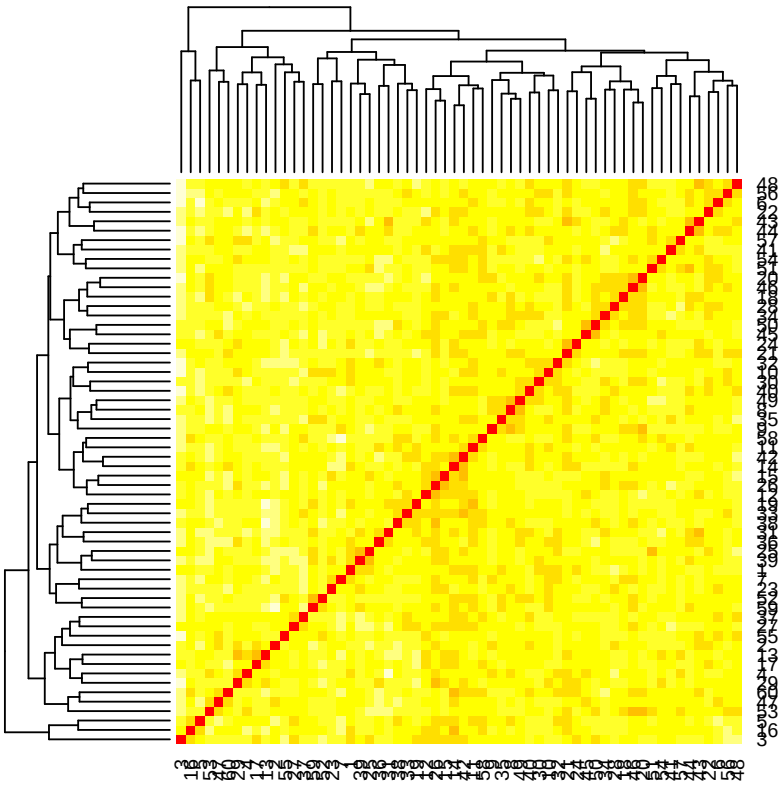


```
km.out = kmeans(DF, 3, nstart = 20)

cluster_kmean.label <- km.out$cluster

TB <- table(cluster_kmean.label, class.label)
library(pander)
pander(TB)
```

1	2	3
0	0	20

1	2	3
20	0	0
0	20	0

```
# Interesting that setting the means to the same value did not give the
# class overlap to cause clustering to get even one label wrong. We look
# into the way we generate the covariance matrix and mean vectors next
M <- diag(runif(50))
v <- rnorm(50, 1)
X1 <- mvrnorm(20, mu = v, Sigma = M)
X2 <- mvrnorm(20, mu = v * (1.1), Sigma = M)
X3 <- mvrnorm(20, mu = v * (0.9), Sigma = M)

class.label <- rbind(matrix(data = 1, nrow = 20, ncol = 1), matrix(data = 2,
    nrow = 20, ncol = 1), matrix(data = 3, nrow = 20, ncol = 1))

DF <- rbind(X1, X2, X3)

dist.mat <- as.matrix(dist(DF, method = "euclidian"))

heatmap(dist.mat)
```



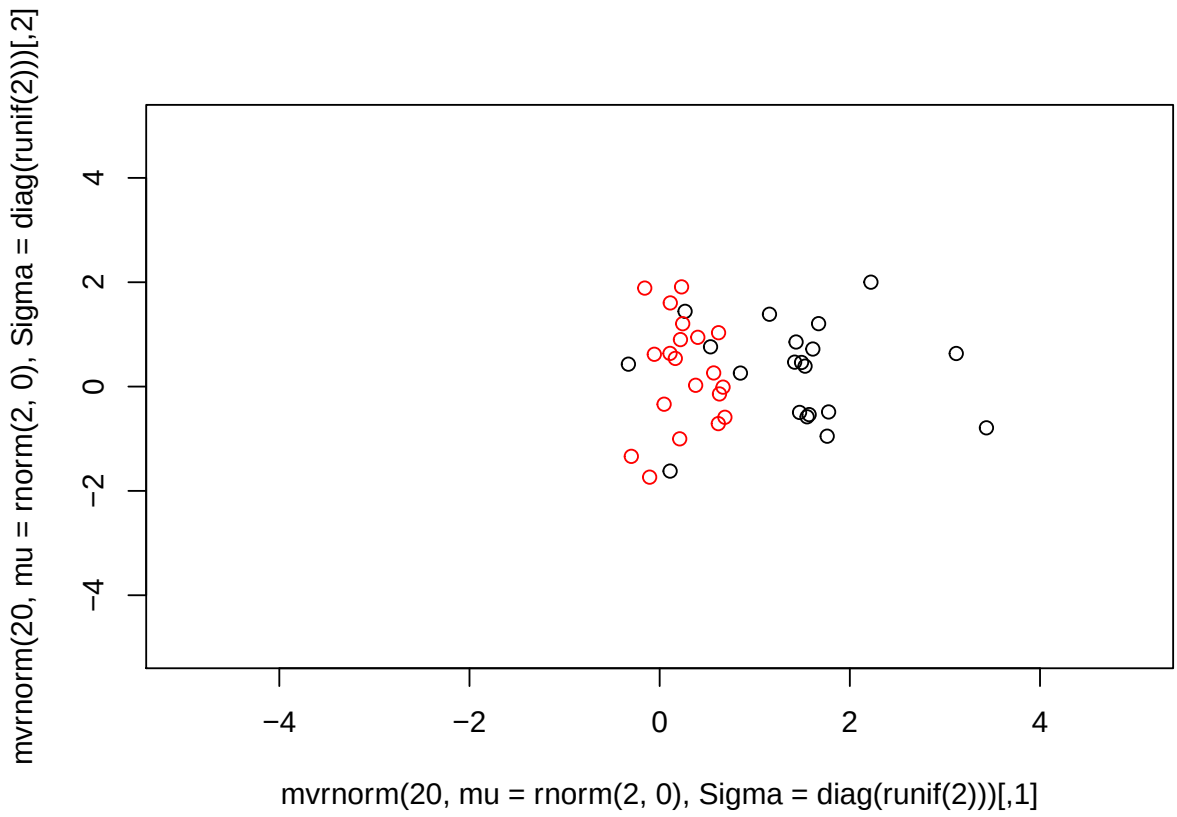
```r
km.out = kmeans(DF, 3, nstart = 20)

cluster_kmean.label <- km.out$cluster

TB <- table(cluster_kmean.label, class.label)
library(pander)
pander(TB)
```

1	2	3
5	5	10
6	9	4
9	6	6

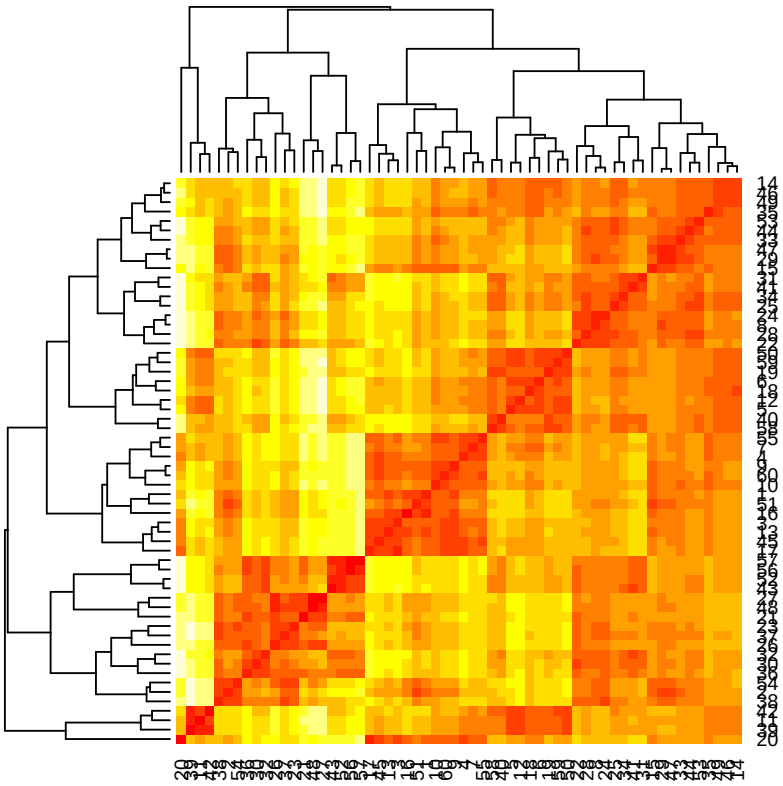
By generating the classes with the same random mean vector and usng an offset from that to separate the classes we have been able to introduce some class overlap in 50 dimensions.

This plot demonstated the effect of dimension increase. In 2 dimensions we get class overlap using the method at the beginning of the problem.

```r
plot(mvrnorm(20, mu = rnorm(2, 0), Sigma = diag(runif(2))), xlim = c(-5, 5),
    ylim = c(-5, 5))
points(mvrnorm(20, mu = rnorm(2, 0), Sigma = diag(runif(2))), col = "red")
```



```r
X1 <- mvrnorm(20, mu = rnorm(2, 0), Sigma = diag(runif(2)))
X2 <- mvrnorm(20, mu = rnorm(2, 0), Sigma = diag(runif(2)))
X3 <- mvrnorm(20, mu = rnorm(2, 0), Sigma = diag(runif(2)))

class.label <- rbind(matrix(data = 1, nrow = 20, ncol = 1), matrix(data = 2,
    nrow = 20, ncol = 1), matrix(data = 3, nrow = 20, ncol = 1))

DF <- rbind(X1, X2, X3)

dist.mat <- as.matrix(dist(DF, method = "euclidian"))

heatmap(dist.mat)
```



```r
km.out = kmeans(DF, 3, nstart = 20)

cluster_kmean.label <- km.out$cluster

TB <- table(cluster_kmean.label, class.label)
library(pander)
pander(TB)
```

1	2	3
2	13	8

1	2	3
11	0	4
7	7	8

The code above shows the results of k-means in 2 dimensions.

Chapter 10

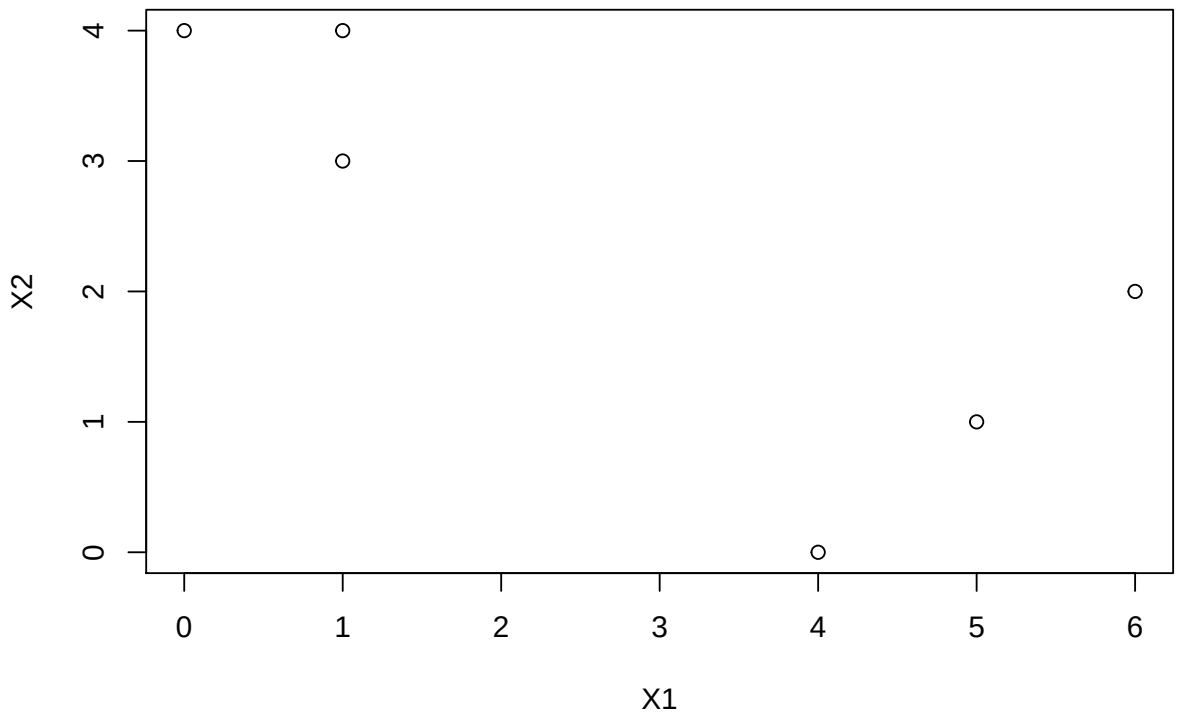
Problem 3

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.

a)

Plot the observations.

```
X1 <- c(1, 1, 0, 5, 6, 4)
X2 <- c(4, 3, 4, 1, 2, 0)
plot(X1, X2)
```



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.

```
DF <- data.frame(X1 = X1, X2 = X2)
class1 <- sample(nrow(DF), floor(nrow(DF)/2))

class <- matrix(0, nrow = nrow(DF), ncol = 1)
class[class1] <- 1
DF$class <- class
library(pander)
pander(DF)
```

X1	X2	class
1	4	0
1	3	1
0	4	0
5	1	1
6	2	0
4	0	1

c)

Compute the centroid for each cluster.

```
DFClass1 <- DF[DF$class == 1, ]
DFClass1$class <- NULL
centroid1 <- colMeans(as.matrix(DFClass1))


DFClass0 <- DF[DF$class == 0, ]
DFClass0$class <- NULL
centroid0 <- colMeans(as.matrix(DFClass0))
```

d)

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

```
DFDist <- rbind(DFClass0, DFClass1, centroid0, centroid1)

# Boo Duplicate Row names not allowed. Oh well we understand why
index <- (c(rep("class 0", nrow(DFClass0)), c(rep("class 1", nrow(DFClass1)),
    "Centroid0", "Centroid1")))
# row.names(DFDist)<-index

row.names(DFDist) <- c("C01", "C02", "C03", "C11", "C12", "C13", "Centroid0",
    "Centroid1")
```

```
dmat <- as.matrix(dist(DFDist))

centroid0Index <- which(index == "Centroid0")

centroid1Index <- which(index == "Centroid1")

indicatorCentroid1_isCloser <- dmat[centroid1Index, ] < dmat[centroid0Index,
    ]
indicatorCentroid1_isCloser
```

```
##       C01       C02       C03       C11       C12       C13 Centroid0
##     FALSE     FALSE      TRUE     FALSE      TRUE      TRUE     FALSE
## Centroid1
##      TRUE
```

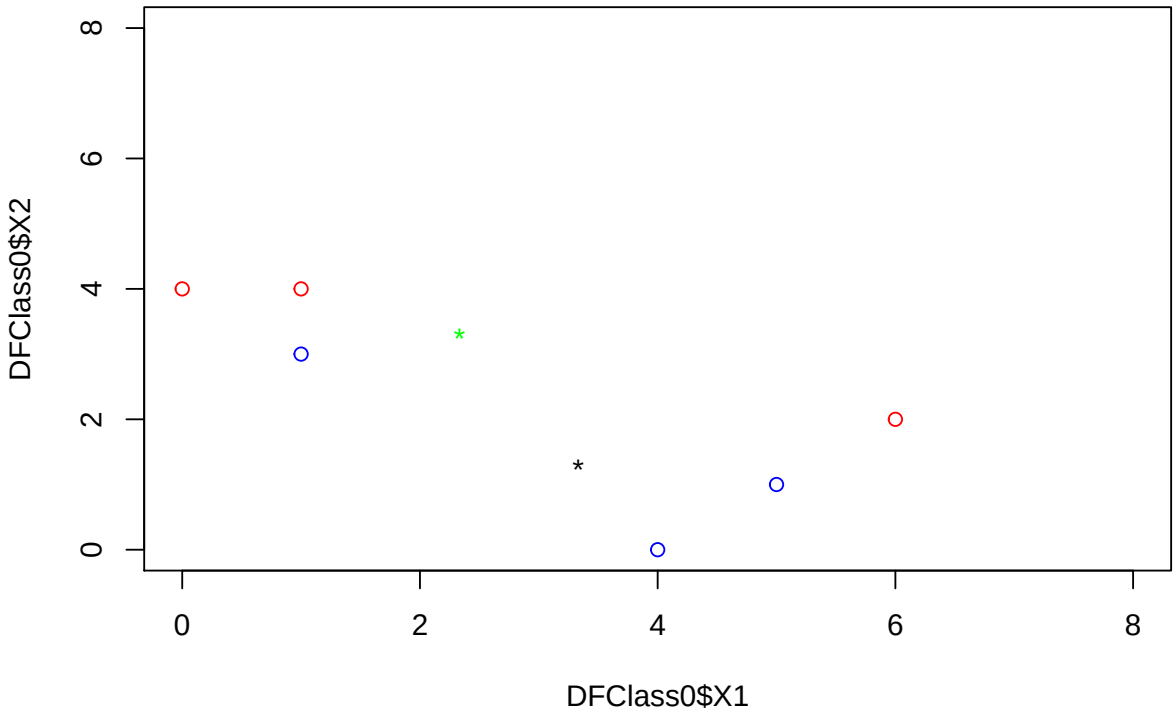
```
cluster1 <- which(indicatorCentroid1_isCloser[c(-7, -8)])
cluster0 <- which(indicatorCentroid1_isCloser[c(-7, -8)] == FALSE)

pander(cluster0, caption = "Cluster 0")
```

C01	C02	C11
1	2	4

```
pander(cluster1, caption = "Cluster 1")
```

C03	C12	C13
3	5	6

```
plot(DFClass0$X1, DFClass0$X2, col = "red", xlim = c(0, 8), ylim = c(0, 8))
points(DFClass1$X1, DFClass1$X2, col = "blue")
points(centroid0[1], centroid0[2], pch = "*", col = "green")
points(centroid1[1], centroid1[2], pch = "*", col = "black")
```

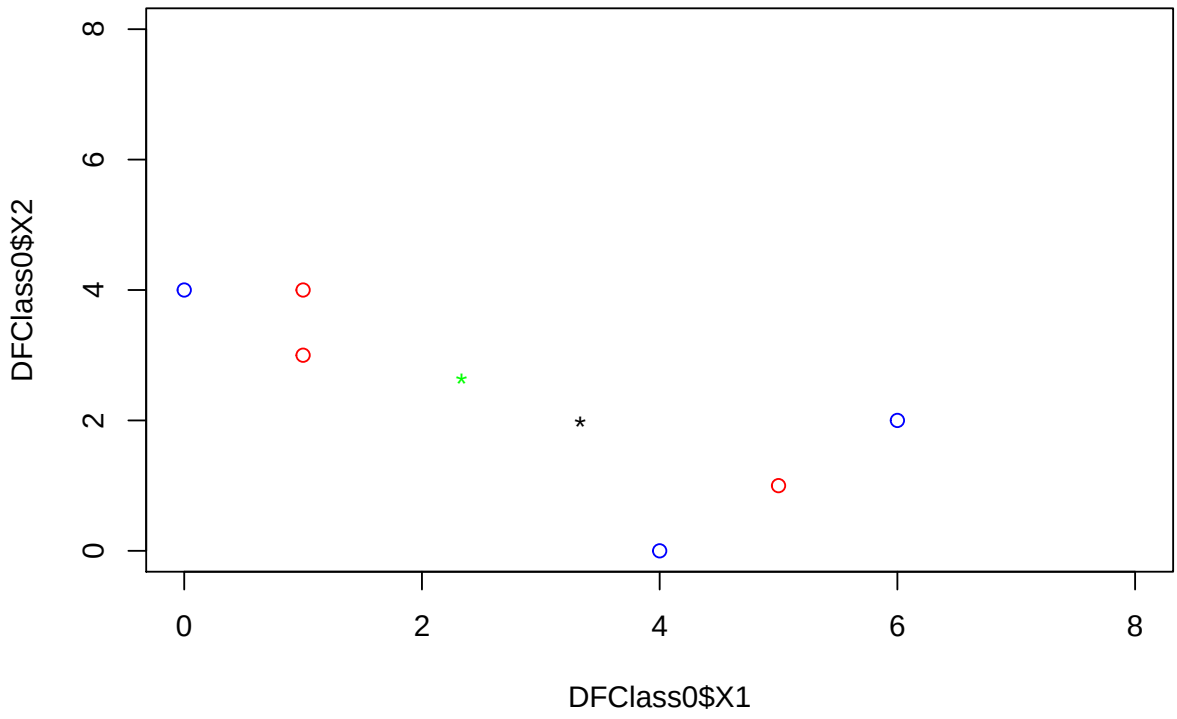


e)

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

```
oldCluster0 <- cluster0
oldCluster1 <- cluster1
numIter = 10
for (i in 1:numIter) {
    DFClass1 <- DF[cluster1, ]
    centroid1 <- colMeans(as.matrix(DFClass1))

    DFClass0 <- DF[cluster0, ]
    centroid0 <- colMeans(as.matrix(DFClass0))
    DFDist <- rbind(DFClass0, DFClass1, centroid0, centroid1)

    index <- (c(rep("class 0", nrow(DFClass0)), c(rep("class 1", nrow(DFClass1)),
        "Centroid0", "Centroid1")))

    row.names(DFDist) <- c("C01", "C02", "C03", "C11", "C12", "C13", "Centroid0",
        "Centroid1")

    dmat <- as.matrix(dist(DFDist))

    centroid0Index <- which(index == "Centroid0")
```

```
    centroid1Index <- which(index == "Centroid1")

    indicatorCentroid1_isCloser <- dmat[centroid1Index, ] < dmat[centroid0Index,
        ]

    cluster1 <- which(indicatorCentroid1_isCloser[c(-7, -8)])
    cluster0 <- which(indicatorCentroid1_isCloser[c(-7, -8)] == FALSE)

    pander(cluster0, caption = "Cluster 0")

    pander(cluster1, caption = "Cluster 1")

    numIter <- numIter - 1
    plot(DFClass0$X1, DFClass0$X2, col = "red", xlim = c(0, 8), ylim = c(0,
        8))
    points(DFClass1$X1, DFClass1$X2, col = "blue")
    points(centroid0[1], centroid0[2], pch = "*", col = "green")
    points(centroid1[1], centroid1[2], pch = "*", col = "black")

    numIter
}
```

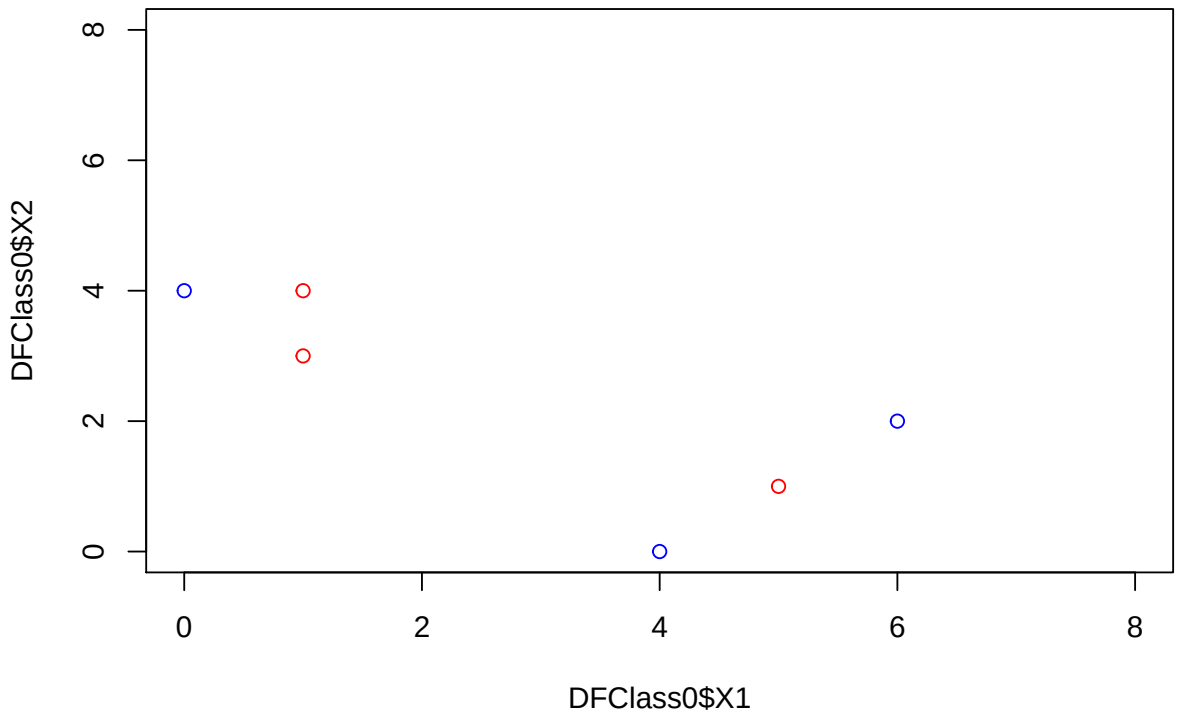


```
# We don't have to put the code above in a loop out of luck.  We'd like to
# revisit this and put the code in a loop.
```

f)

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

```
plot(DFClass0$X1, DFClass0$X2, col = "red", xlim = c(0, 8), ylim = c(0, 8))
points(DFClass1$X1, DFClass1$X2, col = "blue")
```



We MUST have a bug in our code. Sadly, we're out of time.