STA6703 SML HW9

Christopher Marais & Yu Chen

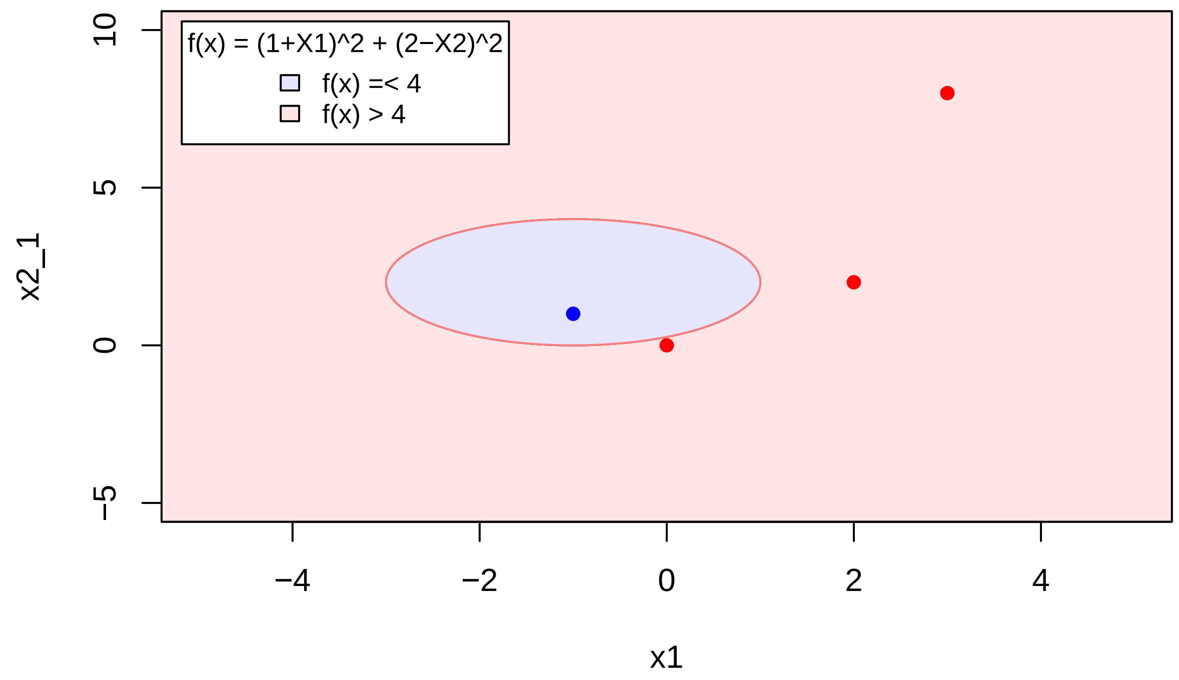
Chapter 9

Question 2

a. - c.)

```
# generate data
x1 = seq(-3,1,0.001)
x2_1 = 2-sqrt(4-(1+x1)^2)
x2_2 = 2+sqrt(4-(1+x1)^2)

# define colors
blue = rgb(0.9,0.9,1,1)
darkred = rgb(1,0.5,0.5,1)
red = rgb(1,0,0,0.1)

# plot circle function
{plot(x1,
     x2_1,
     type="l",
     ylim=c(-5,10),
     xlim=c(-5,5))
lines(x1,x2_2)

# change background color
rect(par("usr")[1],
     par("usr")[3],
     par("usr")[2],
     par("usr")[4],
     col = red)

# change color inside function
polygon(c(x1, rev(x1)),
        c(x2_1, rev(x2_2)),
        col = blue,
        border = darkred)

# add points
points(c(0,-1,2,3),c(0,1,2,8),pch=16,col=c("red","blue","red","red"))
```

```r
# add the legend
legend("topleft",
       inset=.02,
       title = "f(x) = (1+X1)^2 + (2-X2)^2",
       c("f(x) =< 4","f(x) > 4"),
       fill=c(blue, red),
       cex=0.8,
       bg="white")}
```

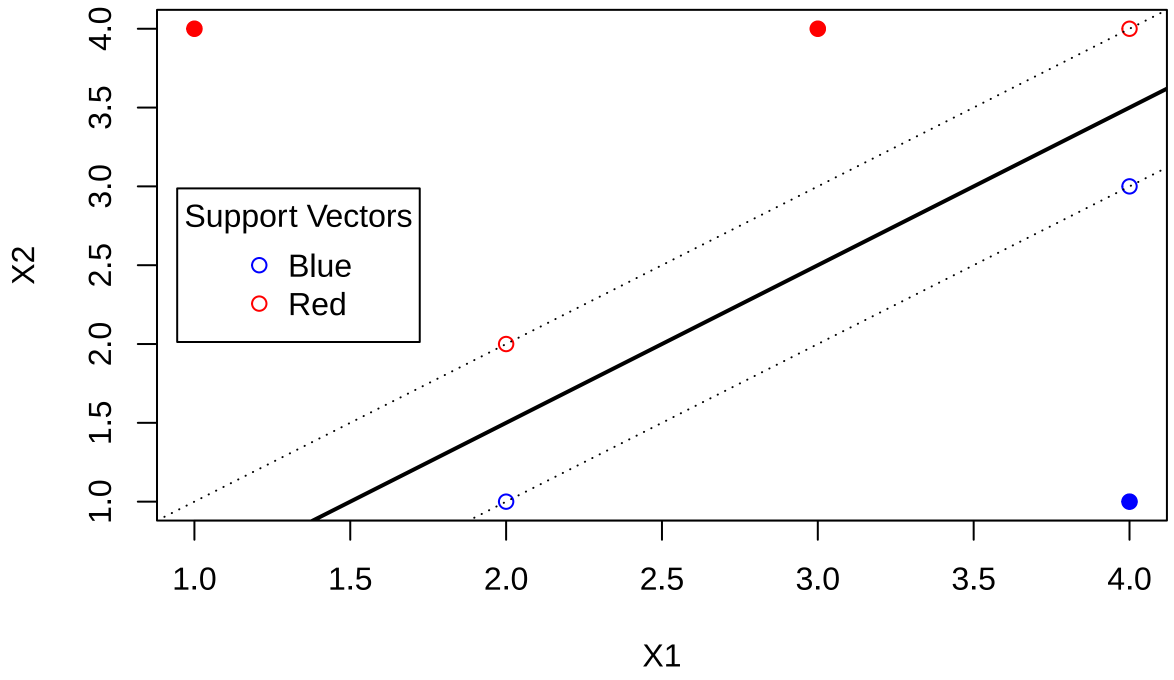


d.)

Question 3

a., b., d., e.)

```r
# define data
n = 7
p = 2

data_df = data.frame(X1=c(3,2,4,1,2,4,4),
                X2=c(4,2,4,4,1,3,1),
                Y=c('red','red','red','red','blue','blue','blue'))
# plot
{plot(data_df[,c(1,2)],col=data_df$Y, pch=1)
abline(-0.5,1, lwd=2)
abline(-1, 1, col='black',lty='dotted')
abline(0, 1, col='black',lty='dotted')
points(c(3,1),c(4,4), col="red",pch=19)
points(c(4),c(1), col="blue",pch=19)
legend("left",
       inset=.02,
       title = "Support Vectors",
       pch=c(1, 1),
       legend=c("Blue","Red"),
       col=c("blue", "red"),
       bg="white")}
```



c.)

$$-\frac{1}{2} + X_1 - X_2 > 0 \rightarrow Blue$$
$$-\frac{1}{2} + X_1 - X_2 \leq 0 \rightarrow Red$$

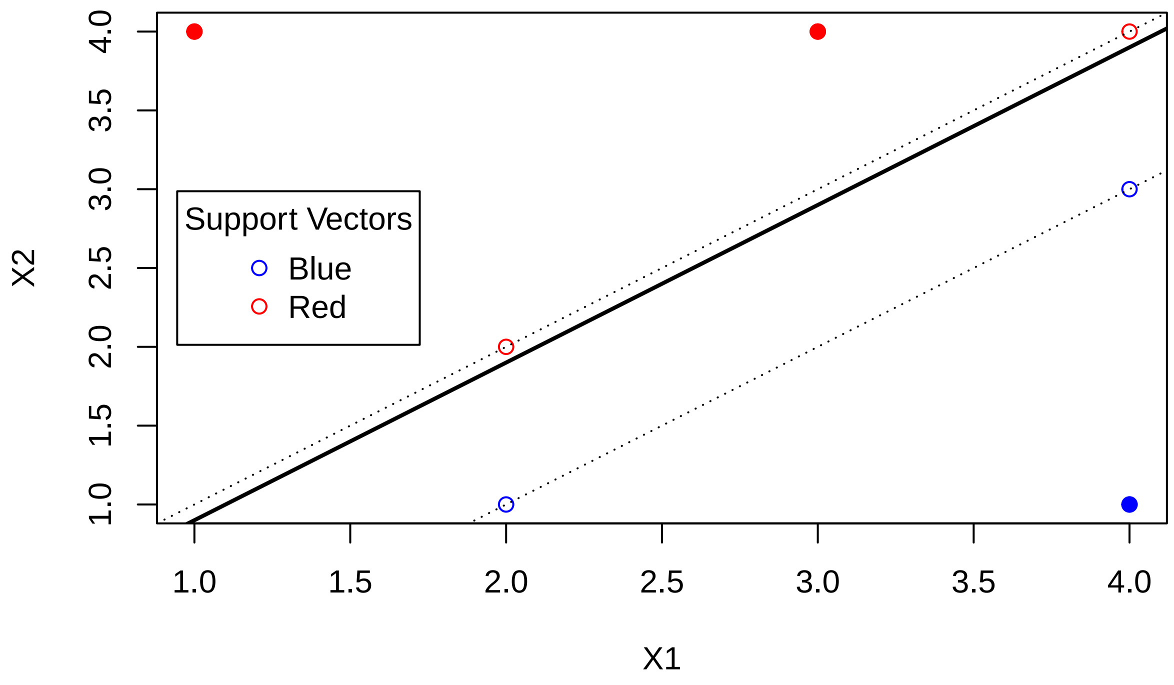
f.)

Changing the seventh point does not affect the support vectors and therefore also not the maximal margin hyperplane.

g.)

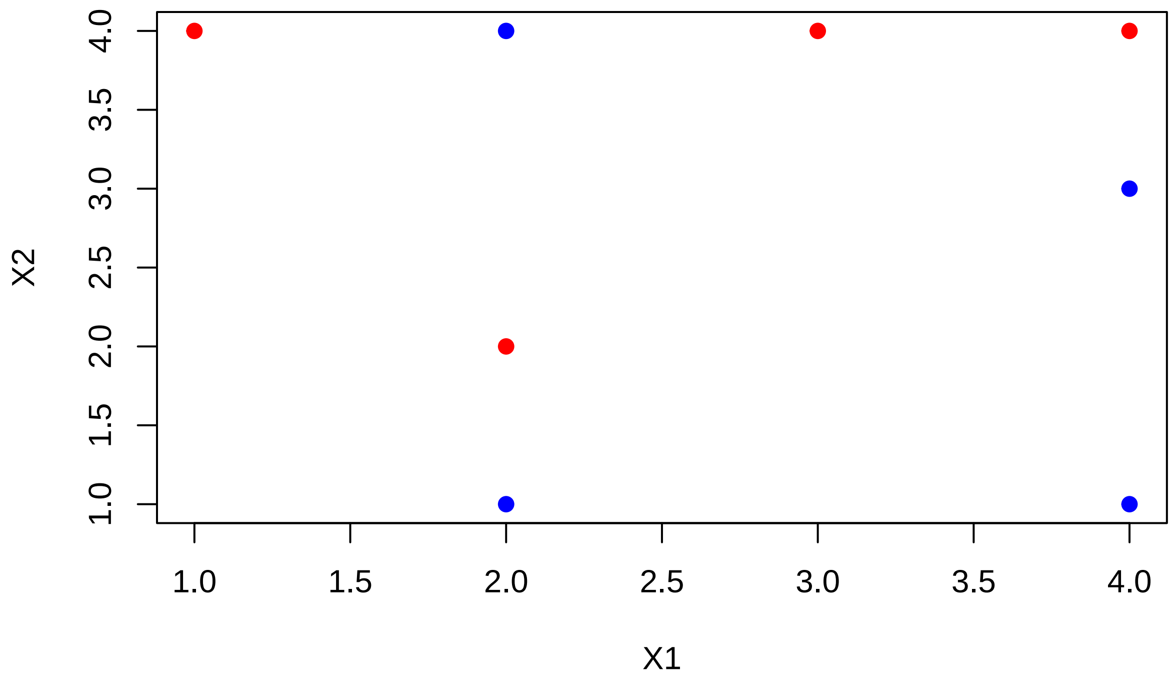
$$-0.1 + X_1 - X_2 = 0$$

```r
# plot
{plot(data_df[,c(1,2)],col=data_df$Y, pch=1)
abline(-0.1,1, lwd=2)
abline(-1, 1, col='black',lty='dotted')
abline(0, 1, col='black',lty='dotted')
points(c(3,1),c(4,4), col="red",pch=19)
points(c(4),c(1), col="blue",pch=19)
legend("left",
       inset=.02,
       title = "Support Vectors",
       pch=c(1, 1),
       legend=c("Blue","Red"),
       col=c("blue", "red"),
       bg="white")}
```



h.)

```r
# plot
{plot(data_df[,c(1,2)],col=data_df$Y, pch=19)
points(c(2),c(4), col="blue",pch=19)}
```



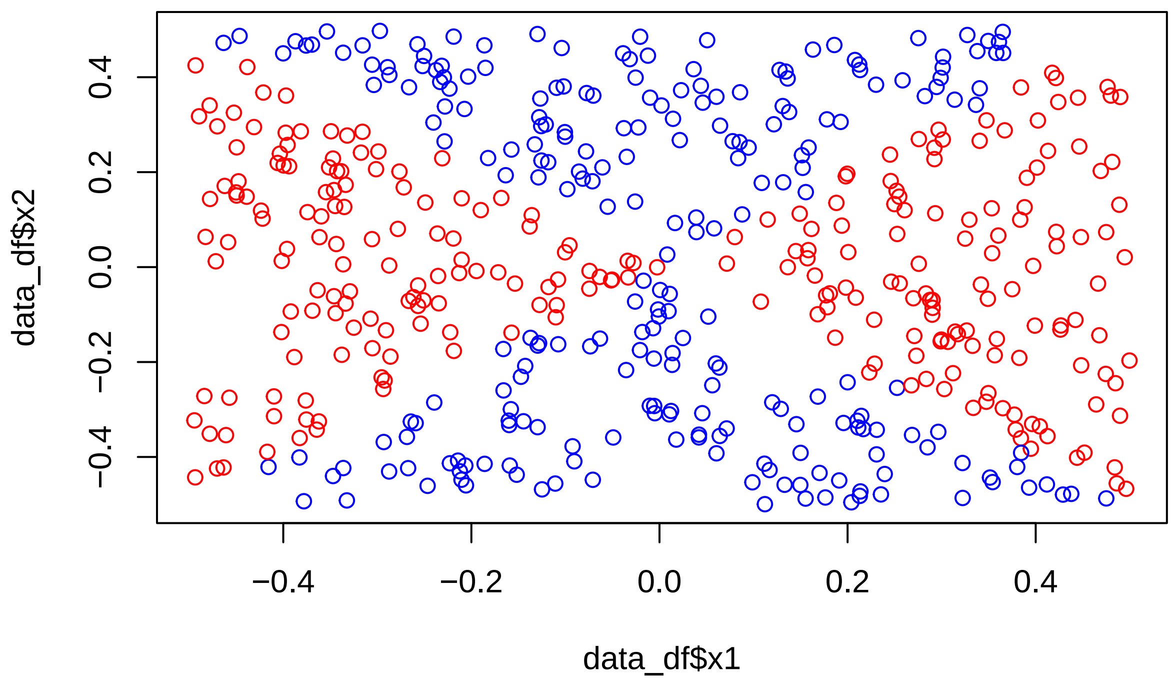
Question 5

a.)

```
# generate data
n = 500
x1= runif (n) -0.5
x2= runif (n) -0.5
y=1*( x1^2- x2 ^2 > 0)

data_df = data.frame(x1,x2,y)
```

b.)

```
plot(data_df$x1,data_df$x2,col=ifelse(y,'red','blue'))
```



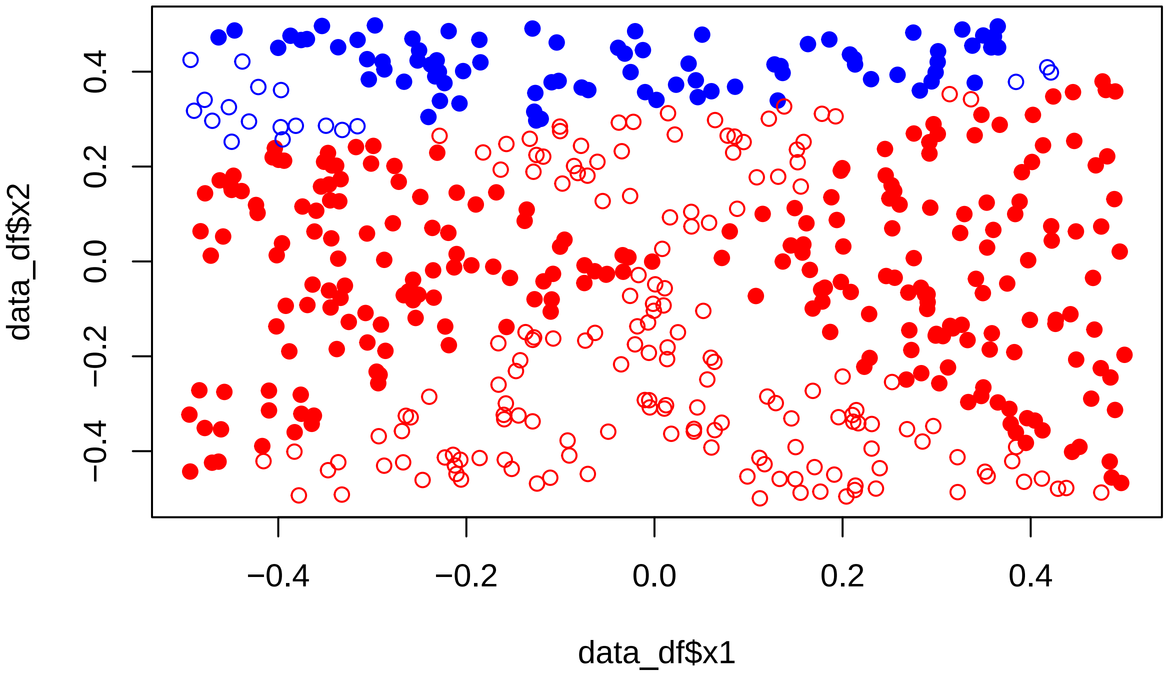
c.)

```
glm_fit = glm(y~. ,family='binomial', data=data_df)
glm_fit
```

```
##
## Call:  glm(formula = y ~ ., family = "binomial", data = data_df)
##
## Coefficients:
## (Intercept)           x1           x2
##     0.05718      0.02895     -0.18376
##
## Degrees of Freedom: 499 Total (i.e. Null);  497 Residual
## Null Deviance:       692.8
## Residual Deviance: 692.4     AIC: 698.4
```

d.)

```
glm_pred = predict(glm_fit,data_df[1:2])

plot(data_df$x1,
     data_df$x2,
     col=ifelse(glm_pred > 0,'red','blue'),
     pch=ifelse(as.integer(glm_pred > 0) == data_df$y,19,1))
```

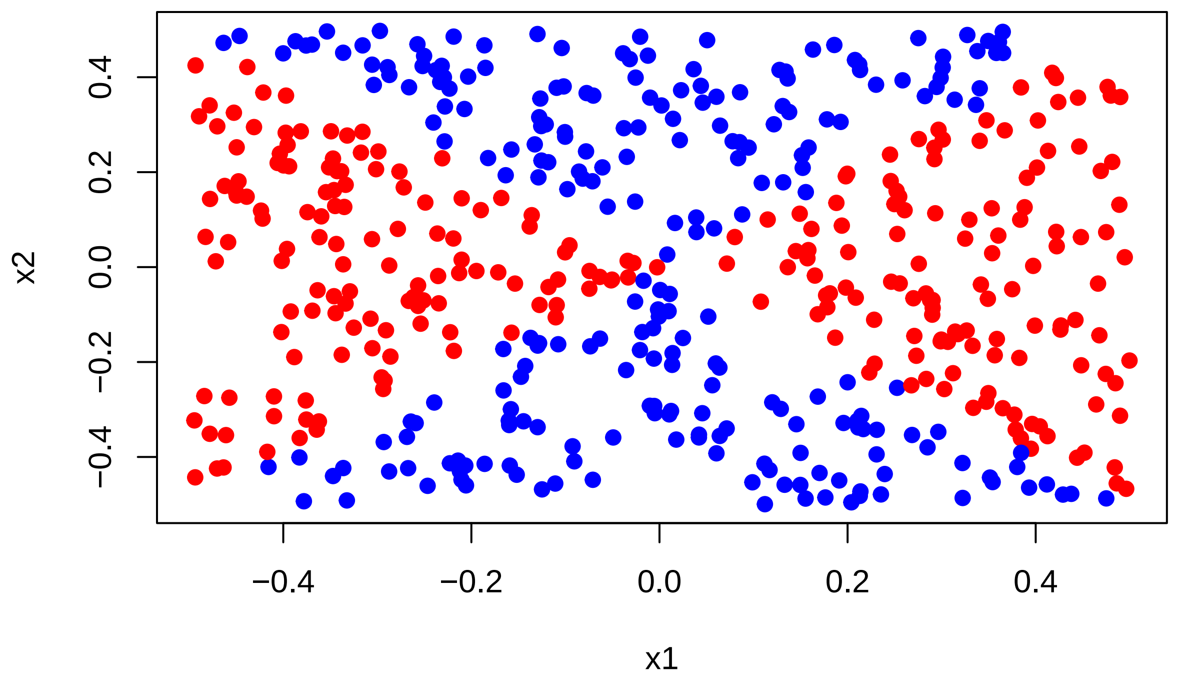


e.)

```
glm_fit=glm(y~poly(x1,2)+poly(x2,2),
            family='binomial',
            data=data_df)
```

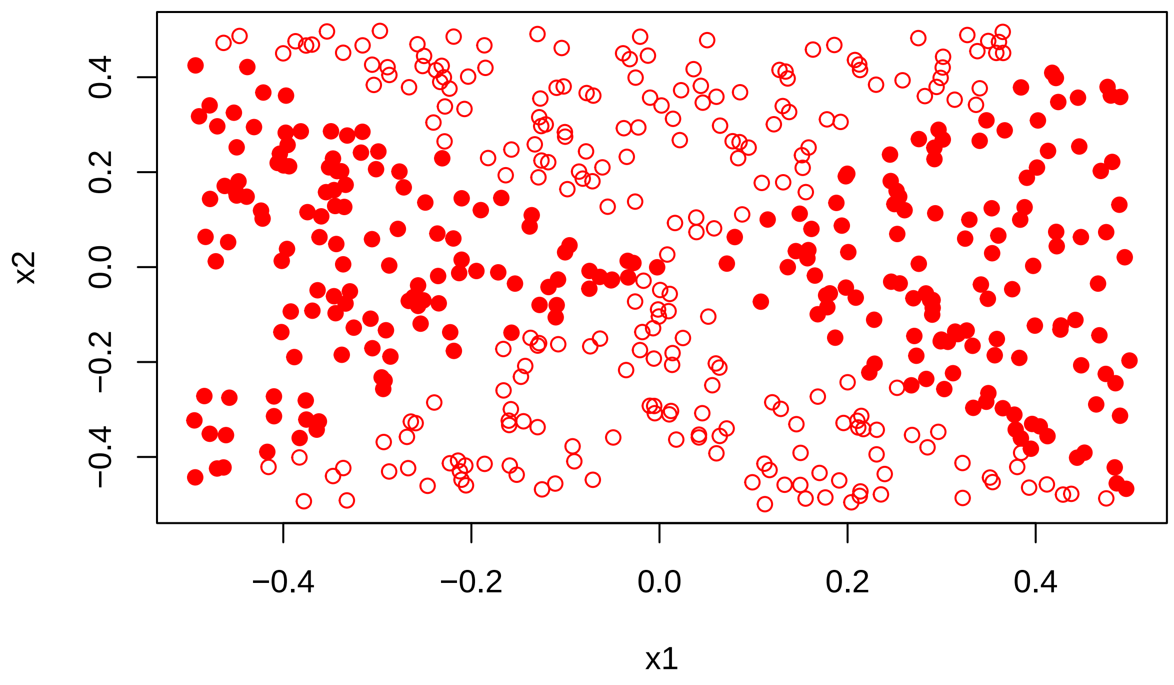
f.)

```
glm_pred=predict(glm_fit,data_df[1:2])

plot(x1,
     x2,
     col=ifelse(glm_pred > 0,'red','blue'),
     pch=ifelse(as.integer(glm_pred>0) == y,19,1))
```



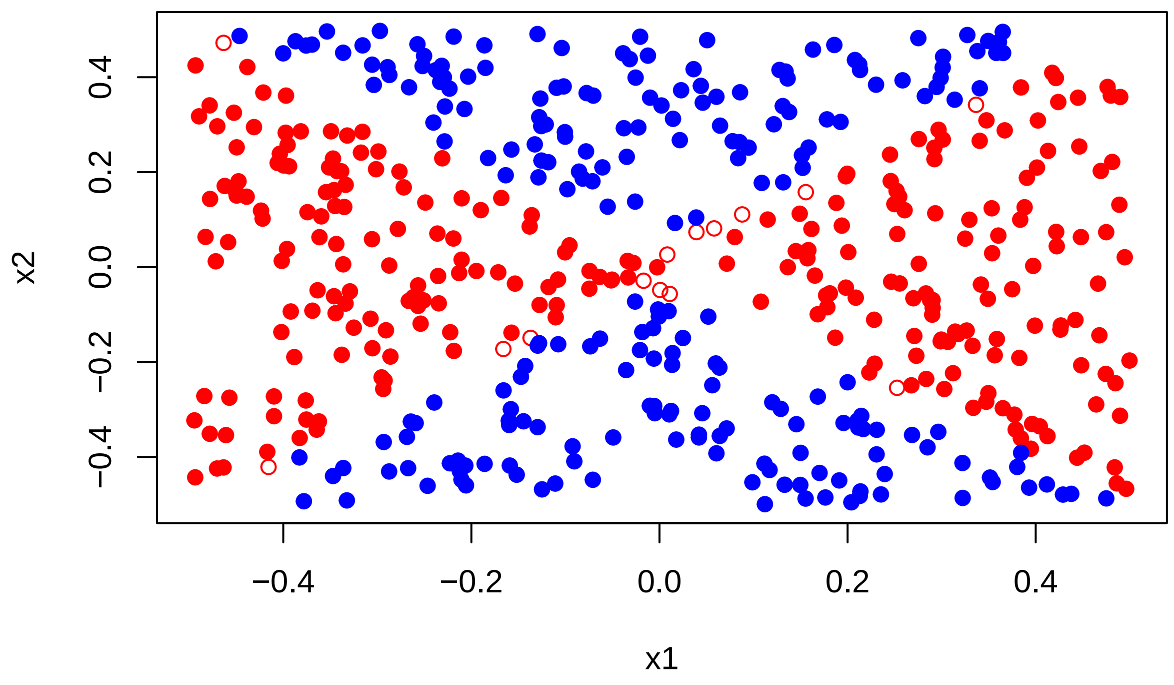
g.)

```
library(e1071)

data_df$y = as.factor(data_df$y)

svm_fit=svm(y~.,data=data_df,kernel='linear')

svm_pred=predict(svm_fit,data.frame(x1,x2),type='response')

plot(x1,
     x2,
     col=ifelse(svm_pred!=0,'red','blue'),
     pch=ifelse(svm_pred == y,19,1))
```



h.)

```
svm_fit=svm(y~.,data=data_df,kernel='polynomial', degree=2)

svm_pred=predict(svm_fit,data.frame(x1,x2),type='response')

plot(x1,
     x2,
     col=ifelse(svm_pred!=0,'red','blue'),
     pch=ifelse(svm_pred == y,19,1))
```



i.)

Question 8

a.)

```
# generate data
train=sample(1:1070,800)
test=(1:1070)[-train]
```

b.)

```
library(ISLR)

svm_fit=svm(Purchase~.,data=OJ,subset=train,cost=0.01,kernel='linear')

summary(svm_fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, cost = 0.01, kernel = "linear",
##     subset = train)
```

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  443
##
##  ( 222 221 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

c.)

```
# train error rate
svm_pred=predict(svm_fit,OJ[train,])

table(OJ[train,'Purchase'],svm_pred)
```

```
##     svm_pred
##       CH  MM
##   CH 442  48
##   MM  83 227
```

```
train_mean_err = mean(OJ$Purchase[train] != svm_pred)



# test error rate
svm_pred=predict(svm_fit,OJ[test,])

table(OJ[test,'Purchase'],svm_pred)
```

```
##     svm_pred
##       CH  MM
##   CH 144  19
##   MM  26  81
```

```
test_mean_err = mean(OJ$Purchase[test] != svm_pred)

err_vec = c()
err_vec=cbind(err_vec,'train'=train_mean_err)
err_vec=cbind(err_vec,'test'=test_mean_err)
err_vec
```

```
##        train      test
## [1,] 0.16375 0.1666667
```

d.)

```
svm_tune=tune(svm,Purchase~.,
              data=OJ[train,],
              ranges=data.frame(cost=seq(0.01,10,25)),
              kernel='linear')

summary(svm_tune)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.17375
```

```
err_vec=cbind(err_vec,'CV'=svm_tune$best.performance)
```

e.)

```
svm_pred=predict(svm_tune$best.model,OJ[train,])

table(OJ[train,'Purchase'],svm_pred)
```

```
##     svm_pred
##       CH  MM
##   CH 442  48
##   MM  83 227
```

```
train_mean_err_tuned = mean(OJ$Purchase[train] != svm_pred)

err_vec=cbind(err_vec,'train_tuned'=train_mean_err_tuned)



svm_pred=predict(svm_tune$best.model,OJ[test,])

table(OJ[test,'Purchase'],svm_pred)
```

```
##     svm_pred
##       CH  MM
##   CH 144  19
##   MM  26  81
```

```
test_mean_err_tuned = mean(OJ$Purchase[test] != svm_pred)

err_vec=cbind(err_vec,'test_tuned'=test_mean_err_tuned)

err_vec
```

```
##         train      test      CV train_tuned test_tuned
## [1,] 0.16375 0.1666667 0.17375     0.16375  0.1666667
```

f.)

g.)

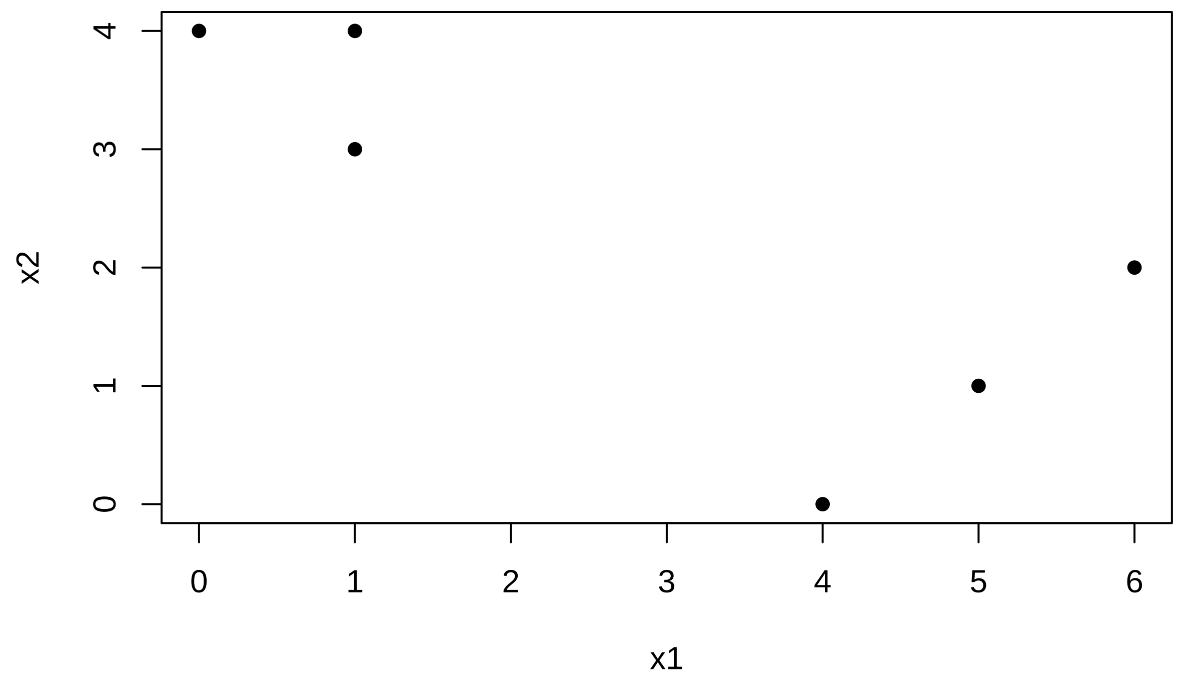
h.)

Chapter 10

Question 3

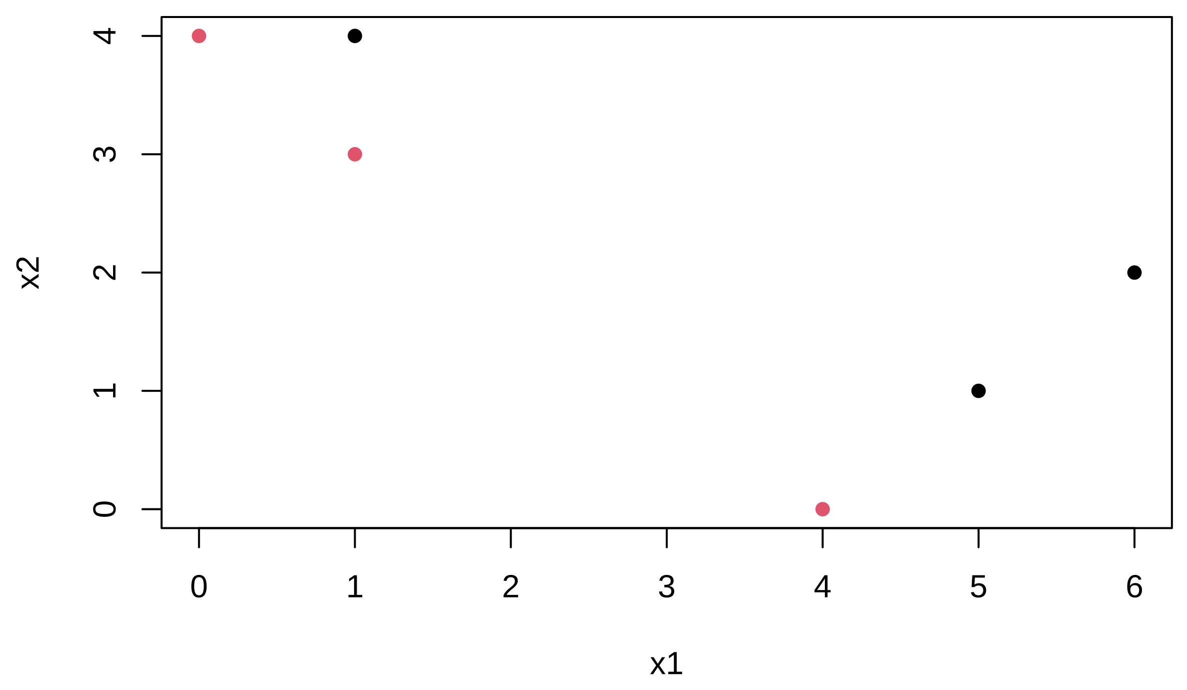
a.)

```
# parameters
K = seq(2)
n = 6
p = 2

# data
obs = seq(1,n)
x1 = c(1,1,0,5,6,4)
x2 = c(4,3,4,1,2,0)
df = data.frame(x1,x2)

plot(df, pch=16)
```



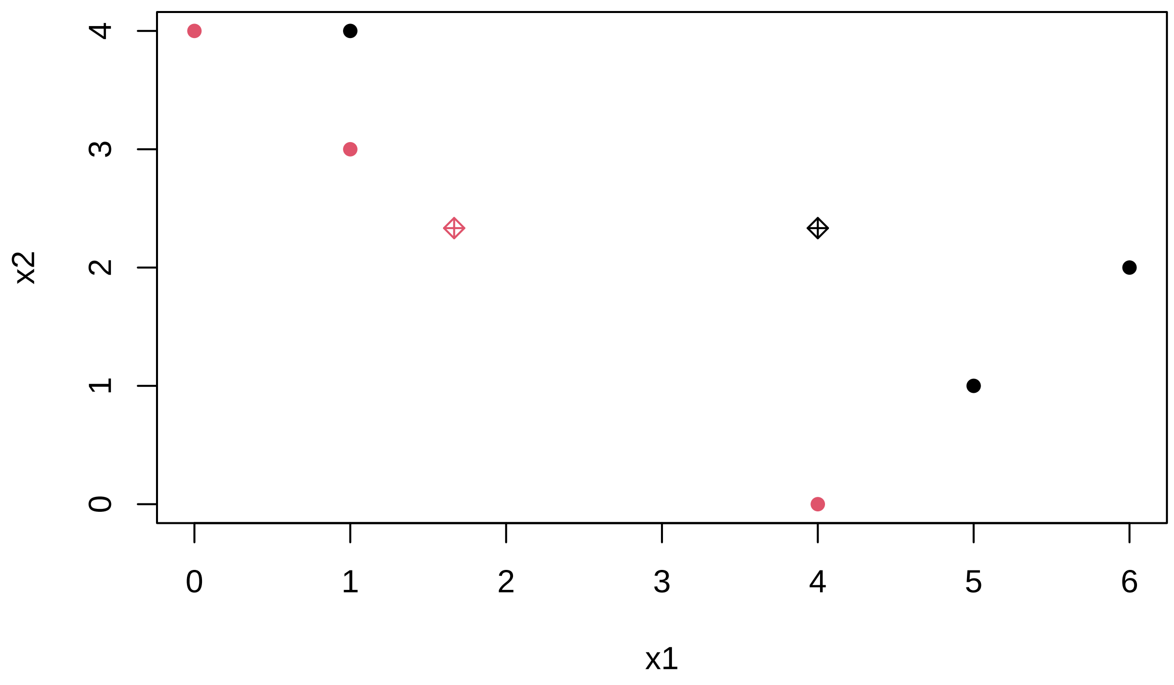
b.)

```
# initialize class
set.seed(0)
df$label = sample(x=K,
                  size=6,
                  replace = TRUE,
                  prob=rep(1/length(K), length(K)))

plot(df[K], col=df$label, pch=16)
```



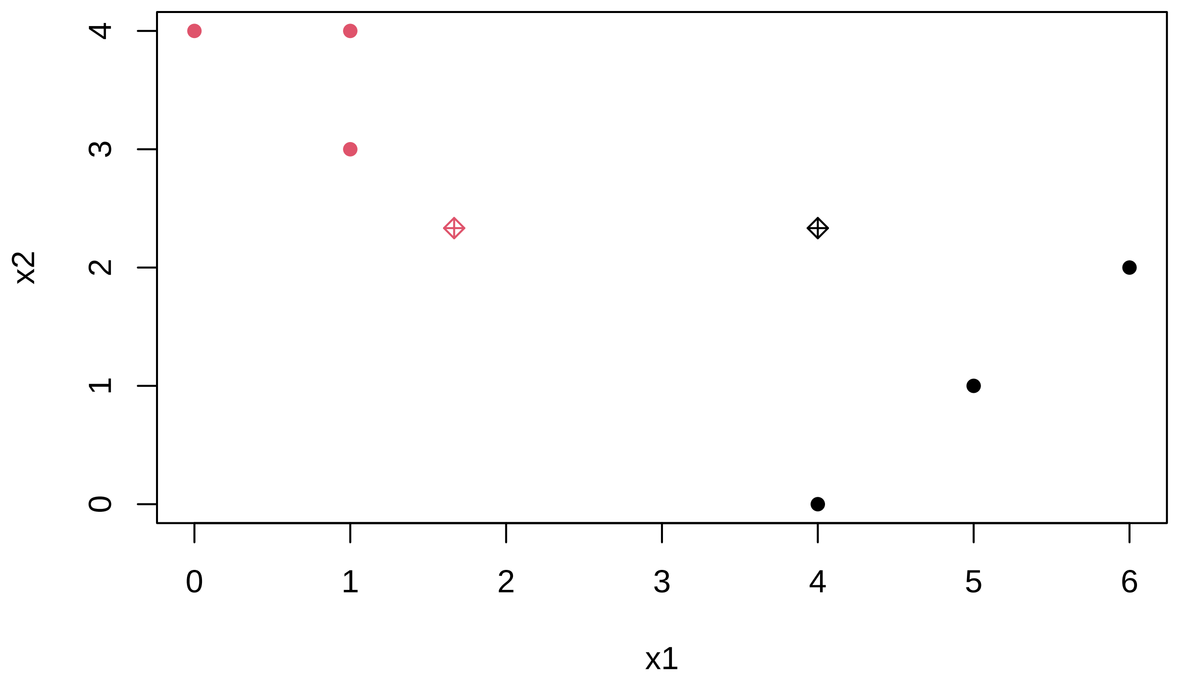
c.)

```r
# centroid function
comp_centroid <- function(k=K, data=df){
  centroid_df = data.frame()
  for(i in k){
    class_df = data[data$label == i,]
    centroid_k = t(data.frame(c(mean(class_df$x1), mean(class_df$x2))))
    centroid_df = rbind(centroid_df, centroid_k)
  }

  rownames(centroid_df) = k
  colnames(centroid_df) = colnames(data[-ncol(data)])

  # plot results
  {plot(df[K], col=df$label, pch=16)
  points(centroid_df, col=rownames(centroid_df), pch=9)}

  return(centroid_df)
}

# calculate centroid
centroid_df = comp_centroid(k=K, data=df)
```



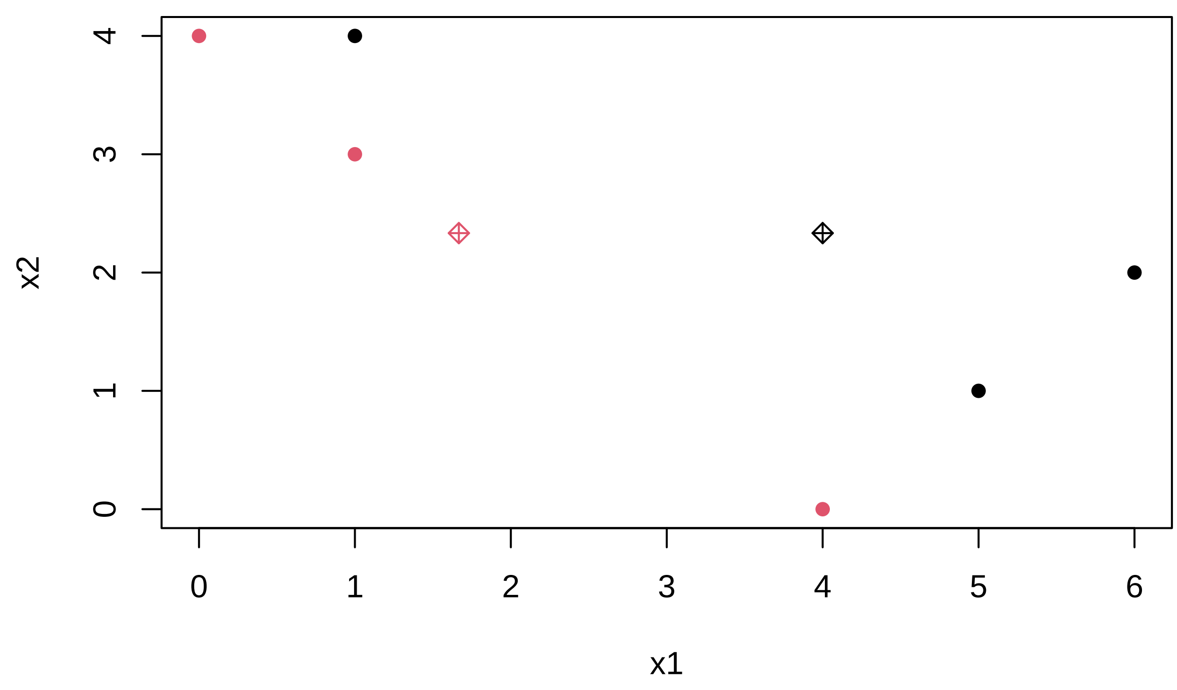
d.)

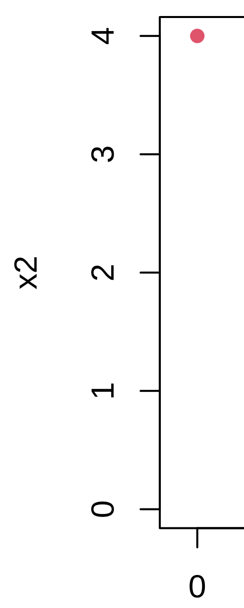
```r
# centroid distance and label update function
closest_centroid <- function(points_data, centroids){
  K_len=nrow(centroids)
  n=nrow(points_data)
  coord_df = rbind(points_data[-ncol(points_data)], centroids)
  dist_mat = as.matrix(dist(coord_df, method="euclidean"))
  dist_df = as.data.frame(dist_mat[1:n,(n+1):(n+K_len)])
  colnames(dist_df) = seq(1:K_len)
  points_data$label =  c(t(apply(dist_df,1,which.min)))

  # plot results
  {plot(points_data[K], col=points_data$label, pch=16)
  points(centroids, col=rownames(centroids), pch=9)}

  return(points_data)
}

# update label to closest centroid
df_updated = closest_centroid(points_data=df, centroids=centroid_df)
```



e.)

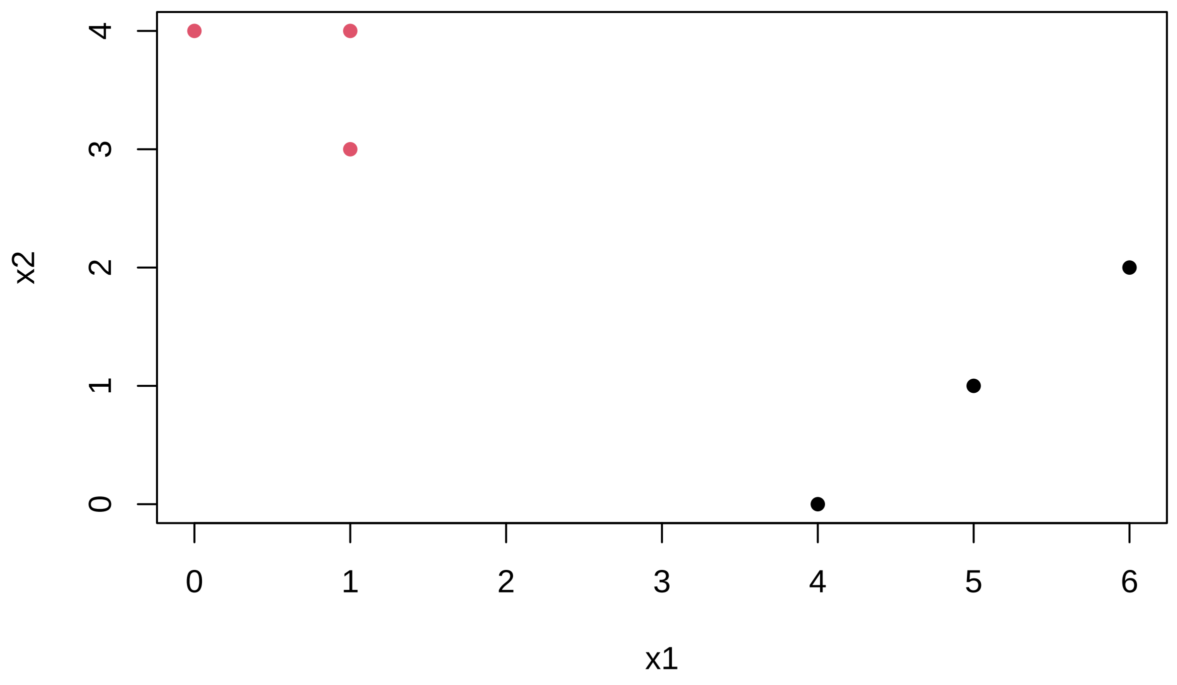
```
# repeat centroid and label updates until no change in labels
df_updated = data.frame(label=(rep(0, nrow(df))))
while(!all(df$label == df_updated$label)){
  centroid_df = comp_centroid(k=K, data=df)
  df_updated = closest_centroid(points_data=df, centroids=centroid_df)
  df <- df_updated
}
```





f.)

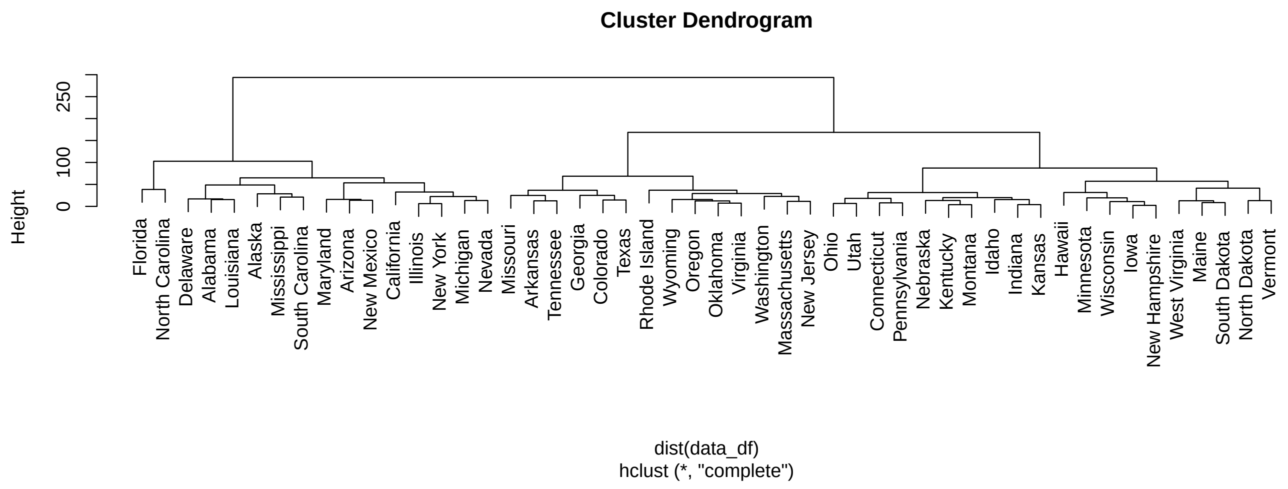
```r
# plot final classes by color without centroids
plot(df[K], col=df$label, pch=16)
```



Question 9

a.)

```
data_df = USArrests

h_clust <- hclust(dist(data_df),method="complete")
plot(h_clust)
```



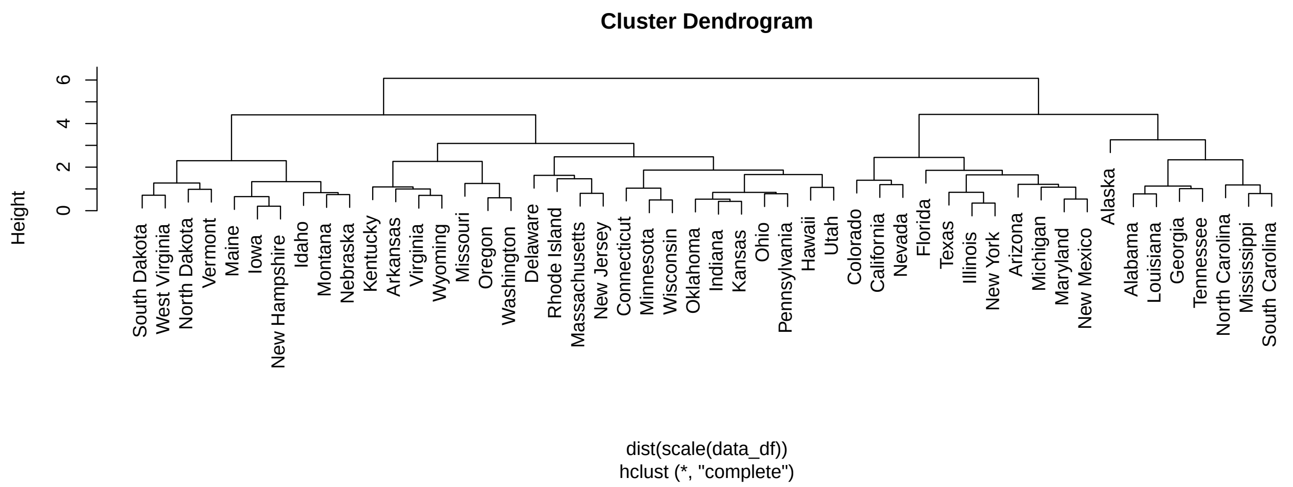
b.)

```r
h_clust_cut = cutree(h_clust,k=3)
h_clust_cut
```

```
##        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.)

```r
h_clust_scale = hclust(dist(scale(data_df)), method="complete")

plot(h_clust_scale)
```



d.)

Scaling makes the dendogram much shorter. Scaling is necessary because different units have been used for the different features. Scaling should be done before the inter-observation dissimilarities are computed to compensate for the different units used among the features.

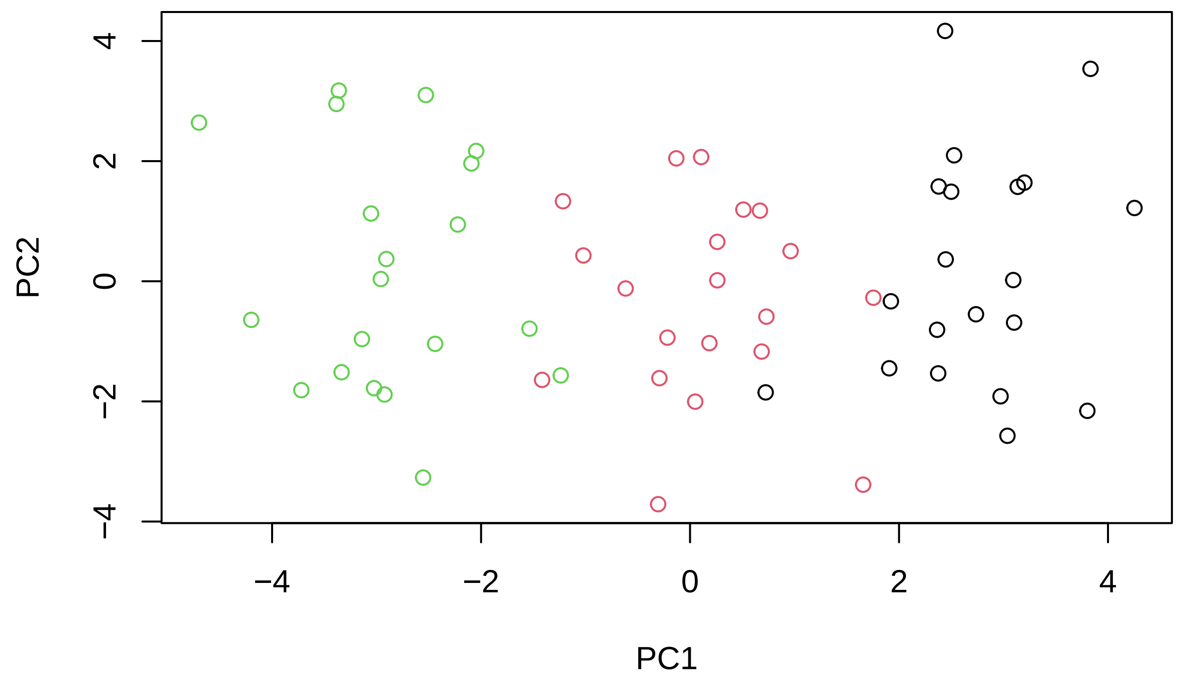
Question 10

a.)

```
set.seed(0)

data_df = data.frame(matrix(c(
                      rnorm(20*50,mean = -0.3),
                      rnorm(20*50,mean = 0),
                      rnorm(20*50,mean = 0.3)),
                    ncol = 50,
                    byrow = T))

data_df$label <- rep(1:3,each=20)
```

b.)

```
pca_comp = prcomp(data_df)

plot(pca_comp$x[,c(1,2)],col=data_df$label)
```



c.)

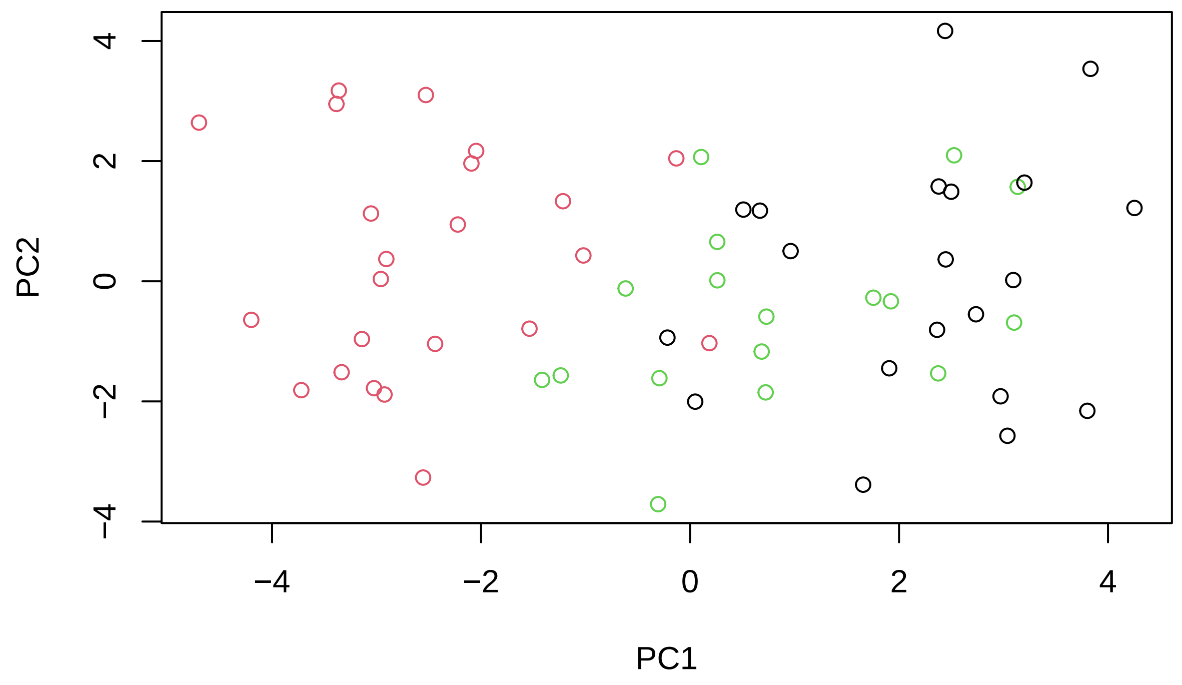
```
set.seed(0)

kmeans_comp = kmeans(data_df,3)

table(kmeans_comp$cluster)
```

```
##
##  1  2  3
## 20 23 17
```

```
table(data_df$label)
```

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

```
plot(pca_comp$x[,c(1,2)],col=kmeans_comp$cluster)
```



It seems there are 3 misclassified observations. K-means aslo seemsto swap around two classes.

d.)

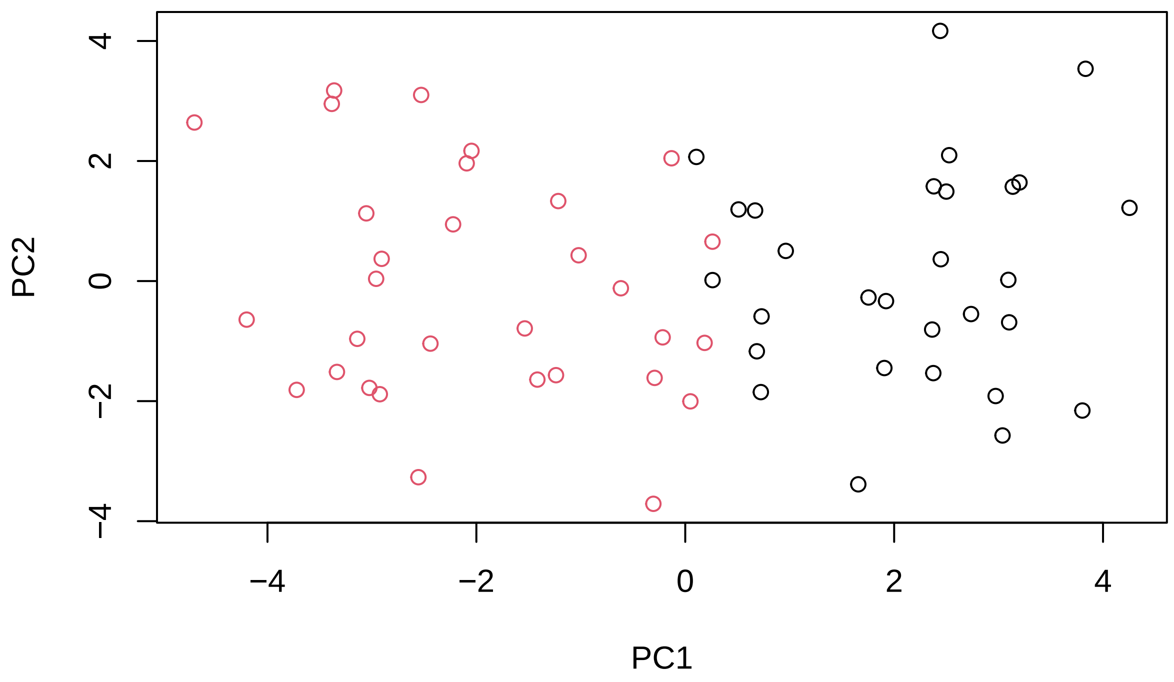
```
set.seed(0)

kmeans_comp=kmeans(data_df,2)

table(kmeans_comp$cluster)
```

```
##
##  1  2
## 29 31
```

```
table(data_df$label)
```

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

```
plot(pca_comp$x[,c(1,2)],col=kmeans_comp$cluster)
```



K-means divides the middle cluster almost evenly to the two other classes that are found on either side.

e.)

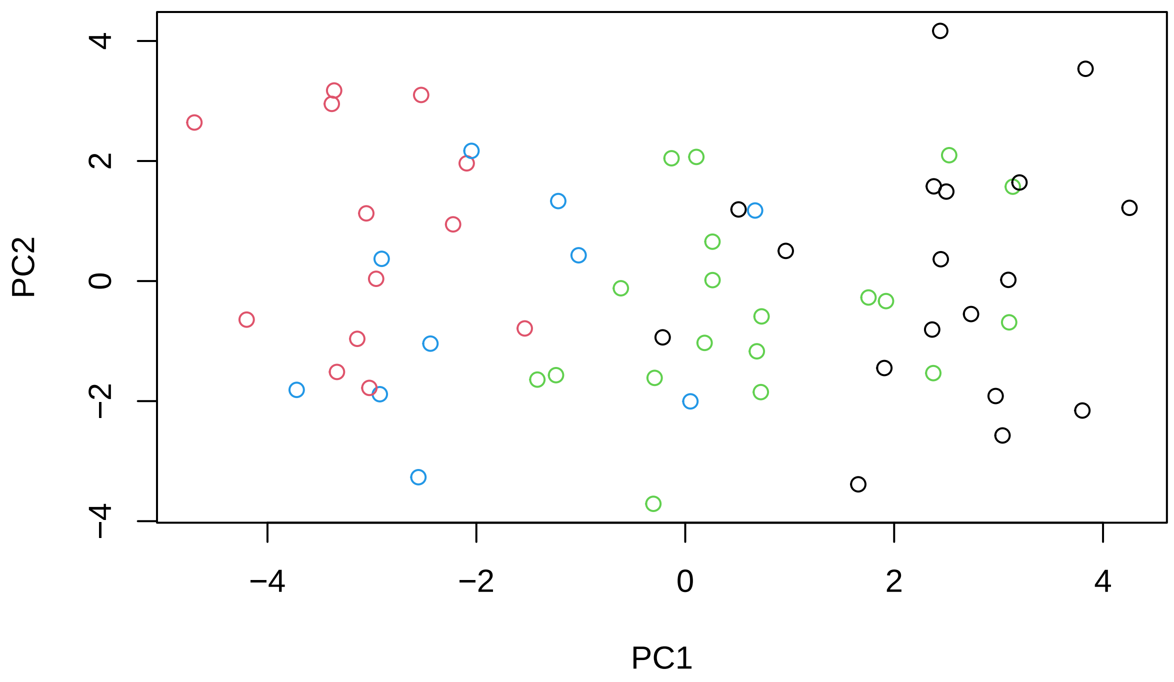
```
set.seed(0)

kmeans_comp=kmeans(data_df,4)

table(kmeans_comp$cluster)
```

```
##
##  1  2  3  4
## 18 13 19 10
```

```
table(data_df$label)
```

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

```
plot(pca_comp$x[,c(1,2)],col=kmeans_comp$cluster)
```



When using 4 clusters it is no longer clear where the classes are and it seems more fragmented and less accurate to the true data.

f.)

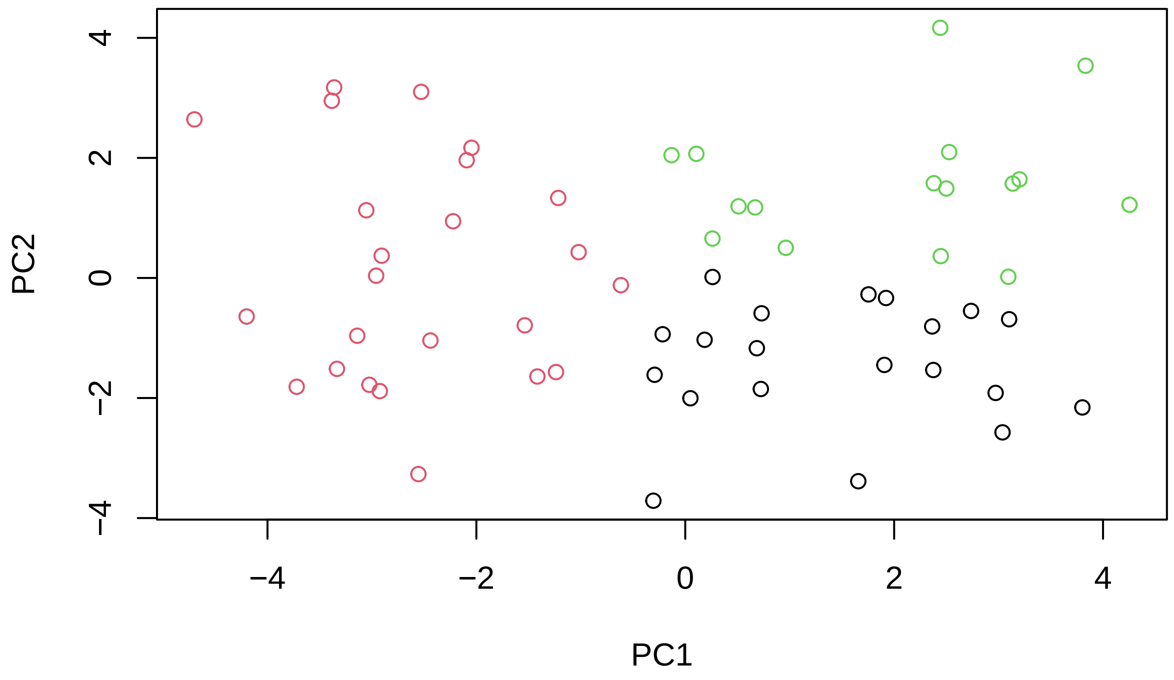
```
set.seed(0)

kmeans_comp=kmeans(pca_comp$x[,c(1,2)],3)

table(kmeans_comp$cluster)
```

```
##
##  1  2  3
## 20 24 16
```

```
table(data_df$label)
```

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

```
plot(pca_comp$x[,c(1,2)],col=kmeans_comp$cluster)
```



We can see that the data is less accurately classified when only using a small portion of the features. the msising features carry some inforamtion that goes missing when left out. There seesm to be an overlap in the clusters now.

g.)

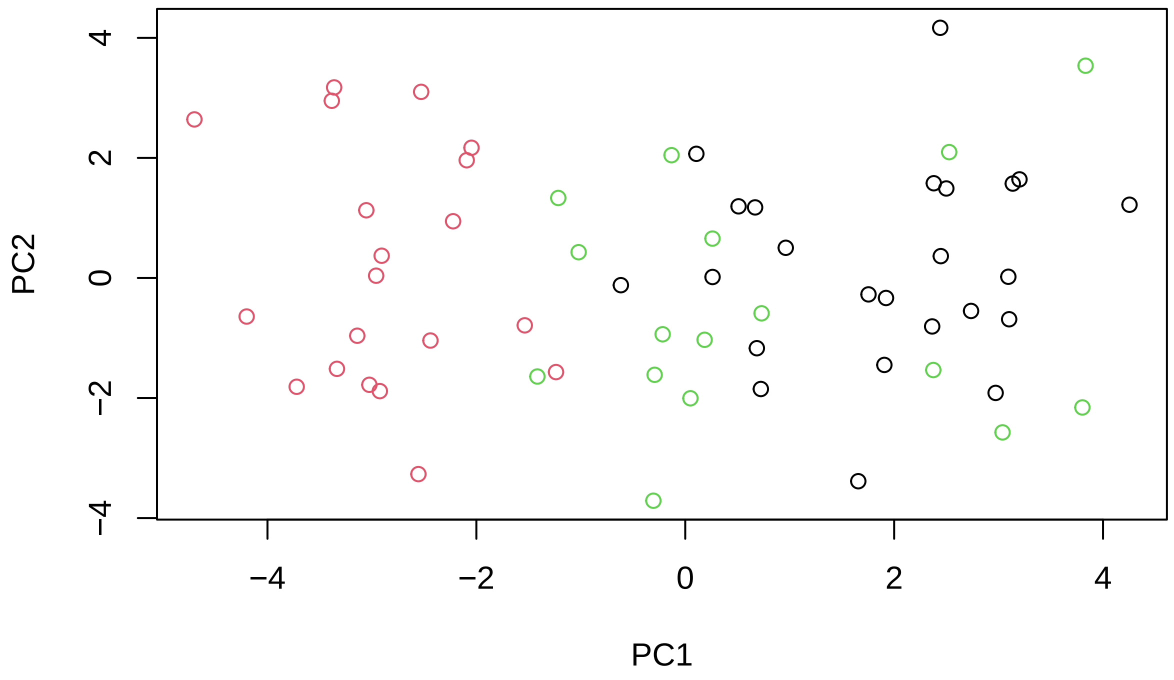
```
set.seed(0)

kmeans_comp=kmeans(scale(data_df,center = T,scale = T),3)

table(kmeans_comp$cluster)
```

```
##
##  1  2  3
## 24 20 16
```

```
table(data_df$label)
```

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

```
plot(pca_comp$x[,c(1,2)],col=kmeans_comp$cluster)
```



There is a large overlap in the clusters and the algorithm seems otreally not perform very well.