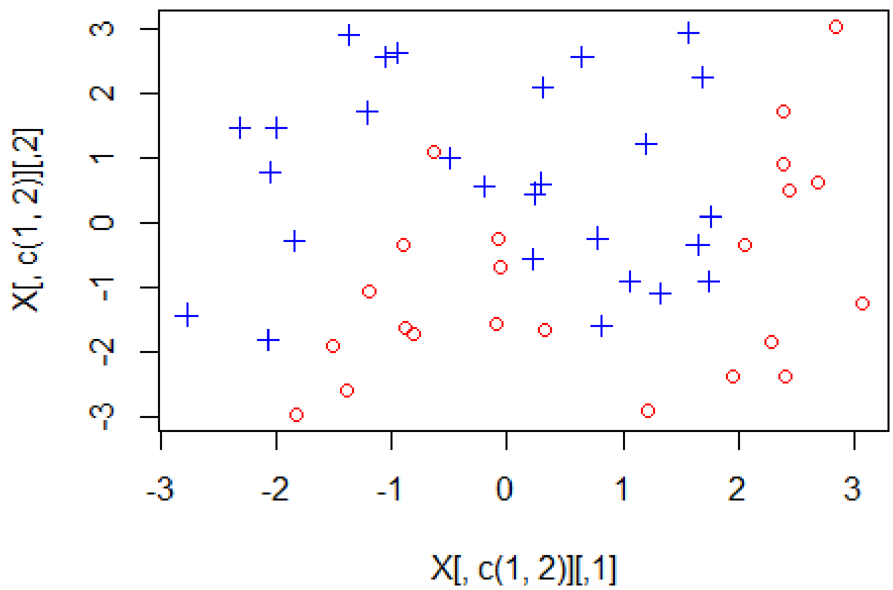
B565_hw6

2

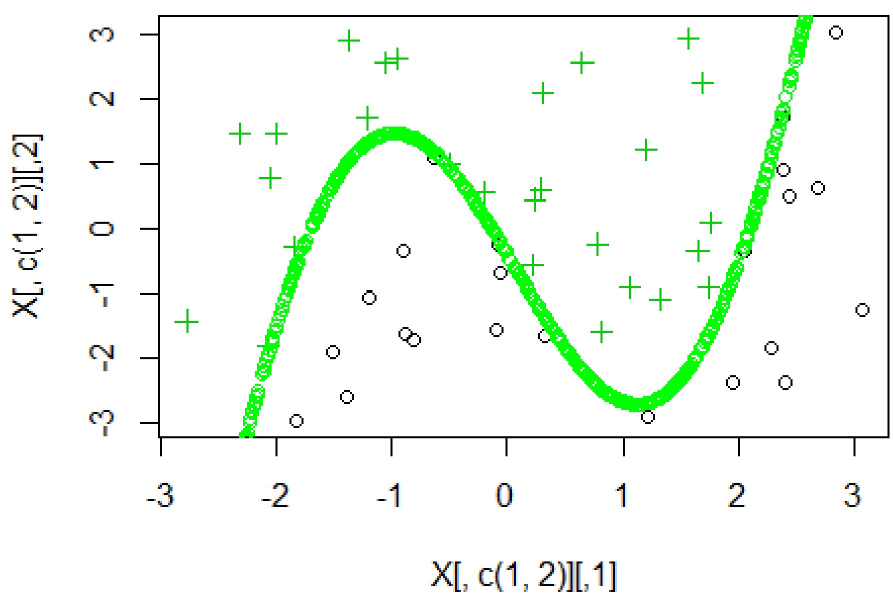
```
library("quadprog")

# (a)
# Create w vector
set.seed(100)
w = rnorm(5,0,1)

# Create X of 50X5 matrix, b and Y as class of 50X1 vector
set.seed(100)
X = matrix(0,nrow = 50, ncol = 5)
Y = rep(0,50)

# Update values of X columns
X[,1] = runif(50,min=-3.14,max=3.14)
X[,2] = runif(50,min=-3.14,max=3.14)
X[,3] = cos(X[,1])
X[,4] = sin(X[,1])
X[,5] = rep(1,50)


# Assign sign to Y based on contraints
Y = as.vector(sign(X%*%w))


# Plot data X1 with different colours for respective classes
plot(X[,c(1,2)],pch=(Y+2),col=(Y+3),cex=1)
```




```r
# (b)
# Solve using quadratic programming

Dmat = diag(5)
Dmat[5,5] = 0.0000001
dvec = rep(0,5)
bvec = rep(1,50)

Amat = as.matrix(X)
A = Amat*Y
result = solve.QP(Dmat,dvec,t(A),bvec)

what = result$solution[1:5]

what

## [1] -37.358591  12.728791  -5.443255  75.626658  10.469340

# Predict for class using result obtained from QP
predict = as.vector(sign(X%*%what))
predict

##  [1] -1 -1 -1  1  1 -1 -1 -1  1  1  1 -1  1 -1  1  1  1 -1 -1  1  1  1  1
## [24]  1  1  1  1 -1  1 -1 -1 -1  1 -1 -1 -1  1  1 -1  1  1 -1  1 -1  1 -1
## [47]  1 -1 -1  1

Y
```

```
##   [1] -1 -1 -1  1  1 -1 -1 -1  1  1  1 -1  1 -1  1  1  1 -1 -1  1  1  1  1
## [24]  1  1  1  1 -1  1 -1 -1 -1  1 -1 -1 -1  1  1 -1  1  1 -1  1 -1  1 -1
## [47]  1 -1 -1  1
```

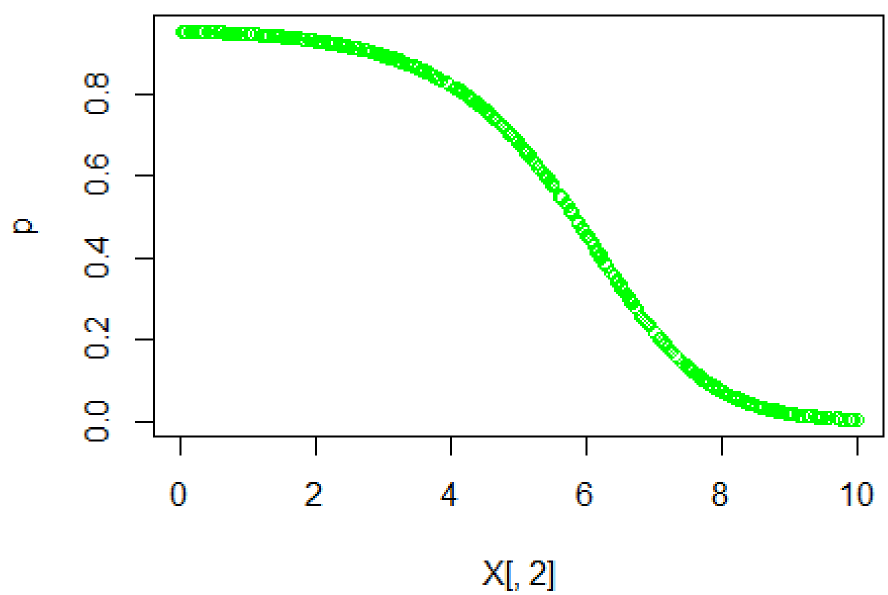
```
# (c)
# Plot decision boundary which maximize margin

a = runif(1000,min=-3.14,max=3.14)
b = (-what[5] - what[1]*a - what[3]*cos(a) - what[4]*sin(a))/what[2]

plot(X[,c(1,2)],pch=(Y+2),col=(Y+2),cex=1)
lines(a,b,col="green",type="p")
```



3

```
library("MASS")
# (a)

# Create data matrix
a = scan("D:/Data Mining/hw6/chipotle.dat")
data = matrix(a,byrow=TRUE,ncol=2)
data1 = head(data,500)


# Creating resulting dataframe using cbind
df = cbind(rep(1,nrow(data1)),data1[,1],data1[,1]^2,data1[,2])
```

```r
colnames(df) = c("a","X1","X2","Y")

# X vector
X = df[,c(1,2,3)]

# Initialising w_0

w = matrix(c(3,-.05,-0.08),ncol=1)

# Calculating probability and ploting w.r.t x

p = 1/(1+exp(-t(w) %*% t(X)))
plot(X[,2],p,col="green")
```



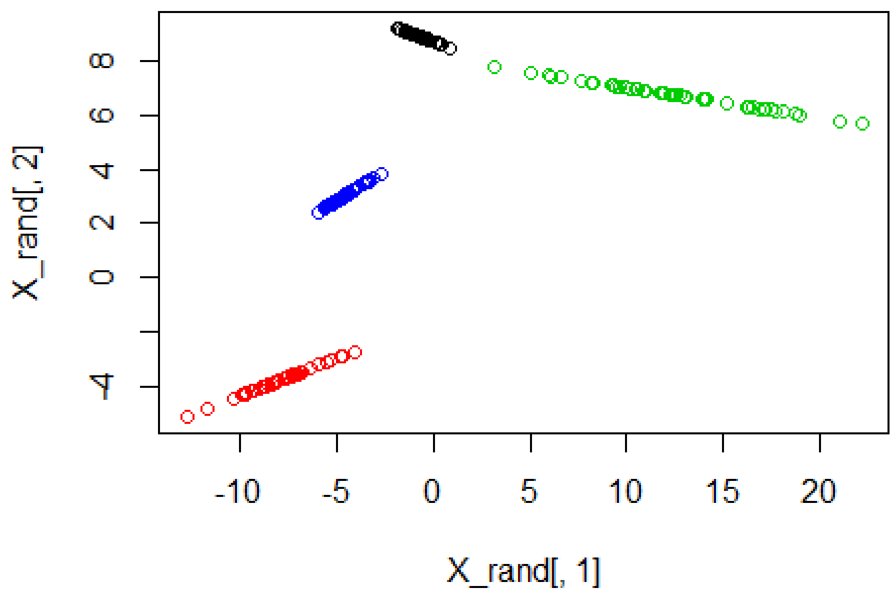

```r
# (b)

# Gradient descent to optmize w
class = df[,4]
set.seed(100)
w = rep(0,3)
for (i in 1:1000) {
  p = 1/(1+exp(-t(w) %*% t(X)))
  grad = t(X) %*% t(class - p)
  w = w + .01 * grad
  pred = ifelse(p >0.5, 1, 0)
  accuracy = sum(pred==class)/length(class)
```

```r
}

w

##          [,1]
## a   37.87352
## X1 229.77711
## X2 -28.88233

# Pred class 1 if p > 0.5 else class 0 and calculate accuracy
pred = ifelse(p >0.5, 1, 0)
accuracy = sum(pred==class)/length(class)
accuracy

## [1] 0.818

# (c)

# Initialising w
class1 = df[,4]
set.seed(100)
w1 = rep(0,3)

# Newton ralphson method to optmize w
for (i in 1:20) {
  p1 = 1/(1+exp(-t(w1) %*% t(X)))
  p1 = as.vector(p1)
  D = diag(p1*(1-p1))
  H = -t(X) %*% D %*% X
  grad1 = t(X) %*% (class1 - p1)
  w1 = w1 - ginv(H)%*%grad1
}

w1

##             [,1]
## [1,]  2.2067068
## [2,]  0.3801537
## [3,] -0.1259208

# Pred class 1 if p > 0.5 else class 0 and calculate accuracy

pred1 = ifelse(p1 >=0.5, 1, 0)
accuracy1 = sum(pred1==class1)/length(class1)
accuracy1

## [1] 0.854
```

4

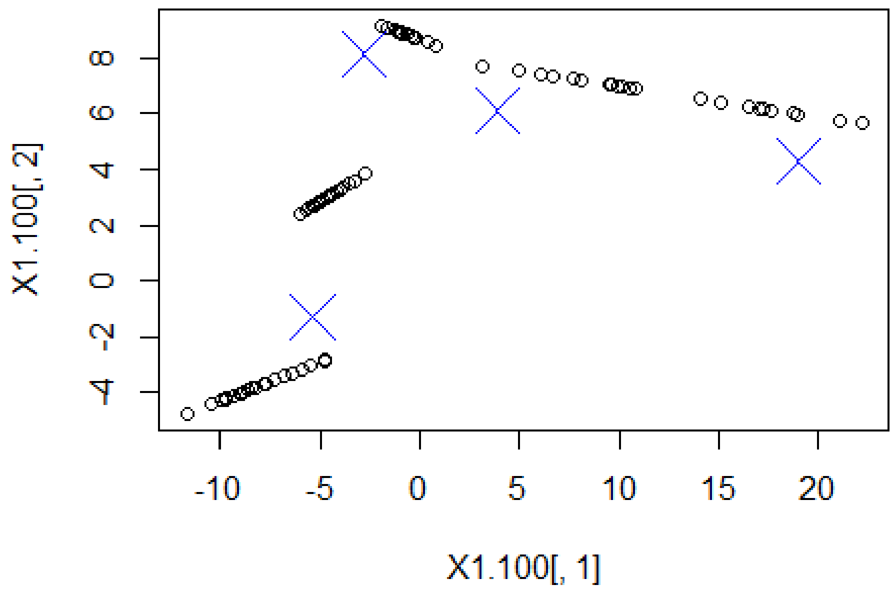
```
# (a)

# Generated 1st cluster points
X1 = matrix(0, nrow = 50, ncol=2)
C1 = rep(1,50)
t1 = matrix(rnorm(2),ncol = 2, nrow = 2)
b1 = matrix(rnorm(2,0,10),ncol = 1, nrow = 2)
for (i in 1:50){
  z1 = matrix(rnorm(2),ncol = 1, nrow = 2)

  X1[i,] = t1%*%z1 + b1
}
X1 = cbind(X1,C1)

# Generated 2nd cluster points
X2 = matrix(0, nrow = 50, ncol=2)
C2 = rep(2,50)
t2 = matrix(rnorm(2),ncol = 2, nrow = 2)
b2 = matrix(rnorm(2,0,10),ncol = 1, nrow = 2)
for (i in 1:50){
  z2 = matrix(rnorm(2),ncol = 1, nrow = 2)

  X2[i,] = t2%*%z2 + b2
}

X2 = cbind(X2,C2)


# Generated 3rd cluster points
X3 = matrix(0, nrow = 50, ncol=2)
C3 = rep(3,50)
t3 = matrix(rnorm(2),ncol = 2, nrow = 2)
b3 = matrix(rnorm(2,0,10),ncol = 1, nrow = 2)
for (i in 1:50){
  z3 = matrix(rnorm(2),ncol = 1, nrow = 2)

  X3[i,] = t3%*%z3 + b3
}
X3 = cbind(X3,C3)


# Generated 4th cluster points
X4 = matrix(0, nrow = 50, ncol=2)
C4 = rep(4,50)
t4 = matrix(rnorm(2),ncol = 2, nrow = 2)
b4 = matrix(rnorm(2,0,10),ncol = 1, nrow = 2)
for (i in 1:50){
```

```
  z4 = matrix(rnorm(2),ncol = 1, nrow = 2)

  X4[i,] = t4%*%z4 + b4
}
X4 = cbind(X4,C4)


# Shuffling data for non-biased initialisation of prototype points
X = rbind(X1,X2,X3,X4)
X_rand = X[sample(nrow(X)),]

# Plotting data points of different cluster with different colour
plot(X_rand[,1],X_rand[,2],col=X_rand[,3])
```



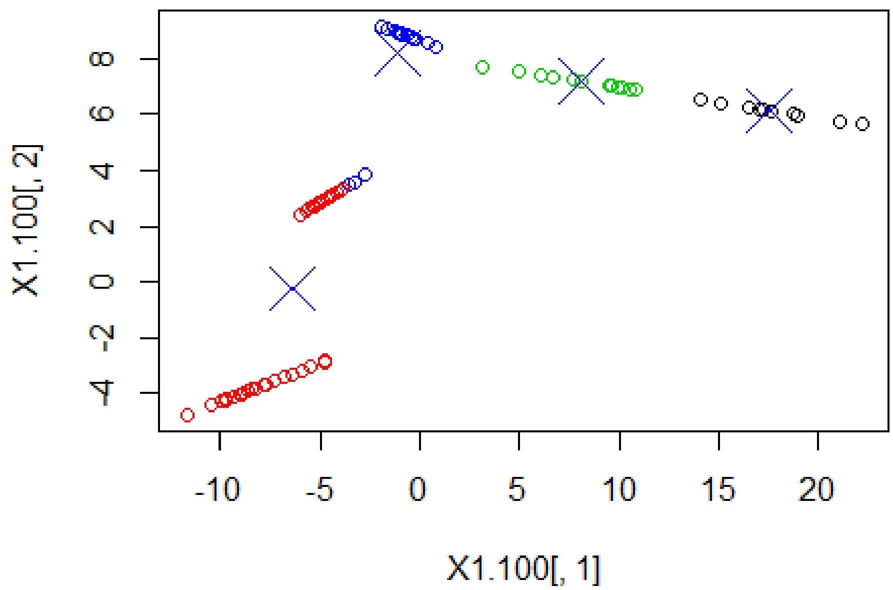

```
# (c)

X.100 = X[sample(nrow(X),100),]


K = 4
X1.100 = cbind(X.100[,c(1,2)],rep(1,100))
#m1 = X1.100[sample(nrow(X1.100),4),c(1,2)]
m1 = matrix(0,nrow=4,ncol=2)
m1[,1] = runif(4,min=min(X1.100[,1]),max=max(X1.100[,1]))
m1[,2] = runif(4,min=min(X1.100[,2]),max=max(X1.100[,2]))
```

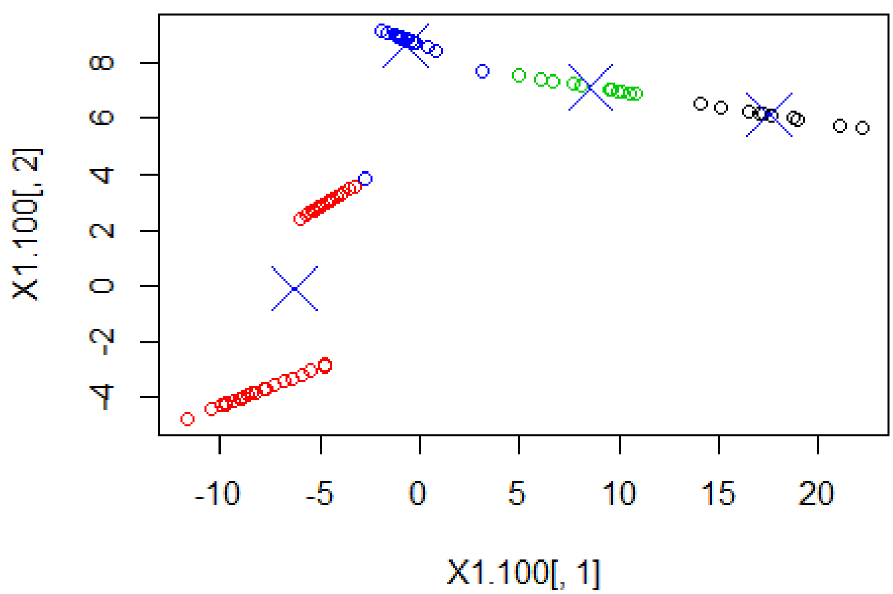
```r
# Plotting the scatter plot with inital cluster same for all points
plot(X1.100[,1], X1.100[,2],col=X1.100[,3])
lines(m1[,1],m1[,2],col="blue",pch=4,cex=3,type="p")
```




```r
# Looping through 50 iteration with user input of cluster formation
dist_matrix1 = matrix(0,nrow=100,ncol= 4)

H = c()
steps = c()
mnew = matrix(0,ncol=2,nrow=4)
for (i in 1:100){
  for (j in 1:nrow(m1)) {
  dist_matrix1[,j] = sqrt((X1.100[,1]-m1[j,1])^2+(X1.100[,2]-m1[j,2])^2)
  }
  cluster = apply(dist_matrix1, 1, which.min)
  X1.100[,3] = cluster
  for (k in 1:nrow(m1)){
  subset = X1.100[X1.100[,3] == k,]
  m1[k,1] = mean(subset[,1])
  m1[k,2] = mean(subset[,2])
  }
H1 = sum(apply(dist_matrix1, 1, min))
plot(X1.100[,1], X1.100[,2],col=X1.100[,3])
lines(m1[,1],m1[,2],col="blue",pch=4,cex=3,type="p")
```

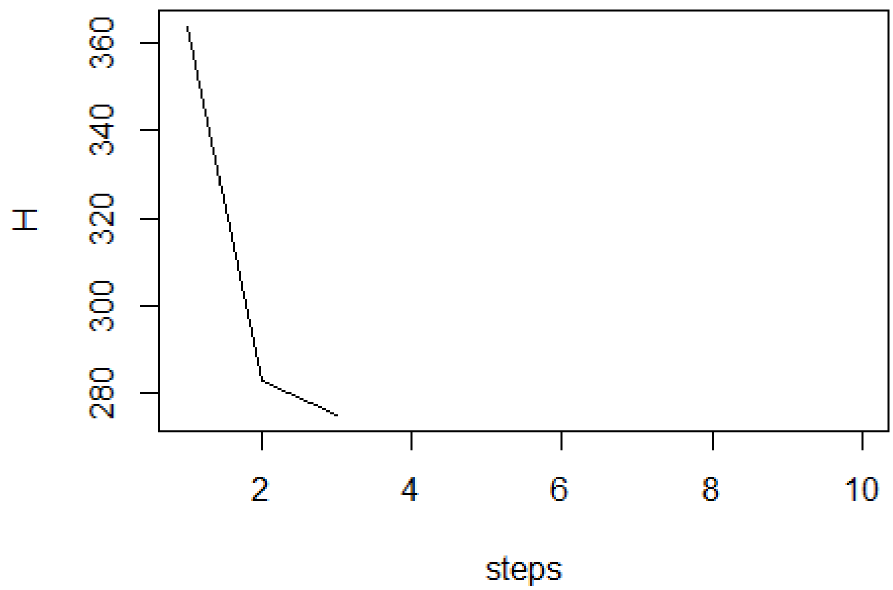
```
 if (all(m1 == mnew) == TRUE){
   H = c(H,H1)
   steps = c(steps,i)
   K = i
   cat("Number of steps to converge: ", K)
   break
 } else {
   H = c(H,H1)
   steps = c(steps,i)
   mnew = m1

 }
}
```






```
## Number of steps to converge:  3
```

```
plot(steps,H,type="l",xlim = c(1,10))
```




```
K

## [1] 3
```