harinris_HW6_ex

Harin Rishabh

2023-12-10

2 a)

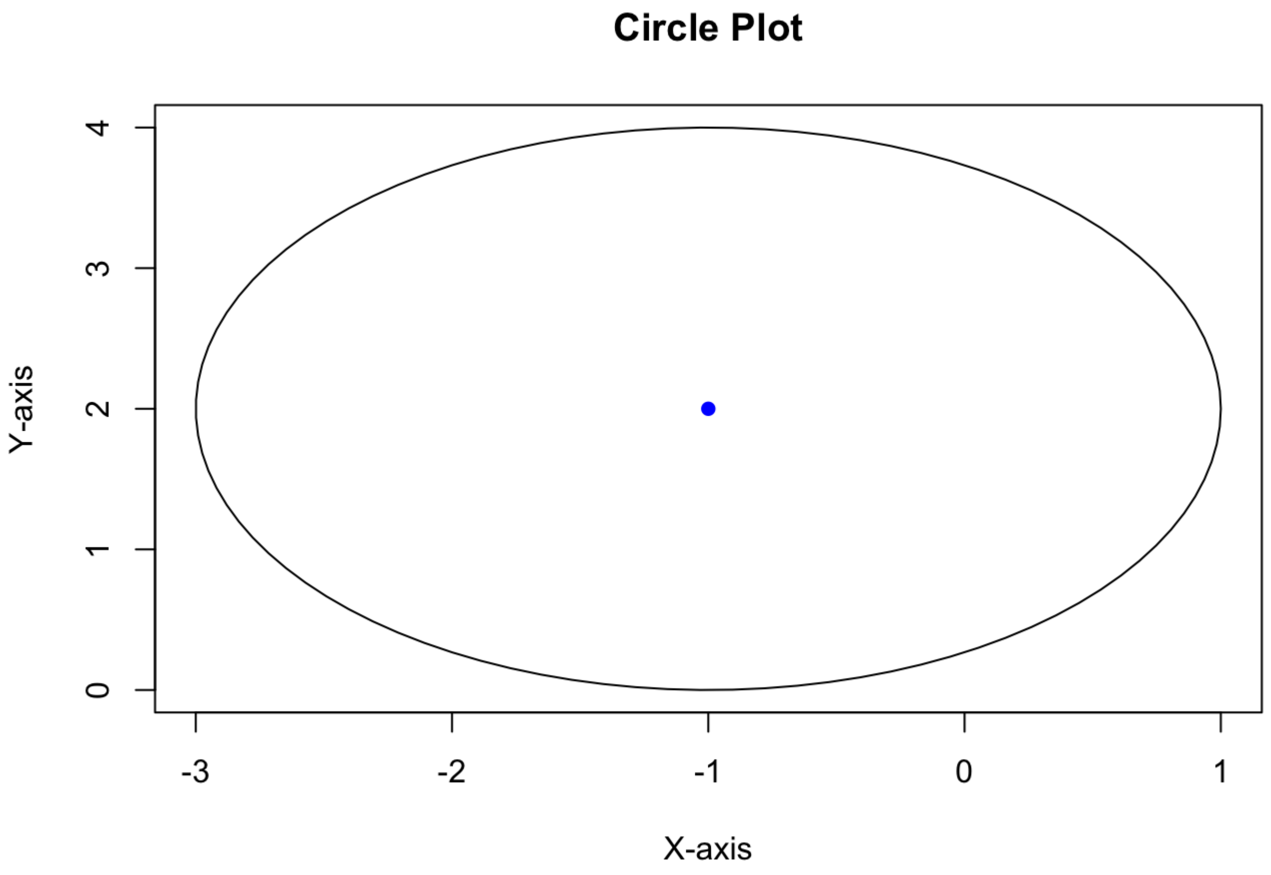
Answer -

The equation in the question can be rewritten as below. This is the equation of a circle.

$$(1 + \mathrm{X1})^2 + (2 - \mathrm{X2})^2 = 4 \Rightarrow (\mathrm{X1} - (-1))^2 + (\mathrm{X2} - 2)^2 = 2^2$$

Plotting the circle-

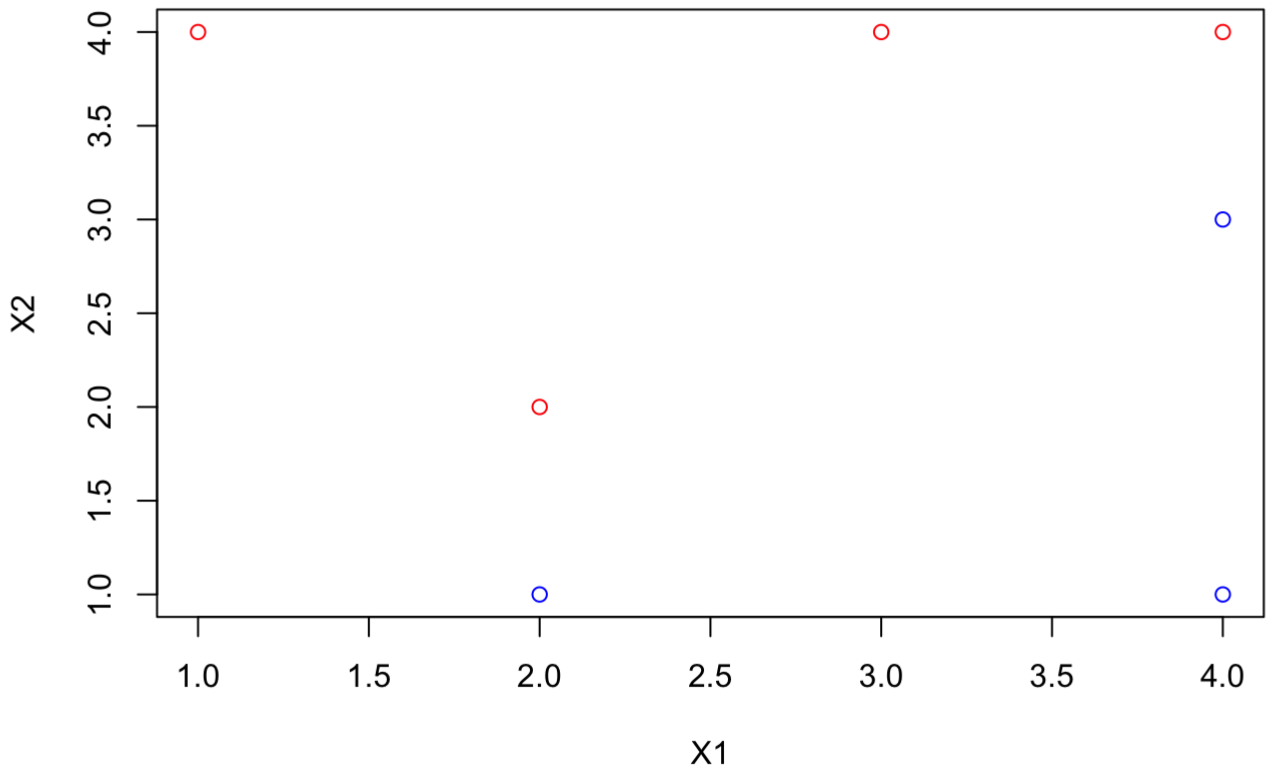
```r
h = -1
k = 2
r = 2

theta <- seq(0, 2*pi, length.out = 100)
x <- h + r * cos(theta)
y <- k + r * sin(theta)

plot(x, y, type = "l", xlab = "X-axis", ylab = "Y-axis", main = "Circle Plot")
points(h, k, col = "blue", pch = 16)
```



3 a)

Answer -

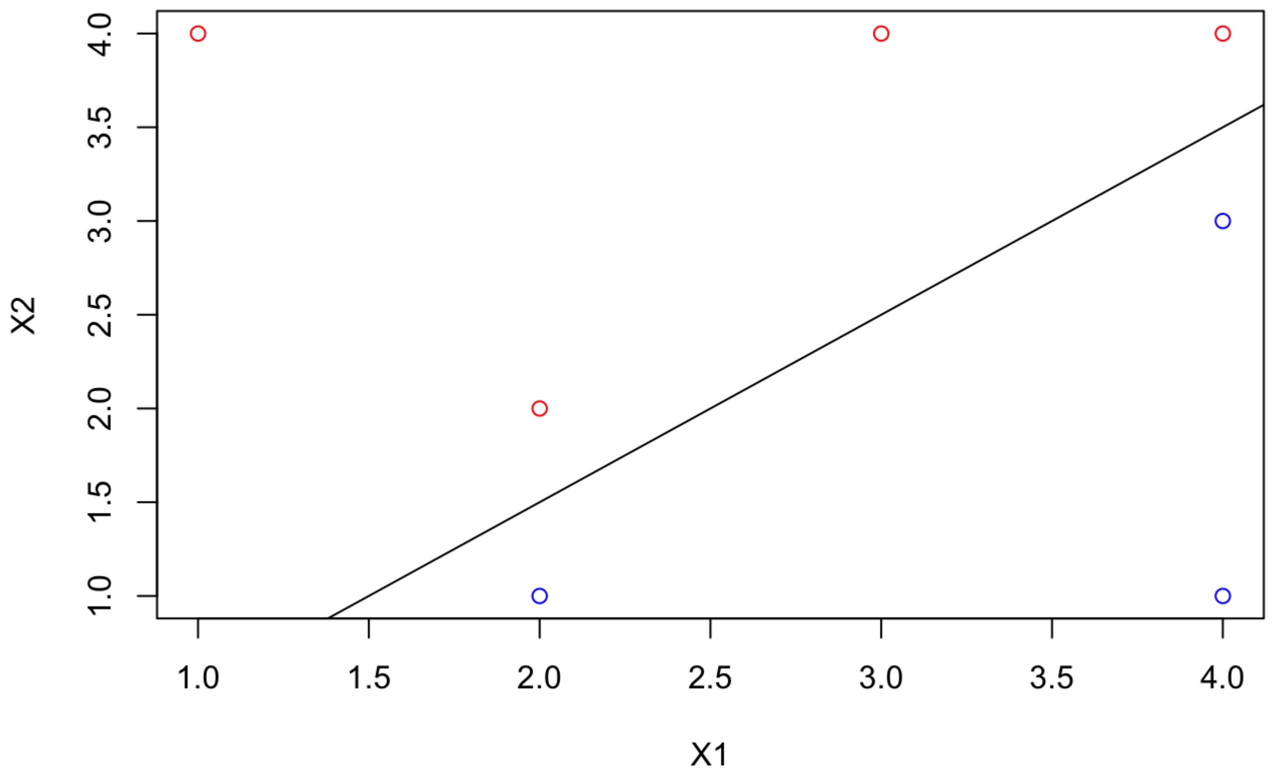
```
X1 = c(3, 2, 4, 1, 2, 4, 4)
X2 = c(4, 2, 4, 4, 1, 3, 1)
Y = c("Red", "Red", "Red", "Red", "Blue", "Blue", "Blue")
df <- data.frame(X1, X2, Y)
plot(df[ , c(1,2)], col=df$Y)
```



3 b)

Answer - On inspecting the graph above, we can see that the best line would be $X2 = X1 + (-0.5)$

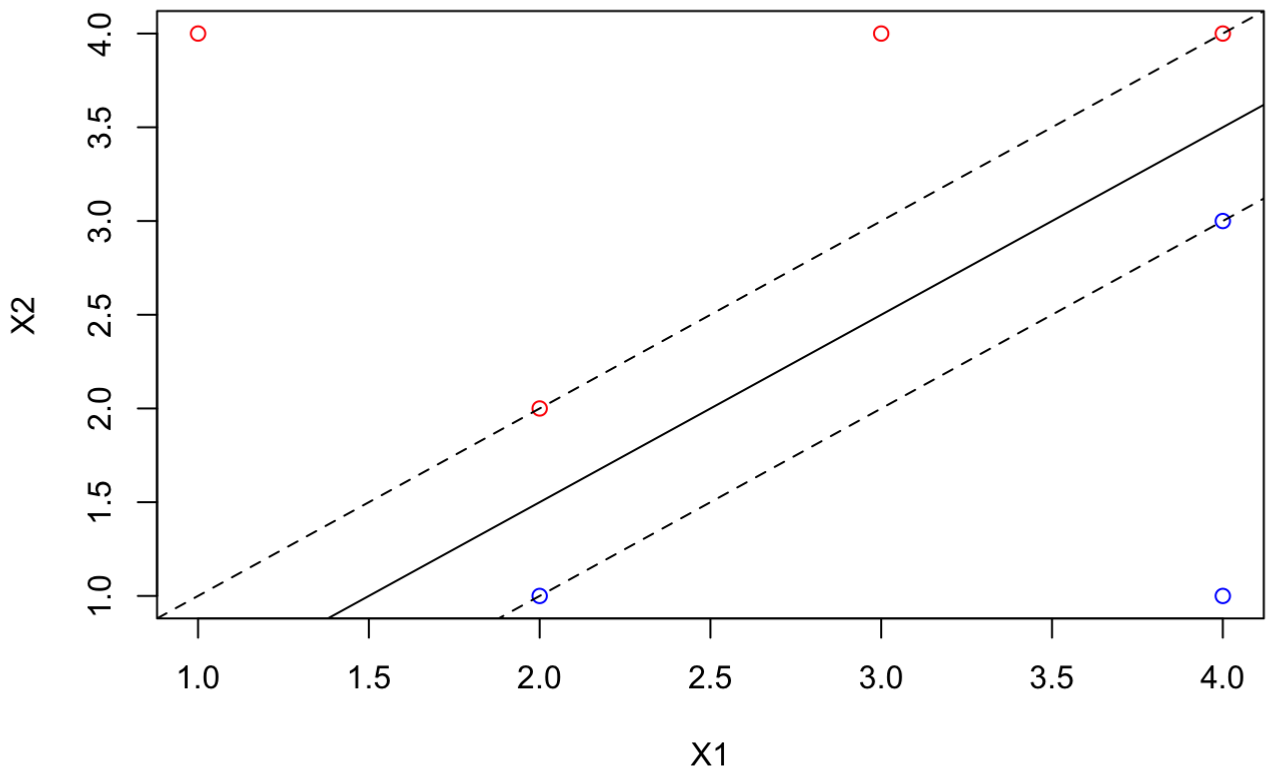
```
plot(df[ , c(1,2)], col=df$Y)
abline(-0.5, 1)
```



3 c)
Answer - Since the optimal separating hyperplane is given by the equation $X2 = X1 + (-0.5)$, from the plot above we can say that the prediction for the maximal margin classifier will be Red when $X2 - X1 + 0.5 > 0$ and blue otherwise. Therefore the values for β0, β1, β2 are 0.5, -1, 1.

3 d)
Answer -

```
plot(df[ , c(1,2)], col=df$Y)
abline(-0.5, 1)
abline(0, 1, lty=2)
abline(-1, 1, lty=2)
```

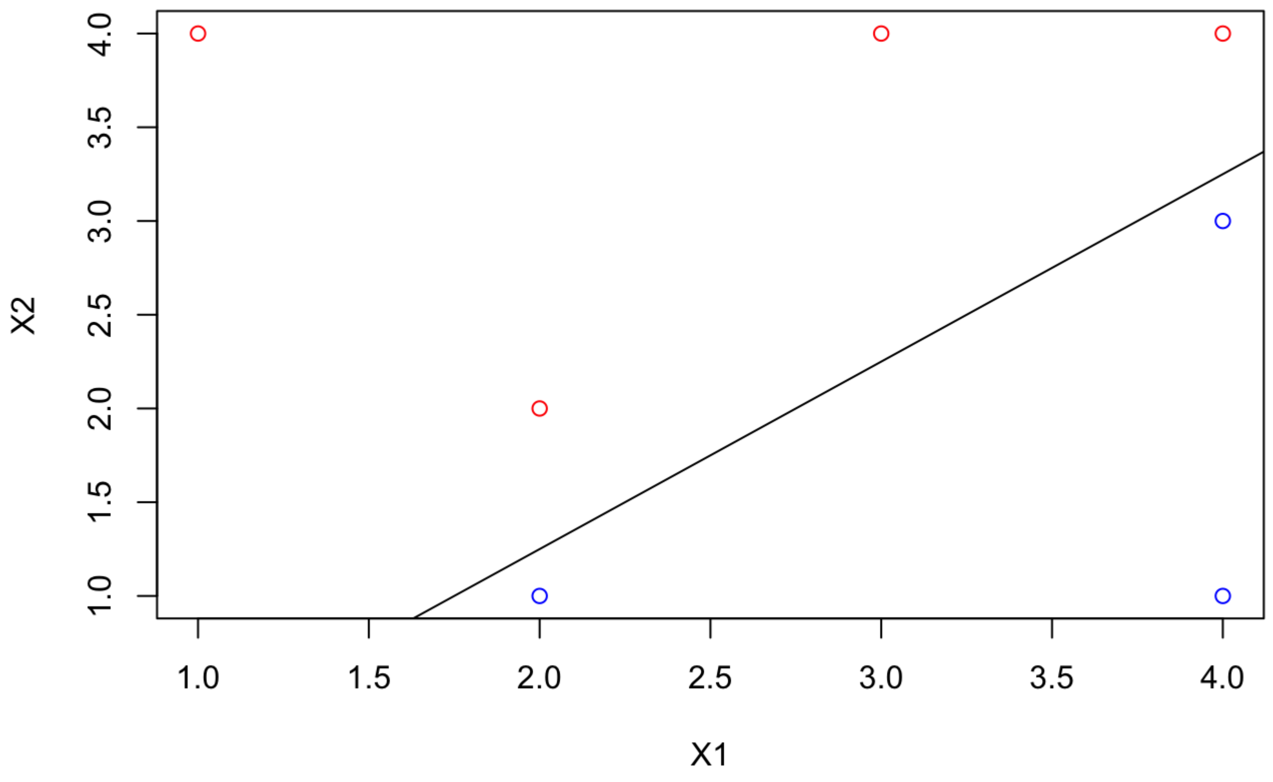


3 e)
Answer - The support vectors are observations - 2, 3, 5 and 6.

3 f)
Answer - The 7th observation is not a support vector. Moving it will not have any affect on the maximal margin hyperplane.

3 g)
On plotting the line equation, $X2 = X1 + (-0.75)$, we can see that even though all the red and blue observations are separated, the boundary is not optimal.

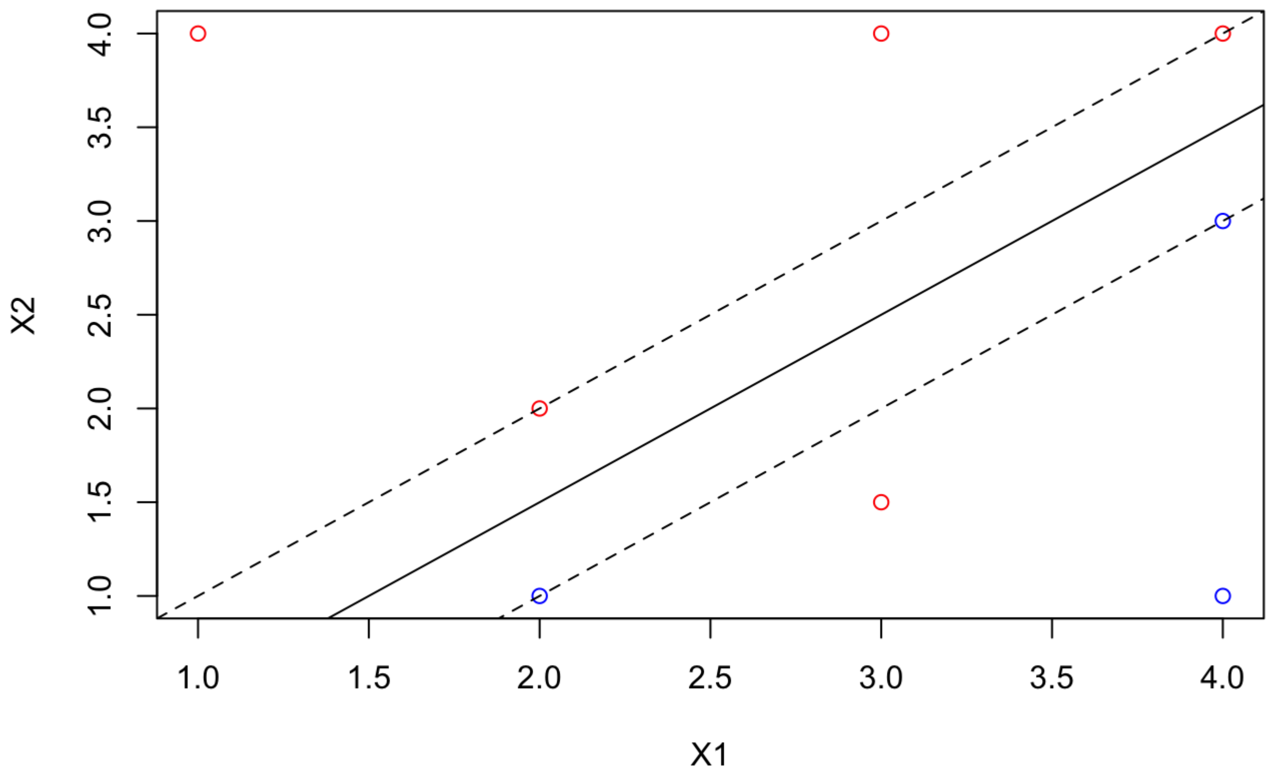
```
plot(df[ , c(1,2)], col=df$Y)
abline(-0.75, 1)
```



3 h)

Answer - Adding a point (3, 1.5) makes the 2 classes inseparable.

```
plot(df[ , c(1,2)], col=df$Y)
abline(-0.5, 1)
abline(0, 1, lty=2)
abline(-1, 1, lty=2)
points(3, 1.5, col="red")
```



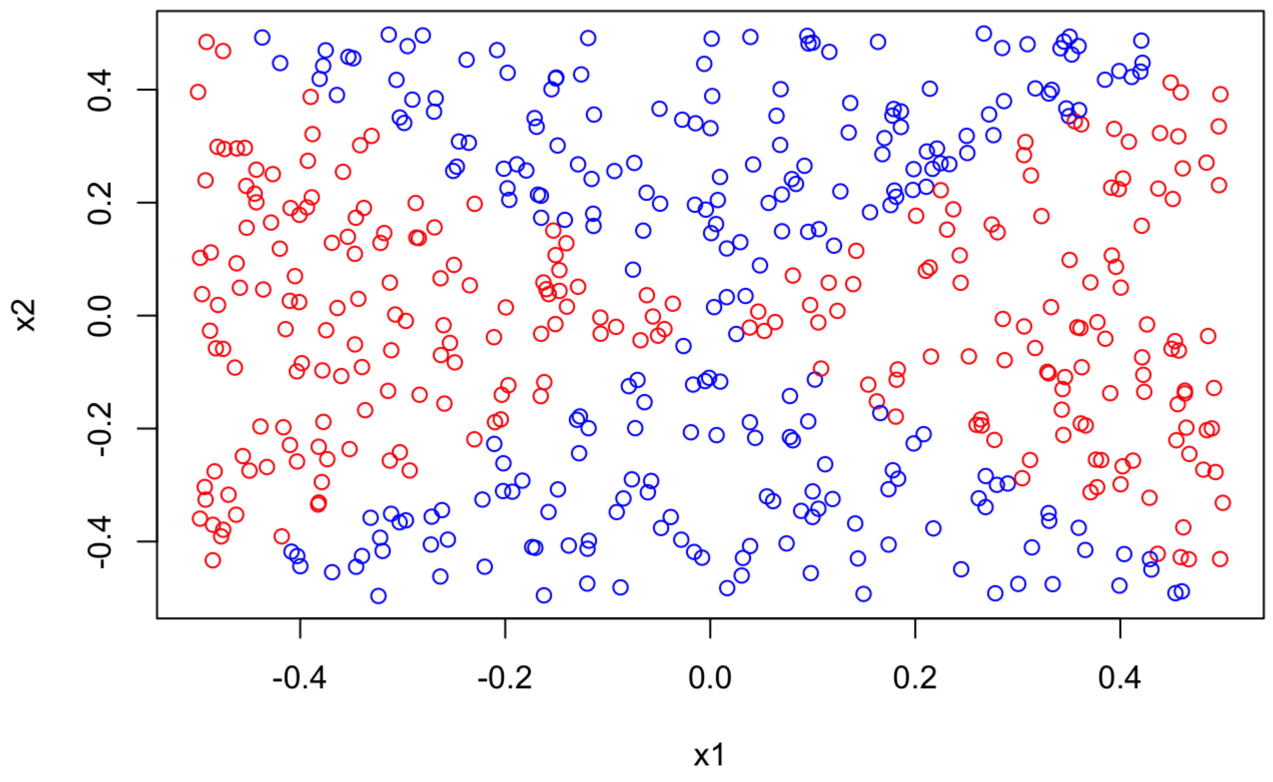
5 a)

Answer-

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

5 b)

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



5 c)

Answer -

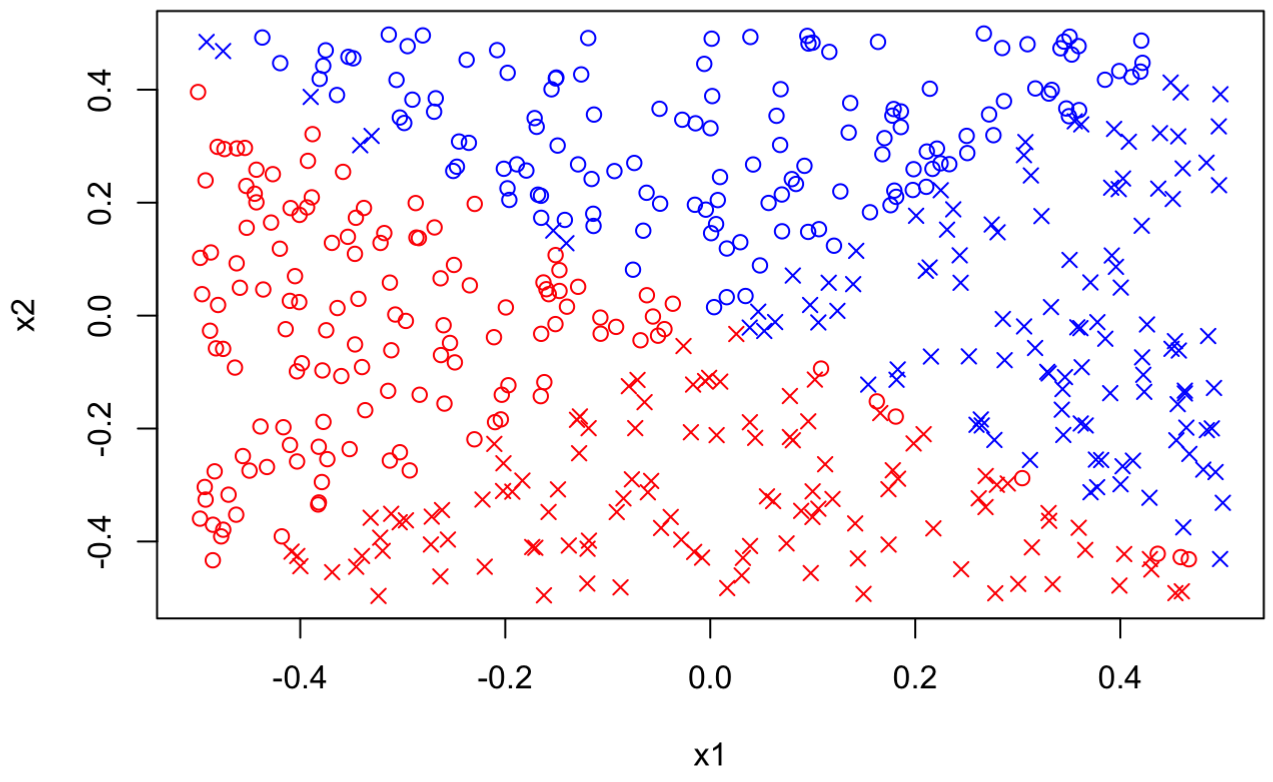
```
df = data.frame(x1,x2,y)
df.fit = glm(y~. , df, family='binomial')
df.fit
```

```
##
## Call:  glm(formula = y ~ ., family = "binomial", data = df)
##
## Coefficients:
## (Intercept)           x1           x2
##   0.0006241   -0.4301484   -0.4894237
##
## Degrees of Freedom: 499 Total (i.e. Null);  497 Residual
## Null Deviance:      693.1
## Residual Deviance: 688.7     AIC: 694.7
```

5 d)

Answer - From the plot below we can see that the decision boundary is linear.

```
df.pred=predict(df.fit, data.frame(x1,x2))
plot(x = x1, y = x2,


     col = ifelse(df.pred > 0, 'red', 'blue'),


     pch = ifelse(as.integer(df.pred > 0) == y, 1, 4)
)
```



5 e)

Answer -

```
glm.fit = glm(y~poly(x1,2)+poly(x2,2), df, family='binomial')
```

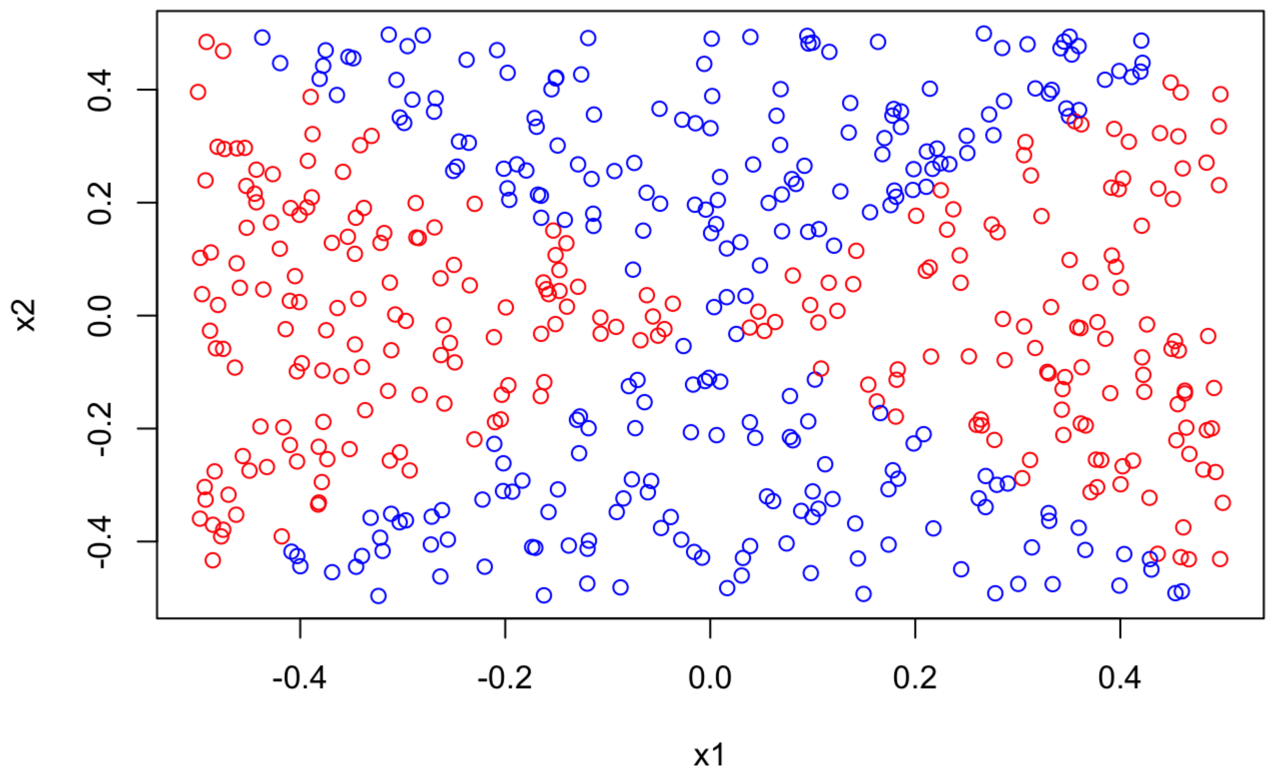
```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

5 f)

Answer - From the graph below, we can see that the decision boundary is clearly non-linear. All observations have been classified correctly.

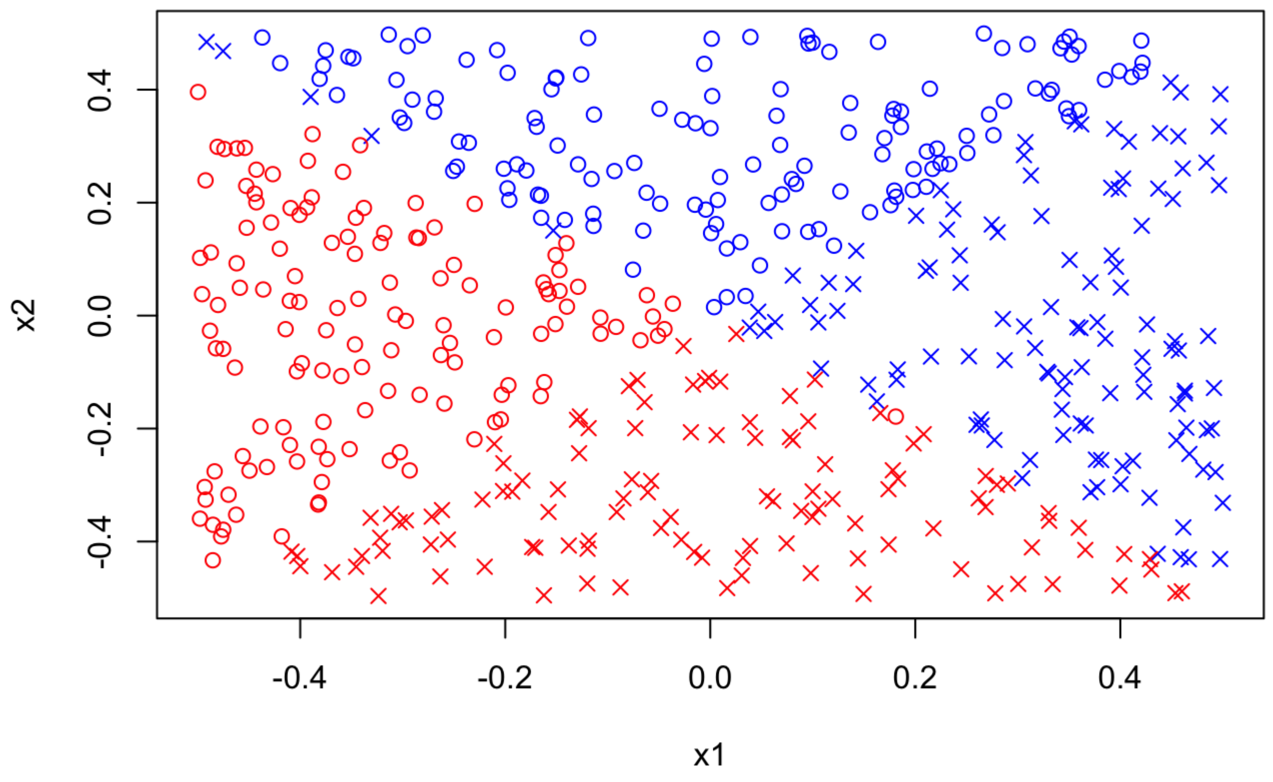
```
glm.pred=predict(glm.fit,data.frame(x1,x2))     # returns the log-odds.
plot(x = x1, y = x2,

     col = ifelse(glm.pred > 0, 'red', 'blue'),

     pch = ifelse(as.integer(glm.pred > 0) == y, 1, 4)
)
```



5 g)

Answer-

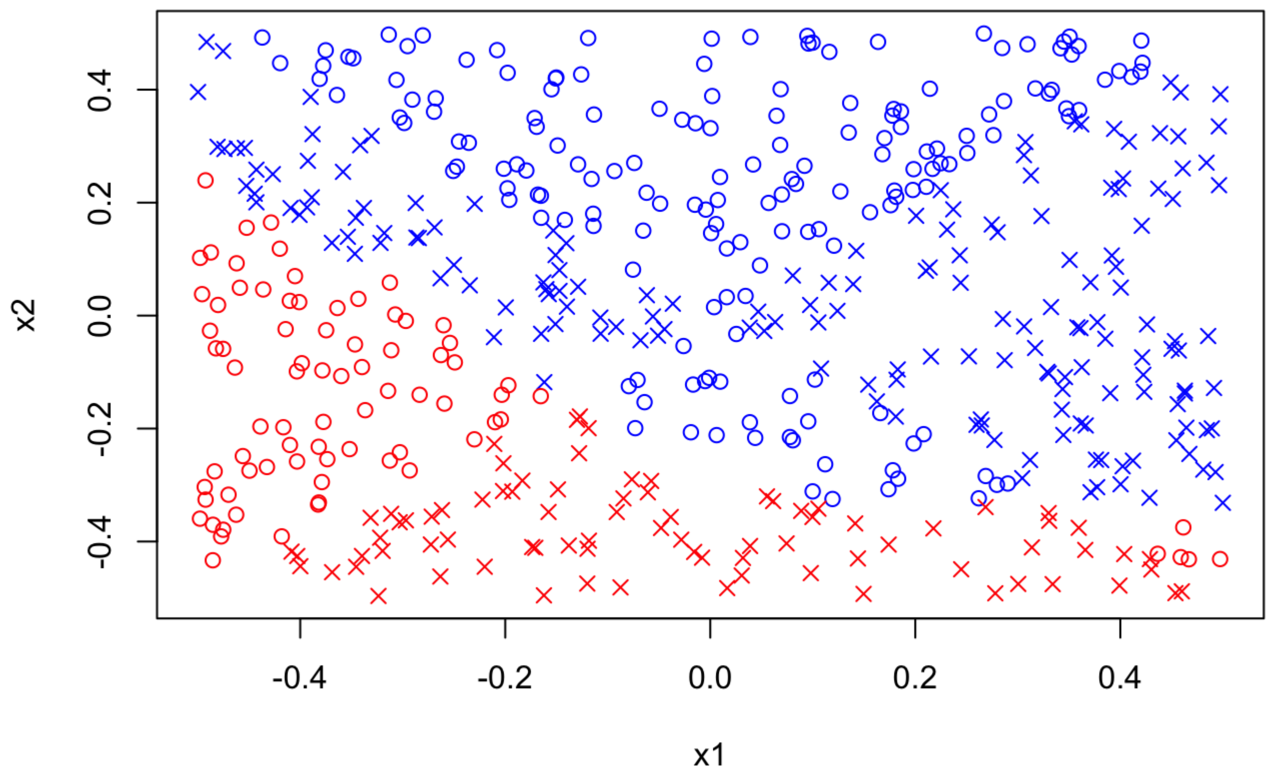
```
library(e1071)
svm.fit <- svm(y ~ x1 + x2, data = data.frame(x1,x2,y=as.factor(y)), kernel = "linea
r", scale = FALSE, cost = 0.001)
svm.pred <- predict(svm.fit, data.frame(x1,x2), type='response')
plot(x1,x2,

     col=ifelse(svm.pred == 1,'red','blue'),

     pch=ifelse(svm.pred == y,1,4))
```



5 h)

Answer-

```
svm.fit <- svm(y ~ x1 + x2, data = data.frame(x1,x2,y=as.factor(y)), kernel = "polyno
mial", scale = FALSE, cost = 0.001)
svm.pred <- predict(svm.fit, data.frame(x1,x2), type='response')
plot(x1,x2,

     col=ifelse(svm.pred == 1,'red','blue'),

     pch=ifelse(svm.pred == y,1,4))
```



5 i)

Answer - Clearly the polynomial logarithmic model is the most accurate. It had zero misclassifications. This is not surprising because we knew the equation before-hand and generated the data artificially.

8 a)

Answer -

```
library(ISLR2)
attach(OJ)
data = OJ
indices = sample(1:nrow(data), size = 0.8 * nrow(data))
train_data <- data[indices, ]
test_data <- data[-indices, ]
```

8 b)

Answer - From the summary below, we can see that there are 467 support vectors. These are almost evenly distributed between the classifications.

```
svm.fit = svm(Purchase~., data=train_data, cost=0.01, kernel='linear')
summary(svm.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_data, cost = 0.01, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  465
##
##  ( 233 232 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

8 c)

Answer -

```
svm.pred=predict(svm.fit,train_data)
print(paste("Train error rate = ", mean(predict(svm.fit, train_data) != train_data$Pu
rchase)))
```

```
## [1] "Train error rate =  0.167056074766355"
```

```
print(paste("Test error rate = ", mean(predict(svm.fit, test_data) != test_data$Purch
ase)))
```

```
## [1] "Test error rate =  0.182242990654206"
```

8 d)

Answer -

```
svm.tune=tune(svm, Purchase~. ,data=train_data, ranges=data.frame(cost=seq(0.01,10)),
kernel='linear')
summary(svm.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  5.01
##
## - best performance: 0.1681806
##
## - Detailed performance results:
##    cost     error dispersion
## 1  0.01 0.1728044 0.05464042
## 2  1.01 0.1705472 0.04920372
## 3  2.01 0.1705609 0.04896436
## 4  3.01 0.1717100 0.04695606
## 5  4.01 0.1693570 0.04792554
## 6  5.01 0.1681806 0.04883004
## 7  6.01 0.1693570 0.04727941
## 8  7.01 0.1681806 0.04851408
## 9  8.01 0.1693570 0.04887873
## 10 9.01 0.1705198 0.04853491
```

8 e)

Answer -

```
svm.pred.best = predict(svm.tune$best.model, train_data)
print(paste("Train error rate = ", mean(predict(svm.tune$best.model, train_data) != t
rain_data$Purchase)))
```

```
## [1] "Train error rate =  0.164719626168224"
```

```
print(paste("Test error rate = ", mean(predict(svm.tune$best.model, test_data) != tes
t_data$Purchase)))
```

```
## [1] "Test error rate =  0.168224299065421"
```

8 f)

Answer -

```
svm.fit = svm(Purchase~., data=train_data, cost=0.01, kernel='radial')
svm.pred=predict(svm.fit,train_data)
print(paste("Train error rate with radial kernel = ", mean(predict(svm.fit, train_dat
a) != train_data$Purchase)))
```

```
## [1] "Train error rate with radial kernel =  0.400700934579439"
```

```
print(paste("Test error rate with radial kernel = ", mean(predict(svm.fit, test_data)
!= test_data$Purchase)))
```

```
## [1] "Test error rate with radial kernel =  0.345794392523364"
```

```
svm.tune=tune(svm, Purchase~. ,data=train_data, ranges=data.frame(cost=seq(0.01,10)),
kernel='radial')
svm.pred.best = predict(svm.tune$best.model, train_data)
print(paste("Train error rate with radial kernel and tuning = ", mean(predict(svm.tun
e$best.model, train_data) != train_data$Purchase)))
```

```
## [1] "Train error rate with radial kernel and tuning =  0.157710280373832"
```

```
print(paste("Test error rate with radial kernel and tuning = ", mean(predict(svm.tune
$best.model, test_data) != test_data$Purchase)))
```

```
## [1] "Test error rate with radial kernel and tuning =  0.163551401869159"
```

8 g)

Answer -

```
svm.fit = svm(Purchase~., data=train_data, cost=0.01, kernel='polynomial', degree=2)
svm.pred=predict(svm.fit,train_data)
print(paste("Train error rate with polynomial kernel = ", mean(predict(svm.fit, train
_data) != train_data$Purchase)))
```

```
## [1] "Train error rate with polynomial kernel =  0.377336448598131"
```

```
print(paste("Test error rate with polynomial kernel = ", mean(predict(svm.fit, test_d
ata) != test_data$Purchase)))
```

```
## [1] "Test error rate with polynomial kernel =  0.336448598130841"
```

```
svm.tune=tune(svm, Purchase~. ,data=train_data, ranges=data.frame(cost=seq(0.01,10)),
kernel='polynomial', degree=2)
svm.pred.best = predict(svm.tune$best.model, train_data)
print(paste("Train error rate with polynomial kernel and tuning = ", mean(predict(sv
m.tune$best.model, train_data) != train_data$Purchase)))
```

```
## [1] "Train error rate with polynomial kernel and tuning =  0.151869158878505"
```

```
print(paste("Test error rate with polynomial kernel and tuning = ", mean(predict(svm.
tune$best.model, test_data) != test_data$Purchase)))
```

```
## [1] "Test error rate with polynomial kernel and tuning =  0.163551401869159"
```

8 h)

Answer - The performances of all 3 models are very similar after using cross-validation but we should go ahead with linear SVM on this data set as it has the least test error rate.