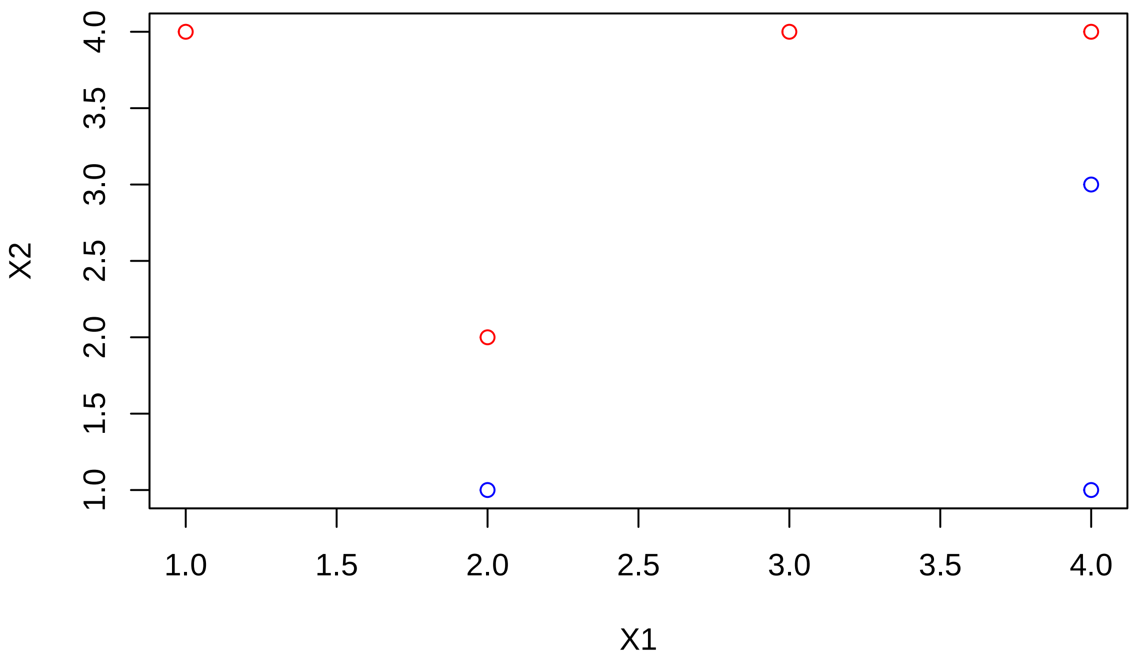
# SVM Homework

Tao He

3/1/2022

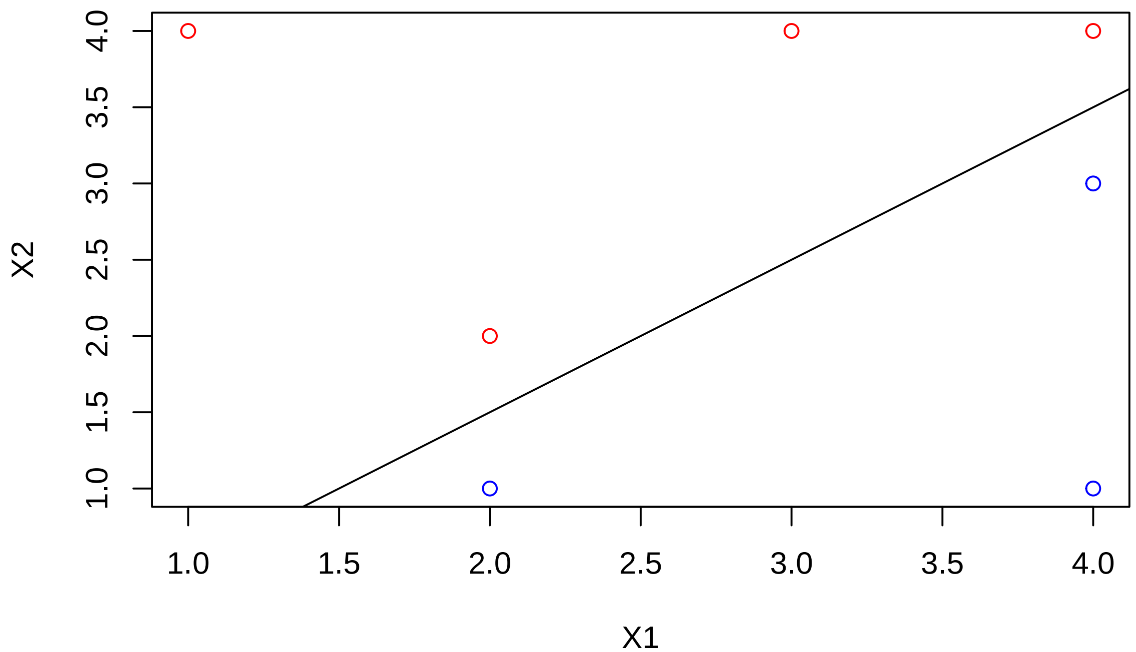
9.3 (a)

```
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(X1, X2, col = Y)
```




(b)

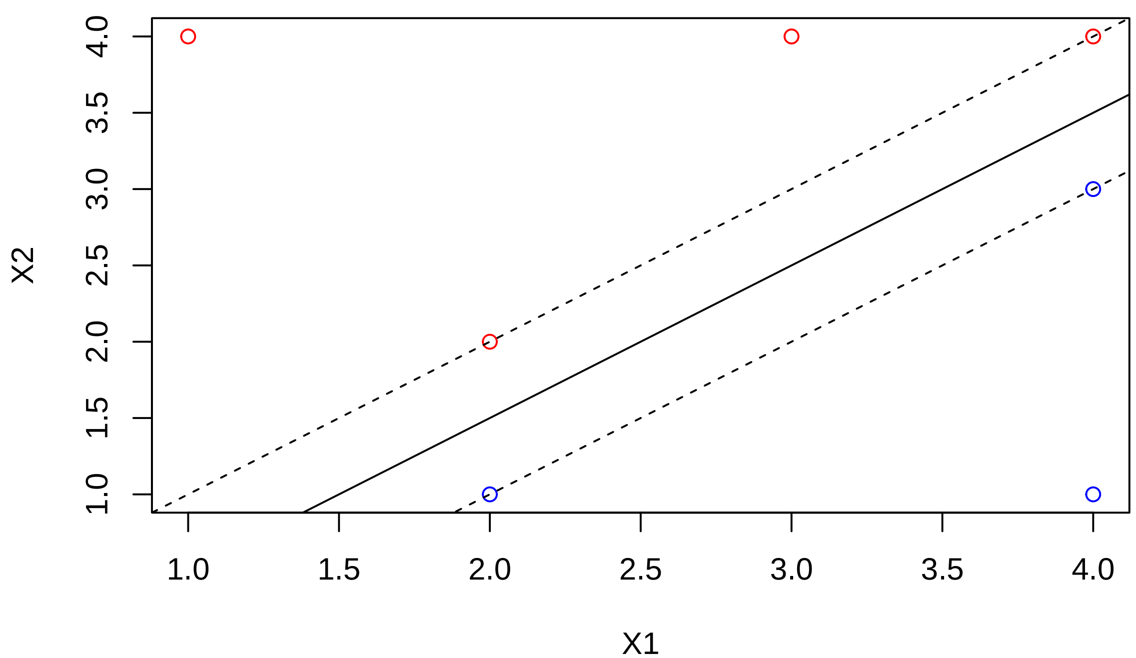
```
plot(X1, X2, col = Y)
abline(-0.5, 1)
```




(c) ***sol.n:*** The classification rule is "Classify to"red" if X1 - X2 - 0.5 < 0, and classify to "blue" otherwise.

(d)

```
plot(X1, X2, col = Y)
abline(-0.5, 1)
abline(-1, 1, lty = 2)
abline(0, 1, lty = 2)
```



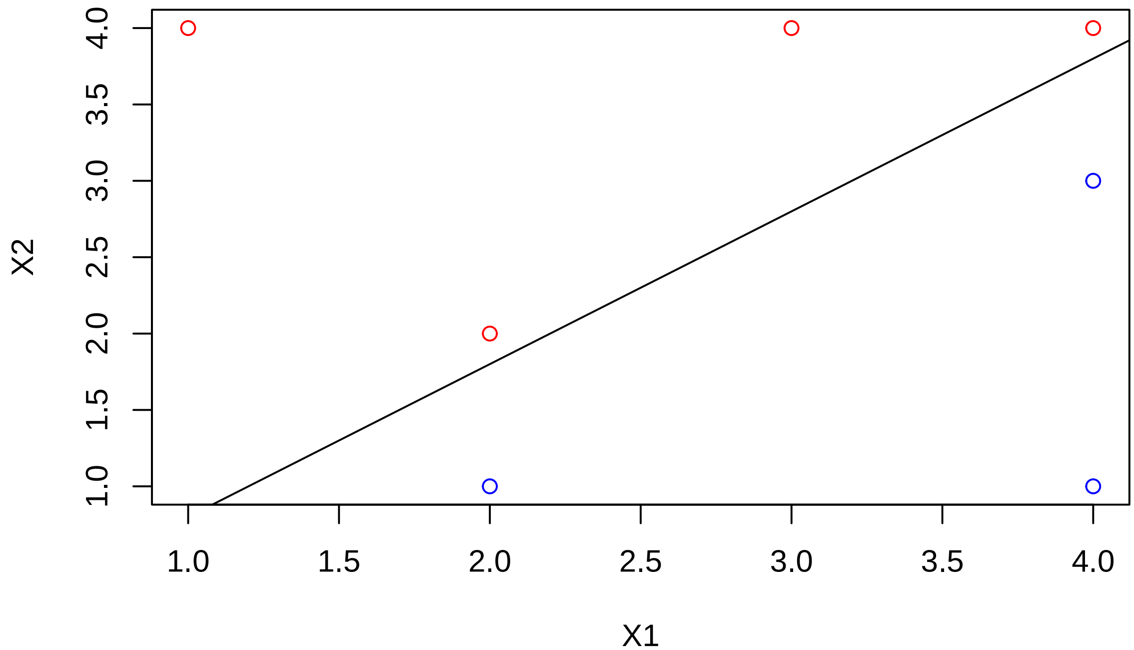

The margin for the maximal margin hyperplane is 1/4.

(e) ***sol.n:*** The support vectors are the points (2,1), (2,2), (4,3) and (4,4).

(f) ***sol.n:*** If we move the seventh observation (4,1), the maximal margin hyperplane would not be changed as it is not a support vector.

(g)

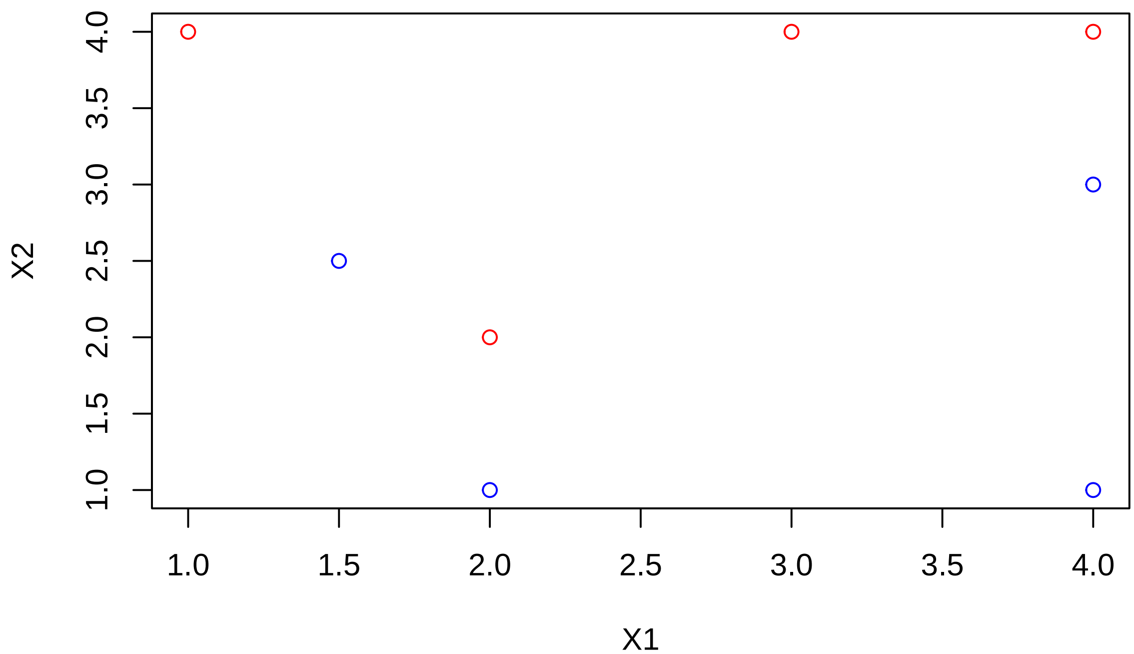
```
plot(X1, X2, col = Y)
abline(-0.2, 1)
```




The equation for this hyperplane is X1 - X2 -0.2 = 0

(h)

```
plot(X1, X2, col = Y)
points(c(1.5), c(2.5), col = "blue")
```

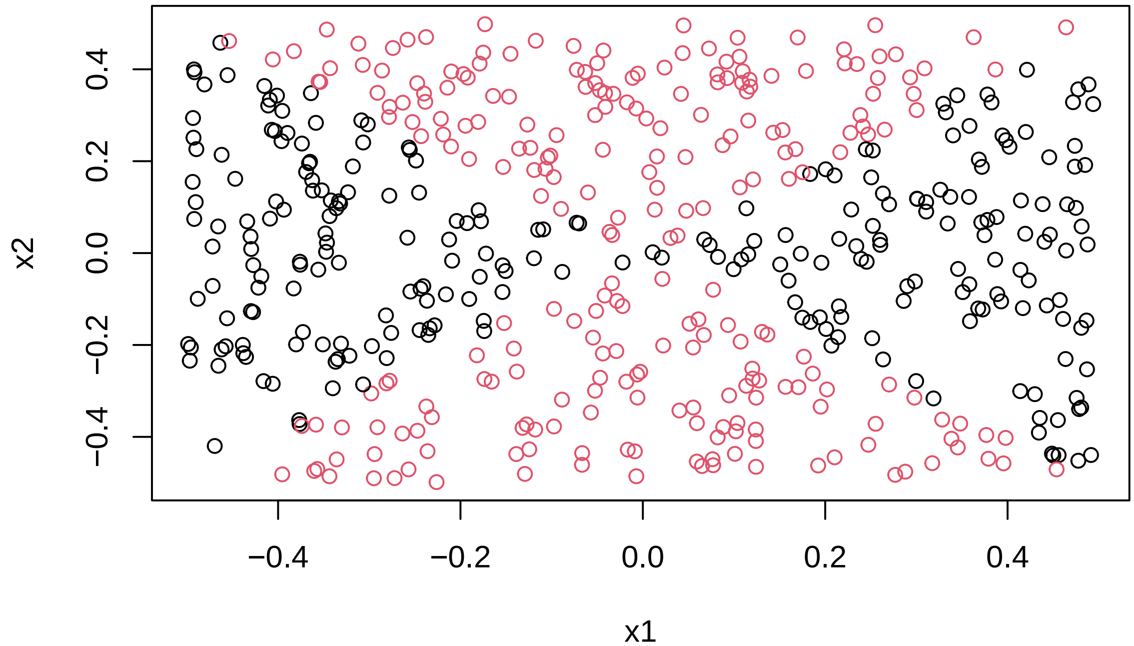



9.5 (a)

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

(b)

```r
df <- data.frame(x1= x1, x2= x2, y=as.factor(y))
plot(x1,x2,col = (2 - y))
```




(c)

```r
logit.fit <- glm(y ~ x1 + x2, family = "binomial")
summary(logit.fit)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = "binomial")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.215  -1.174   1.145   1.174   1.202
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.009702   0.089635   0.108    0.914
## x1          -0.034680   0.311514  -0.111    0.911
## x2          -0.134885   0.319251  -0.423    0.673
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 693.14  on 499  degrees of freedom
## Residual deviance: 692.95  on 497  degrees of freedom
## AIC: 698.95
##
```

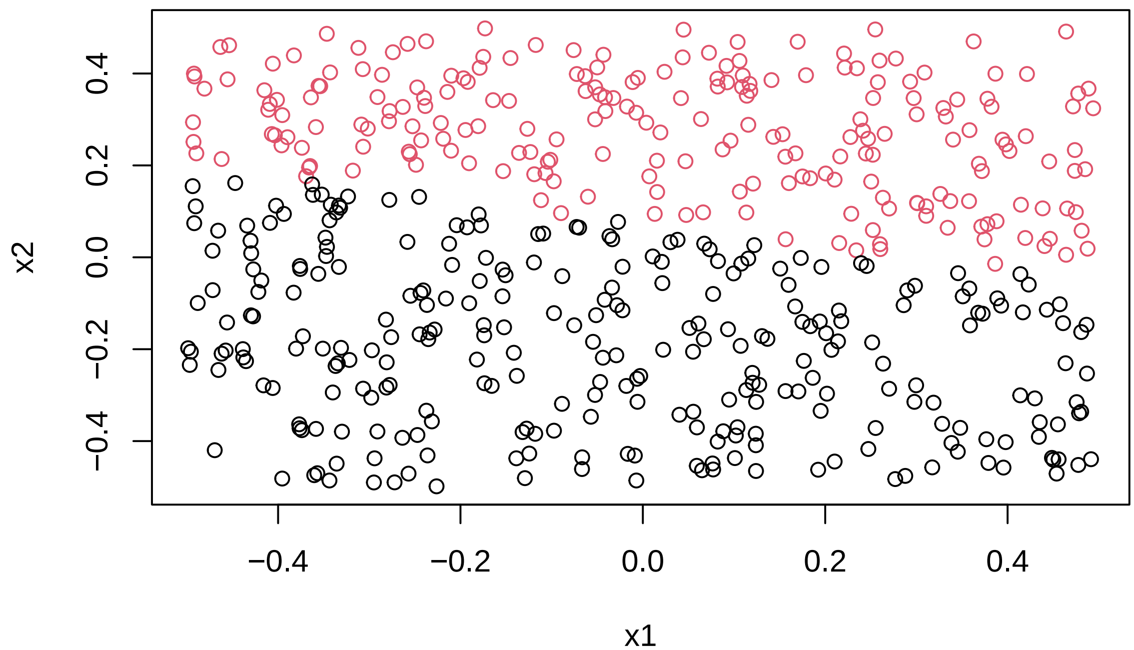
```
## Number of Fisher Scoring iterations: 3
```

(d)

```
logit.probs <- predict(logit.fit, newdata = df, type = "response")
logit.preds <- rep(0, 500)
logit.preds[logit.probs > 0.5] <- 1

table(preds = logit.preds, truth = df$y)
```

```
##      truth
## preds   0   1
##     0 130  92
##     1 119 159
```

```
plot(x1, x2, col = (2-logit.preds))
```




The decision boundary is linear.

(e)

```
logit.fit1 <- glm(y ~ poly(x1, 2, raw = T) + poly(x2, 2, raw = T), family = "binomial")
summary(logit.fit1)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2, raw = T) + poly(x2, 2, raw = T),
##     family = "binomial")
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -0.007730   0.000000   0.000000   0.000000   0.006888
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)            -3.306e+00  7.952e+01  -0.042    0.967
```

```
## poly(x1, 2, raw = T)1 -2.340e+02  2.474e+04  -0.009    0.992
## poly(x1, 2, raw = T)2  1.543e+05  1.056e+06   0.146    0.884
## poly(x2, 2, raw = T)1  1.055e+02  2.466e+04   0.004    0.997
## poly(x2, 2, raw = T)2 -1.551e+05  1.062e+06  -0.146    0.884
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9314e+02  on 499  degrees of freedom
## Residual deviance: 1.3712e-04  on 495  degrees of freedom
## AIC: 10
##
## Number of Fisher Scoring iterations: 25
```

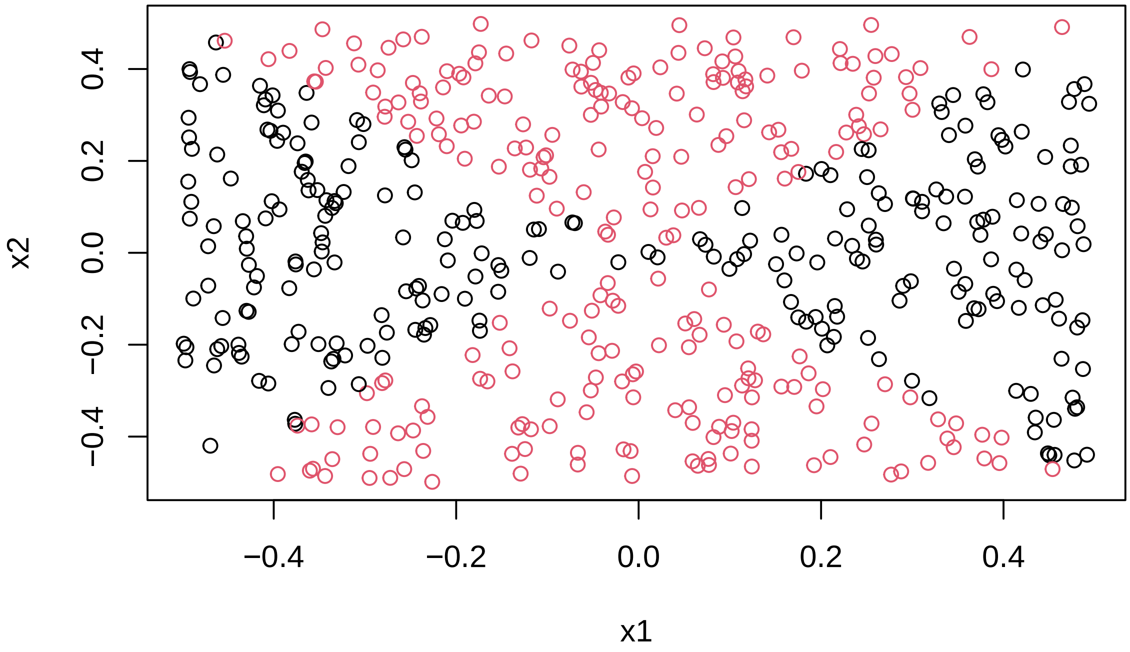
None of the variables are statistically significant.

(f)

```r
logit.fit1.probs <- predict(logit.fit1, newdata = df, type = "response")
logit.fit1.preds <- rep(0, 500)
logit.fit1.preds[logit.fit1.probs > 0.5] <- 1

table(preds = logit.fit1.preds, truth = df$y)
```

```
##      truth
## preds   0   1
##     0 249   0
##     1   0 251
```

```r
plot(x1, x2, col = (2- logit.fit1.preds))
```



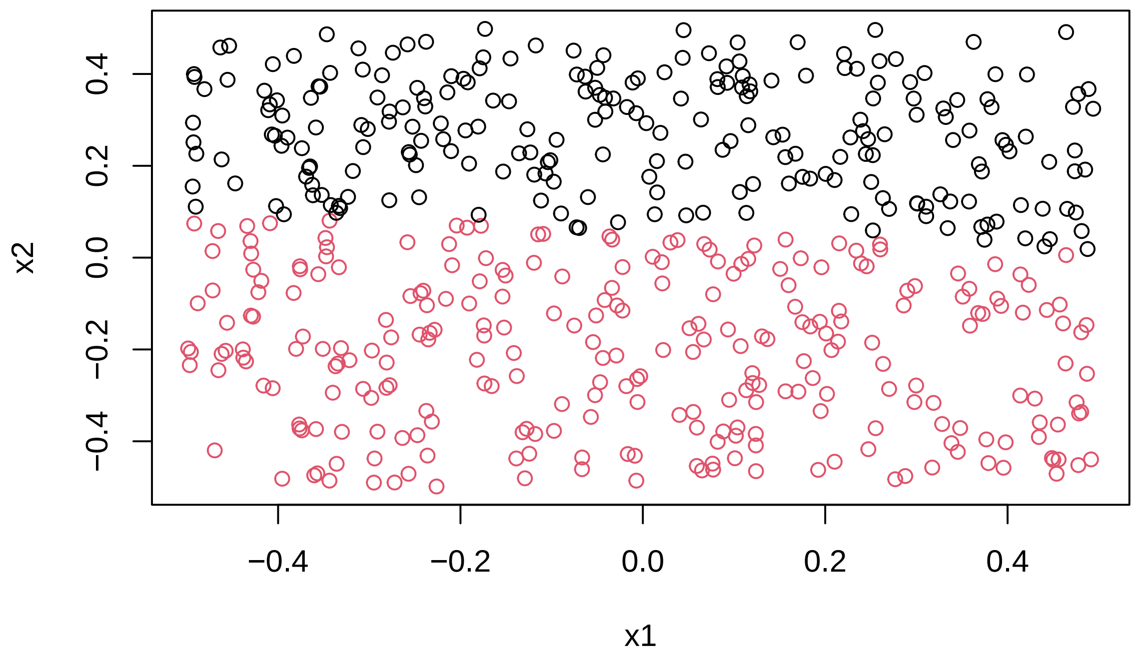

The decision boundary should be obviously non-linear. Quadratic transformations of x1 and x2 result in perfect separation.

(g)

```r
# best model
# linear
tune.out <- tune(svm, y ~ x1 + x2, data = df, kernel = "linear", ranges = list(cost = c(0.001, 0.01, 0.
bestmod <- tune.out$best.model

pred <- predict(bestmod, newdata = df, type = "response")
table(predict = pred, truth = df$y)
```

```
##        truth
## predict   0   1
##       0 131 103
##       1 118 148
```

```r
plot(x1, x2, col = pred)
```

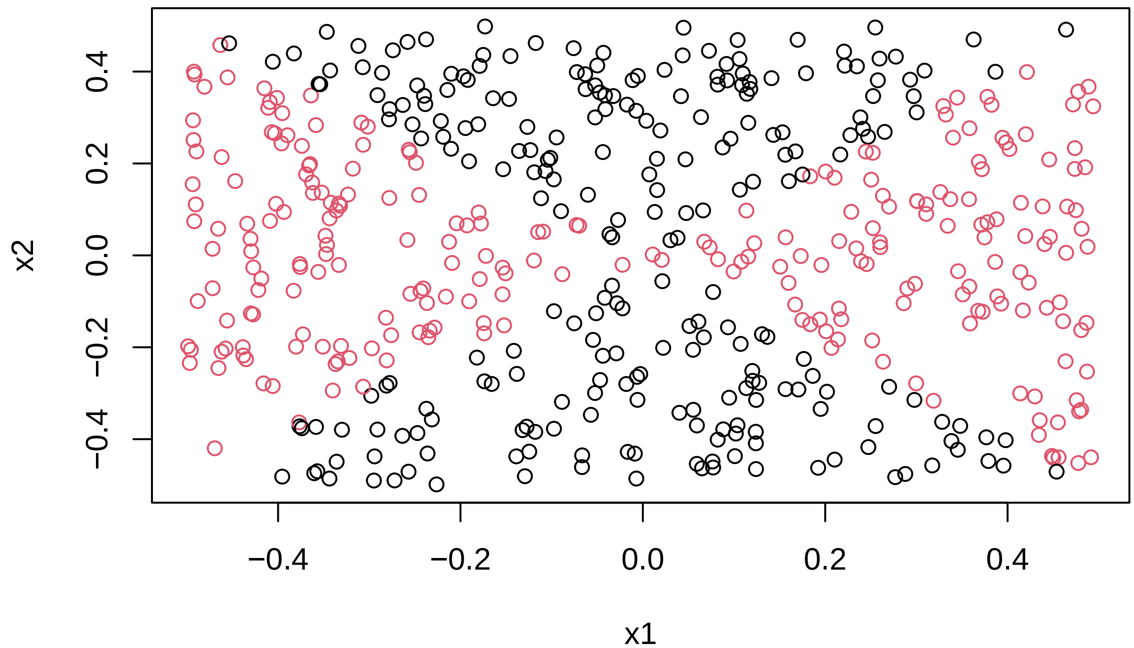



(h)

```r
# non-linear
tune.out <- tune(svm, y ~ x1 + x2, data = df, kernel = "radial", ranges = list(cost = c(0.1, 1, 10, 100
bestmod <- tune.out$best.model

pred <- predict(bestmod, newdata = df, type = "response")
table(predict = pred, truth = df$y)
```

```
##        truth
## predict   0   1
##       0 248   1
##       1   1 250
```

```r
plot(x1, x2, col = pred)
```




(i) ***sol.n:*** We may conclude that both SVM with non-linear kernel and logistic regression are useful for finding non-linear decision boundaries.

On the other words, SVM with linear kernel and logistic regression without any interaction term perform badly during finding non-linear decision boundaries.

9.7 (a)

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

(b)

```
set.seed(679)
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "linear", ranges = list(cost = c(0.01, 0.1, 1
bestmod <- tune.out$best.model
bestmod
```

```
##
## Call:
## best.tune(method = svm, train.x = mpglevel ~ ., data = Auto, ranges = list(cost = c(0.01,
##     0.1, 1, 5, 10, 100, 1000)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  56
```

A cost of 1 perform best.

(c)

```
# radial
set.seed(679)
```

```
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "radial", ranges = list(cost = c(0.01, 0.1, 1
tune.out$best.parameters
```

```
##   cost gamma
## 6  100  0.01
```

```
# polynomial
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "polynomial", ranges = list(cost = c(0.01, 0.
tune.out$best.parameters
```

```
##   cost degree
## 6  100      2
```

For a radial kernel, the lowest cross-validation error is obtained for a gamma of 0.01 and a cost of 100.

For a polynomial kernel, the lowest cross-validation error is obtained for a degree of 2 and a cost of 100.

(d)

9.8 (a)

```
set.seed(679)
train <- sample(nrow(OJ), 800)
OJ.train <- OJ[train,]
OJ.test <- OJ[-train,]
```

(b)

```
svm.linear <- svm(Purchase ~ ., data = OJ, subset = train, kernel = "linear", cost = 0.01)
summary(svm.linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, kernel = "linear", cost = 0.01,
##     subset = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  427
##
##  ( 213 214 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Support vector classifier creates 427 support vectors out of 800 training points. Out of these, 213 belong to level CH and remaining 214 belong to level MM.

(c)

```
# training error
pred <- predict(svm.linear, newdata = OJ.train, type = "response")
```

```r
result <- table(OJ.train$Purchase, pred)
result
```

```
##     pred
##       CH  MM
##   CH 440  47
##   MM  73 240
```

```r
training.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,2])
training.error.rate
```

```
## [1] 0.15
```

The training error rate os 0.15

```r
# test error
pred <- predict(svm.linear, newdata = OJ.test, type = "response")
result <- table(OJ.test$Purchase, pred)
result
```

```
##     pred
##       CH  MM
##   CH 142  24
##   MM  30  74
```

```r
test.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,2])
test.error.rate
```

```
## [1] 0.2
```

The test error rate is 0.2.

(d)

```r
set.seed(679)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", ranges = list(cost = 10^seq(-2,
bestmod <- tune.out$best.parameters
bestmod
```

```
##   cost
## 1 0.01
```

The best cost is 0.01.

(e)

```r
svm.linear <- svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = tune.out$best.parameter$cost)
train.pred <- predict(svm.linear, newdata = OJ.train, type = "response")
result <- table(OJ.train$Purchase, train.pred)
result
```

```
##     train.pred
##       CH  MM
##   CH 440  47
##   MM  73 240
```

```r
training.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,
training.error.rate
```

```
## [1] 0.15
```

```r
test.pred <- predict(svm.linear, newdata = OJ.test, type = "response")
result <- table(OJ.test$Purchase, test.pred)

result
```

```
##     test.pred
##       CH  MM
##   CH 142  24
##   MM  30  74
```

```r
test.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,2])
test.error.rate
```

```
## [1] 0.2
```

(f)

```r
svm.radial <- svm(Purchase ~ ., kernel = "radial", data = OJ.train)
summary(svm.radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  362
##
##  ( 176 186 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
train.pred <- predict(svm.radial, newdata = OJ.train, type = "response")
result <- table(OJ.train$Purchase, train.pred)
result
```

```
##     train.pred
##       CH  MM
##   CH 447  40
##   MM  74 239
```

```r
training.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,
training.error.rate
```

```
## [1] 0.1425
```

```r
test.pred <- predict(svm.radial, newdata = OJ.test, type = "response")
result <- table(OJ.test$Purchase, test.pred)

result
```

```
##     test.pred
##       CH  MM
##   CH 145  21
##   MM  33  71
```

```
test.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,2])
test.error.rate
```

```
## [1] 0.2
```

```
set.seed(679)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", ranges = list(cost = 10^seq(-2,
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##       cost
##  0.5623413
##
## - best performance: 0.15625
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39125 0.04678927
## 2   0.01778279 0.39125 0.04678927
## 3   0.03162278 0.34875 0.05756940
## 4   0.05623413 0.17875 0.02889757
## 5   0.10000000 0.17750 0.03270236
## 6   0.17782794 0.16250 0.02946278
## 7   0.31622777 0.15750 0.02838231
## 8   0.56234133 0.15625 0.02716334
## 9   1.00000000 0.16750 0.02513851
## 10  1.77827941 0.16750 0.03395258
## 11  3.16227766 0.16875 0.02960973
## 12  5.62341325 0.17000 0.03872983
## 13 10.00000000 0.17750 0.02934469
```

```
svm.radial <- svm(Purchase ~ ., kernel = "radial", data = OJ.train, cost = tune.out$best.parameter$cost)
summary(svm.radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial", cost = tune.out$best.parameter$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.5623413
##
## Number of Support Vectors:  391
##
```

```
##  ( 193 198 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
train.pred <- predict(svm.radial, newdata = OJ.train, type = "response")
result <- table(OJ.train$Purchase, train.pred)
result
```

```
##     train.pred
##       CH  MM
##   CH 449  38
##   MM  76 237
```

```r
training.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,2])
training.error.rate
```

```
## [1] 0.1425
```

```r
test.pred <- predict(svm.radial, newdata = OJ.test, type = "response")
result <- table(OJ.test$Purchase, test.pred)

result
```

```
##     test.pred
##       CH  MM
##   CH 144  22
##   MM  33  71
```

```r
test.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,2])
test.error.rate
```

```
## [1] 0.2037037
```

(g)

```r
svm.poly <- svm(Purchase ~ ., kernel = "polynomial", data = OJ.train, degree = 2)
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  448
##
##  ( 219 229 )
##
```

```
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.poly, newdata = OJ.train, type = "response")
result <- table(OJ.train$Purchase, train.pred)
result
```

```
##     train.pred
##       CH  MM
##   CH 458  29
##   MM 106 207
```

```
training.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,
training.error.rate
```

```
## [1] 0.16875
```

```
test.pred <- predict(svm.radial, newdata = OJ.test, type = "response")
result <- table(OJ.test$Purchase, test.pred)

result
```

```
##     test.pred
##       CH  MM
##   CH 144  22
##   MM  33  71
```

```
test.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,2])
test.error.rate
```

```
## [1] 0.2037037
```

```
set.seed(679)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "polynomial", degree = 2, ranges = list(c
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.16625
##
## - Detailed performance results:
##          cost   error dispersion
## 1   0.01000000 0.39000 0.04116363
## 2   0.01778279 0.36875 0.05111602
## 3   0.03162278 0.35625 0.04535738
## 4   0.05623413 0.34500 0.05244044
## 5   0.10000000 0.31750 0.04005205
## 6   0.17782794 0.22500 0.03996526
```

```
## 7   0.31622777 0.20125 0.03793727
## 8   0.56234133 0.19875 0.03408018
## 9   1.00000000 0.18500 0.02993047
## 10  1.77827941 0.17875 0.03335936
## 11  3.16227766 0.17375 0.03030516
## 12  5.62341325 0.16875 0.03186887
## 13 10.00000000 0.16625 0.02949223
```

```
svm.poly <- svm(Purchase ~ ., kernel = "polynomial", degree = 2, data = OJ.train, cost = tune.out$best.p
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial",
##     degree = 2, cost = tune.out$best.parameter$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  330
##
##  ( 164 166 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.poly, newdata = OJ.train, type = "response")
result <- table(OJ.train$Purchase, train.pred)
result
```

```
##     train.pred
##       CH  MM
##   CH 455  32
##   MM  76 237
```

```
training.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,
training.error.rate
```

```
## [1] 0.135
```

```
test.pred <- predict(svm.radial, newdata = OJ.test, type = "response")
result <- table(OJ.test$Purchase, test.pred)

result
```

```
##     test.pred
##       CH  MM
##   CH 144  22
##   MM  33  71
```

```r
test.error.rate <- (result[2,1] + result[1,2])/(result[1,1] + result[1,2] + result[2,1] + result[2,2])
test.error.rate
```

```
## [1] 0.2037037
```

(h) ***sol.n:*** In conclusion, radial kernel could produce minimum misclassification error on both train and test data.