SVM Homework

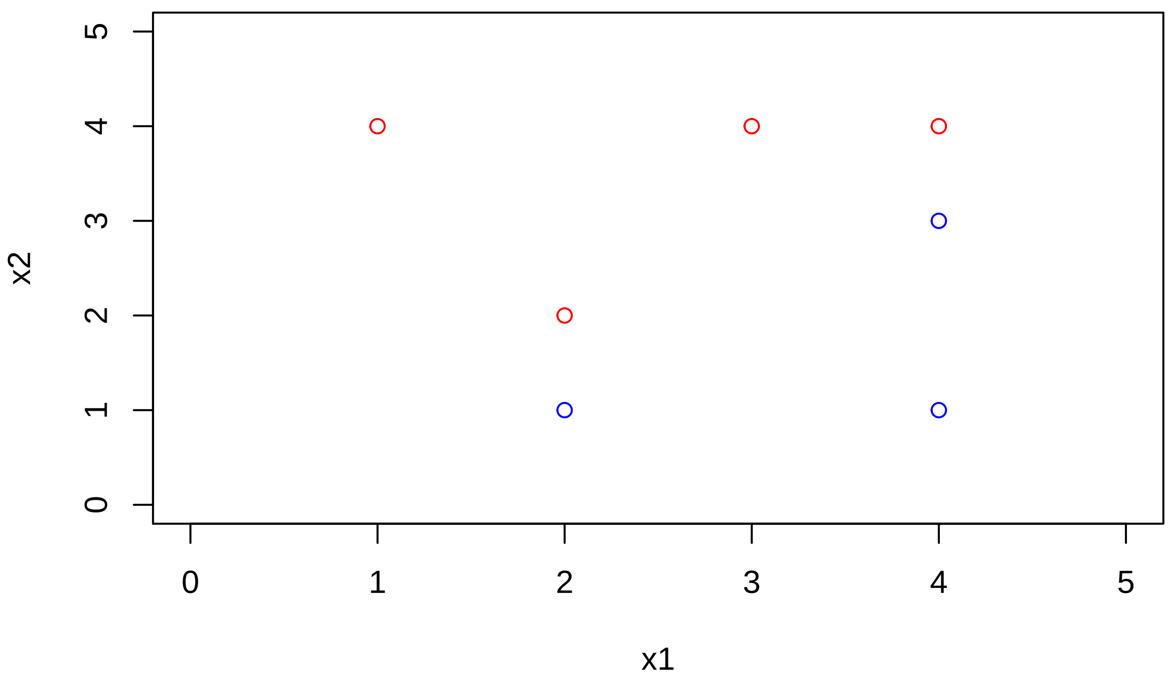
Megha Pandit

March 13, 2019

Problem 3

(a)

```
x1 <- c(3, 2, 4, 1, 2, 4, 4)
x2 <- c(4, 2, 4, 4, 1, 3, 1)
cols <- c("red", "red", "red", "red", "blue", "blue", "blue")
plot(x1, x2, col = cols, xlim = c(0,5), ylim = c(0,5))
```



(b)

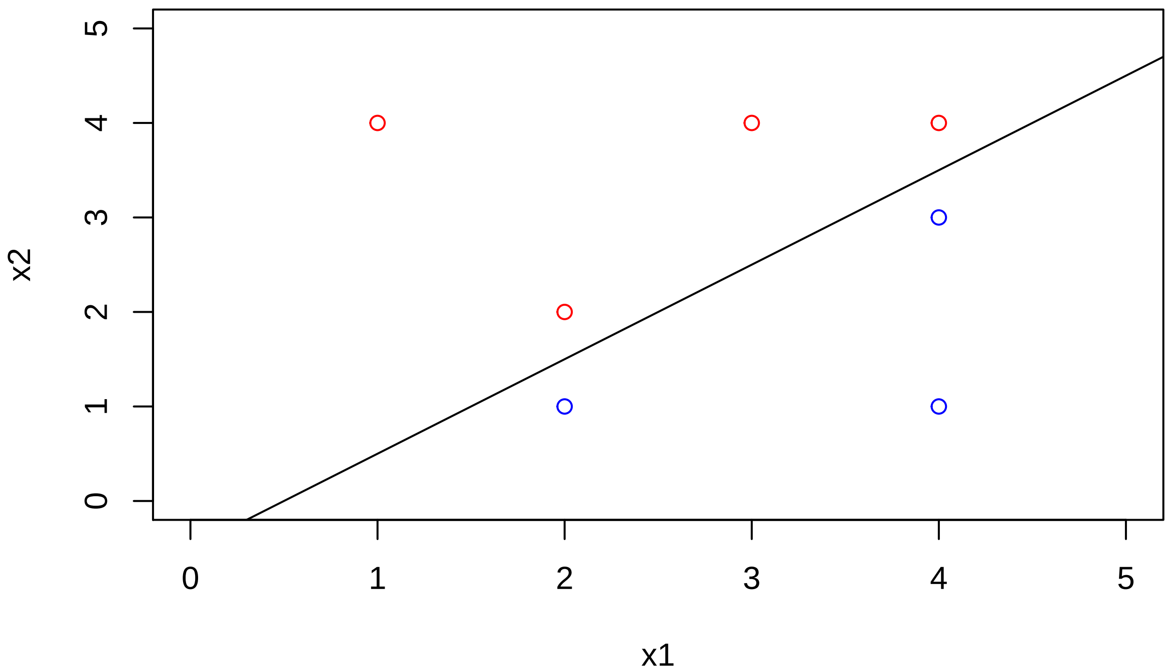
Since the two classes are very distinctly separable, we can see from the plot that the hyperplace must lie in between the points {(2,2), (4,4)} and {(2,1), (4,3)}. Therefore, the hyperplane will pass through the points (2, 1.5) and (4, 3.5). The equation for the line passing through these

two points is:

$$x2 = -0.5 + x1$$

Therefore, the intercept and slope are -0.5 and 1 respectively.

```
#Optimal separating hyperplane
plot(x1, x2, col = cols, xlim = c(0,5), ylim = c(0,5))
abline(-0.5, 1)
```

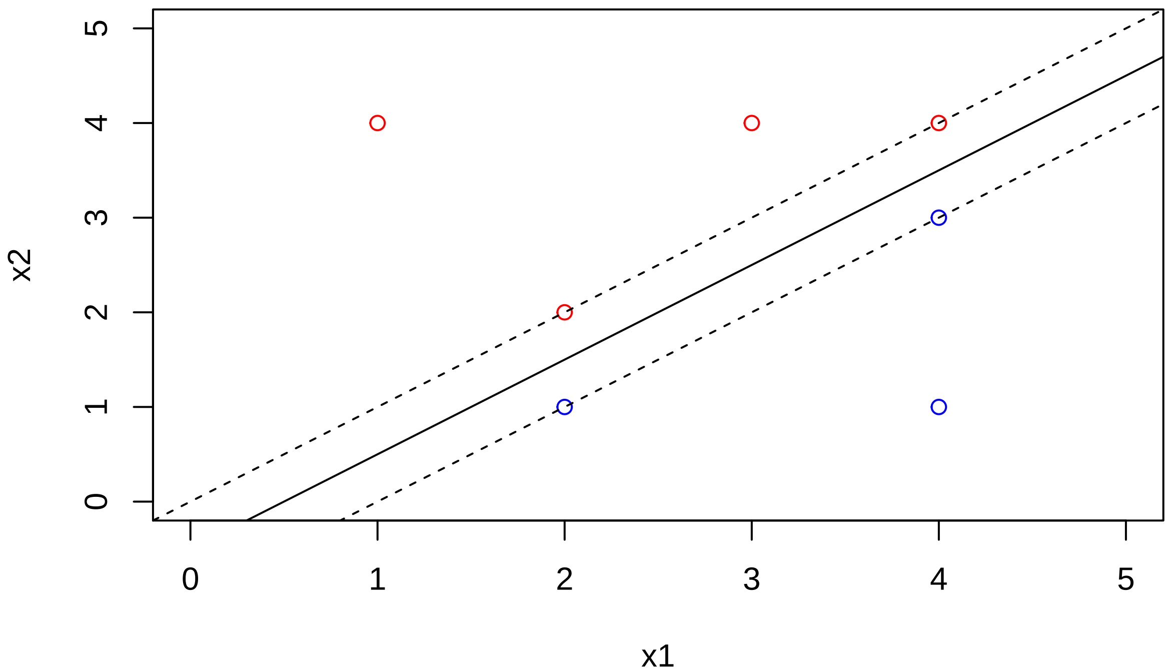


(c)

The classification rule here would be: Classify as Red if $x2 - x1 + 0.5 > 0$ **, and, classify as Blue if** $x2 - x1 + 0.5 < 0$

(d)

```
#Margin for maximal margin hyperplane
plot(x1, x2, col = cols, xlim = c(0,5), ylim = c(0,5))
abline(-0.5, 1)
abline(-1, 1, lty = 2)
abline(0, 1, lty = 2)
```



(e)

The support vectors for the maximal margin classifier are the points (2,1), (2,2), (4,3) and (4,4).

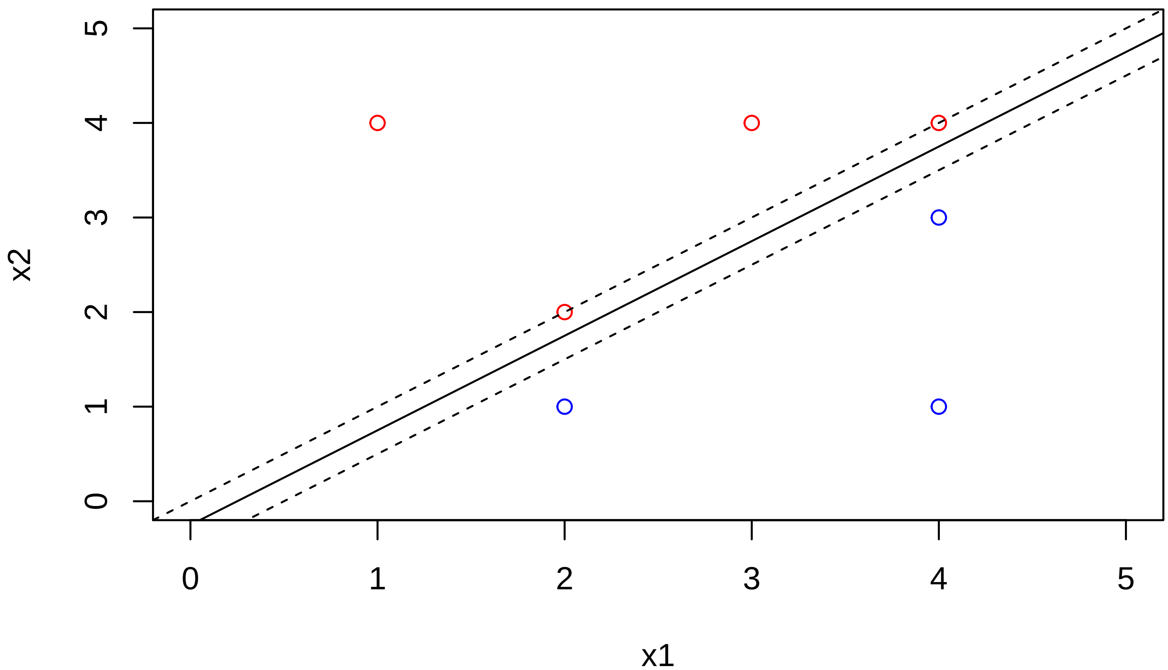
(f)

Since the 7th observation is not a support vector, a slight change in its position will not affect the maximal margin hyperplane. This is evident even from the above plot.

(g)

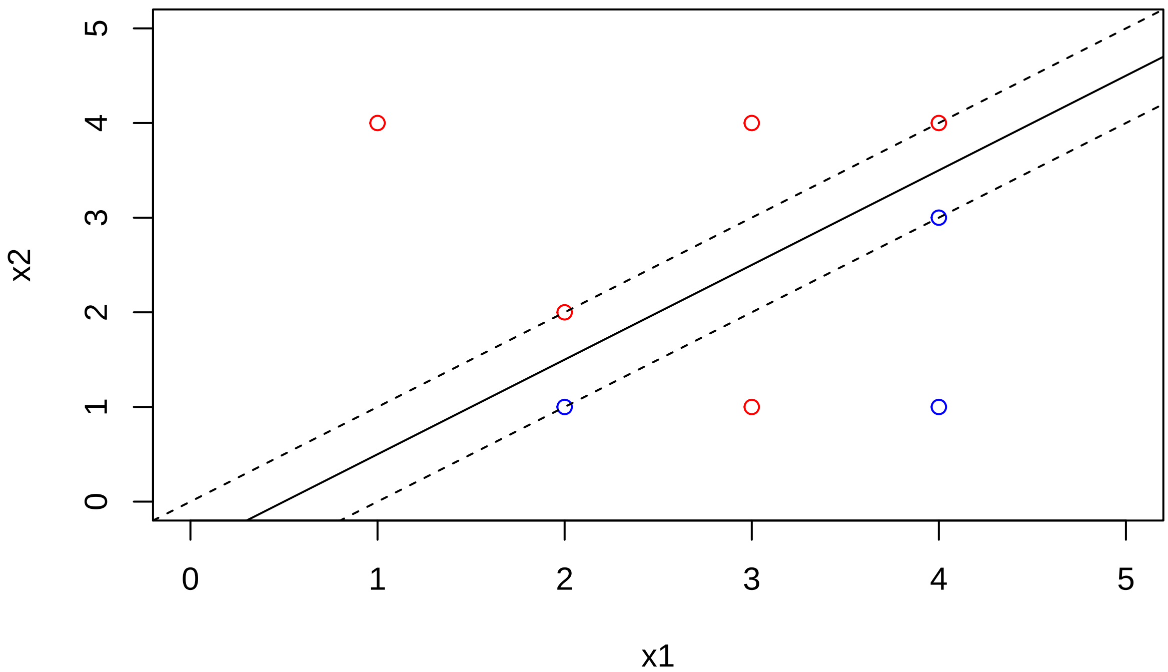
The equation $x2 = -0.25 + x1$ will also separate all the observations but is not an optimal hyperplane because the margin is smaller than the optimal option.

```
plot(x1, x2, col = cols, xlim = c(0,5), ylim = c(0,5))
abline(-0.25, 1)
abline(0, 1, lty = 2)
abline(-0.5, 1, lty = 2)
```



(h)

```r
plot(x1, x2, col = cols, xlim = c(0,5), ylim = c(0,5))
abline(-0.5, 1)
abline(-1, 1, lty = 2)
abline(0, 1, lty = 2)
points(3,1, col = "red")
```



From the above plot, we see that after adding the new point, the hyperplane cannot separate the classes.

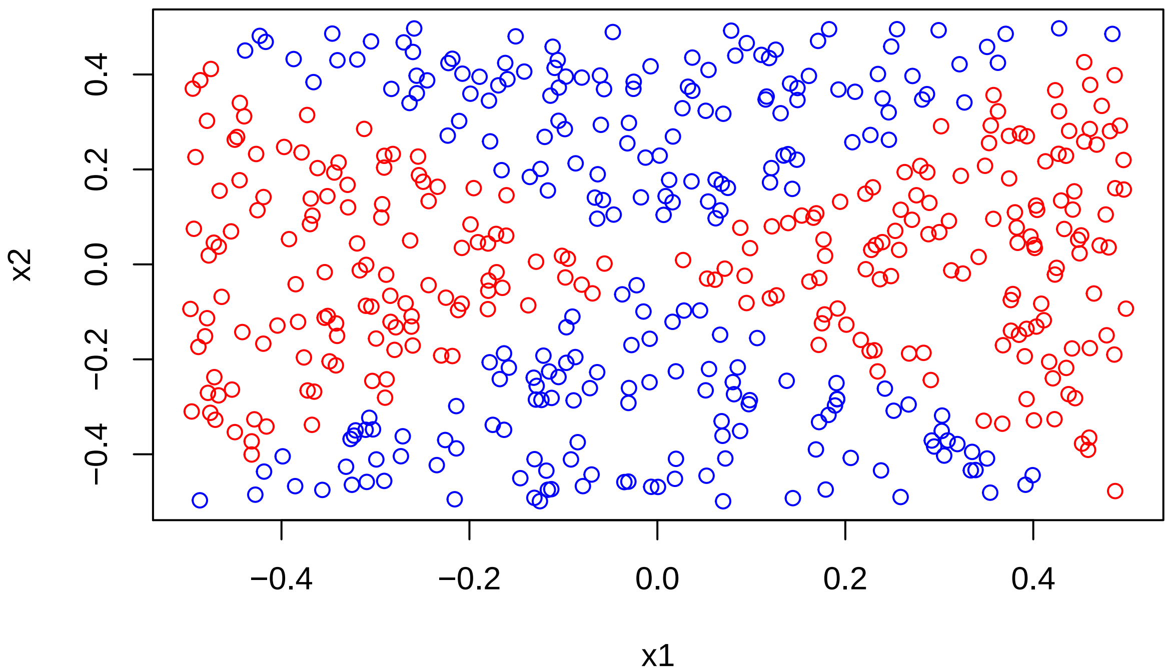
Problem 5

(a)

```r
set.seed(9)
x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- 1*(x1^2 - x2^2 > 0)
```

(b)

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



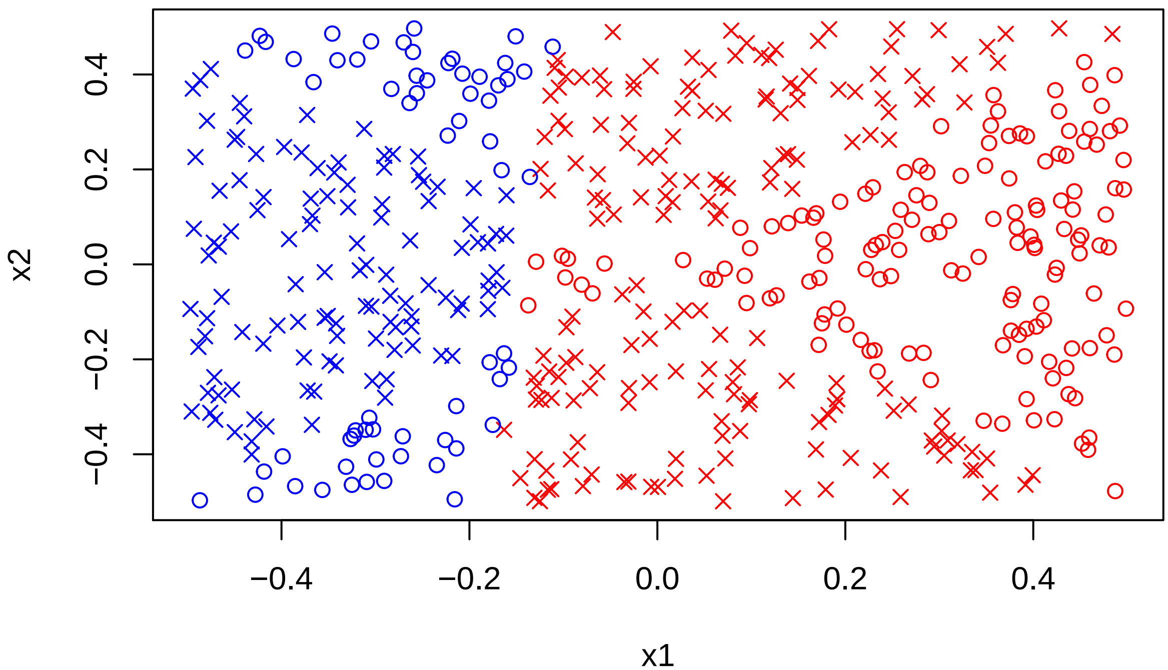
(c)

```
#Logistic regression
df <- data.frame(x1, x2, y)
fit_glm <- glm(y ~ x1 + x2, data = df, family = binomial)
fit_glm
```

```
##
## Call:  glm(formula = y ~ x1 + x2, family = binomial, data = df)
##
## Coefficients:
## (Intercept)           x1           x2
##     0.05514      0.38587     -0.02653
##
## Degrees of Freedom: 499 Total (i.e. Null);  497 Residual
## Null Deviance:       692.8
## Residual Deviance: 691.2     AIC: 697.2
```

(d)

```
pred_fit <- predict(fit_glm, data.frame(x1,x2))
plot(x1, x2, col = ifelse(pred_fit > 0, "red", "blue"), pch = ifelse(as.integer(pred_fit > 0) == y, 1,4
```



In the above plot, the circles are the observations that have been classified correctly and the the crosses are the ones that are misclassified. The decision boundary looks linear.

(e)

```r
fit_glm1 <- glm(y ~ poly(x1, 2) + poly(x2, 2), data = df, family = binomial)
```

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

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

```r
summary(fit_glm1)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2) + poly(x2, 2), family = binomial,
##     data = df)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -1.079e-03  -2.000e-08   2.000e-08   2.000e-08   9.076e-04
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)       43.78    3063.63   0.014    0.989
## poly(x1, 2)1    1360.39  102905.10   0.013    0.989
```

```
## poly(x1, 2)2  21374.91  785951.63   0.027    0.978
## poly(x2, 2)1   -119.10   88918.85  -0.001    0.999
## poly(x2, 2)2 -21333.50  788724.67  -0.027    0.978
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9276e+02  on 499  degrees of freedom
## Residual deviance: 2.4730e-06  on 495  degrees of freedom
## AIC: 10
##
## Number of Fisher Scoring iterations: 25
```

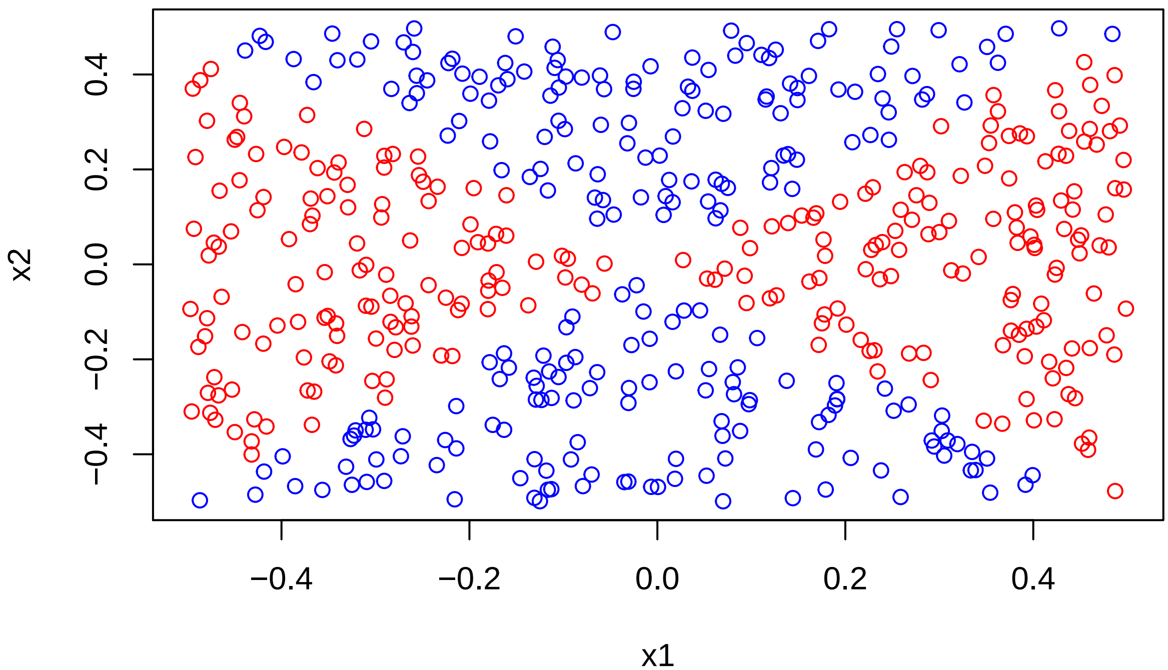
```
fit_glm2 <- glm(y ~ x1 + x2 + x1*x2, data = df, family = binomial)
summary(fit_glm2)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2 + x1 * x2, family = binomial, data = df)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.342  -1.199   1.050   1.144   1.291
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.05206    0.08983   0.580    0.562
## x1           0.38234    0.31036   1.232    0.218
## x2          -0.01969    0.31760  -0.062    0.951
## x1:x2        0.64537    1.12041   0.576    0.565
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.76  on 499  degrees of freedom
## Residual deviance: 690.87  on 496  degrees of freedom
## AIC: 698.87
##
## Number of Fisher Scoring iterations: 3
```

None of the coefficient estimates are statistically significant.

(f)

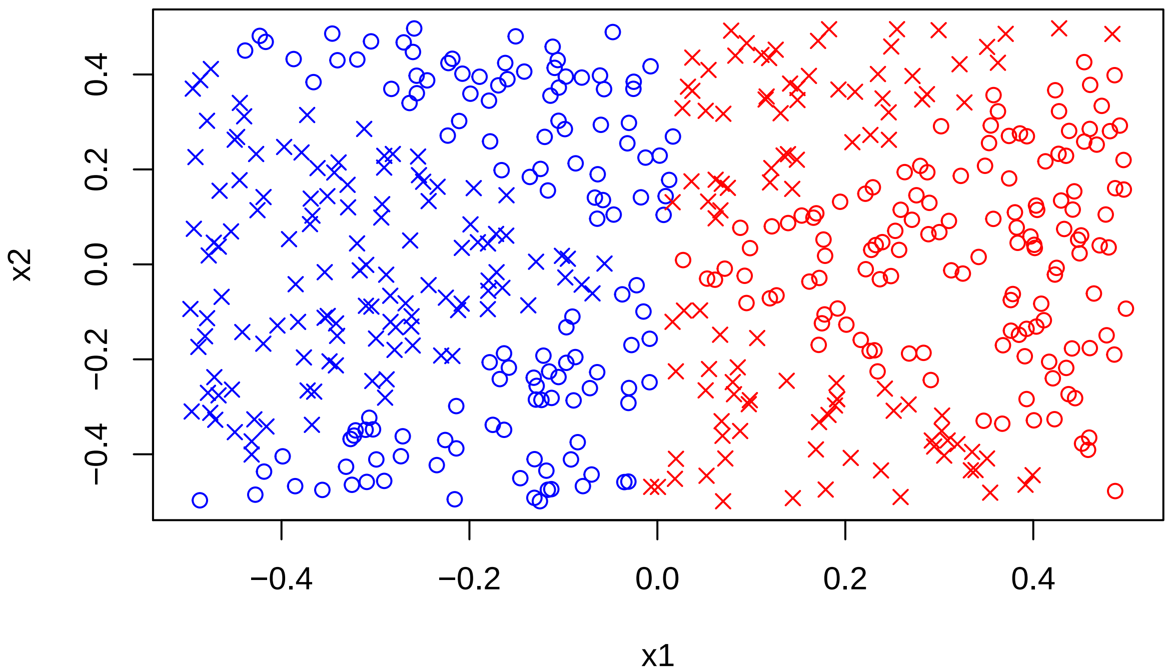
```
pred_fit1 <- predict(fit_glm1, df)
plot(x1, x2, col = ifelse(pred_fit1 > 0, "red", "blue"), pch = ifelse(as.integer(pred_fit1 > 0) == y, 1
```



The plot above shows a non-linear decision boundary and all the observations are correctly classified. Also, the decision boundary is similar to the true decision boundary.

(g)

```
#Support Vector Classifier
df$y <- as.factor(df$y)
fit_svc <- svm(y ~ x1 + x2, data = df, kernel = "linear")
pred_svc <- predict(fit_svc, df, type = "response")
plot(x1, x2, col = ifelse(pred_svc != 0, "red", "blue"), pch = ifelse(pred_svc == y, 1,4))
```

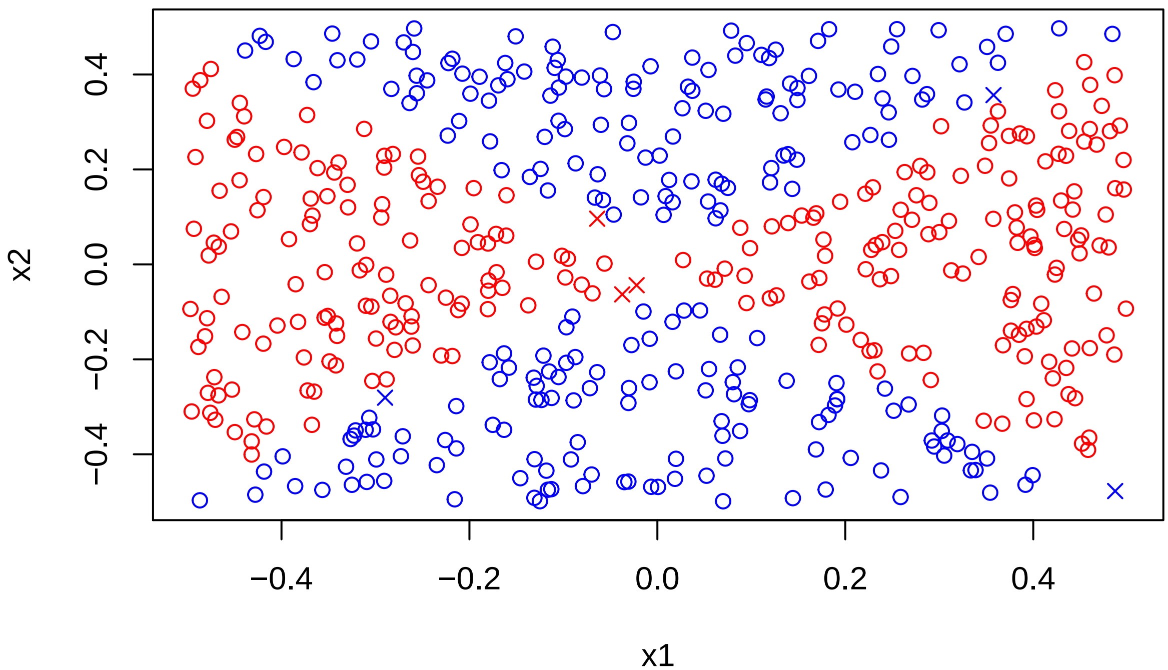


In the above plot, the circles represent observations that have been classified correctly and crosses represent the observations that have been misclassified.

(h)

```
#SVM with non-linear kernel

fit_svm <- svm(y ~ x1 + x2, data = df, kernel = "polynomial", degree = 2)
pred_svm <- predict(fit_svm, df, type = "response")
plot(x1, x2, col = ifelse(pred_svm != 0, "red", "blue"), pch = ifelse(pred_svm == y, 1,4))
```



There is a drastic improvement in classification compared to the linear kernel.

(i)

From the results, we see that SVM with a polynomial kernel of degree 2 performs quite well but still misclassifies some observations. In contrast, logistic regression with non-linear functions of predictors (polynomials of degree 2) does not misclassify any observations and definitely performs better than SVM.

Problem 7

(a)

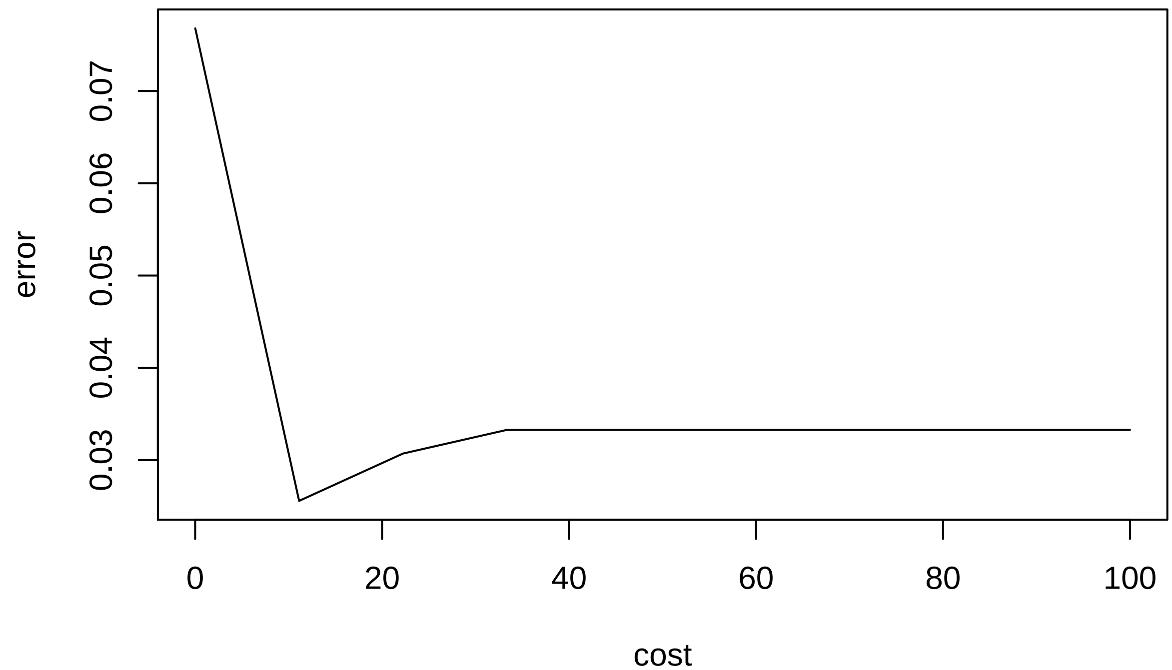
```
data("Auto")
Auto$Y <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
Auto$Y <- as.factor(Auto$Y)
```

(b)

```r
set.seed(9)
cost <- data.frame(cost = seq(0.01, 100, length.out = 10))
svm_tune <- tune(svm, Y ~ ., data = Auto, kernel = "linear", ranges = cost)
summary(svm_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##    cost
##   11.12
##
## - best performance: 0.02557692
##
## - Detailed performance results:
##       cost      error dispersion
## 1     0.01 0.07679487 0.04850079
## 2    11.12 0.02557692 0.02417544
## 3    22.23 0.03070513 0.02649650
## 4    33.34 0.03326923 0.03211170
## 5    44.45 0.03326923 0.03211170
## 6    55.56 0.03326923 0.03211170
## 7    66.67 0.03326923 0.03211170
## 8    77.78 0.03326923 0.03211170
## 9    88.89 0.03326923 0.03211170
## 10 100.00 0.03326923 0.03211170
```

```r
plot(svm_tune$performances[,c(1,2)], type = "l")
```



Cost = 11.12 seems to perform the best. This is also seen in the plot.

(c)

```
#Polynomial Kernel
para <- data.frame(cost = seq(0.01, 100, length.out = 5), degree = seq(1, 100, length.out = 5))

svm_poly <- tune(svm, Y ~ ., data = Auto, kernel = "polynomial", ranges = para)
summary(svm_poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      1
##
## - best performance: 0.01519231
##
## - Detailed performance results:
##         cost degree      error dispersion
## 1     0.0100   1.00 0.56115385 0.02988888
```

```
## 2   25.0075   1.00 0.05089744 0.03945905
## 3   50.0050   1.00 0.02782051 0.04182274
## 4   75.0025   1.00 0.02025641 0.02601613
## 5  100.0000   1.00 0.01519231 0.02135480
## 6    0.0100  25.75 0.56115385 0.02988888
## 7   25.0075  25.75 0.56115385 0.02988888
## 8   50.0050  25.75 0.56115385 0.02988888
## 9   75.0025  25.75 0.56115385 0.02988888
## 10 100.0000  25.75 0.56115385 0.02988888
## 11   0.0100  50.50 0.56115385 0.02988888
## 12  25.0075  50.50 0.56115385 0.02988888
## 13  50.0050  50.50 0.56115385 0.02988888
## 14  75.0025  50.50 0.56115385 0.02988888
## 15 100.0000  50.50 0.56115385 0.02988888
## 16   0.0100  75.25 0.56115385 0.02988888
## 17  25.0075  75.25 0.56115385 0.02988888
## 18  50.0050  75.25 0.56115385 0.02988888
## 19  75.0025  75.25 0.56115385 0.02988888
## 20 100.0000  75.25 0.56115385 0.02988888
## 21   0.0100 100.00 0.56115385 0.02988888
## 22  25.0075 100.00 0.56115385 0.02988888
## 23  50.0050 100.00 0.56115385 0.02988888
## 24  75.0025 100.00 0.56115385 0.02988888
## 25 100.0000 100.00 0.56115385 0.02988888
```

Cost of 100 with degree 1 seems to perform the best.

```r
#Radial Kernel

params <- data.frame(cost=seq(0.01,100,length.out = 5),gamma=seq(0.1,100,length.out = 5))
svm_radial <- tune(svm, Y ~ ., data = Auto, kernel = "radial", ranges = params)
summary(svm_radial)
```

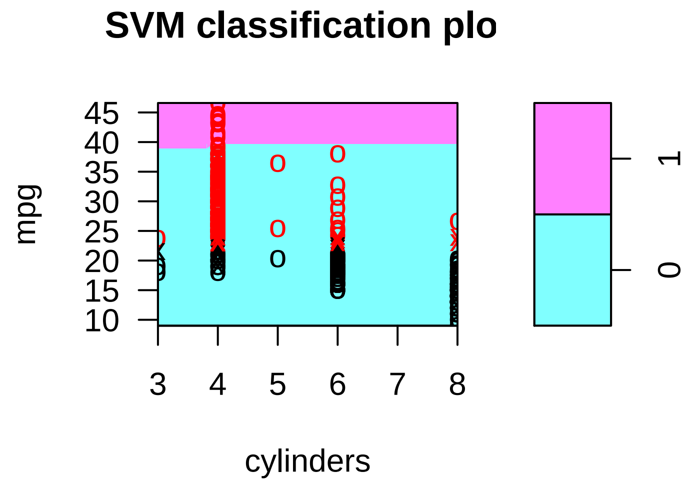
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##     cost gamma
##  25.0075   0.1
##
## - best performance: 0.02532051
##
## - Detailed performance results:
##        cost   gamma      error dispersion
## 1    0.0100   0.100 0.19365385 0.08464044
## 2   25.0075   0.100 0.02532051 0.03129777
## 3   50.0050   0.100 0.02532051 0.03355075
## 4   75.0025   0.100 0.02532051 0.03355075
## 5  100.0000   0.100 0.02532051 0.03355075
## 6    0.0100  25.075 0.56115385 0.03834765
## 7   25.0075  25.075 0.54596154 0.03688365
## 8   50.0050  25.075 0.54596154 0.03688365
## 9   75.0025  25.075 0.54596154 0.03688365
```

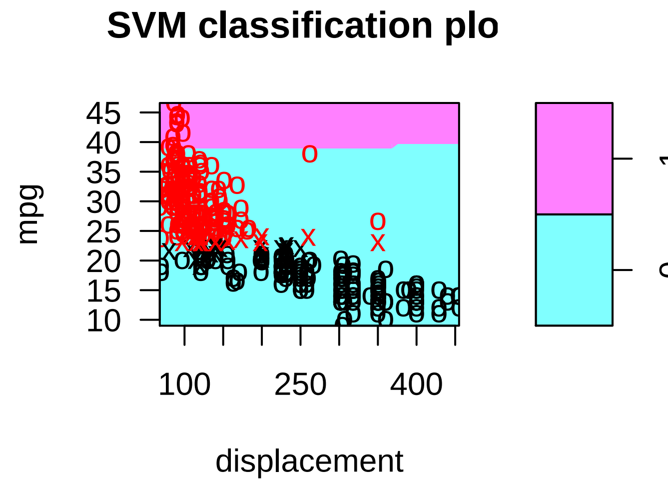
```
## 10 100.0000  25.075 0.54596154 0.03688365
## 11   0.0100  50.050 0.56115385 0.03834765
## 12  25.0075  50.050 0.55608974 0.03870442
## 13  50.0050  50.050 0.55608974 0.03870442
## 14  75.0025  50.050 0.55608974 0.03870442
## 15 100.0000  50.050 0.55608974 0.03870442
## 16   0.0100  75.025 0.56115385 0.03834765
## 17  25.0075  75.025 0.56115385 0.03834765
## 18  50.0050  75.025 0.56115385 0.03834765
## 19  75.0025  75.025 0.56115385 0.03834765
## 20 100.0000  75.025 0.56115385 0.03834765
## 21   0.0100 100.000 0.56115385 0.03834765
## 22  25.0075 100.000 0.56115385 0.03834765
## 23  50.0050 100.000 0.56115385 0.03834765
## 24  75.0025 100.000 0.56115385 0.03834765
## 25 100.0000 100.000 0.56115385 0.03834765
```

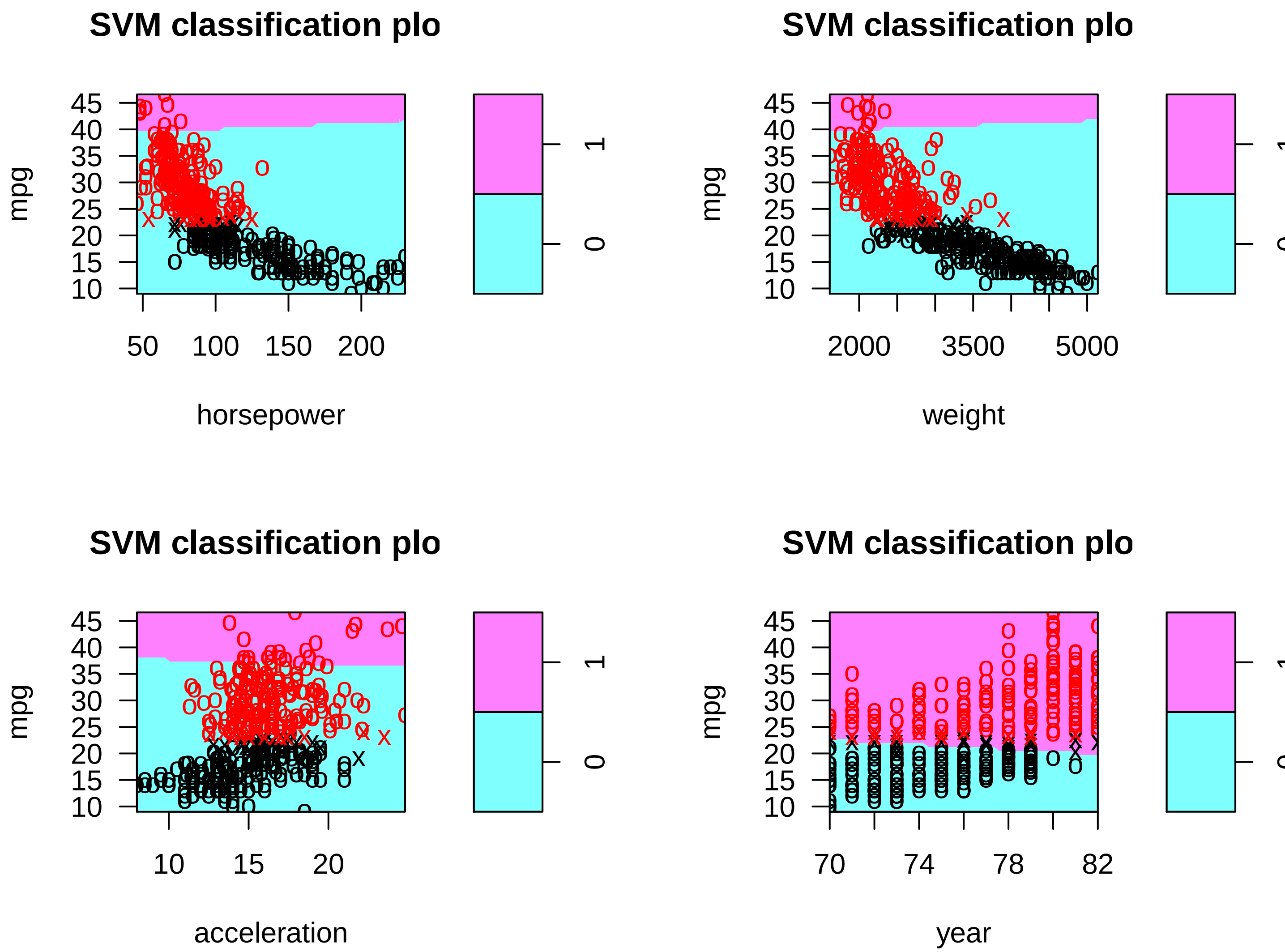
(d)

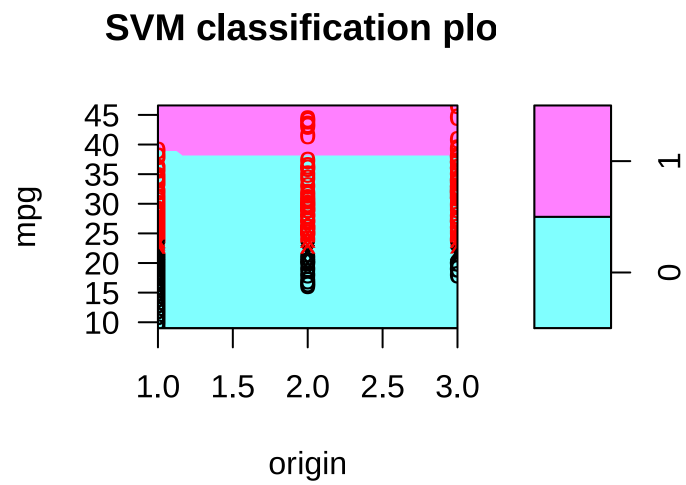
```r
linear <- svm(Y ~ ., data = Auto, kernel = "linear", cost = 11.12)
polynomial <- svm(Y ~ ., data = Auto, kernel = "polynomial", cost = 100, degree = 1)
radial <- svm(Y ~ ., data = Auto, kernel = "radial", cost = 25.0075, gamma = 0.1)
pair_plot <- function(a){
  for (name in names(Auto)[!(names(Auto) %in% c("mpg", "Y", "name"))])
    plot(a, Auto, as.formula(paste("mpg~", name, sep = "")))
}

pair_plot(linear)
```



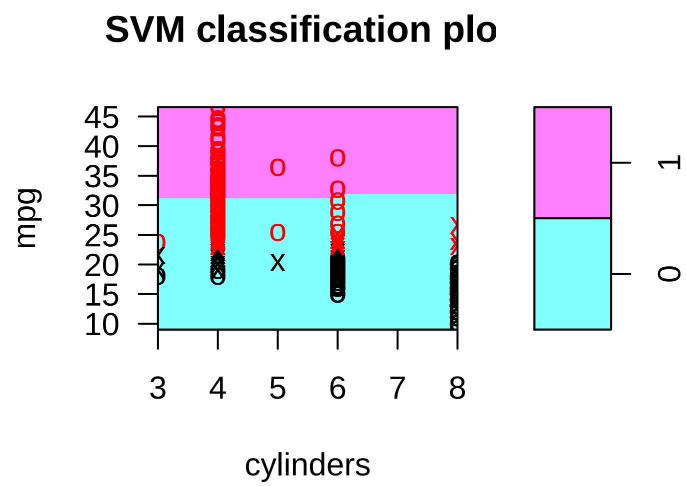


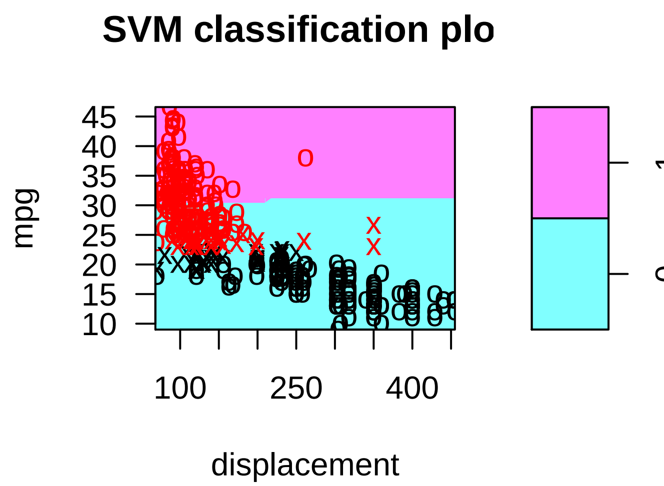

SVM classification plo
mpg
45
40
35
30
25
20
15
10
50
100
150
200
horsepower
1
0
SVM classification plo
mpg
2000
3500
5000
weight
SVM classification plo
mpg
10
15
20
acceleration
SVM classification plo
mpg
70
74
78
82
year

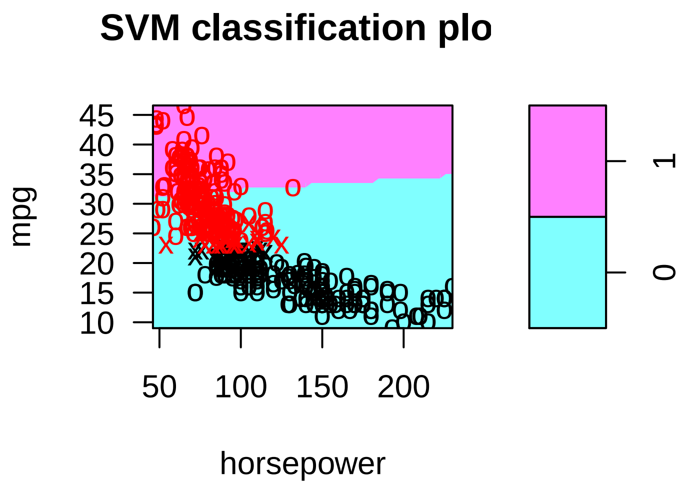


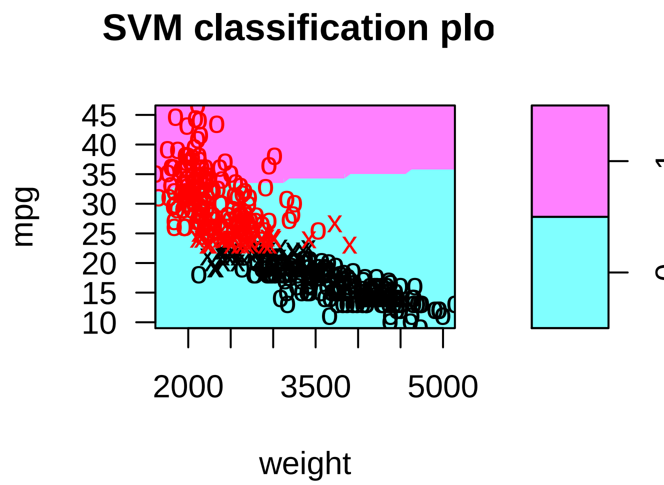
The above are the SVM classification plots for linear kernel.

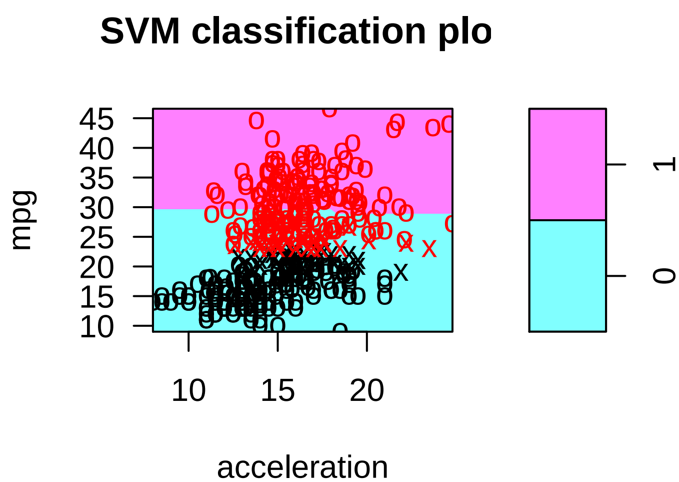
```r
pair_plot(polynomial)
```

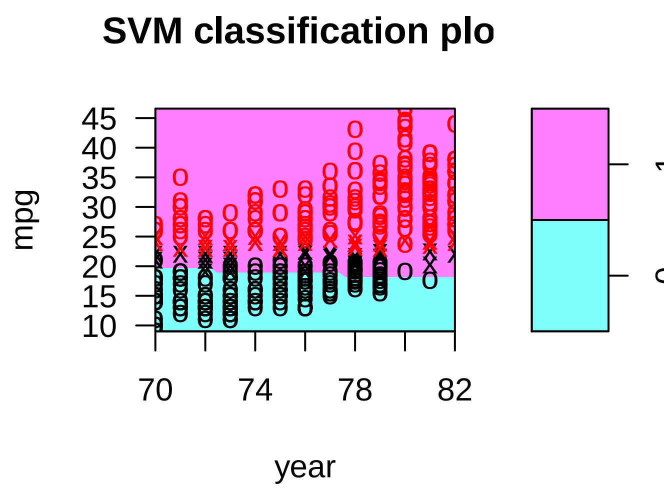


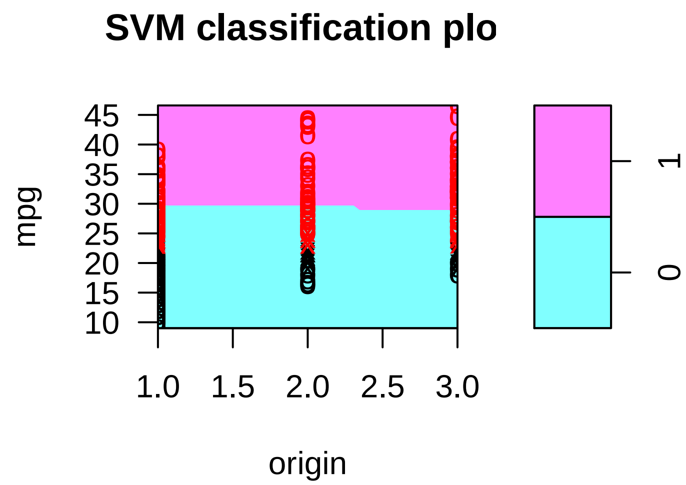



SVM classification plo
mpg
45
40
35
30
25
20
15
10
50
100
150
200
horsepower
1
0


SVM classification plo
mpg
45
40
35
30
25
20
15
10
2000
3500
5000
weight

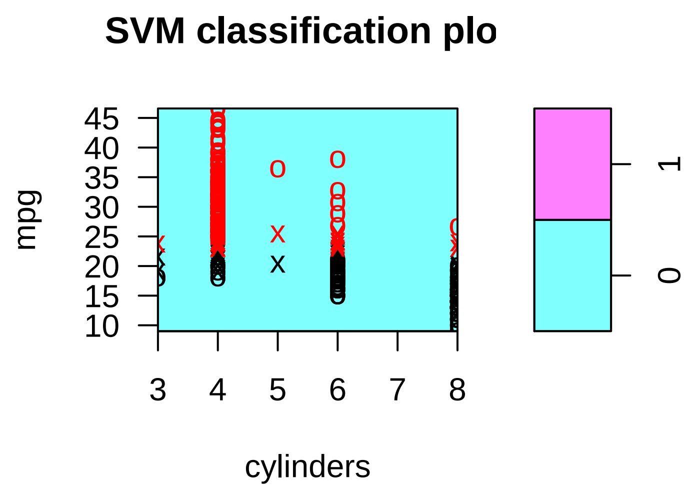

SVM classification plo
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0

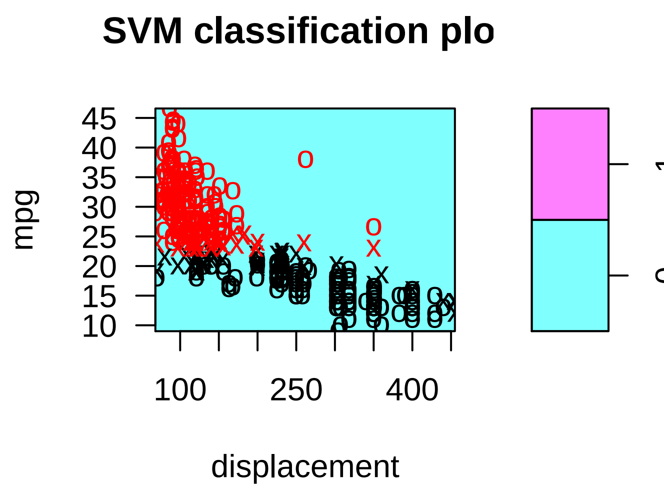

SVM classification plo
mpg
45
40
35
30
25
20
15
10
70
74
78
82
year

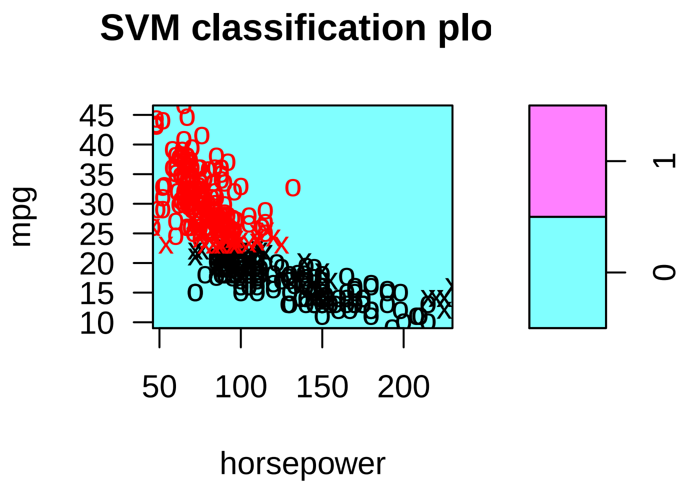


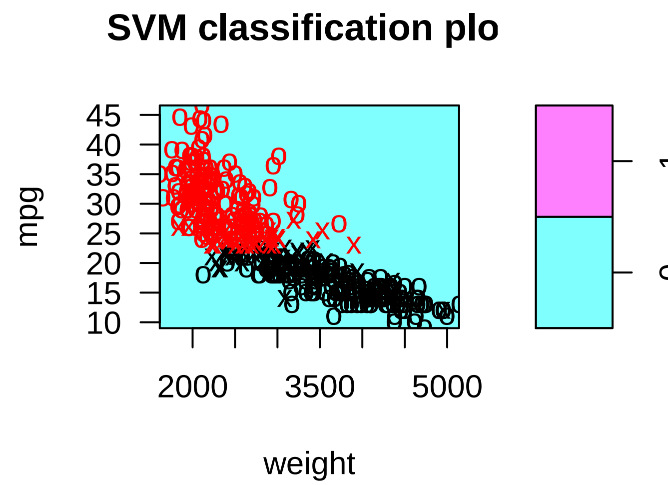
Thde above plots are the SVM classification plots for polynomial kernel.

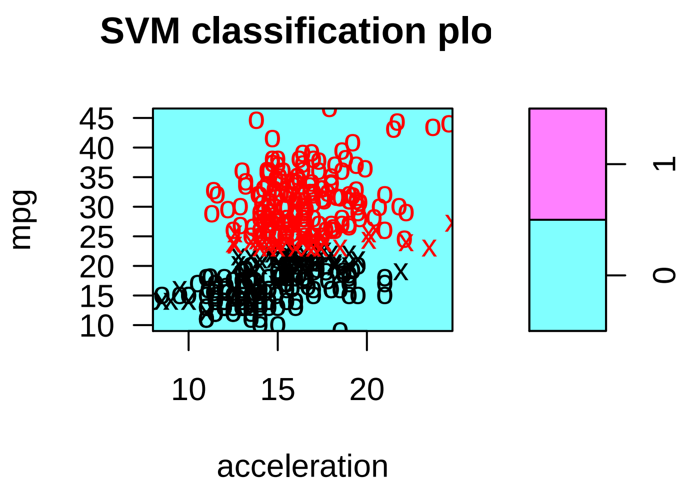
```r
pair_plot(radial)
```

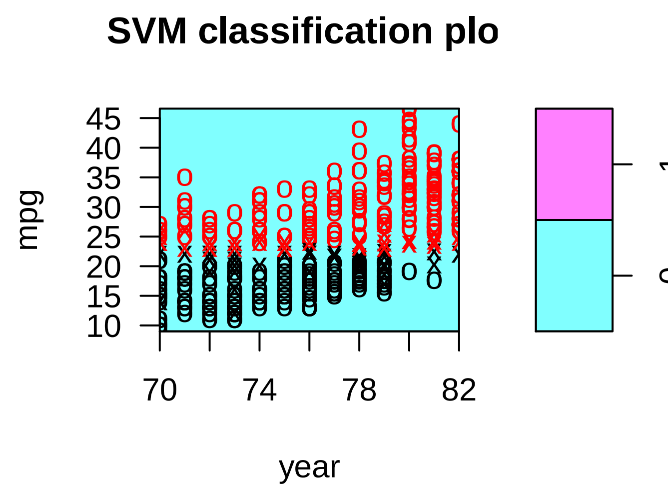


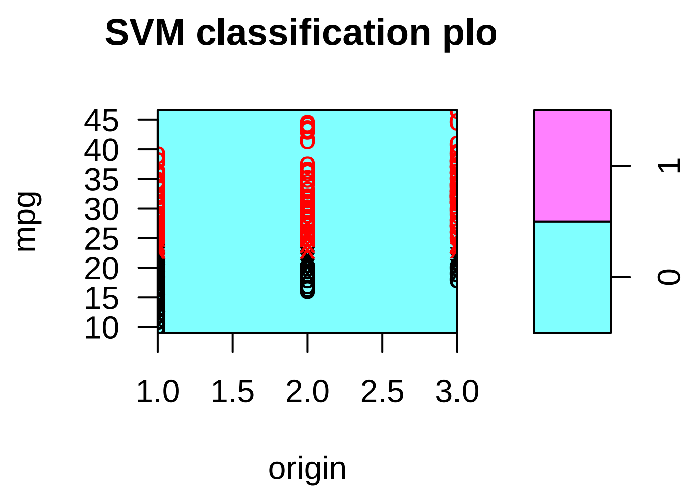



SVM classification plo
mpg
45
40
35
30
25
20
15
10
50
100
150
200
horsepower
1
0


SVM classification plo
mpg
45
40
35
30
25
20
15
10
2000
3500
5000
weight


SVM classification plo
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0


SVM classification plo
mpg
45
40
35
30
25
20
15
10
70
74
78
82
year



Problem 8

(a)

```
data("OJ")
set.seed(9)
train_oj <- sample(nrow(OJ), 800)
oj_train <- OJ[train_oj,]
oj_test <- OJ[-train_oj,]
```

(b)

```
oj_svc <- svm(Purchase ~ ., data = oj_train, kernel = "linear", cost = 0.01)
summary(oj_svc)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj_train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  427
```

```
##
##  ( 214 213 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The support vector classifier creates 432 support vectors out of the 800 training observations. Out of the 432 support vectors, 214 belong to level CH and 213 to level MM.

(c)

```r
#Training error rate

pred_train <- predict(oj_svc, oj_train)
table(pred_train, oj_train$Purchase)
```

```
##
## pred_train  CH  MM
##         CH 428  70
##         MM  61 241
```

```r
#Test error rate

pred_test <- predict(oj_svc, oj_test)
table(pred_test, oj_test$Purchase)
```

```
##
## pred_test  CH  MM
##        CH 143  33
##        MM  21  73
```

```r
(tr_error<- (70+61)/(428+70+61+241))
```

```
## [1] 0.16375
```

```r
(te_error <- (33+21)/(143+33+21+73))
```

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

The training error rate is 16.37% and the test error rate is 20%.

(d)

```r
#For optimal cost

oj_tune <- tune(svm, Purchase ~ ., data = oj_train, kernel = "linear", ranges =
                  data.frame(cost = seq(0.01, 10, length.out = 25)))
summary(oj_tune)
```

```
##
## Parameter tuning of 'svm':
```

```
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##     cost
##  1.25875
##
## - best performance: 0.1575
##
## - Detailed performance results:
##        cost   error dispersion
## 1   0.01000 0.16750 0.05075814
## 2   0.42625 0.16250 0.04082483
## 3   0.84250 0.16250 0.04039733
## 4   1.25875 0.15750 0.03446012
## 5   1.67500 0.15750 0.03827895
## 6   2.09125 0.15750 0.04090979
## 7   2.50750 0.16125 0.04427267
## 8   2.92375 0.16125 0.04427267
## 9   3.34000 0.16250 0.04289846
## 10  3.75625 0.16375 0.04387878
## 11  4.17250 0.16375 0.04387878
## 12  4.58875 0.16250 0.04602234
## 13  5.00500 0.16500 0.04518481
## 14  5.42125 0.16500 0.04779877
## 15  5.83750 0.16500 0.04779877
## 16  6.25375 0.16625 0.04931827
## 17  6.67000 0.16625 0.04931827
## 18  7.08625 0.16875 0.05245699
## 19  7.50250 0.17000 0.05041494
## 20  7.91875 0.16875 0.04973890
## 21  8.33500 0.16875 0.04973890
## 22  8.75125 0.17000 0.04794383
## 23  9.16750 0.16750 0.05210833
## 24  9.58375 0.16750 0.05210833
## 25 10.00000 0.17000 0.04972145
```

From the results, we see that the optimal cost is 3.75625

(e)

```r
#Training error rate
oj_svm <- svm(Purchase ~ ., data = oj_train, kernel = "linear", cost = oj_tune$best.parameters$cost)
svm_train <- predict(oj_svm, oj_train)
table(svm_train, oj_train$Purchase)
```

```
##
## svm_train  CH  MM
##        CH 426  58
##        MM  63 253
```

```r
(tr_err <- (58+64)/(425+58+64+253))
```

```
## [1] 0.1525
```

```r
#Test error rate
svm_test <- predict(oj_svm, oj_test)
table(svm_test, oj_test$Purchase)
```

```
##
## svm_test  CH  MM
##       CH 142  27
##       MM  22  79
```

```r
(te_err <- (27+22)/(142+27+22+79))
```

```
## [1] 0.1814815
```

The training error rate is 15.25% and the test error rate is 18.15%.

(f)

Radial Kernel

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj_train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.05555556
##
## Number of Support Vectors:  362
##
##  ( 184 178 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The SVM with radial kernel creates 624 support vectors out of the 800 training observations. Out of the 624 support vectors, 313 belong to level CH and 311 to level MM.

```r
#Training error rate

radial_train <- predict(oj_radial, oj_train)
table(radial_train, oj_train$Purchase)
```

```
##
## radial_train  CH  MM
```

```
##           CH 441  70
##           MM  48 241
```

```r
#Test error rate

radial_test <- predict(oj_radial, oj_test)
table(radial_test, oj_test$Purchase)
```

```
##
## radial_test  CH  MM
##          CH 147  34
##          MM  17  72
```

The training error rate is 14.75% and the test error rate is 18.89%.

```r
#For optimal cost

radial_tune <- tune(svm, Purchase ~ ., data = oj_train, kernel = "radial", ranges =
                  data.frame(cost = seq(0.01, 10, length.out = 25)))
summary(radial_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##     cost
##  1.25875
##
## - best performance: 0.17375
##
## - Detailed performance results:
##        cost   error dispersion
## 1   0.01000 0.38875 0.04910660
## 2   0.42625 0.17500 0.03996526
## 3   0.84250 0.17625 0.03701070
## 4   1.25875 0.17375 0.04143687
## 5   1.67500 0.17875 0.04291869
## 6   2.09125 0.17875 0.04210189
## 7   2.50750 0.17875 0.03955042
## 8   2.92375 0.18375 0.04332131
## 9   3.34000 0.18250 0.04216370
## 10  3.75625 0.18250 0.04216370
## 11  4.17250 0.18125 0.04458528
## 12  4.58875 0.18250 0.04456581
## 13  5.00500 0.18375 0.04678927
## 14  5.42125 0.18500 0.04923018
## 15  5.83750 0.18625 0.04910660
## 16  6.25375 0.18750 0.04787136
## 17  6.67000 0.18875 0.04730589
## 18  7.08625 0.19125 0.04641674
## 19  7.50250 0.18875 0.04581439
## 20  7.91875 0.18875 0.04581439
## 21  8.33500 0.19000 0.04706674
## 22  8.75125 0.19250 0.04571956
```

```
## 23  9.16750 0.19250 0.04571956
## 24  9.58375 0.19125 0.04332131
## 25 10.00000 0.19000 0.04199868
```

The optimal cost is 0.42625.

```r
#Training error rate
radial_svm <- svm(Purchase ~ ., data = oj_train, kernel = "radial", cost = radial_tune$best.parameters$c
svm_rad <- predict(radial_svm, oj_train)
table(svm_rad, oj_train$Purchase)
```

```
##
## svm_rad  CH  MM
##      CH 444  70
##      MM  45 241
```

```r
#Test error rate
svm_rad_test <- predict(radial_svm, oj_test)
table(svm_rad_test, oj_test$Purchase)
```

```
##
## svm_rad_test  CH  MM
##           CH 147  34
##           MM  17  72
```

The training error rate is 14.5% and the test error rate is 18.89%.

(g)

Polynomial Kernel

```r
oj_poly <- svm(Purchase ~ ., data = oj_train, kernel = "polynomial", degree = 2)
summary(oj_poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj_train, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  441
##
##  ( 224 217 )
##
##
## Number of Classes:  2
```

```
##
## Levels:
##  CH MM
```

The SVM with polynomial kernel creates 441 support vectors out of the 800 training observations. Out of the 441 support vectors, 224 belong to level CH and 217 to level MM.

```r
#Training error rate

poly_train <- predict(oj_poly, oj_train)
table(poly_train, oj_train$Purchase)
```

```
##
## poly_train  CH  MM
##         CH 452 107
##         MM  37 204
```

```r
#Test error rate

poly_test <- predict(oj_poly, oj_test)
table(poly_test, oj_test$Purchase)
```

```
##
## poly_test  CH  MM
##        CH 149  46
##        MM  15  60
```

```r
(poly_error<- (107+37)/(452+107+37+204))
```

```
## [1] 0.18
```

```r
(ploye_error <- (46+15)/(149+46+15+60))
```

```
## [1] 0.2259259
```

The training error rate is 18% amd the test error rate is 22.59%.

```r
#For optimal cost

poly_tune <- tune(svm, Purchase ~ ., data = oj_train, kernel = "polynomial", degree = 2, ranges =
                  data.frame(cost = seq(0.01, 10, length.out = 25)))
summary(poly_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1775
##
## - Detailed performance results:
##         cost   error dispersion
## 1   0.01000 0.38875 0.06192794
## 2   0.42625 0.20000 0.03996526
## 3   0.84250 0.19875 0.04143687
```

```
## 4   1.25875 0.19500 0.04338138
## 5   1.67500 0.19125 0.04411554
## 6   2.09125 0.19125 0.04372023
## 7   2.50750 0.19125 0.04604120
## 8   2.92375 0.18875 0.04839436
## 9   3.34000 0.19125 0.05138701
## 10  3.75625 0.19375 0.04868051
## 11  4.17250 0.19250 0.04866267
## 12  4.58875 0.19125 0.04641674
## 13  5.00500 0.19125 0.04896498
## 14  5.42125 0.19125 0.05138701
## 15  5.83750 0.18750 0.04526159
## 16  6.25375 0.18750 0.04526159
## 17  6.67000 0.18500 0.04556741
## 18  7.08625 0.18625 0.04767147
## 19  7.50250 0.18500 0.04851976
## 20  7.91875 0.18625 0.04693746
## 21  8.33500 0.18500 0.04816061
## 22  8.75125 0.18250 0.04901814
## 23  9.16750 0.18000 0.05006940
## 24  9.58375 0.18000 0.05006940
## 25 10.00000 0.17750 0.04816061
```

The optimal cost here is 10.

```r
#Training error rate
poly_oj <- svm(Purchase ~ ., data = oj_train, kernel = "polynomial", cost = poly_tune$best.parameters$co
train_poly <- predict(poly_oj, oj_train)
table(train_poly, oj_train$Purchase)
```

```
##
## train_poly  CH  MM
##         CH 449  74
##         MM  40 237
```

```r
(tr_err_poly <- (74+40)/(449+74+40+237))
```

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

```r
#Test error rate
test_poly <- predict(poly_oj, oj_test)
table(test_poly, oj_test$Purchase)
```

```
##
## test_poly  CH  MM
##        CH 147  37
##        MM  17  69
```

```r
(te_err_poly <- (37+17)/(147+37+17+69))
```

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

The training error rate is 14.25% and the test error rate is 20%.

(h)

The support vector classifier or the SVM with linear kernel and with cost 3.75625 gives the best results in terms of the test error rate. It gives the smallest test error rate of 18.15%, among all the approaches.