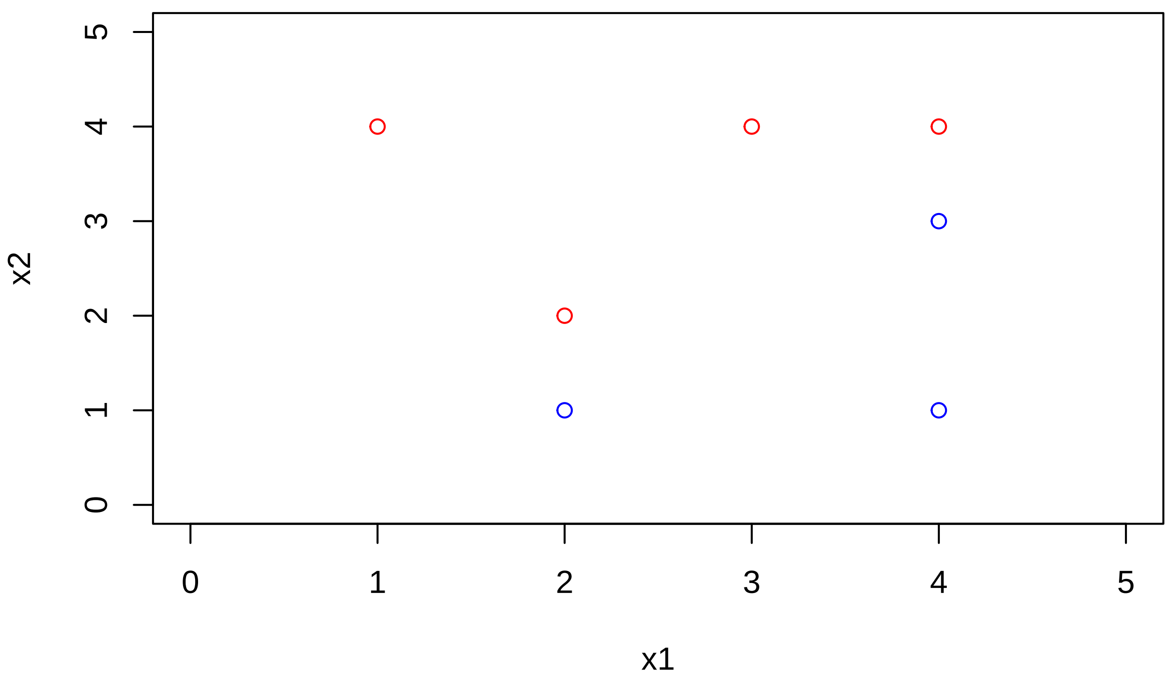
SVM Homework

Pruthvi Bharadwaj

March 10, 2022

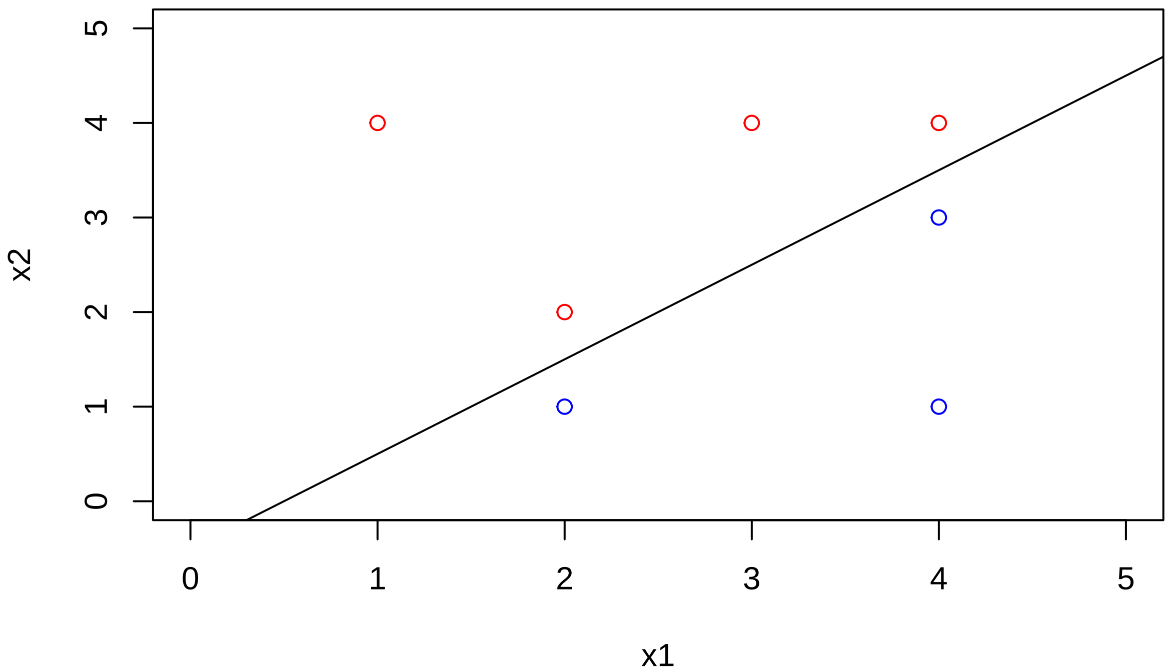
9.3 (a)

```
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)

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



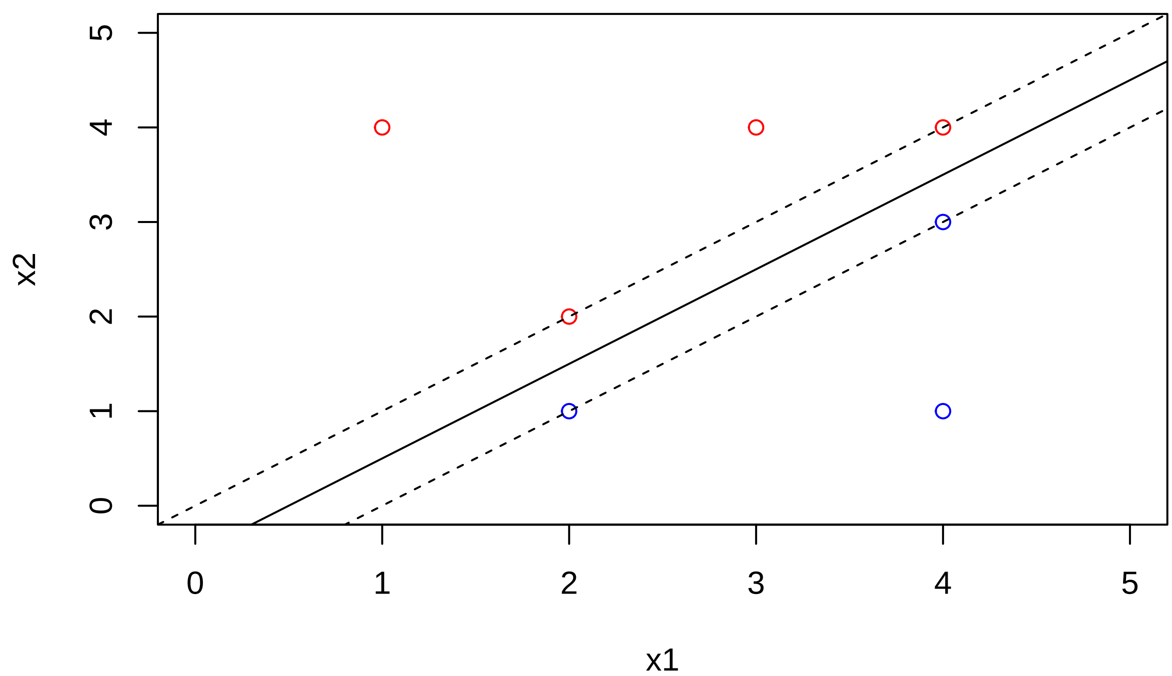
(c)

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)

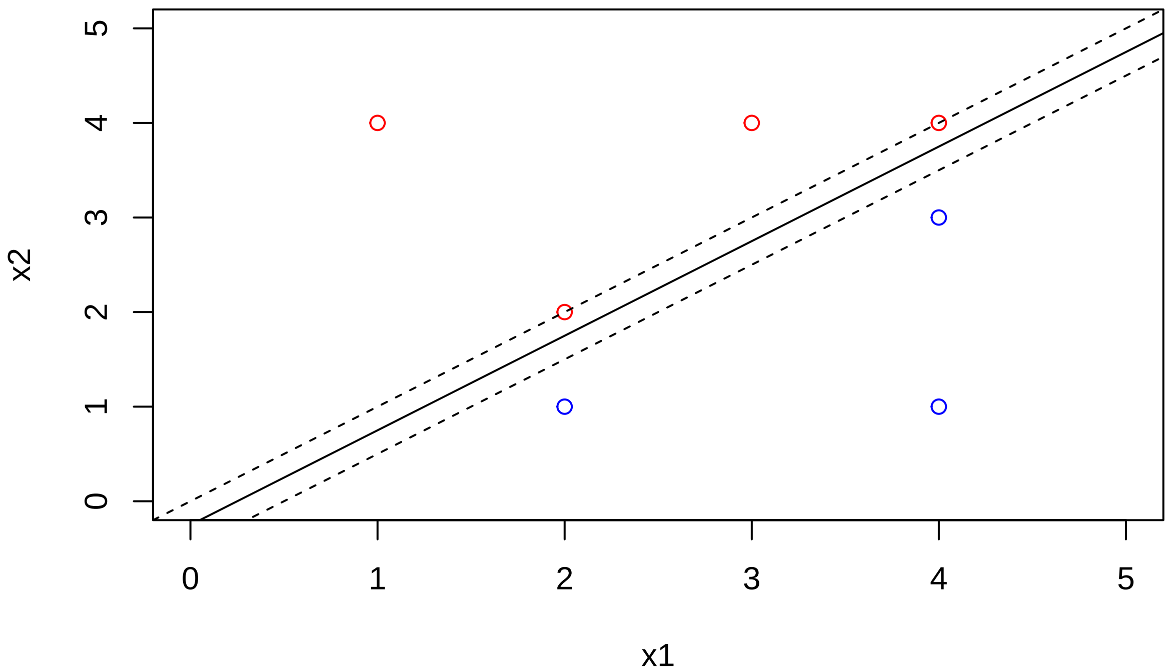
(f)

Since the 7th observation is not a support vector, a slight change in its position will not affect the maximal margin hyperplane.

(g)

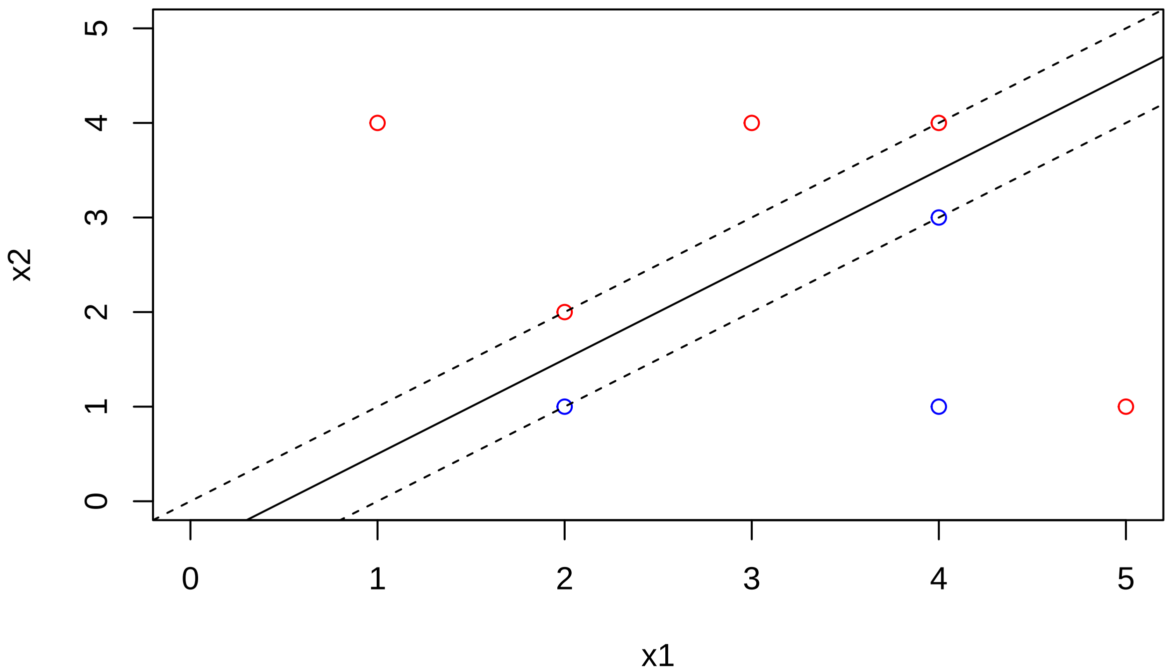
The equation $x2 = -0.25 + x1$ will also separate all the observations but is not an optimal hyperplane because the margin is smaller than the optimal option.

```
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)

```
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(5,1, col = "red")
```

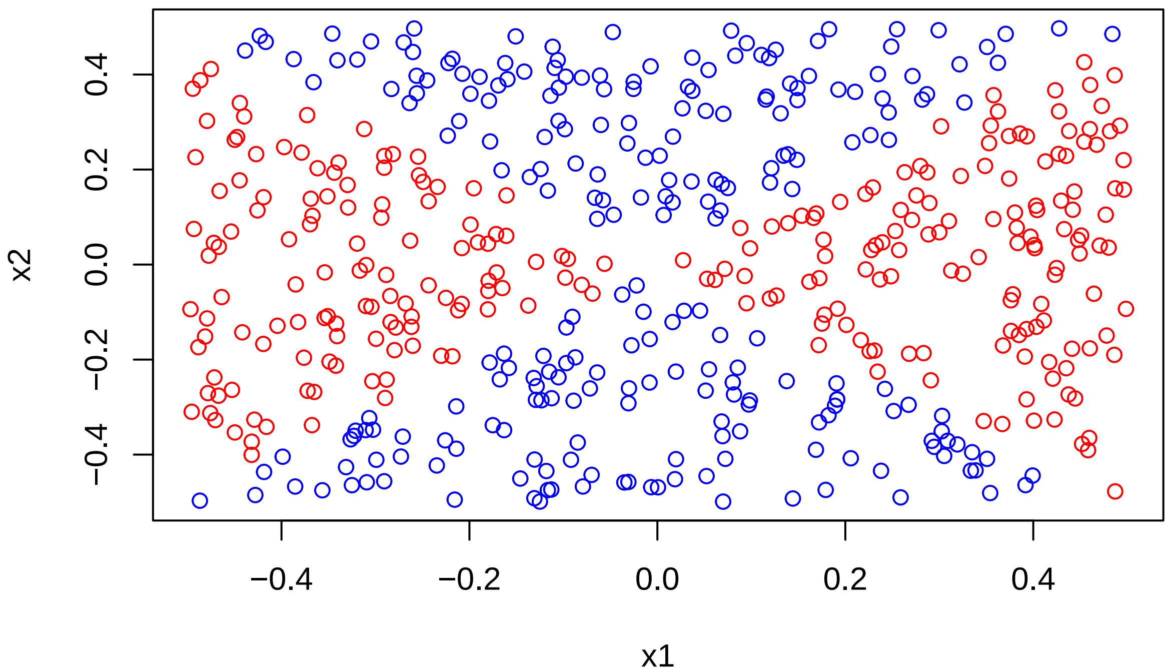


9.5 (a)

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

(b)

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



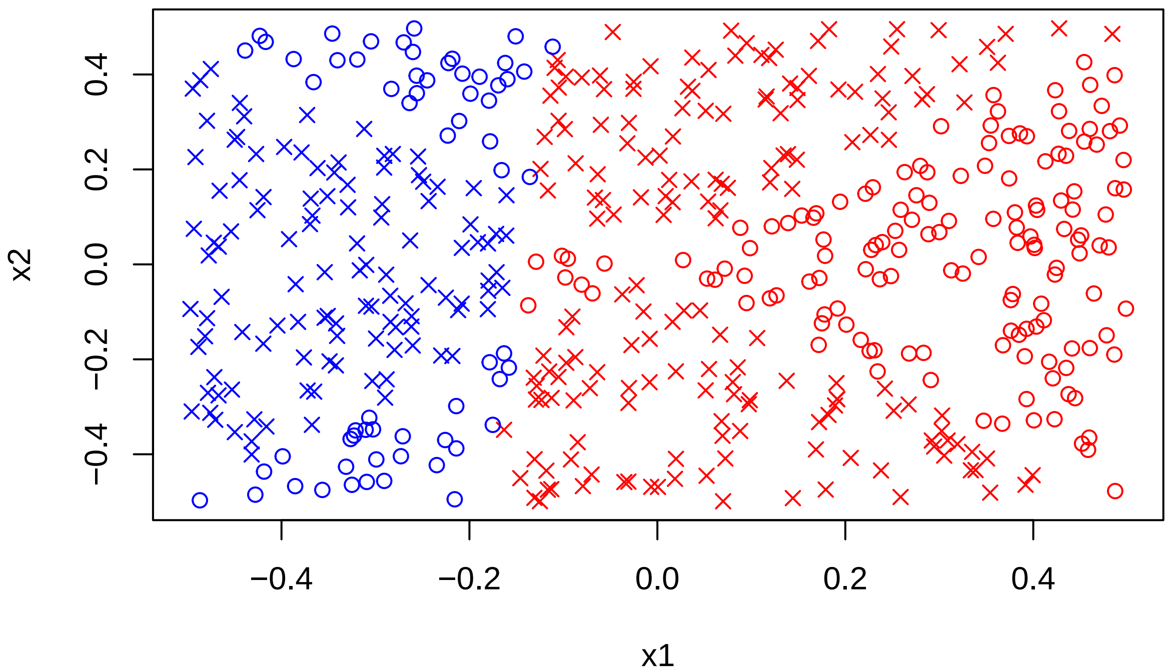
(c)

```
#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
```

```
fit_glm3 <- glm(y ~ x1 + x2 + log(x1) + log(x2), data = df, family = binomial)
```

```
## Warning in log(x1): NaNs produced
```

```
## Warning in log(x2): NaNs produced
```

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

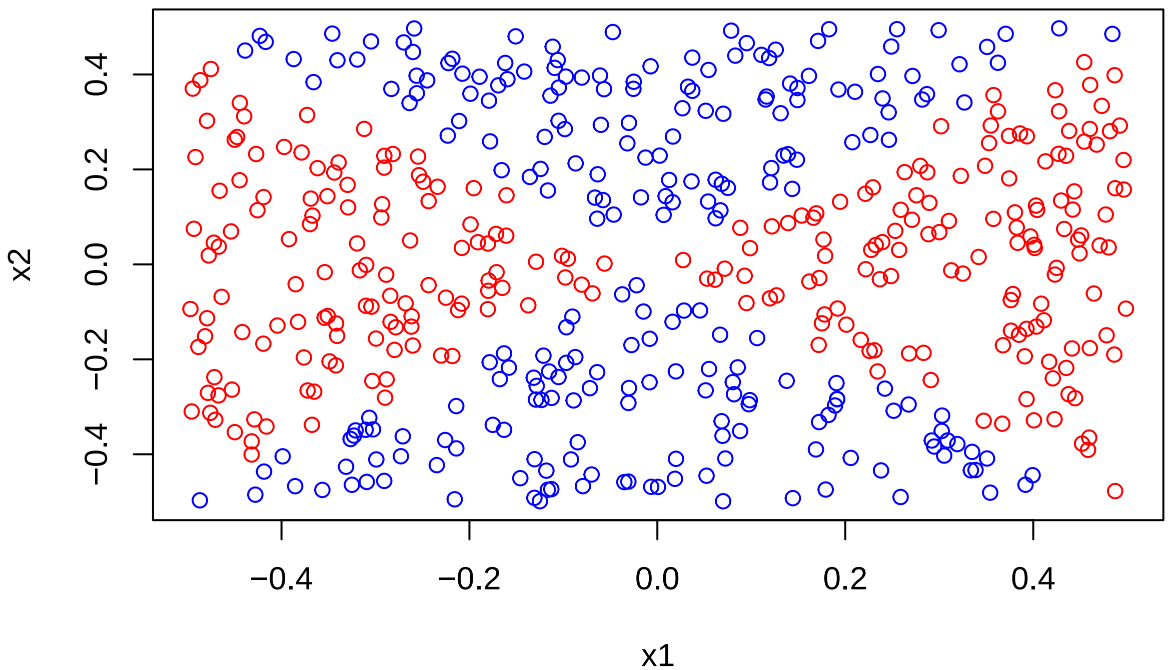
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(fit_glm3)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2 + log(x1) + log(x2), family = binomial,
##     data = df)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -2.314e-04  -2.100e-08   2.100e-08   2.100e-08   2.932e-04
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)     274.39  276445.15   0.001    0.999
## x1             2225.00 2058197.83   0.001    0.999
## x2            -2678.51 1713550.32  -0.002    0.999
## log(x1)         145.08  387545.42   0.000    1.000
## log(x2)         -49.28  246981.59   0.000    1.000
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1.9107e+02  on 138  degrees of freedom
## Residual deviance: 1.7565e-07  on 134  degrees of freedom
##   (361 observations deleted due to missingness)
## AIC: 10
##
## Number of Fisher Scoring iterations: 25
```

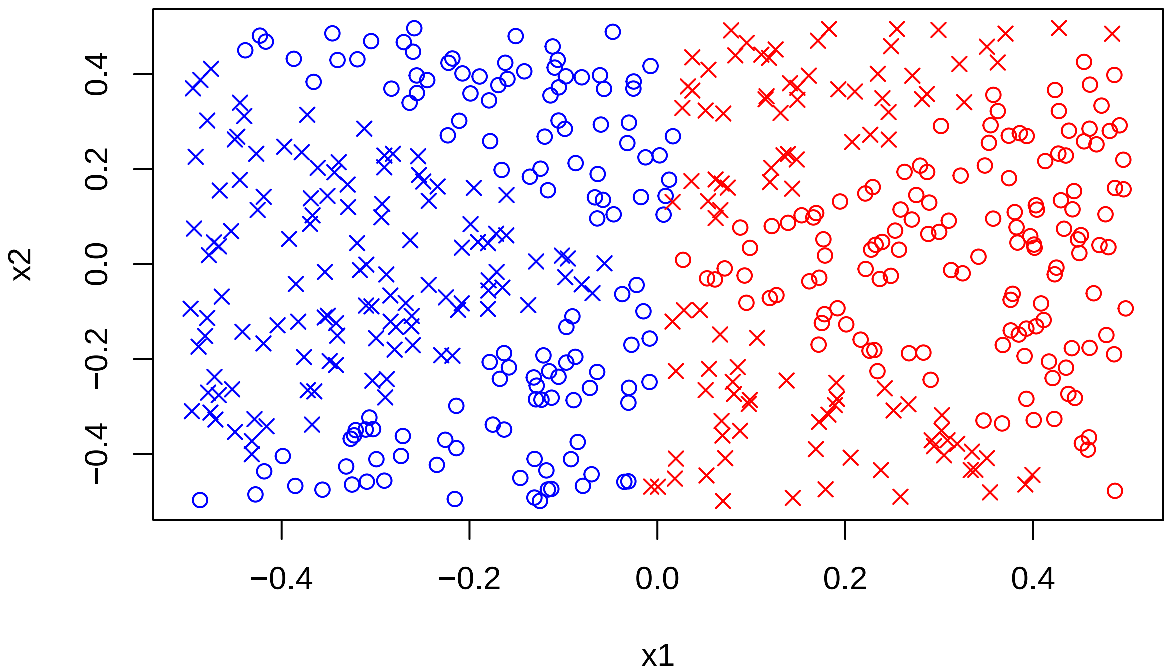
(f)

```
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
```



(g)

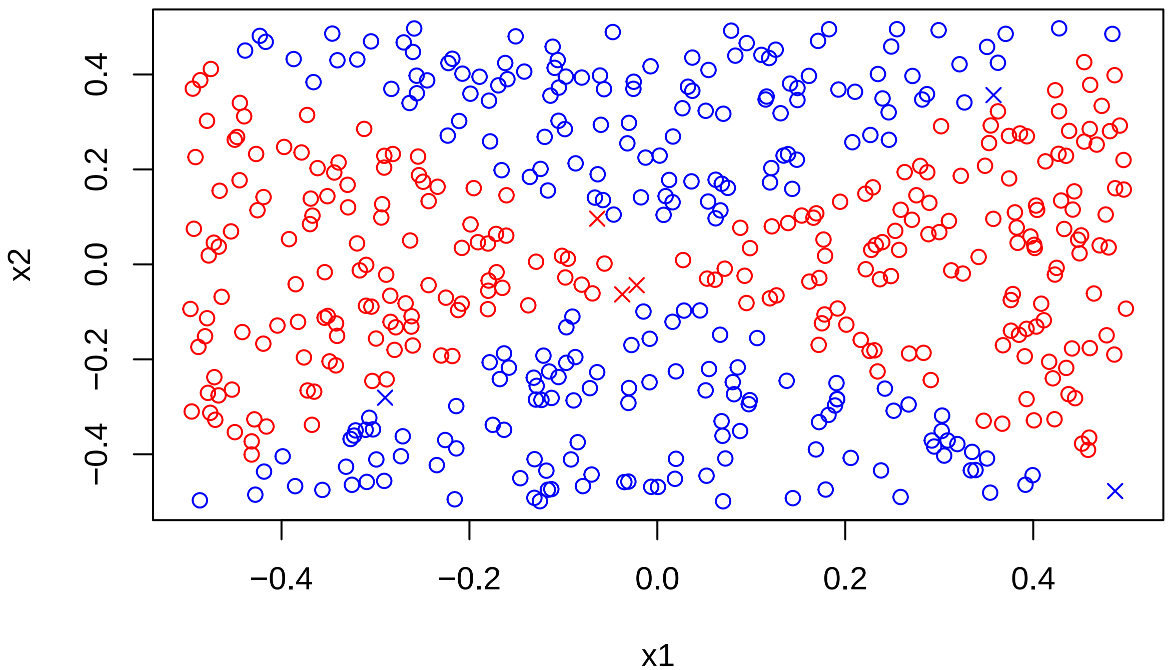
```
#Support Vector Classifier
df$y <- as.factor(df$y)
fit_svc <- svm(y ~ x1 + x2, data = df, kernel = "linear")
pred_svc <- predict(fit_svc, df, type = "response")
plot(x1, x2, col = ifelse(pred_svc != 0, "red", "blue"), pch = ifelse(pred_svc == y, 1,4))
```



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))
```



(i) SVM with a polynomial kernel performs better. But logistic regression with non-linear predictors performs the best

9.7 (a)

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

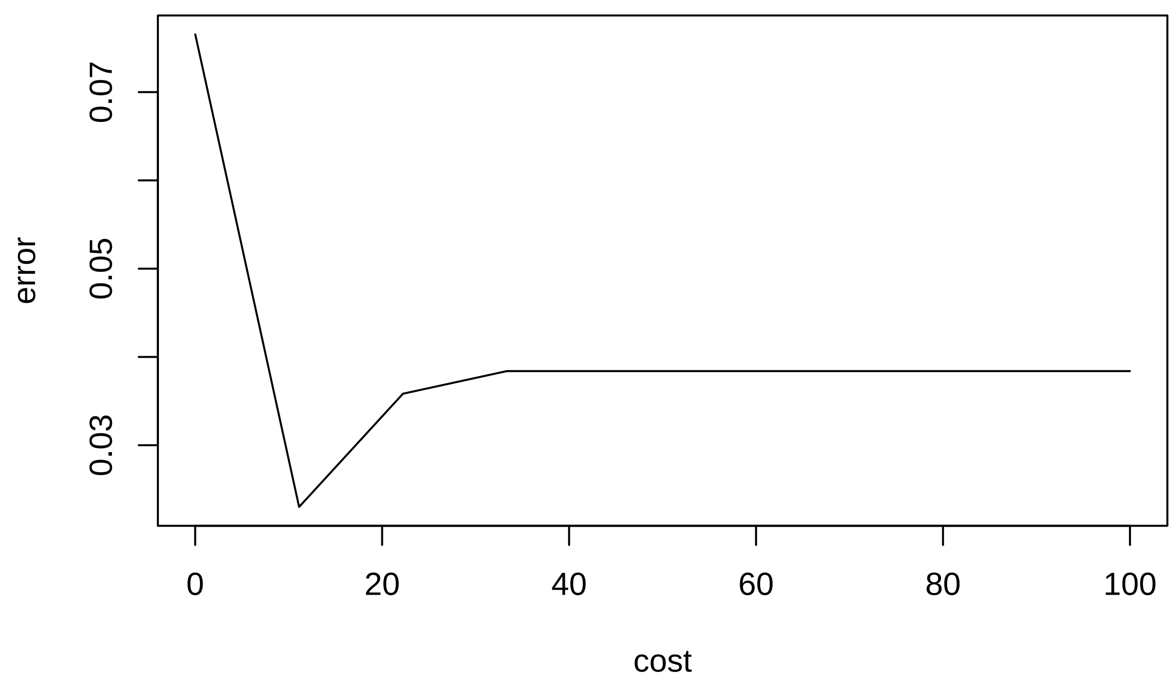
(b)

```
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: 0.02301282
##
## - Detailed performance results:
##      cost      error dispersion
## 1    0.01 0.07653846 0.05100638
## 2   11.12 0.02301282 0.01891104
## 3   22.23 0.03583333 0.03245677
## 4   33.34 0.03839744 0.03872235
## 5   44.45 0.03839744 0.03872235
## 6   55.56 0.03839744 0.03872235
## 7   66.67 0.03839744 0.03872235
## 8   77.78 0.03839744 0.03872235
## 9   88.89 0.03839744 0.03872235
## 10 100.00 0.03839744 0.03872235
```

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



cost=11.12 has the best performance.

(c)

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

```r
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
##  75.0025      1
##
## - best performance: 0.02814103
##
## - Detailed performance results:
##        cost degree      error dispersion
## 1    0.0100   1.00 0.54852564 0.01899199
## 2   25.0075   1.00 0.05102564 0.03419231
## 3   50.0050   1.00 0.03326923 0.02434857
## 4   75.0025   1.00 0.02814103 0.01893035
## 5  100.0000   1.00 0.02814103 0.01893035
## 6    0.0100  25.75 0.54852564 0.01899199
## 7   25.0075  25.75 0.54852564 0.01899199
## 8   50.0050  25.75 0.54852564 0.01899199
## 9   75.0025  25.75 0.54852564 0.01899199
## 10 100.0000  25.75 0.54852564 0.01899199
## 11   0.0100  50.50 0.54852564 0.01899199
## 12  25.0075  50.50 0.54852564 0.01899199
## 13  50.0050  50.50 0.54852564 0.01899199
## 14  75.0025  50.50 0.54852564 0.01899199
## 15 100.0000  50.50 0.54852564 0.01899199
## 16   0.0100  75.25 0.54852564 0.01899199
## 17  25.0075  75.25 0.54852564 0.01899199
## 18  50.0050  75.25 0.54852564 0.01899199
## 19  75.0025  75.25 0.54852564 0.01899199
## 20 100.0000  75.25 0.54852564 0.01899199
## 21   0.0100 100.00 0.54852564 0.01899199
## 22  25.0075 100.00 0.54852564 0.01899199
## 23  50.0050 100.00 0.54852564 0.01899199
## 24  75.0025 100.00 0.54852564 0.01899199
## 25 100.0000 100.00 0.54852564 0.01899199
```

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)
```

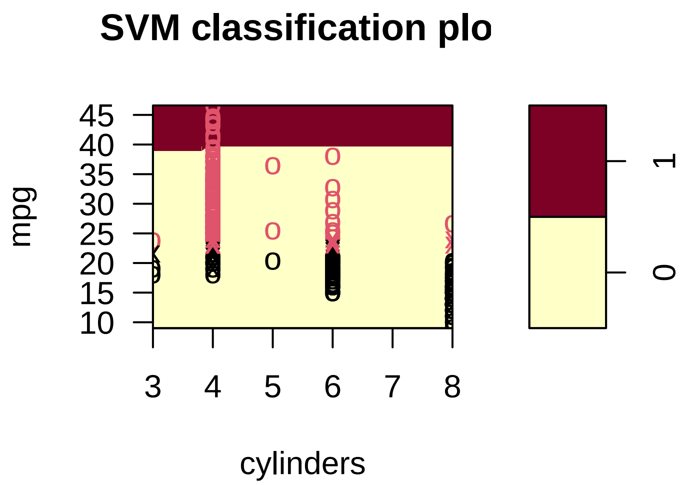
```
##
## Parameter tuning of 'svm':
```

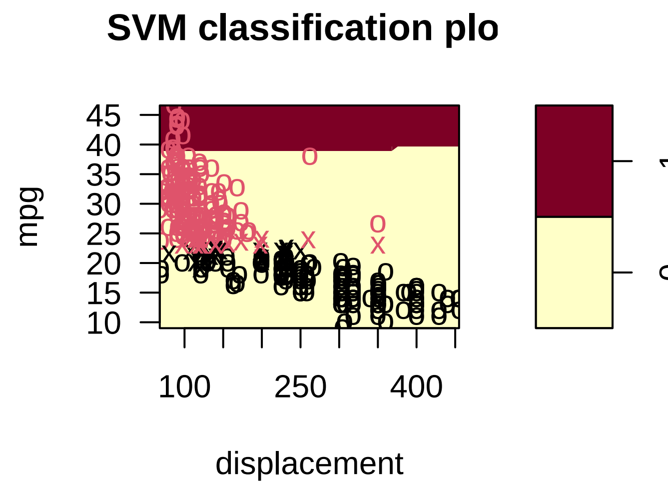
```
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##     cost gamma
##  25.0075   0.1
##
## - best performance: 0.02294872
##
## - Detailed performance results:
##         cost   gamma      error dispersion
## 1     0.0100   0.100 0.20621795 0.08131124
## 2    25.0075   0.100 0.02294872 0.02807826
## 3    50.0050   0.100 0.02807692 0.03059334
## 4    75.0025   0.100 0.03064103 0.02901234
## 5   100.0000   0.100 0.03064103 0.02901234
## 6     0.0100  25.075 0.56858974 0.04679438
## 7    25.0075  25.075 0.54557692 0.06126516
## 8    50.0050  25.075 0.54557692 0.06126516
## 9    75.0025  25.075 0.54557692 0.06126516
## 10  100.0000  25.075 0.54557692 0.06126516
## 11    0.0100  50.050 0.56858974 0.04679438
## 12   25.0075  50.050 0.56089744 0.05867321
## 13   50.0050  50.050 0.56089744 0.05867321
## 14   75.0025  50.050 0.56089744 0.05867321
## 15  100.0000  50.050 0.56089744 0.05867321
## 16    0.0100  75.025 0.56858974 0.04679438
## 17   25.0075  75.025 0.56602564 0.05072705
## 18   50.0050  75.025 0.56602564 0.05072705
## 19   75.0025  75.025 0.56602564 0.05072705
## 20  100.0000  75.025 0.56602564 0.05072705
## 21    0.0100 100.000 0.56858974 0.04679438
## 22   25.0075 100.000 0.56858974 0.04679438
## 23   50.0050 100.000 0.56858974 0.04679438
## 24   75.0025 100.000 0.56858974 0.04679438
## 25  100.0000 100.000 0.56858974 0.04679438
```

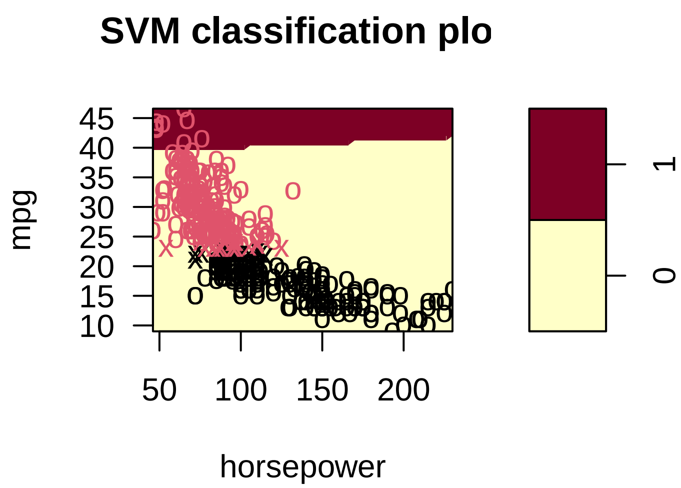
Cost of 25 with gamma 0.1 seems to perform the best

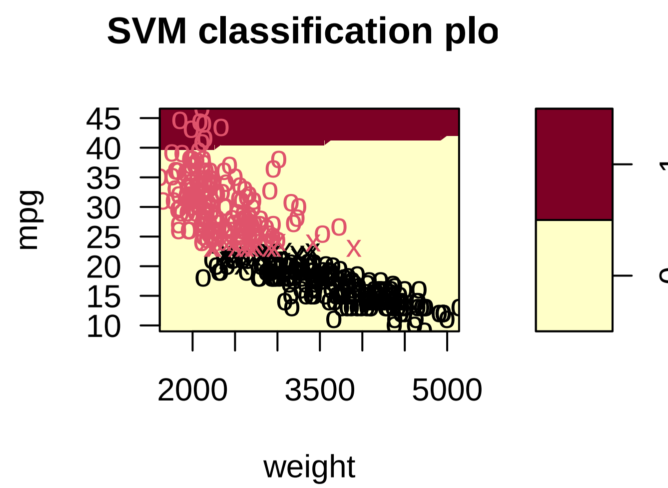
(d)

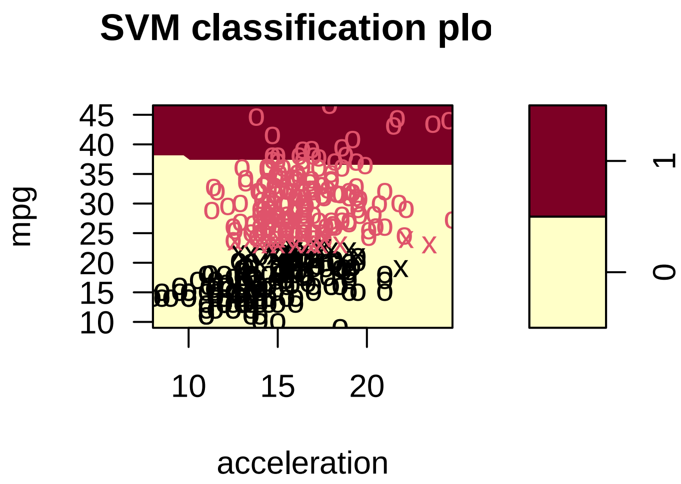
```r
linear <- svm(Y ~ ., data = Auto, kernel = "linear", cost = 11.12)
polynomial <- svm(Y ~ ., data = Auto, kernel = "polynomial", cost = 100, degree = 1)
radial <- svm(Y ~ ., data = Auto, kernel = "radial", cost = 25.0075, gamma = 0.1)


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)
```

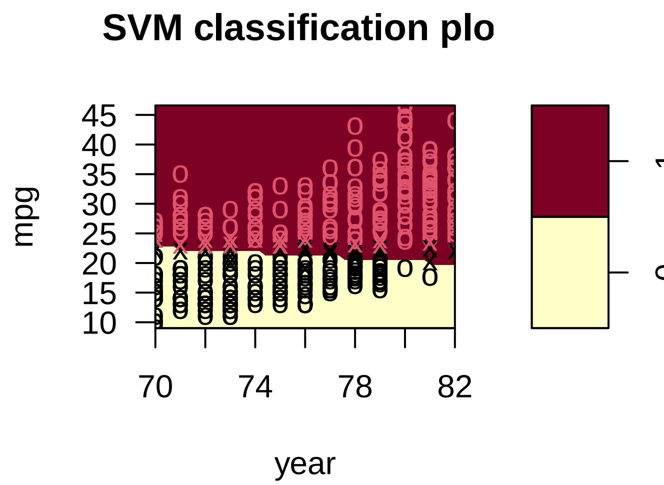

SVM classification plo
mpg
45
40
35
30
25
20
15
10
3
4
5
6
7
8
cylinders
1
0

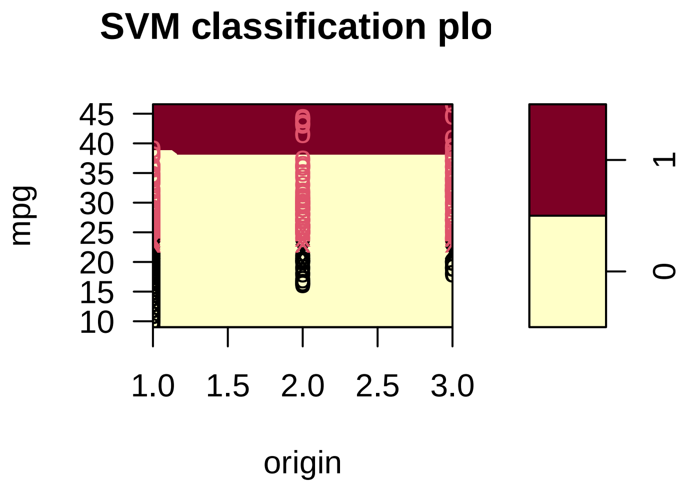

SVM classification plo
mpg
45
40
35
30
25
20
15
10
100
250
400
displacement


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

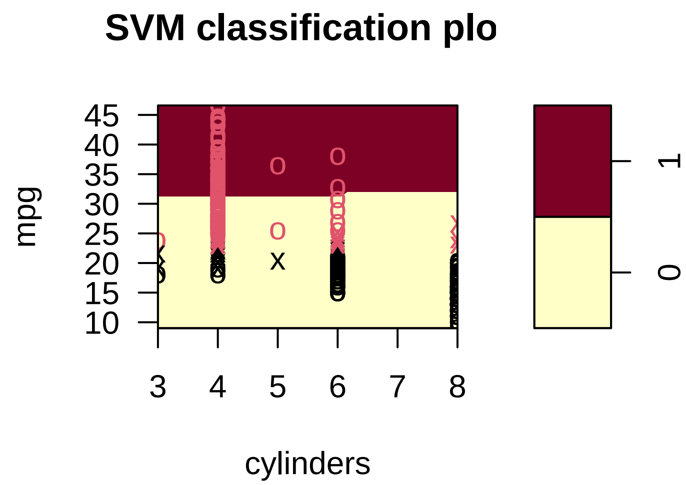


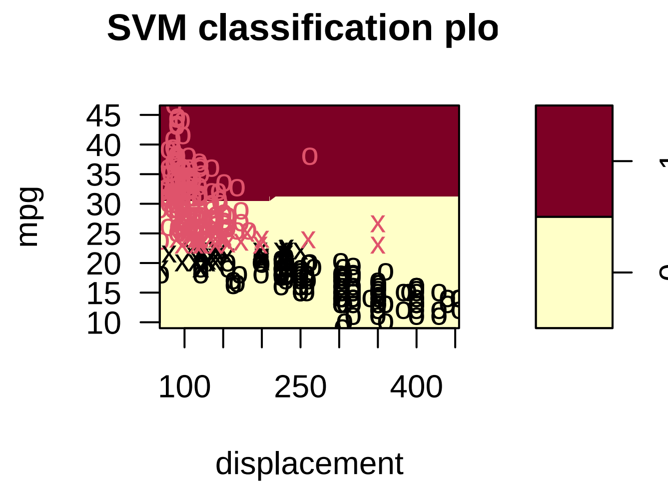


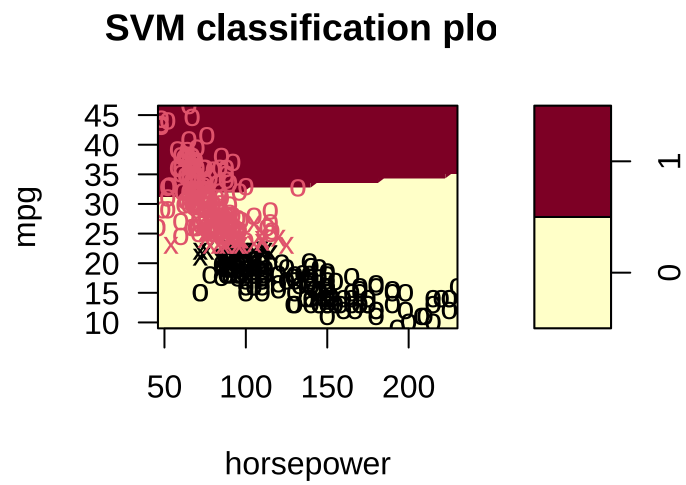


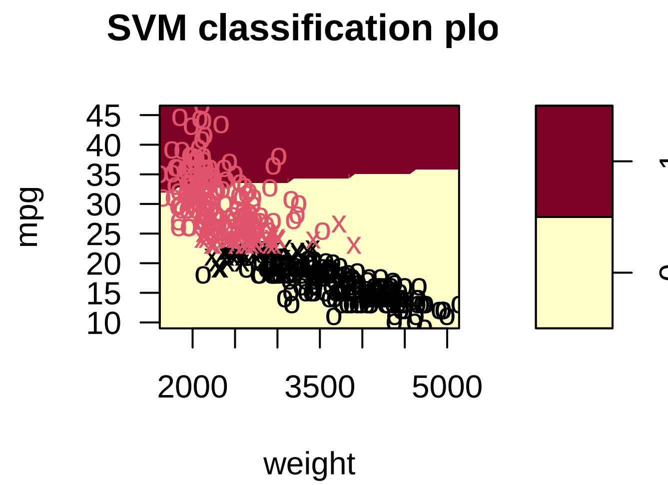
The above are the SVM classification plots for linear kernel.

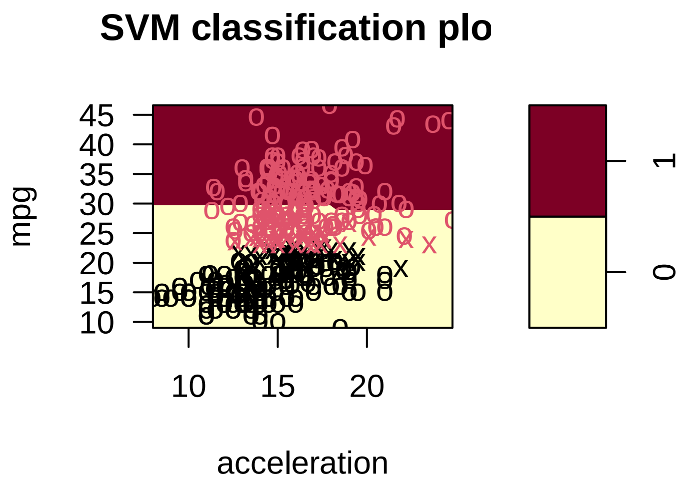
```
pair_plot(polynomial)
```

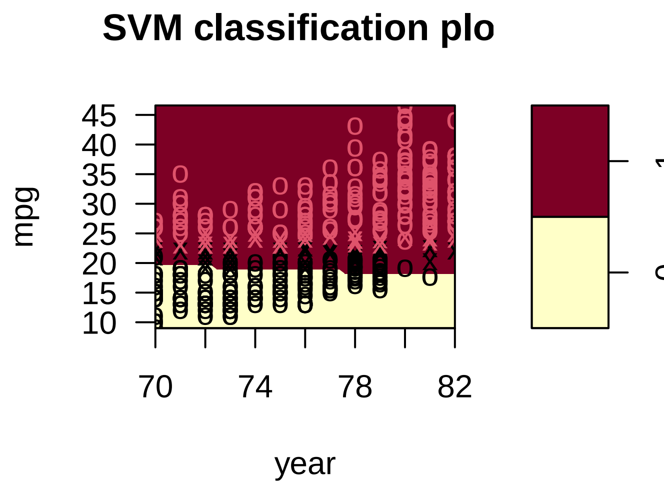

SVM classification plo
mpg
45
40
35
30
25
20
15
10
3
4
5
6
7
8
cylinders
1
0

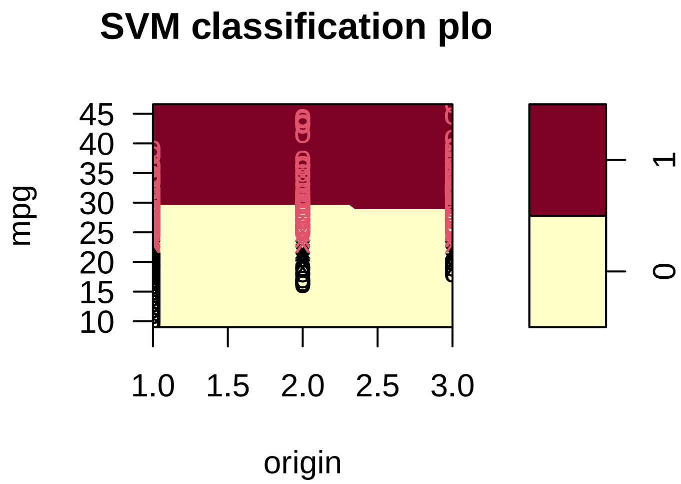

SVM classification plo
mpg
45
40
35
30
25
20
15
10
100
250
400
displacement


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

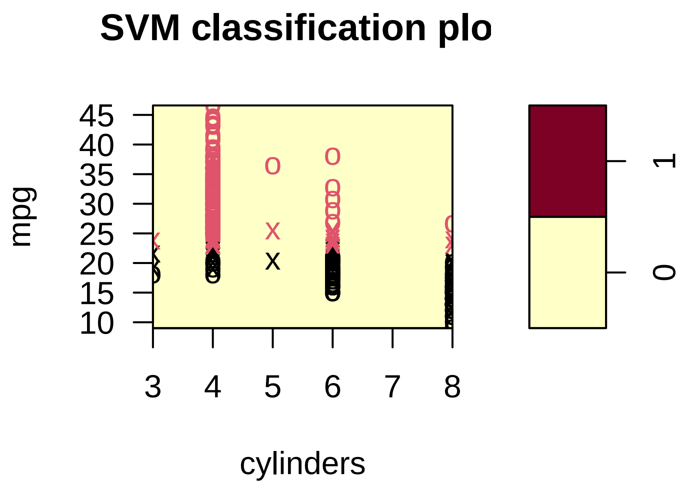


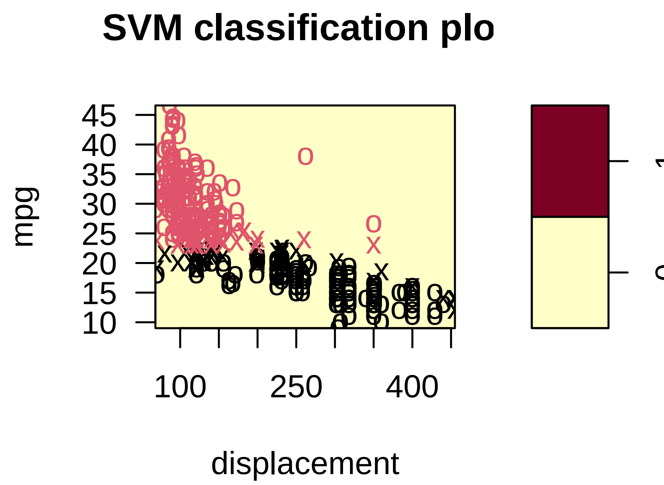


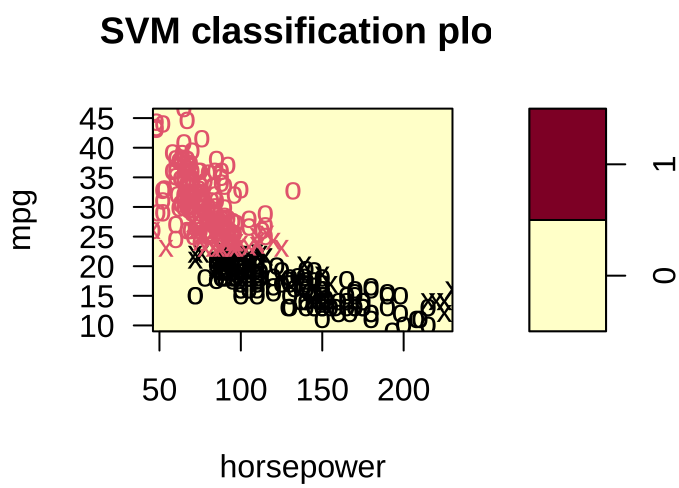


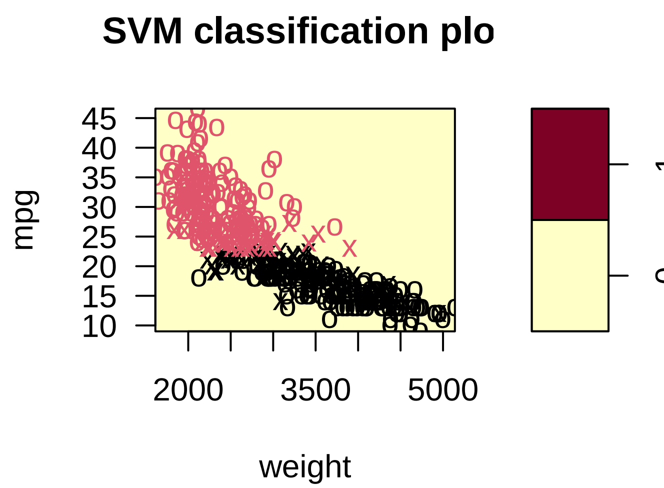
The above plots are the SVM classification plots for polynomial kernel

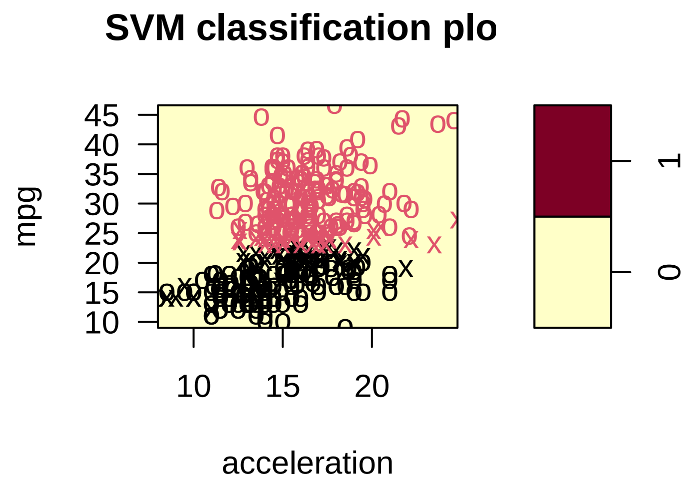
```
pair_plot(radial)
```

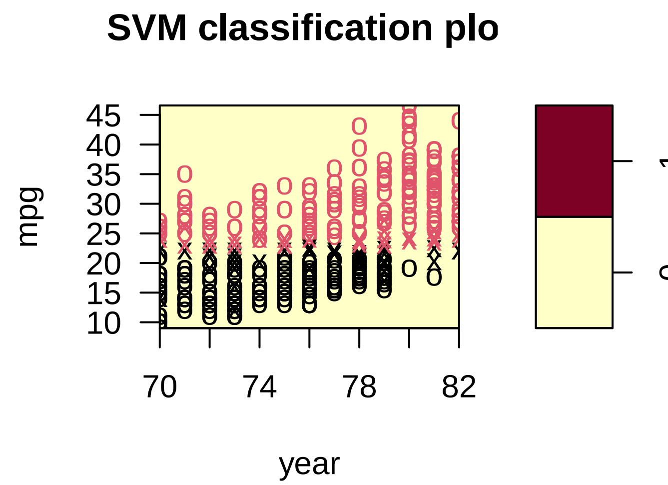

SVM classification plo
mpg
45
40
35
30
25
20
15
10
3
4
5
6
7
8
cylinders
1
0

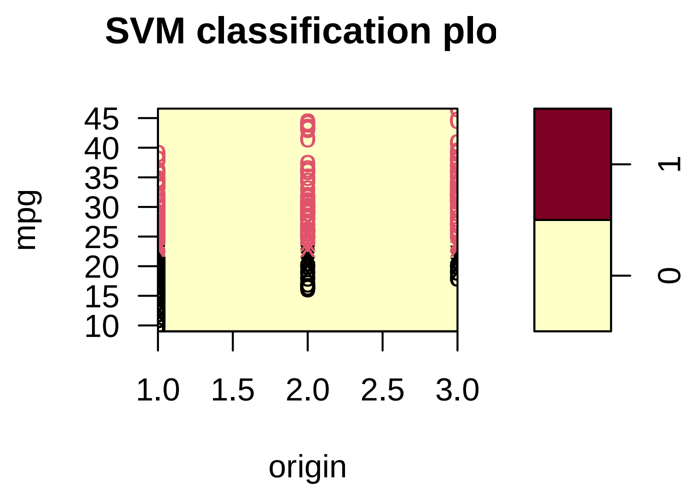

SVM classification plo
mpg
45
40
35
30
25
20
15
10
100
250
400
displacement


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







The above plots are the SVM classification plots for radial kernel

9.8 (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
##
## Number of Support Vectors:  426
##
##  ( 213 213 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The SVC 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 455  77
##         MM  50 218
```

```r
#Test error rate

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

```
##
## pred_test  CH  MM
##        CH 131  39
##        MM  17  83
```

```r
(tr_error<- (50+77)/(455+77+50+218))
```

```
## [1] 0.15875
```

```r
(te_error <- (39+17)/(131+39+17+83))
```

```
## [1] 0.2074074
```

The training error rate is 15.87% and the test error rate is 20.74%

(d)

```
#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
##   3.34
##
## - best performance: 0.1575
##
## - Detailed performance results:
##        cost   error dispersion
## 1   0.01000 0.16625 0.03175973
## 2   0.42625 0.16125 0.03557562
## 3   0.84250 0.16125 0.03408018
## 4   1.25875 0.16125 0.03197764
## 5   1.67500 0.16125 0.03408018
## 6   2.09125 0.16000 0.03670453
## 7   2.50750 0.15875 0.03586723
## 8   2.92375 0.15875 0.03729108
## 9   3.34000 0.15750 0.03593976
## 10  3.75625 0.15750 0.03593976
## 11  4.17250 0.15750 0.03593976
## 12  4.58875 0.15750 0.03593976
## 13  5.00500 0.15875 0.03586723
## 14  5.42125 0.15875 0.03586723
## 15  5.83750 0.15875 0.03586723
## 16  6.25375 0.15875 0.03586723
## 17  6.67000 0.15875 0.03586723
## 18  7.08625 0.15875 0.03586723
## 19  7.50250 0.15875 0.03586723
## 20  7.91875 0.15875 0.03586723
## 21  8.33500 0.15875 0.03586723
## 22  8.75125 0.15875 0.03586723
## 23  9.16750 0.15875 0.03586723
## 24  9.58375 0.15875 0.03586723
## 25 10.00000 0.15875 0.03586723
```

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 456  75
##         MM  49 220
```

```r
(tr_err <- (49+75)/(456+75+49+220))
```

```
## [1] 0.155
```

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

```
##
## svm_test  CH  MM
##        CH 131  35
##        MM  17  87
```

```r
(te_err <- (17+35)/(131+35+17+87))
```

```
## [1] 0.1925926
```

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
##
## Number of Support Vectors:  365
##
##  ( 187 178 )
##
##
## 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.

```
#Training error rate

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

```
##
## radial_train  CH  MM
##           CH 466  73
##           MM  39 222
```

```
(tr_err <- (39+73)/(466+73+39+222))
```

```
## [1] 0.14
```

```
#Test error rate

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

```
##
## radial_test  CH  MM
##          CH 132  37
##          MM  16  85
```

```
(te_err <- (37+16)/(132+37+16+85))
```

```
## [1] 0.1962963
```

The training error rate is 14% and the test error rate is 19.63%.

```
#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: 0.155
##
```

```
## - Detailed performance results:
##        cost   error dispersion
## 1   0.01000 0.36875 0.04218428
## 2   0.42625 0.16125 0.02461509
## 3   0.84250 0.15500 0.02648375
## 4   1.25875 0.15500 0.03016160
## 5   1.67500 0.15500 0.03238227
## 6   2.09125 0.15625 0.03294039
## 7   2.50750 0.15750 0.03593976
## 8   2.92375 0.16125 0.03557562
## 9   3.34000 0.16000 0.03574602
## 10  3.75625 0.16125 0.03408018
## 11  4.17250 0.16250 0.03535534
## 12  4.58875 0.16625 0.03488573
## 13  5.00500 0.16750 0.03343734
## 14  5.42125 0.16625 0.03682259
## 15  5.83750 0.16625 0.03682259
## 16  6.25375 0.16625 0.03488573
## 17  6.67000 0.16625 0.03488573
## 18  7.08625 0.16625 0.03488573
## 19  7.50250 0.16750 0.03593976
## 20  7.91875 0.16875 0.03448530
## 21  8.33500 0.16875 0.03448530
## 22  8.75125 0.16875 0.03448530
## 23  9.16750 0.17000 0.03343734
## 24  9.58375 0.17000 0.03343734
## 25 10.00000 0.17125 0.03283481
```

The optimal cost ranges from 0.8425 to 1.675

```
#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 469  73
##      MM  36 222
```

```
(tr_err <- (36+73)/(469+73+36+222))
```

```
## [1] 0.13625
```

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

```
##
## svm_rad_test  CH  MM
##           CH 132  36
##           MM  16  86
```

```r
(te_err <- (36+16)/(132+36+16+86))
```

```
## [1] 0.1925926
```

The training error rate is 13.625% and the test error rate is 19.26%

(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
##      coef.0:  0
##
## Number of Support Vectors:  431
##
##  ( 219 212 )
##
##
## 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 468 105
##         MM  37 190
```

```r
#Test error rate

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

```
##
## poly_test  CH  MM
##        CH 136  45
##        MM  12  77
```

```
(poly_error<- (105+37)/(468+105+37+190))
```

```
## [1] 0.1775
```

```
(ploye_error <- (45+12)/(136+45+12+77))
```

```
## [1] 0.2111111
```

The training error rate is 17.75% amd the test error rate is 21.11%

```
#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
##  8.335
##
## - best performance: 0.15875
##
## - Detailed performance results:
##        cost   error dispersion
## 1   0.01000 0.36875 0.04903584
## 2   0.42625 0.20375 0.05684103
## 3   0.84250 0.19375 0.05376453
## 4   1.25875 0.18625 0.04875178
## 5   1.67500 0.18500 0.04556741
## 6   2.09125 0.18375 0.04678927
## 7   2.50750 0.18125 0.04458528
## 8   2.92375 0.18000 0.04338138
## 9   3.34000 0.17375 0.04427267
## 10  3.75625 0.17500 0.04039733
## 11  4.17250 0.17375 0.04226652
## 12  4.58875 0.16875 0.04093101
## 13  5.00500 0.17000 0.04133199
## 14  5.42125 0.16375 0.04619178
## 15  5.83750 0.16375 0.04730589
## 16  6.25375 0.16250 0.04602234
## 17  6.67000 0.16125 0.04505013
## 18  7.08625 0.16250 0.04370037
```

```
## 19  7.50250 0.16250 0.04370037
## 20  7.91875 0.16000 0.04362084
## 21  8.33500 0.15875 0.04489571
## 22  8.75125 0.15875 0.04489571
## 23  9.16750 0.15875 0.04489571
## 24  9.58375 0.16250 0.04714045
## 25 10.00000 0.16375 0.04730589
```

The optimal cost here is between 8.335 to 9.1675

```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 471  80
##         MM  34 215
```

```r
(tr_err_poly <- (80+34)/(471+80+34+215))
```

```
## [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 132  39
##        MM  16  83
```

```r
(te_err_poly <- (16+39)/(132+39+16+83))
```

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

The training error rate is 14.25% and the test error rate is 20.37%

(h)

SVM with linear kernel and with cost 3.75625 gives the best results in terms of the test error rate which is 18.15%