Assignment 8

Tina Roha

5/11/2021

Support Vector Machines

Chapter 09 (page 368): Questions 5, 7 and 8

Question-5

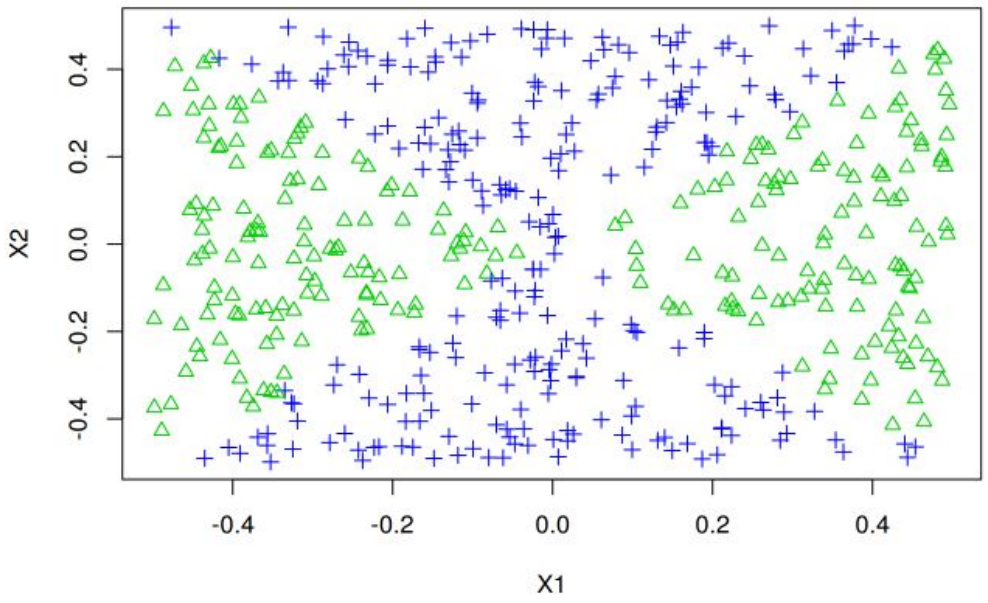
We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

a. Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with a quadratic decision boundary between them. For instance, you can do this as follows: > x1=runif (500) -0.5 > x2=runif (500) -0.5 > y=1*(x1^{2-x2}2 > 0)

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

b. Plot the observations, colored according to their class labels. Your plot should display X1 on the x-axis, and X2 on the yaxis.

```
plot(x1, x2, xlab = "X1", ylab = "X2", col = (4 - y), pch = (3 - y))
```



c. Fit a logistic regression model to the data, using X1 and X2 as predictors.

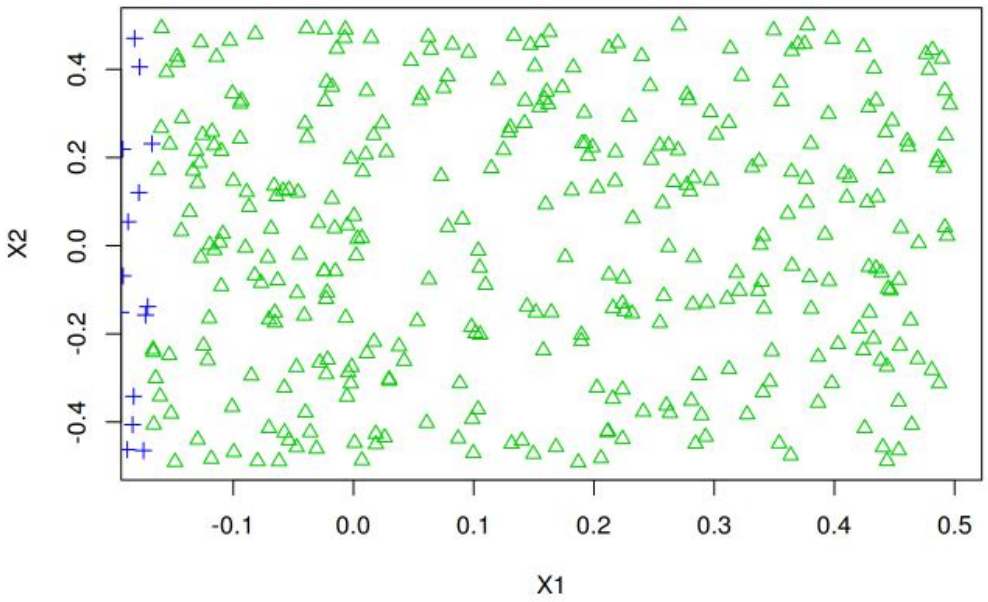
```
logit.fit <- glm(y ~ x1 + x2, family = "binomial")
summary(logit.fit)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = "binomial")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.179  -1.139  -1.112   1.206   1.257
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.087260   0.089579  -0.974    0.330
## x1           0.196199   0.316864   0.619    0.536
## x2          -0.002854   0.305712  -0.009    0.993
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.18  on 499  degrees of freedom
## Residual deviance: 691.79  on 497  degrees of freedom
## AIC: 697.79
##
## Number of Fisher Scoring iterations: 3
```

From the results it can be concluded that none of the variables possess statistical significance.

d. Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be linear.

```
data <- data.frame(x1 = x1, x2 = x2, y = y)
probs <- predict(logit.fit, data, type = "response")
preds <- rep(0, 500)
preds[probs > 0.47] <- 1
plot(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1), xlab = "X1", ylab = "X2")
points(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0))
```



From the results, it is evident that the decision boundary is linear.

e. Now fit a logistic regression model to the data using non-linear functions of X1 and X2 as predictors (e.g. X2 1 , X1×X2, log(X2), and so forth).

```
logitnl.fit <- glm(y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family = "binomial")
```

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

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

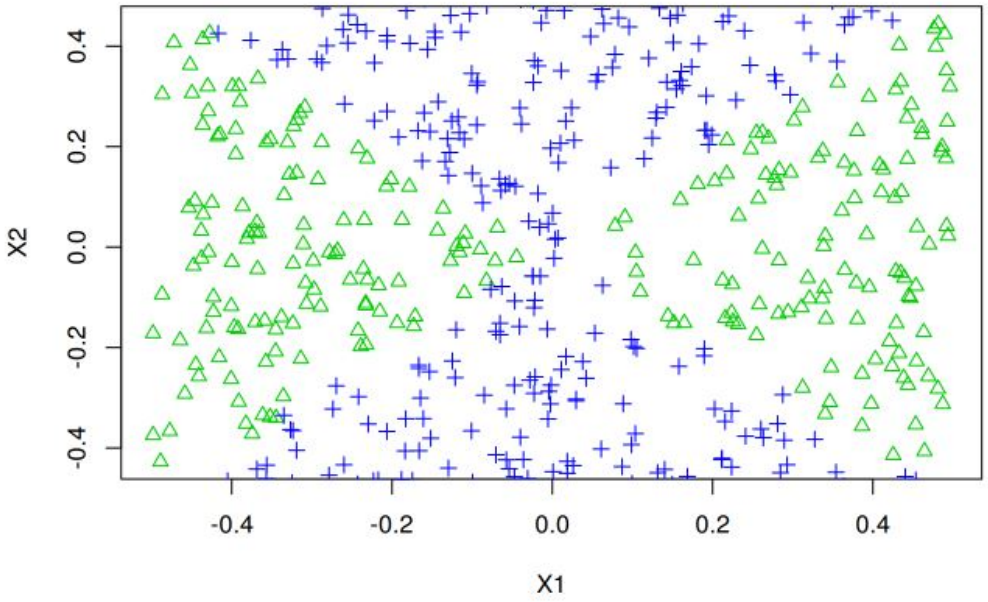
```
summary(logitnl.fit)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family = "binomial")
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -8.240e-04  -2.000e-08  -2.000e-08   2.000e-08   1.163e-03
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -102.2     4302.0  -0.024    0.981
## poly(x1, 2)1   2715.3   141109.5   0.019    0.985
## poly(x1, 2)2  27218.5   842987.2   0.032    0.974
## poly(x2, 2)1   -279.7    97160.4  -0.003    0.998
## poly(x2, 2)2 -28693.0   875451.3  -0.033    0.974
## I(x1 * x2)     -206.4    41802.8  -0.005    0.996
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9218e+02  on 499  degrees of freedom
## Residual deviance: 3.5810e-06  on 494  degrees of freedom
## AIC: 12
##
## Number of Fisher Scoring iterations: 25
```

From the results, it can be concluded that none of the variables possess statistical significance.

f. Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be obviously non-linear. If it is not, then repeat (a)-(e) until you come up with an example in which the predicted class labels are obviously non-linear.

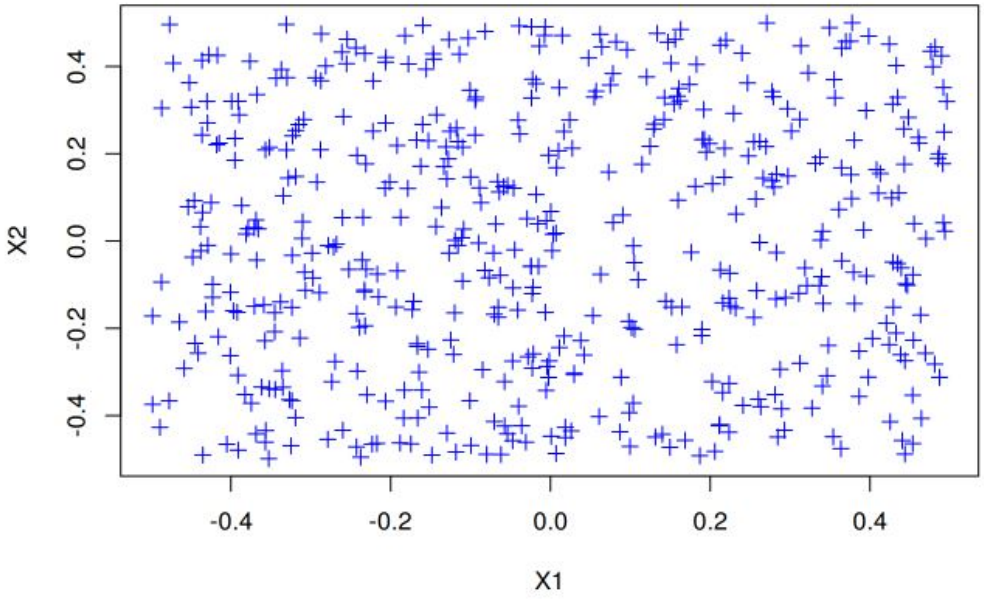
```r
probs <- predict(logitnl.fit, data, type = "response")
preds <- rep(0, 500)
preds[probs > 0.47] <- 1
plot(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1), xlab = "X1", ylab = "X2")
points(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0))
```



Here it is evident that the decision boundary is obviously non-linear.

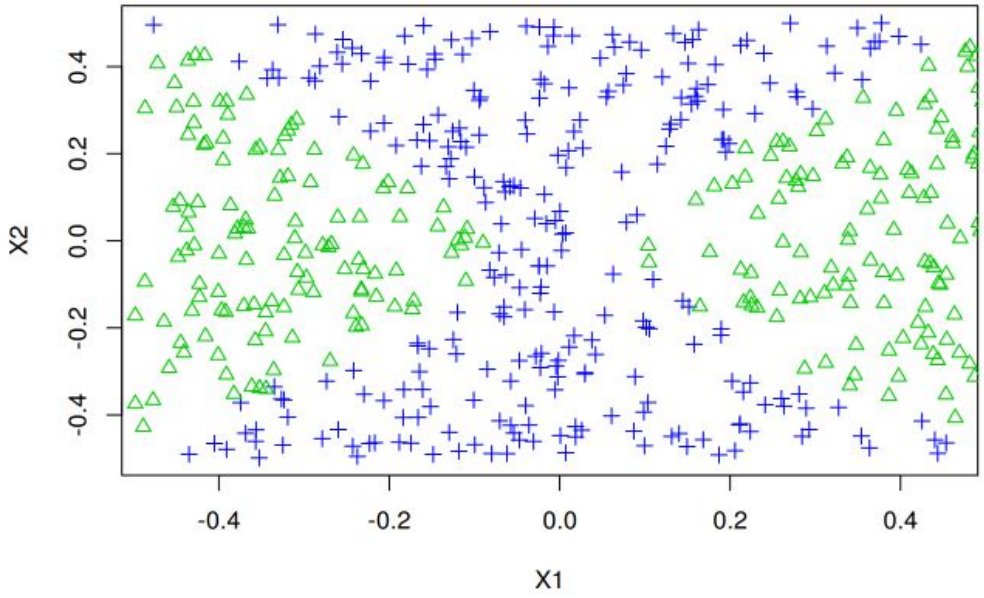
g. Fit a support vector classifier to the data with X1 and X2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
data$y <- as.factor(data$y)
svm.fit <- svm(y ~ x1 + x2, data, kernel = "linear", cost = 0.01)
preds <- predict(svm.fit, data)
plot(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0), xlab = "X1", ylab = "X2")
points(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1))
```



h. Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
data$y <- as.factor(data$y)
svmnl.fit <- svm(y ~ x1 + x2, data, kernel = "radial", gamma = 1)
preds <- predict(svmnl.fit, data)
plot(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0), xlab = "X1", ylab = "X2")
points(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1))
```



i. Comment on your results

Overall, it can be concluded that in order to find non-linear decision boundaries both SVM with non-linear kernel and logistic regression with interaction terms are both equally excellent options. However, if SVM with linear kernel and logistic regression do not have interaction terms they are poor choices when finding non-linear decision boundaries.

Question-7

In this problem, you will use support vector approaches in order to predict whether a given car gets high or low gas mileage based on the Auto data set.

a. Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

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

b. Fit a support vector classifier to the data with various values of cost, in order to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter. Comment on your results.

```
set.seed(1)
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "linear", ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100, 100
0)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.01275641
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-02 0.07403846 0.05471525
## 2 1e-01 0.03826923 0.05148114
## 3 1e+00 0.01275641 0.01344780
## 4 5e+00 0.01782051 0.01229997
## 5 1e+01 0.02038462 0.01074682
## 6 1e+02 0.03820513 0.01773427
## 7 1e+03 0.03820513 0.01773427
```

From the results it can be concluded that a cost of 1 has the best performance.

c. Now repeat (b), this time using SVMs with radial and polynomial basis kernels, with different values of gamma and degree and cost. Comment on your results.

```
set.seed(1)
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "polynomial", ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100),
degree = c(2, 3, 4)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      2
##
## - best performance: 0.3013462
##
## - Detailed performance results:
##     cost degree     error dispersion
## 1  1e-02      2 0.5611538 0.04344202
## 2  1e-01      2 0.5611538 0.04344202
## 3  1e+00      2 0.5611538 0.04344202
## 4  5e+00      2 0.5611538 0.04344202
## 5  1e+01      2 0.5382051 0.05829238
## 6  1e+02      2 0.3013462 0.09040277
## 7  1e-02      3 0.5611538 0.04344202
## 8  1e-01      3 0.5611538 0.04344202
## 9  1e+00      3 0.5611538 0.04344202
## 10 5e+00      3 0.5611538 0.04344202
## 11 1e+01      3 0.5611538 0.04344202
## 12 1e+02      3 0.3322436 0.11140578
## 13 1e-02      4 0.5611538 0.04344202
## 14 1e-01      4 0.5611538 0.04344202
## 15 1e+00      4 0.5611538 0.04344202
## 16 5e+00      4 0.5611538 0.04344202
## 17 1e+01      4 0.5611538 0.04344202
## 18 1e+02      4 0.5611538 0.04344202
```

From the output it can be concluded that, for a polynomial kernel, the smallest cross-validation error is when the degree equals 2 and the cost equals 100.

```
set.seed(1)
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "radial", ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100), gam
ma = c(0.01, 0.1, 1, 5, 10, 100)))
summary(tune.out)
```

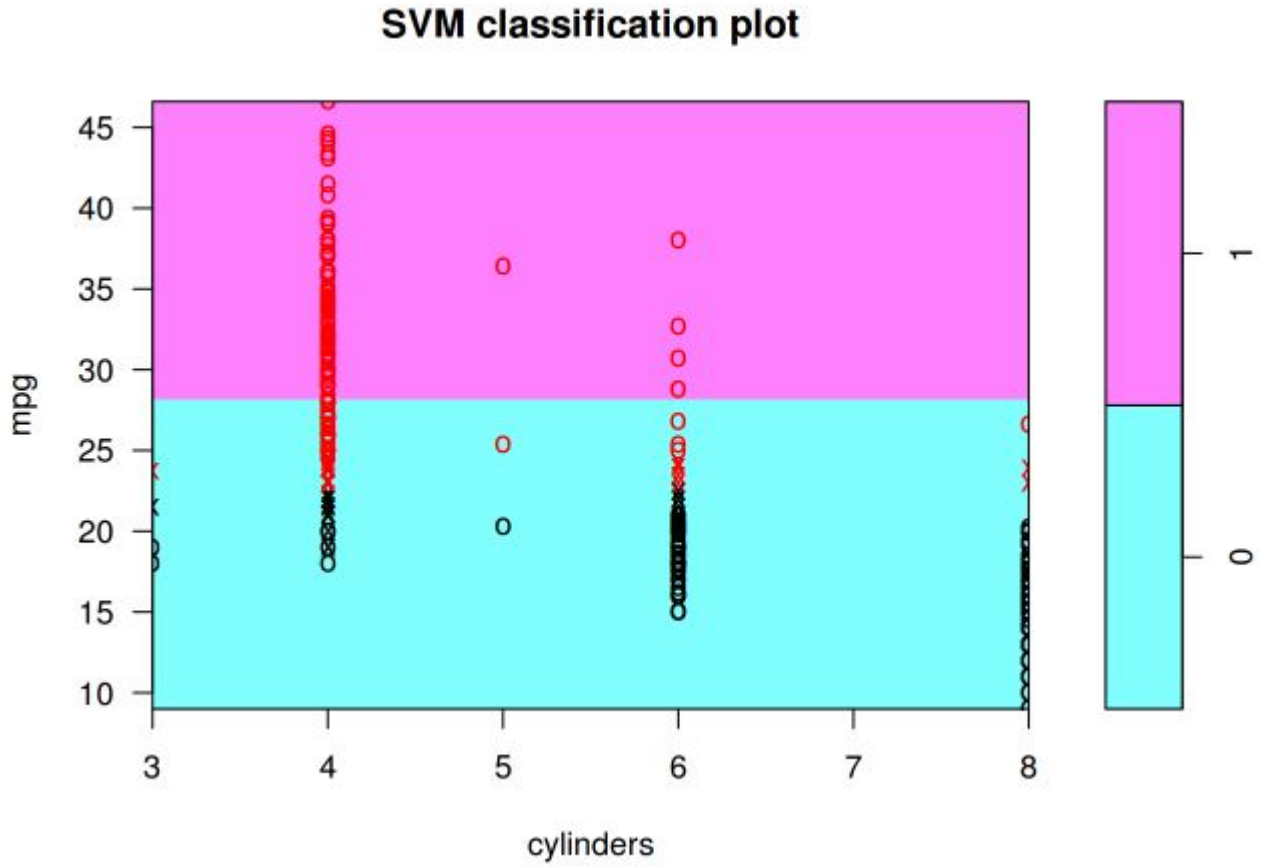
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100  0.01
##
## - best performance: 0.01532051
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-02 1e-02 0.56115385 0.04344202
## 2  1e-01 1e-02 0.09185897 0.03862507
## 3  1e+00 1e-02 0.07147436 0.05103685
## 4  5e+00 1e-02 0.04326923 0.04975032
## 5  1e+01 1e-02 0.02551282 0.03812986
## 6  1e+02 1e-02 0.01532051 0.01788871
## 7  1e-02 1e-01 0.19153846 0.07612945
## 8  1e-01 1e-01 0.07916667 0.05201159
## 9  1e+00 1e-01 0.05608974 0.05092939
## 10 5e+00 1e-01 0.03064103 0.02637448
## 11 1e+01 1e-01 0.02551282 0.02076457
## 12 1e+02 1e-01 0.02807692 0.01458261
## 13 1e-02 1e+00 0.56115385 0.04344202
## 14 1e-01 1e+00 0.56115385 0.04344202
## 15 1e+00 1e+00 0.06634615 0.06187383
## 16 5e+00 1e+00 0.06128205 0.06186124
## 17 1e+01 1e+00 0.06128205 0.06186124
## 18 1e+02 1e+00 0.06128205 0.06186124
## 19 1e-02 5e+00 0.56115385 0.04344202
## 20 1e-01 5e+00 0.56115385 0.04344202
## 21 1e+00 5e+00 0.49224359 0.03806832
## 22 5e+00 5e+00 0.48967949 0.03738577
## 23 1e+01 5e+00 0.48967949 0.03738577
## 24 1e+02 5e+00 0.48967949 0.03738577
## 25 1e-02 1e+01 0.56115385 0.04344202
## 26 1e-01 1e+01 0.56115385 0.04344202
## 27 1e+00 1e+01 0.51775641 0.04471079
## 28 5e+00 1e+01 0.51012821 0.03817175
## 29 1e+01 1e+01 0.51012821 0.03817175
## 30 1e+02 1e+01 0.51012821 0.03817175
## 31 1e-02 1e+02 0.56115385 0.04344202
## 32 1e-01 1e+02 0.56115385 0.04344202
## 33 1e+00 1e+02 0.56115385 0.04344202
## 34 5e+00 1e+02 0.56115385 0.04344202
## 35 1e+01 1e+02 0.56115385 0.04344202
## 36 1e+02 1e+02 0.56115385 0.04344202
```

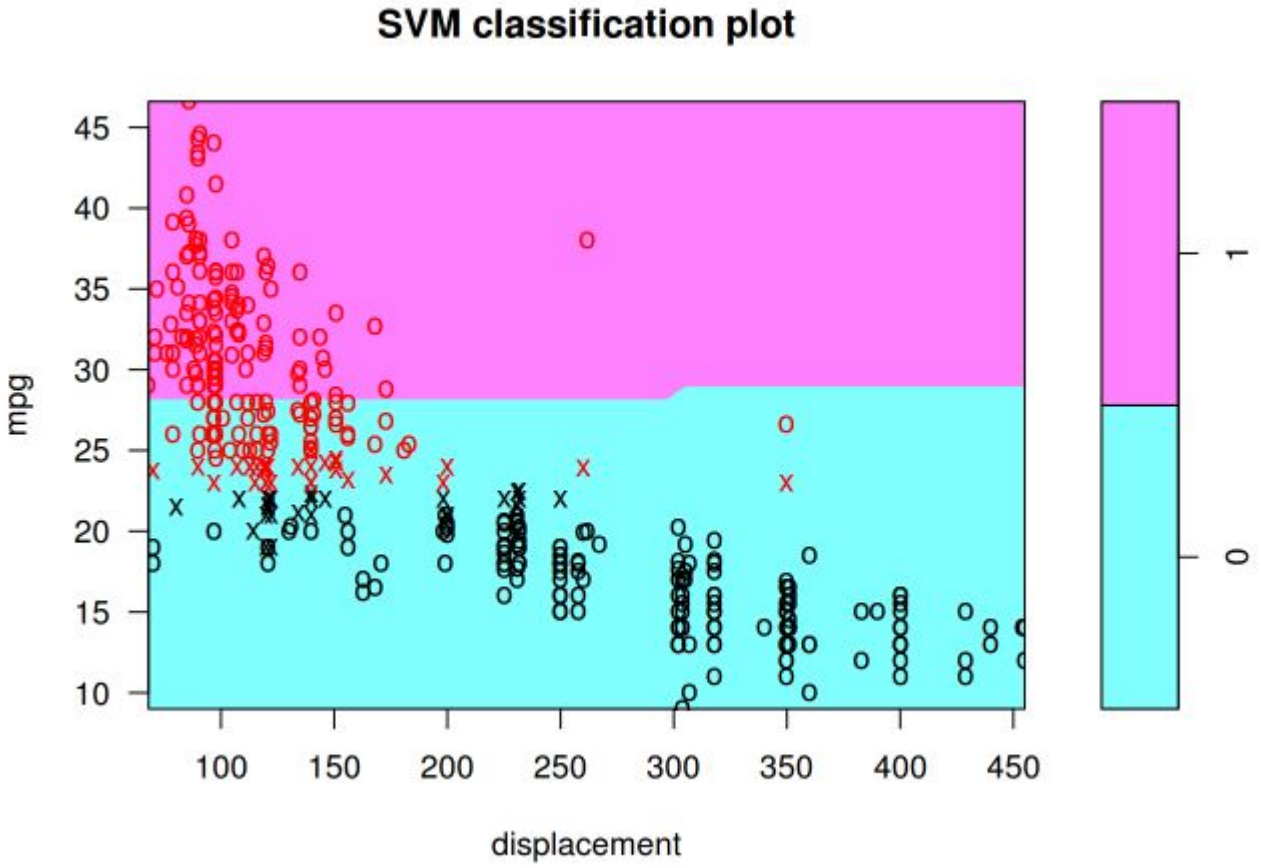
From the output it can be concluded that, for a radial kernel, the smallest cross-validation error is when the gamma equals 0.01 and the cost equals 100.

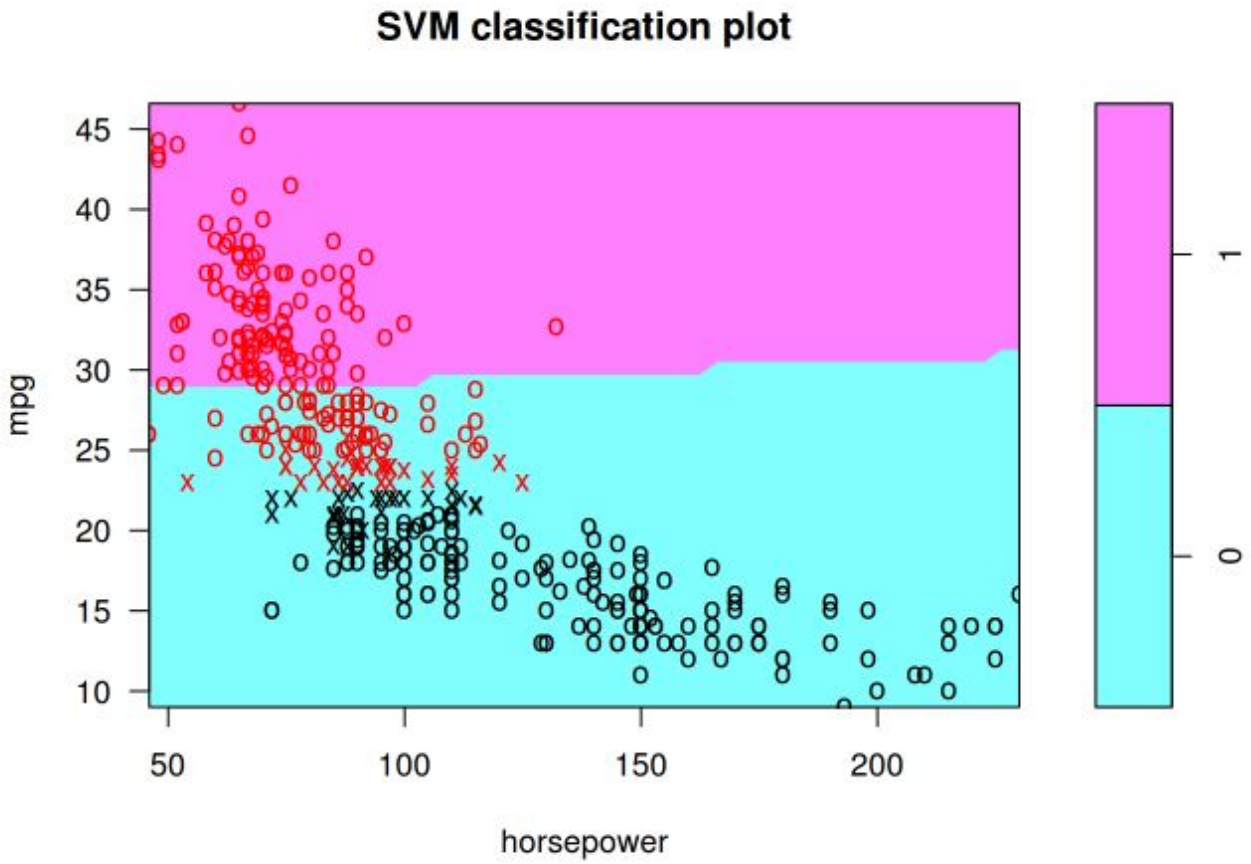
d. Make some plots to back up your assertions in (b) and (c).

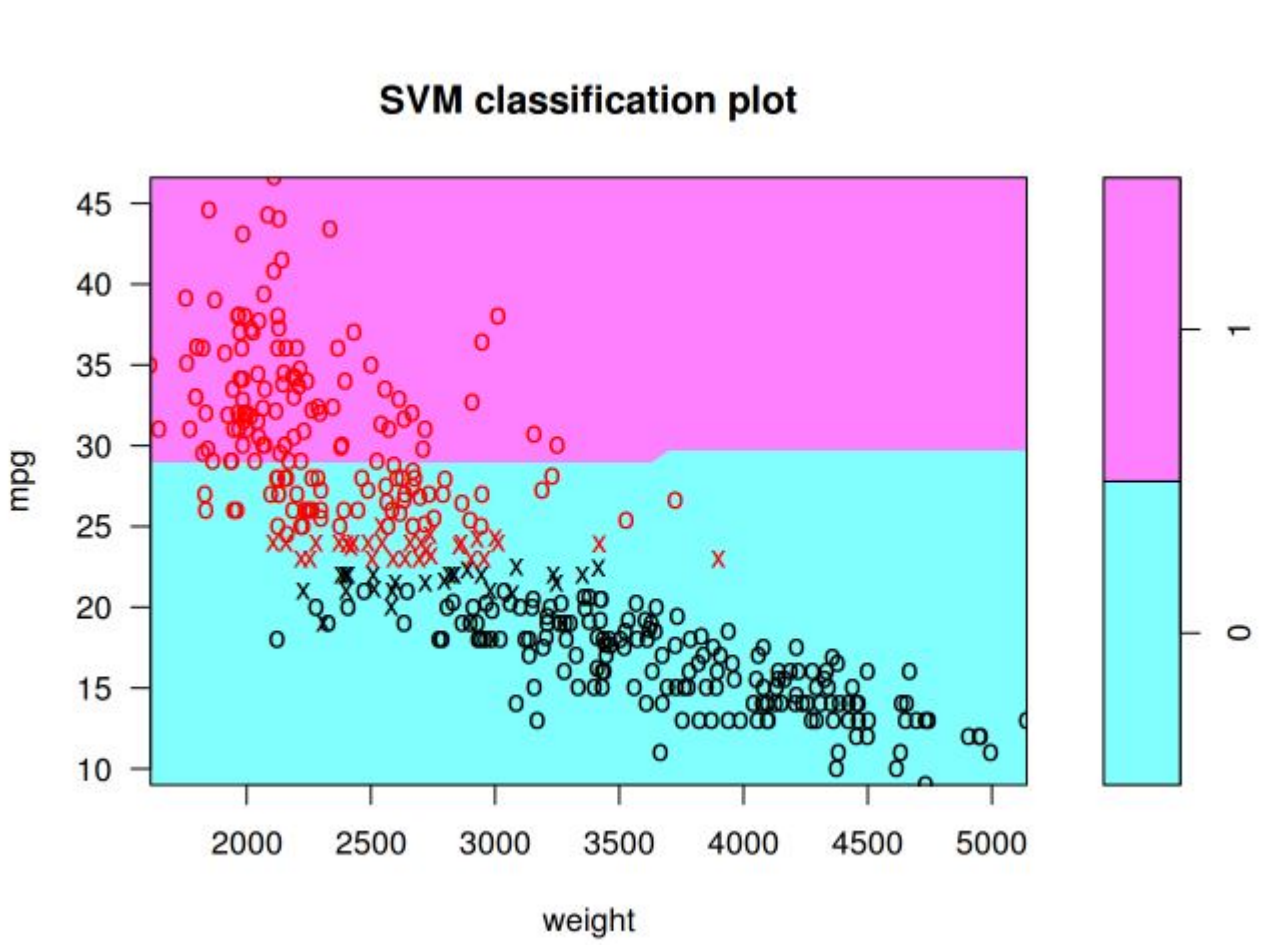
SO MANY PICS (21)

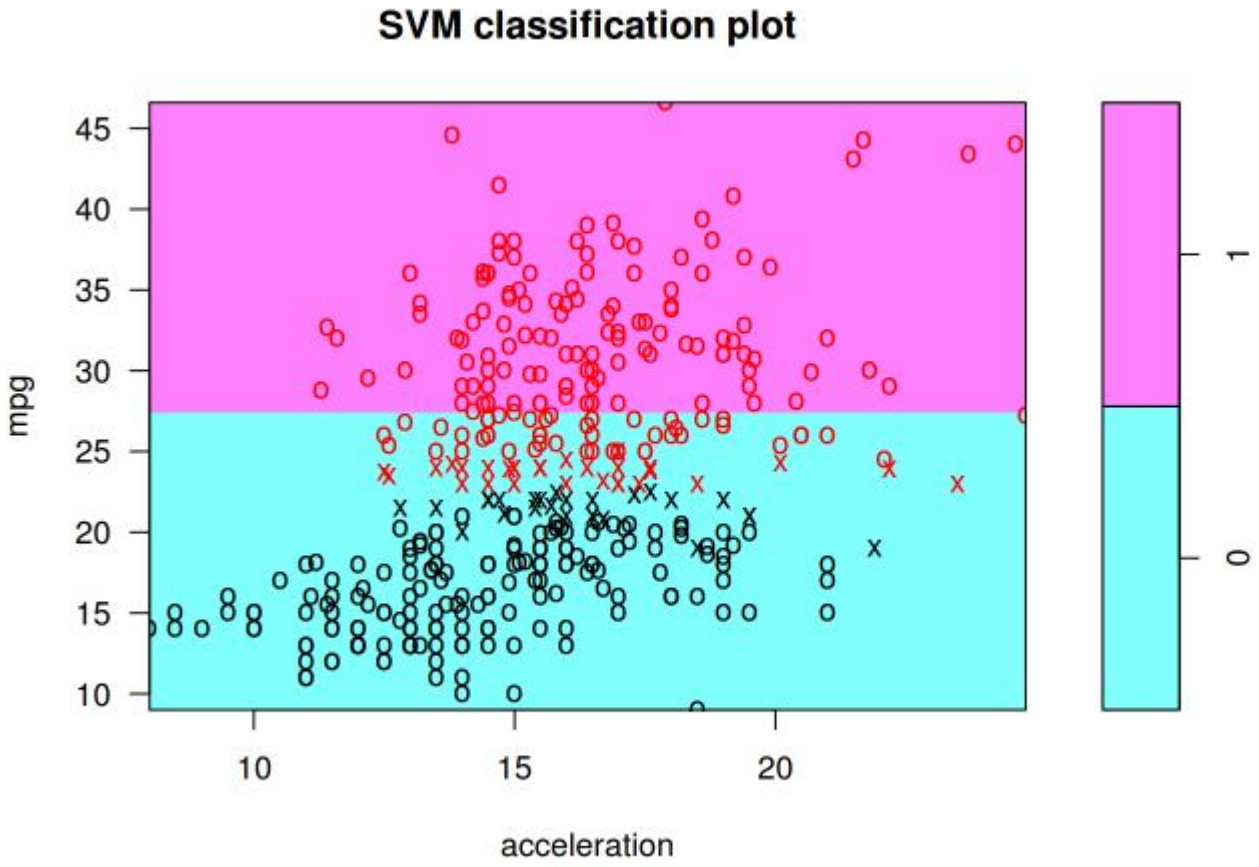
```r
svm.linear <- svm(mpglevel ~ ., data = Auto, kernel = "linear", cost = 1)
svm.poly <- svm(mpglevel ~ ., data = Auto, kernel = "polynomial", cost = 100, degree = 2)
svm.radial <- svm(mpglevel ~ ., data = Auto, kernel = "radial", cost = 100, gamma = 0.01)
plotpairs = function(fit) {
    for (name in names(Auto)[!(names(Auto) %in% c("mpg", "mpglevel", "name"))]) {
        plot(fit, Auto, as.formula(paste("mpg~", name, sep = "")))
    }
}
plotpairs(svm.linear)
```

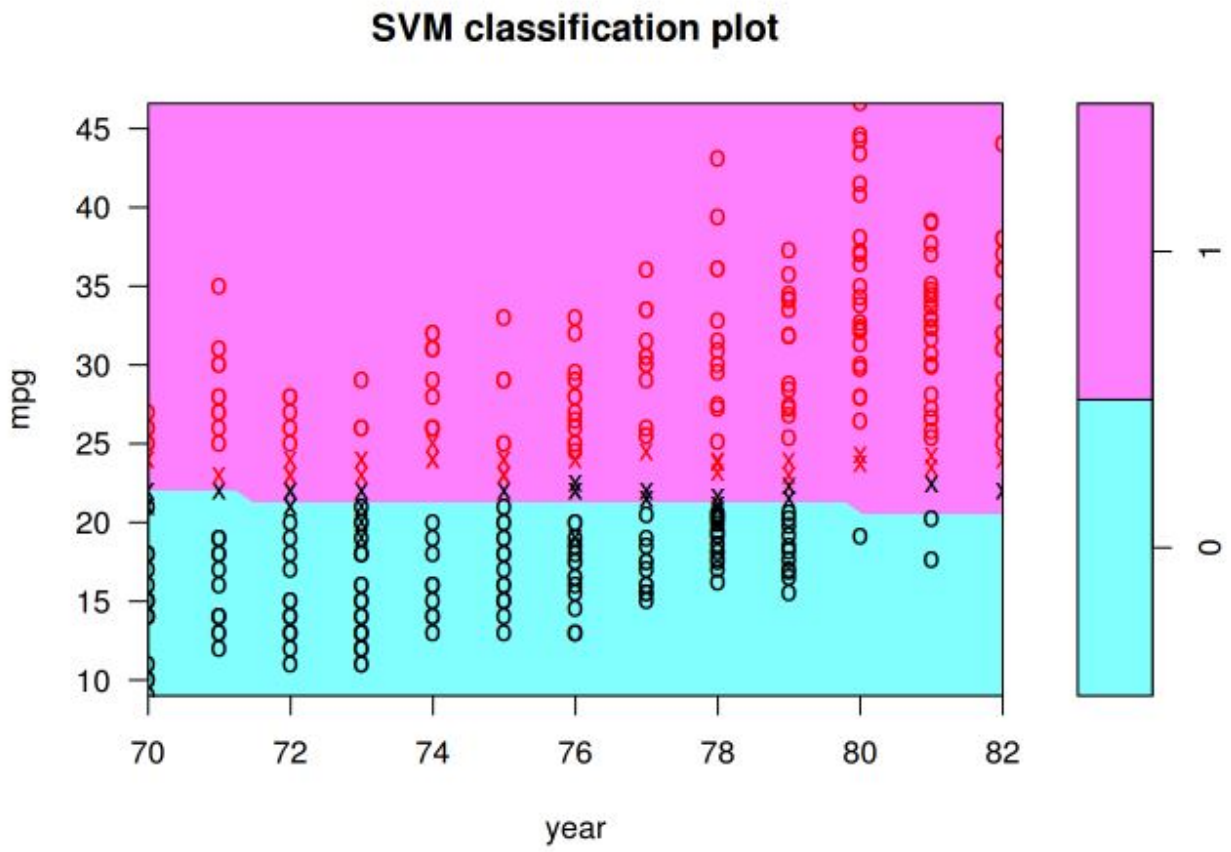

SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0

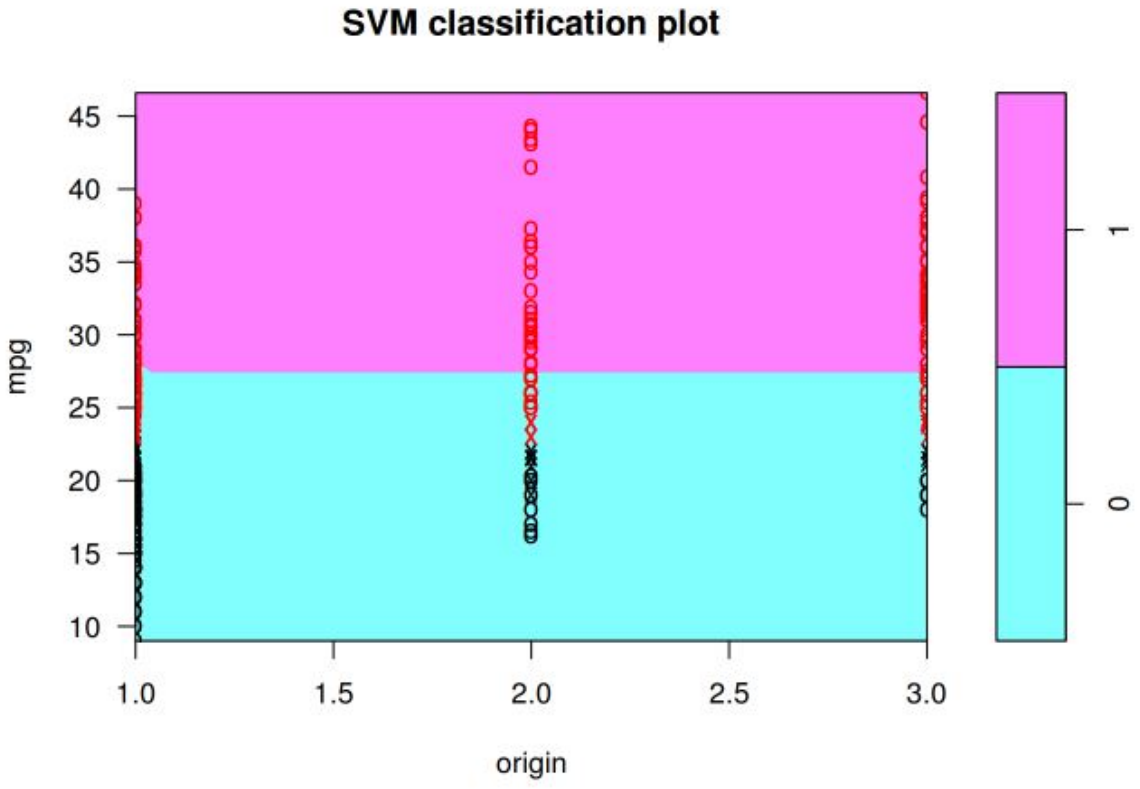

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0


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

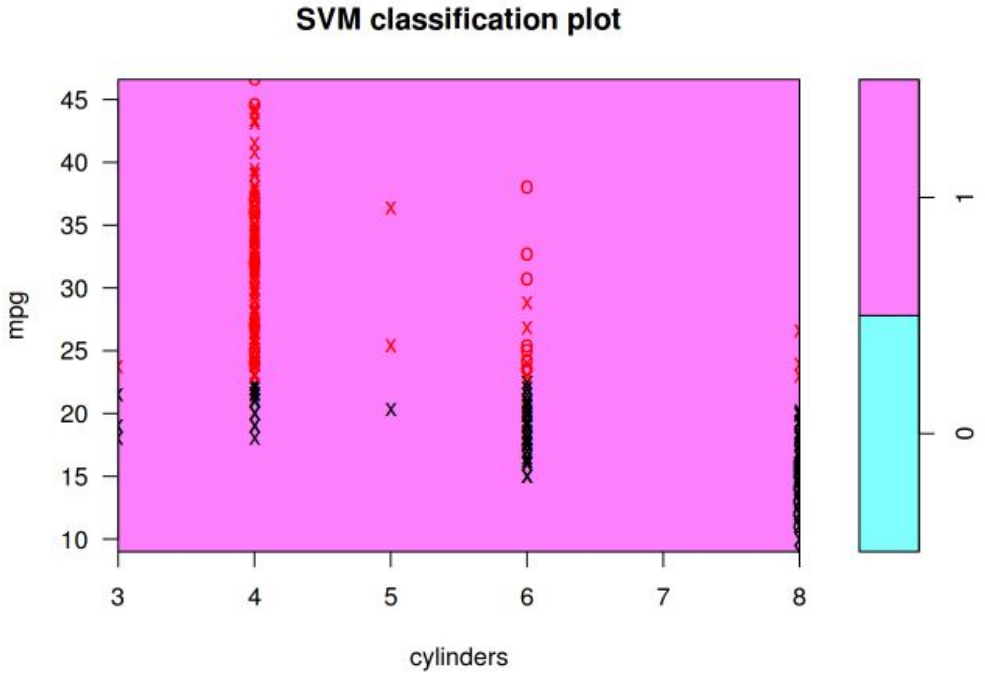

SVM classification plot
mpg
45
40
35
30
25
20
15
10
2000
2500
3000
3500
4000
4500
5000
weight
1
0

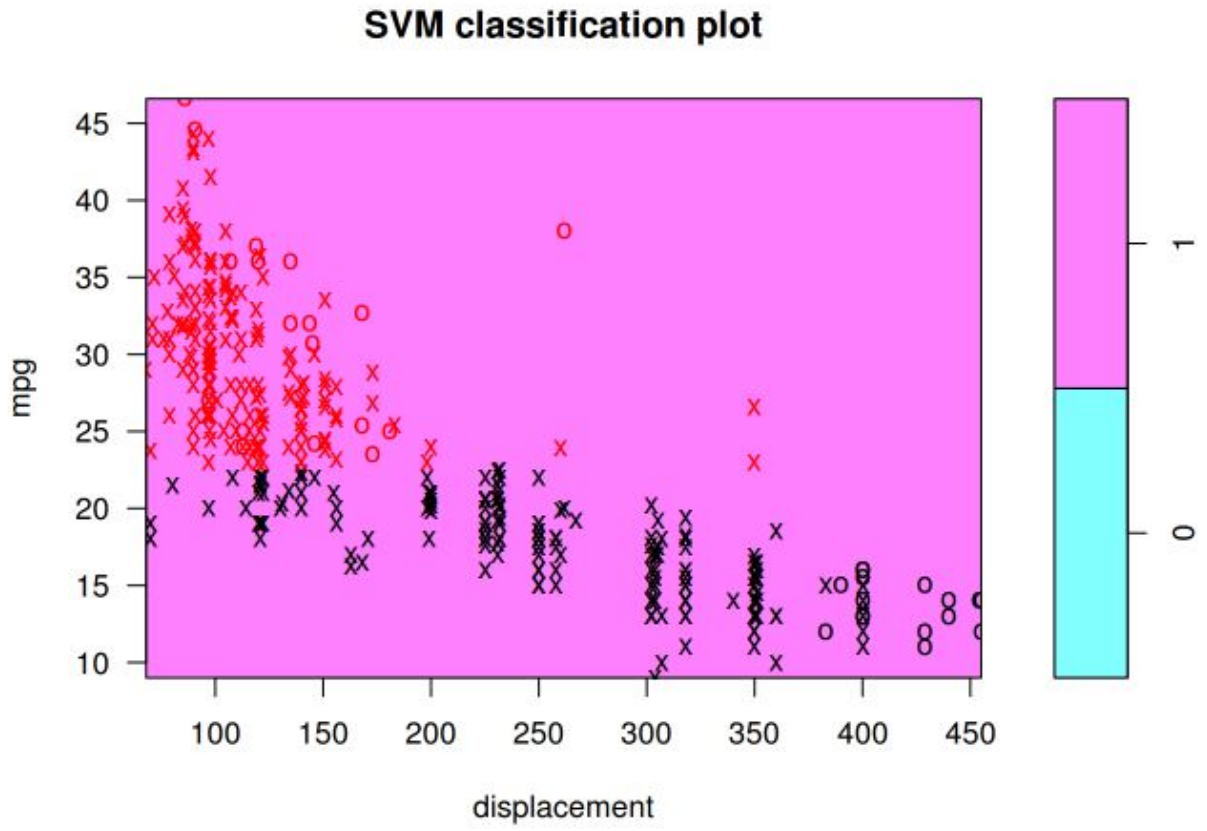

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0

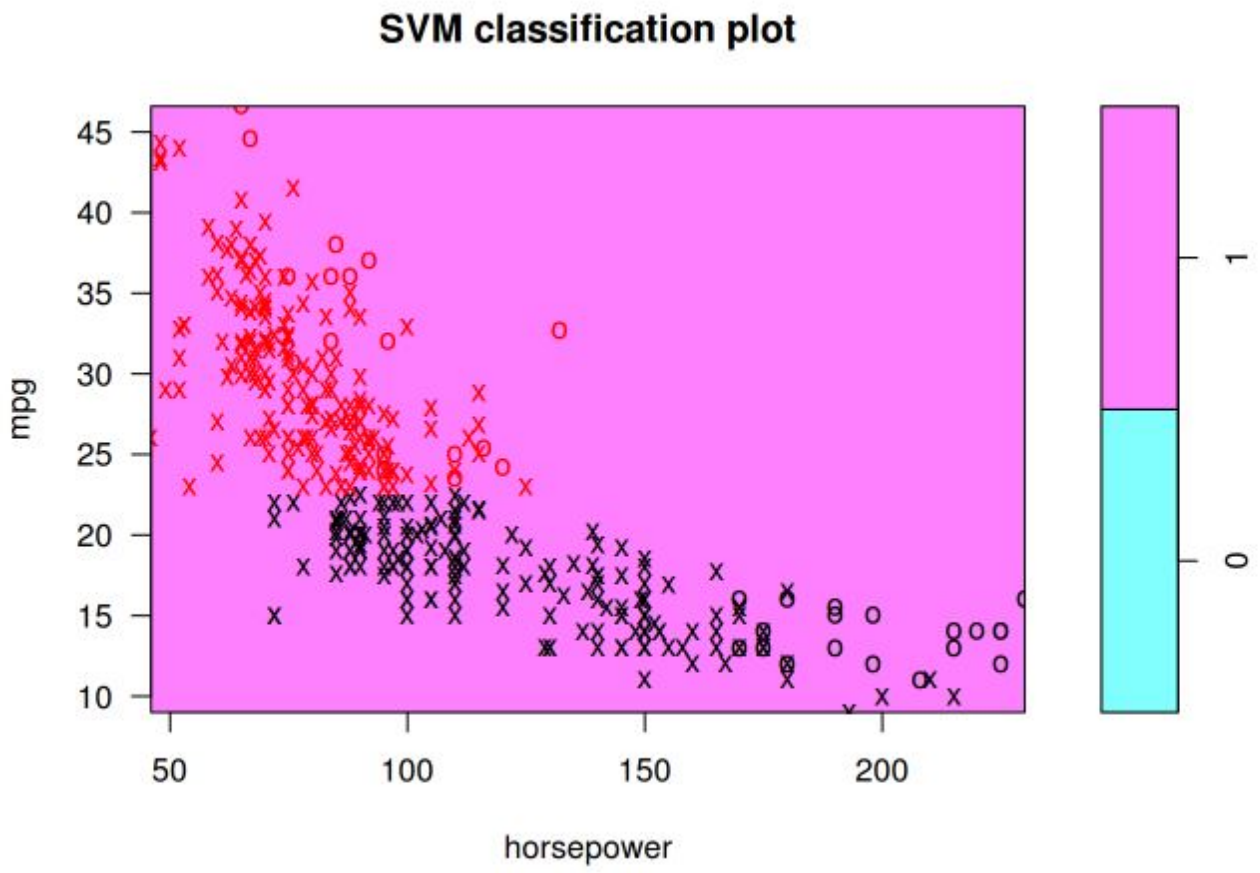

SVM classification plot
mpg
year
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
1
0

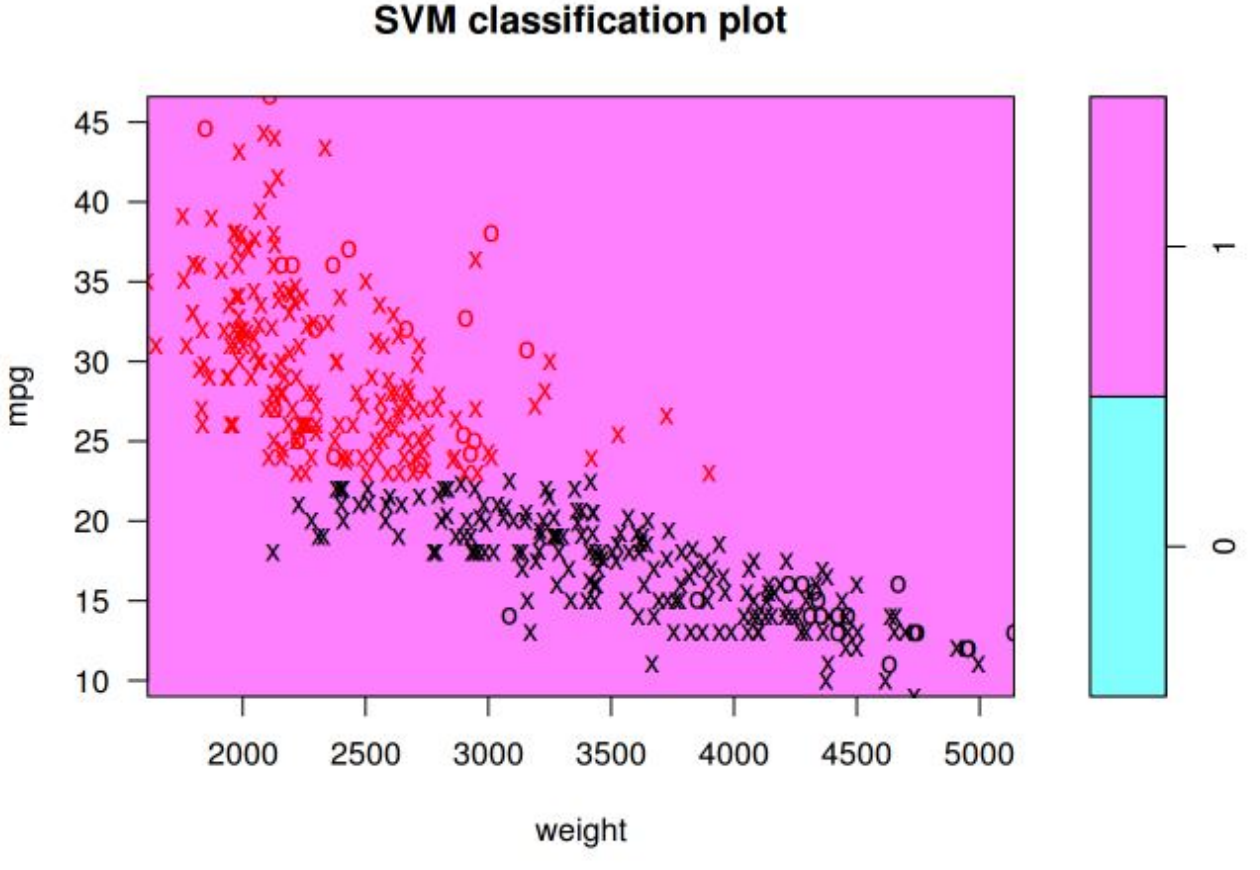

SVM classification plot
mpg
origin
45
40
35
30
25
20
15
10
1.0
1.5
2.0
2.5
3.0
1
0

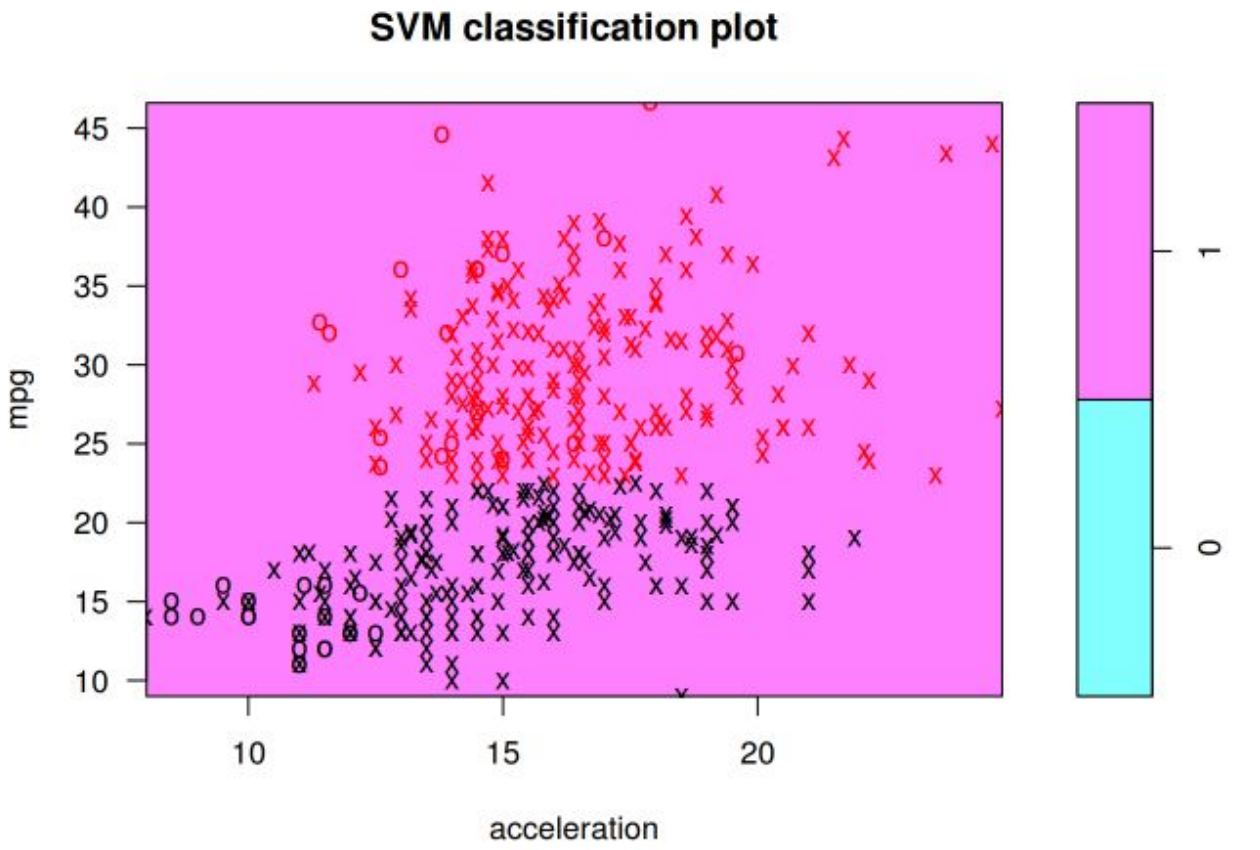
```
plotpairs(svm.poly)
```

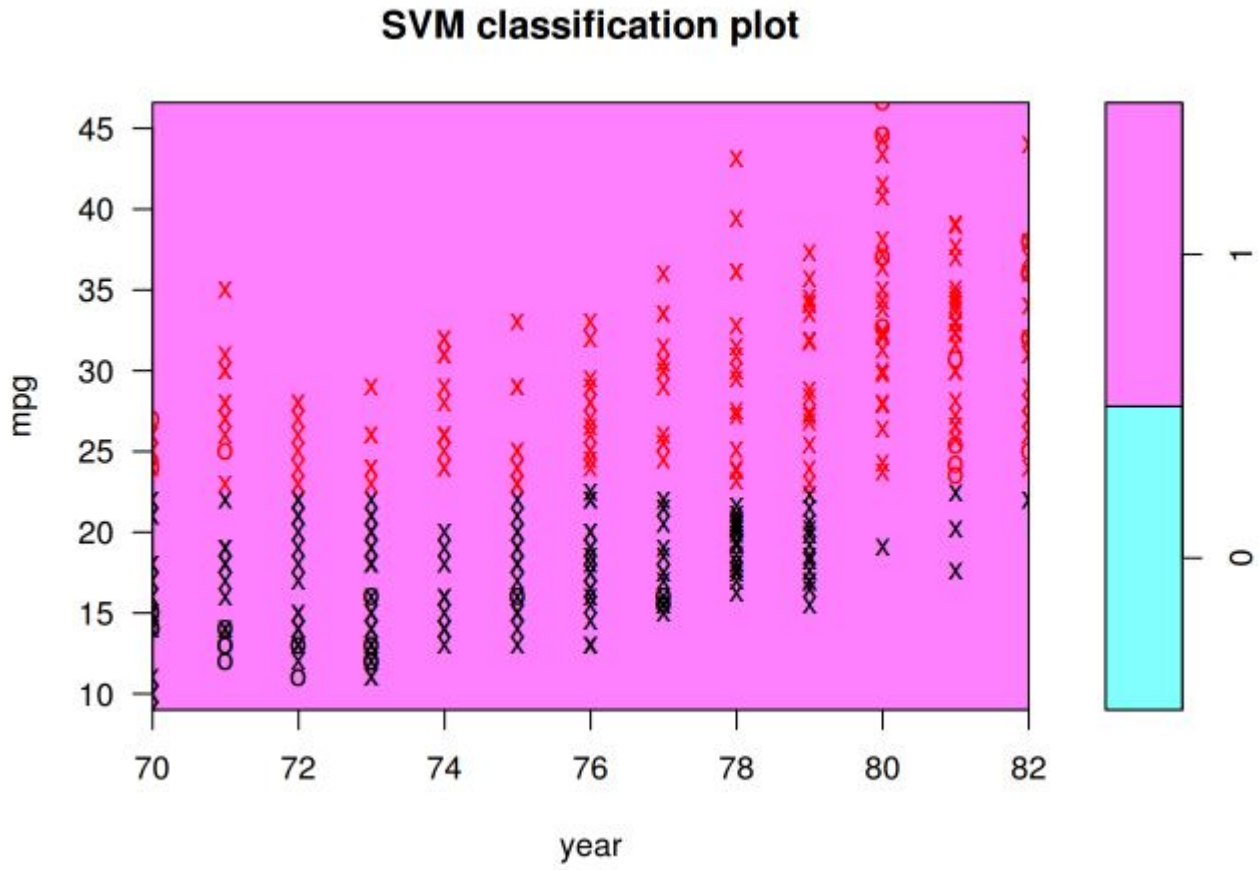


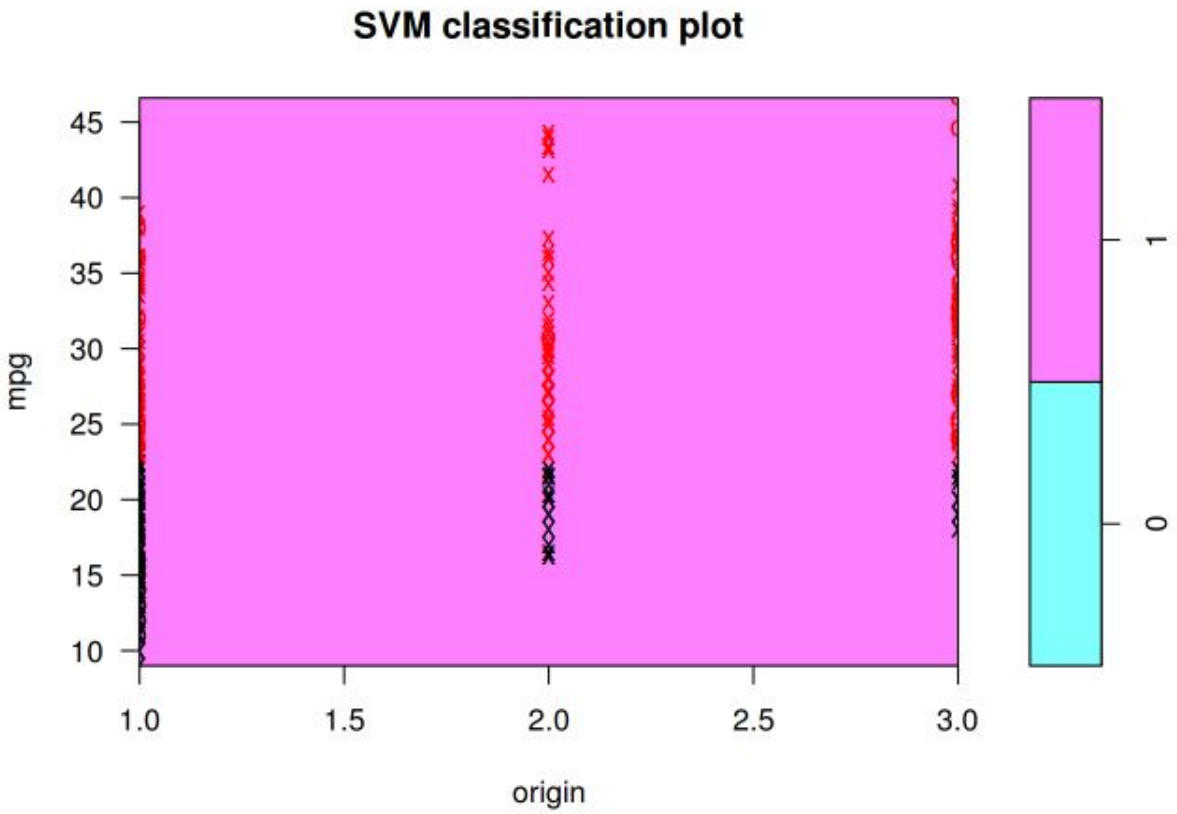



SVM classification plot
mpg
horsepower
45
40
35
30
25
20
15
10
50
100
150
200
1
0

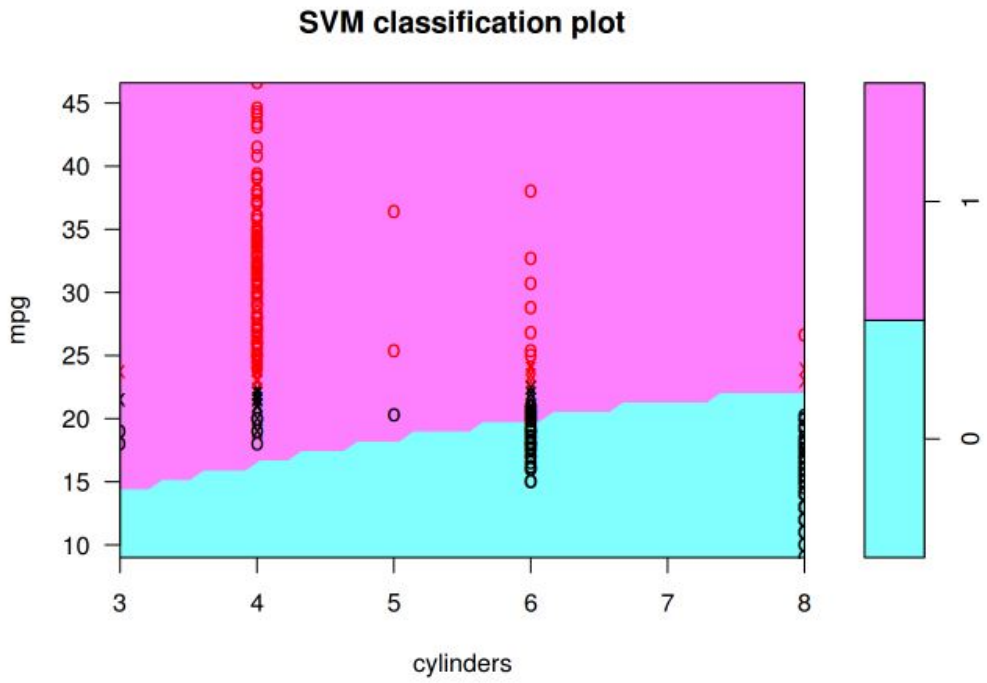

SVM classification plot
mpg
weight
45
40
35
30
25
20
15
10
2000
2500
3000
3500
4000
4500
5000
1
0

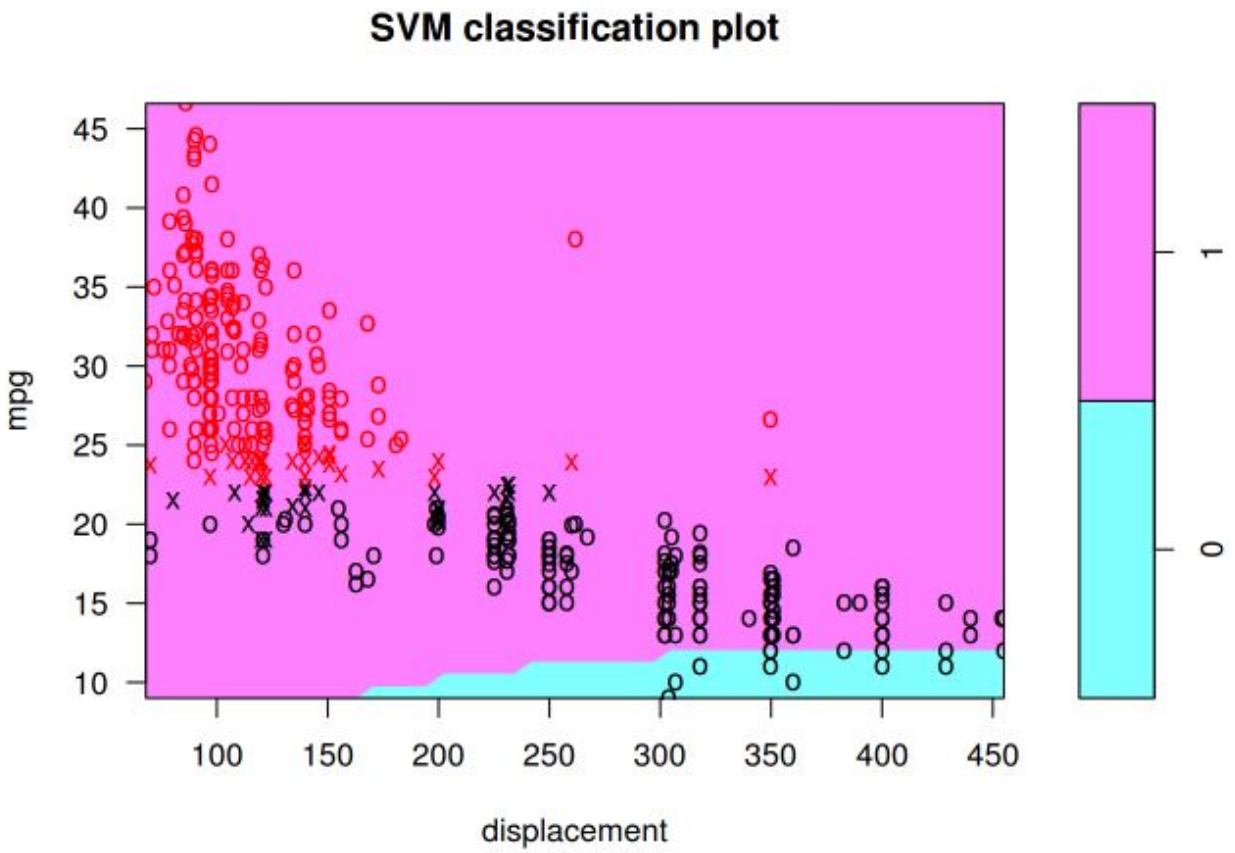

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0

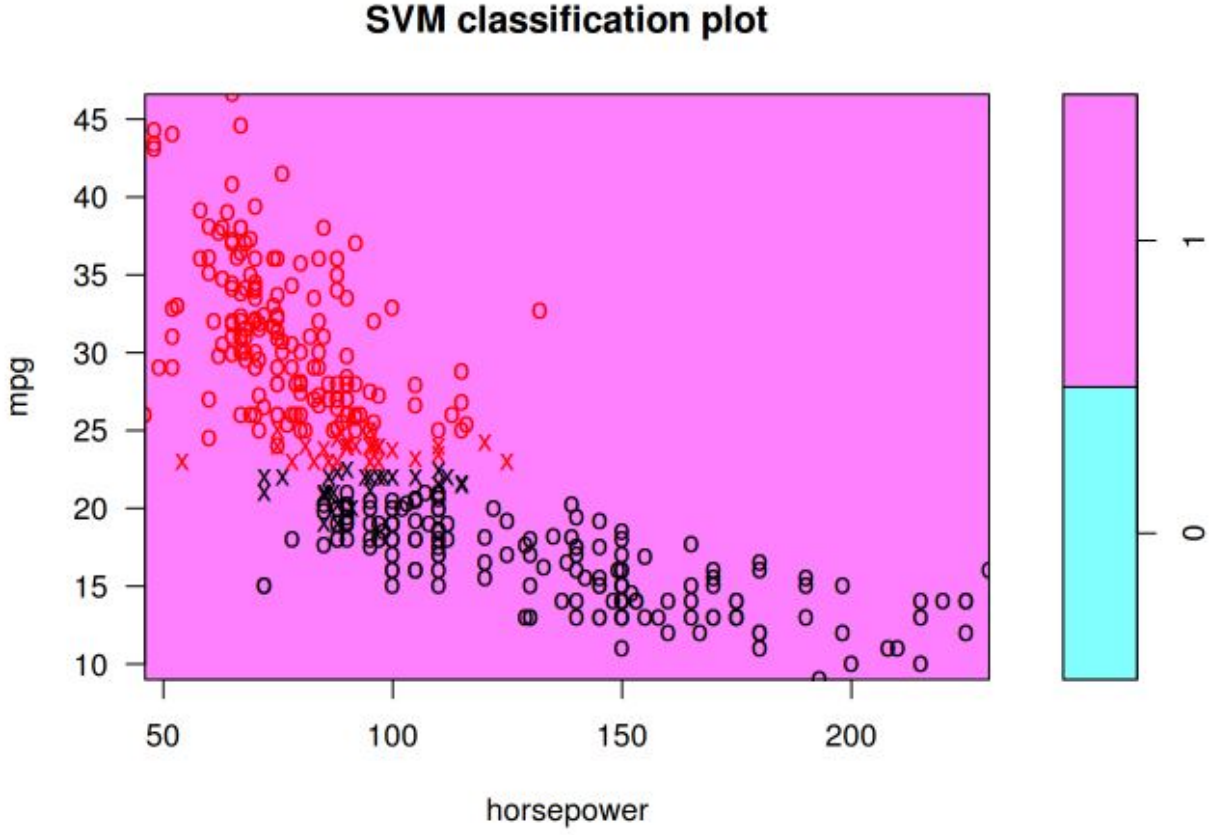

SVM classification plot
mpg
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
year
1
0

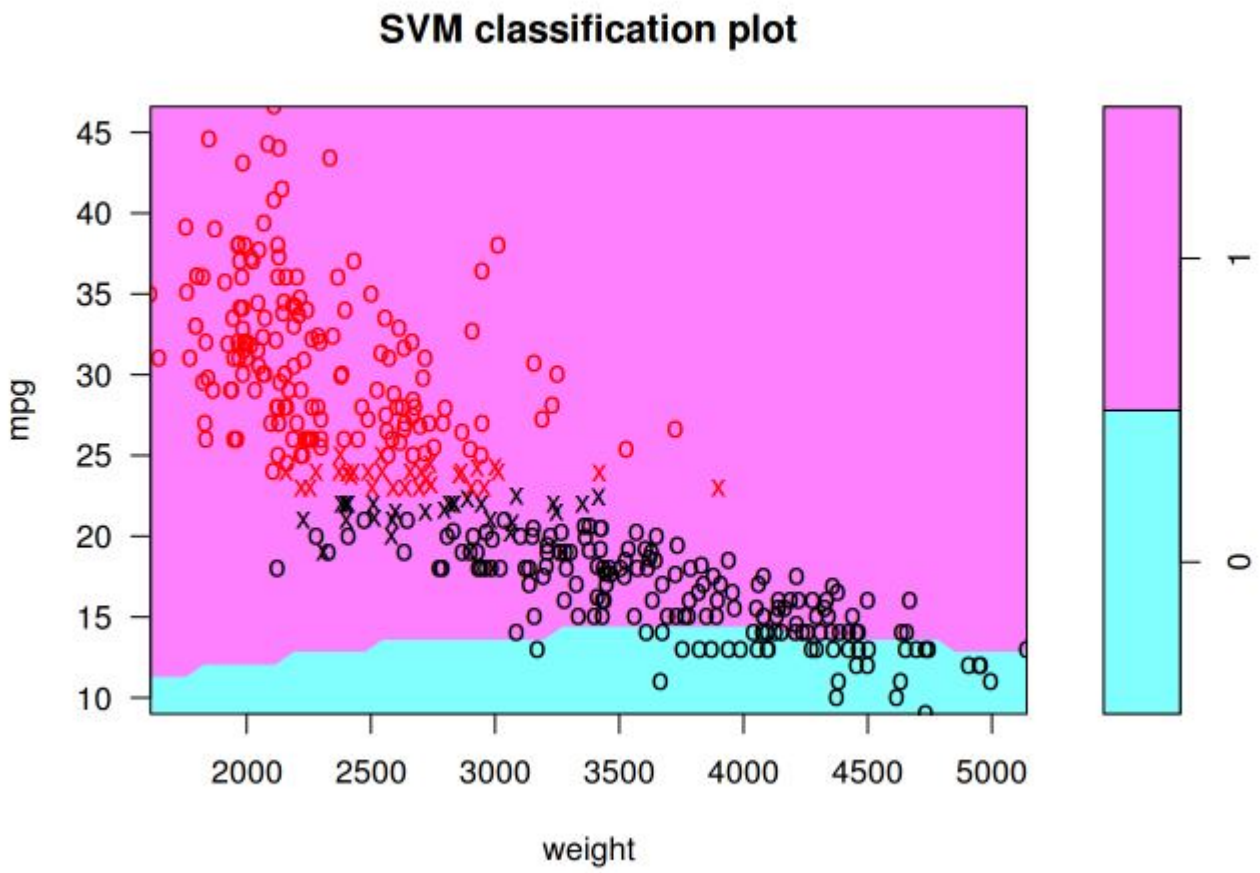


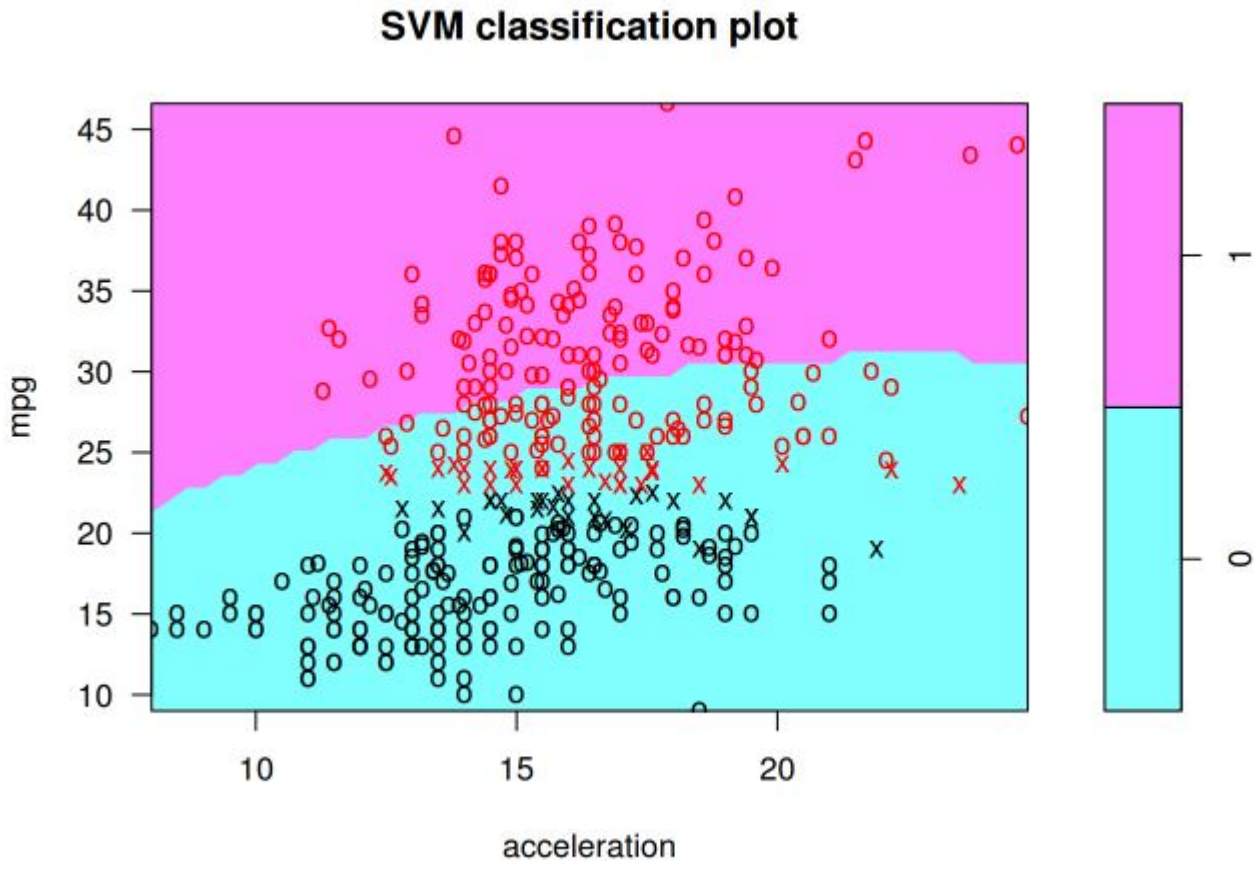

```
plotpairs(svm.radial)
```

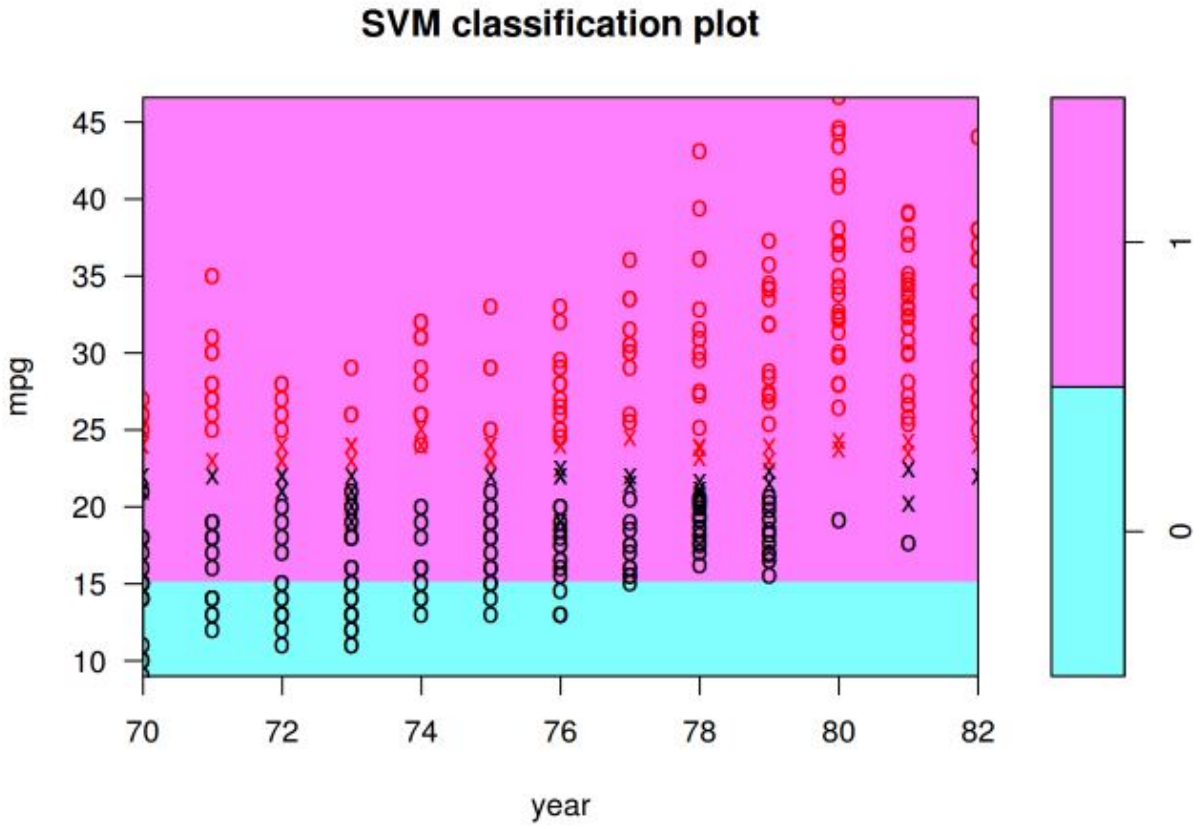


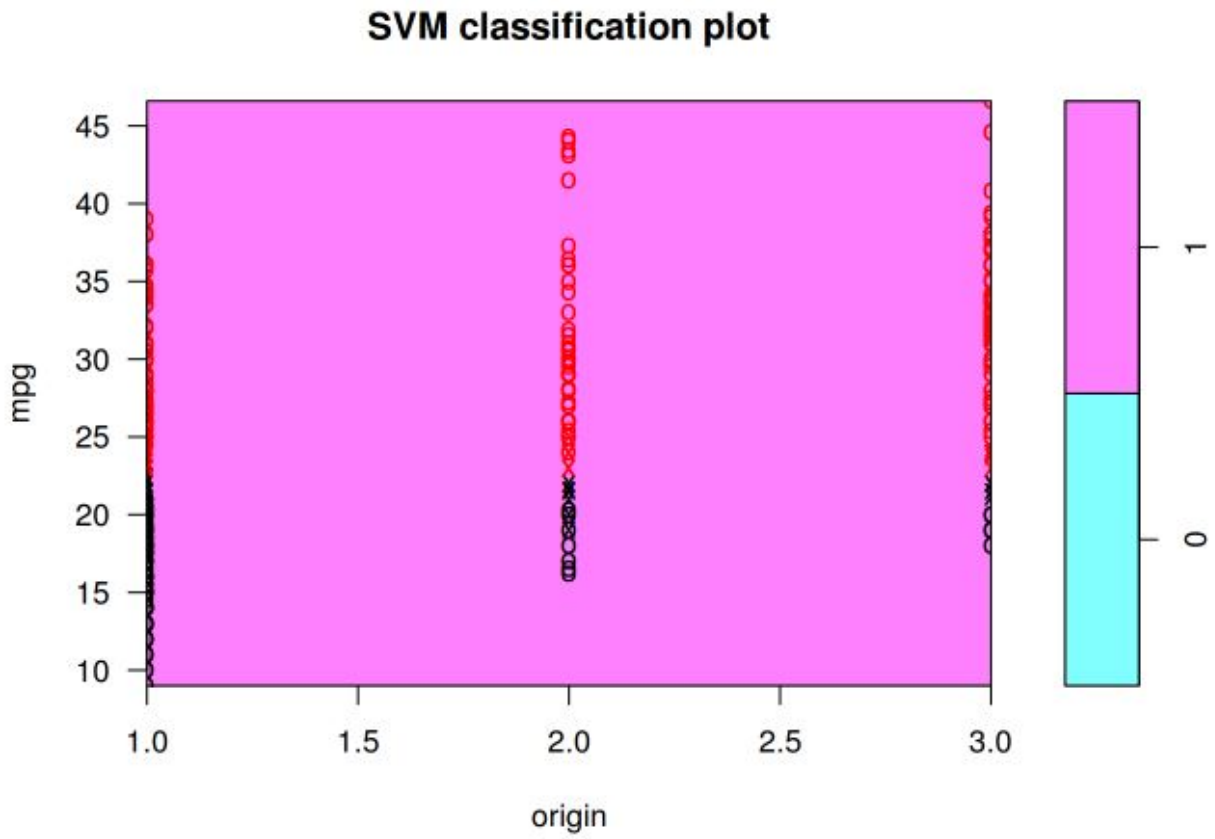

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0


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


SVM classification plot
mpg
45
40
35
30
25
20
15
10
2000
2500
3000
3500
4000
4500
5000
weight
1
0


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


SVM classification plot
mpg
year
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
1
0



Question-8

This problem involves the OJ data set which is part of the ISLR package.

a. Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
set.seed(1)
train <- sample(nrow(OJ), 800)
OJ.train <- OJ[train, ]
OJ.test <- OJ[-train, ]
```

b. Fit a support vector classifier to the training data using cost=0.01, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics, and describe the results obtained.

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

```
##
## 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:  432
##
##  ( 215 217 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The results obtained include that there are 432 support vectors created out of 800 training points. Furthermore, from those 432 support vectors 217 are in the MM level and 215 are in the CH level.

c. What are the training and test error rates?

```
train.pred <- predict(svm.linear, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 439  55
##   MM  78 228
```

```
(78 + 55) / (439 + 228 + 78 + 55)
```

```
## [1] 0.16625
```

```
test.pred <- predict(svm.linear, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 141  18
##   MM  31  80
```

```
(31 + 18) / (141 + 80 + 31 + 18)
```

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

The training and test error rates are 16.6% and 18.1% respectively.

d. Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.

```
set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", ranges = list(cost = 10^seq(-2, 1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.1625
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.17125 0.05172376
## 2   0.01778279 0.16500 0.05197489
## 3   0.03162278 0.16625 0.04604120
## 4   0.05623413 0.16500 0.04594683
## 5   0.10000000 0.16250 0.04787136
## 6   0.17782794 0.16250 0.04249183
## 7   0.31622777 0.16875 0.04379958
## 8   0.56234133 0.16625 0.03998698
## 9   1.00000000 0.16500 0.03670453
## 10  1.77827941 0.16625 0.03682259
## 11  3.16227766 0.16500 0.03717451
## 12  5.62341325 0.16500 0.03525699
## 13 10.00000000 0.16750 0.03917553
```

From the results it can be concluded that having a cost of 0.1 is the most favorable.

e. Compute the training and test error rates using this new value for cost.

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

```
##     train.pred
##       CH  MM
##   CH 438  56
##   MM  71 235
```

```
(71 + 56) / (438 + 235 + 71 + 56)
```

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

```
test.pred <- predict(svm.linear, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 140  19
##   MM  32  79
```

```
(32 + 19) / (140 + 79 + 32 + 19)
```

```
## [1] 0.1888889
```

Utilizing the optimal cost of 0.1, the resulting training and test error rates are 15.8% and 18.8% respectively.

f. Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.

```
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
##       gamma:  0.05555556
##
## Number of Support Vectors:  379
##
##  ( 188 191 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.radial, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 455  39
##   MM  77 229
```

```
(77 + 39) / (455 + 229 + 77 + 39)
```

```
## [1] 0.145
```

```
test.pred <- predict(svm.radial, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 141  18
##   MM  28  83
```

```
(28 + 18) / (141 + 83 + 28 + 18)
```

```
## [1] 0.1703704
```

```r
set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", ranges = list(cost = 10^seq(-2,
    1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.16625
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.38250 0.04533824
## 2   0.01778279 0.38250 0.04533824
## 3   0.03162278 0.37500 0.04894725
## 4   0.05623413 0.21500 0.05886661
## 5   0.10000000 0.17875 0.04860913
## 6   0.17782794 0.17875 0.05497790
## 7   0.31622777 0.17875 0.05981743
## 8   0.56234133 0.17250 0.05458174
## 9   1.00000000 0.16625 0.05001736
## 10  1.77827941 0.16875 0.05008673
## 11  3.16227766 0.17500 0.04787136
## 12  5.62341325 0.18000 0.05244044
## 13 10.00000000 0.18250 0.05596378
```

```r
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:  1
##       gamma:  0.05555556
##
## Number of Support Vectors:  379
##
##  ( 188 191 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.radial, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 455  39
##   MM  77 229
```

```
(77 + 39) / (455 + 229 + 77 + 39)
```

```
## [1] 0.145
```

```
test.pred <- predict(svm.radial, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 141  18
##   MM  28  83
```

```
(28 + 18) / (141 + 83 + 28 + 18)
```

```
## [1] 0.1703704
```

g. Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree=2.

```
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
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  454
##
##  ( 224 230 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.poly, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 461  33
##   MM 105 201
```

```
(105 + 33) / (461 + 201 + 105 + 33)
```

```
## [1] 0.1725
```

```
test.pred <- predict(svm.poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 149  10
##   MM  41  70
```

```
(41 + 10) / (149 + 70 + 41 + 10)
```

```
## [1] 0.1888889
```

```r
set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "polynomial", degree = 2, ranges = list(cost = 10^seq(-2,
    1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.18125
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.38250 0.04533824
## 2   0.01778279 0.36750 0.04972145
## 3   0.03162278 0.36500 0.05458174
## 4   0.05623413 0.33375 0.05070681
## 5   0.10000000 0.32500 0.04677072
## 6   0.17782794 0.25875 0.05952649
## 7   0.31622777 0.21250 0.06123724
## 8   0.56234133 0.21250 0.05743354
## 9   1.00000000 0.19750 0.06687468
## 10  1.77827941 0.19375 0.05376453
## 11  3.16227766 0.19625 0.05653477
## 12  5.62341325 0.18375 0.05434266
## 13 10.00000000 0.18125 0.05245699
```

```r
svm.poly <- svm(Purchase ~ ., kernel = "polynomial", degree = 2, data = OJ.train, cost = tune.out$best.parameter$cost)
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
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  342
##
##  ( 170 172 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.poly, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 450  44
##   MM  72 234
```

```
(72 + 44) / (450 + 234 + 72 + 44)
```

```
## [1] 0.145
```

```
test.pred <- predict(svm.poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 140  19
##   MM  31  80
```

```
(31 + 19) / (140 + 80 + 31 + 19)
```

```
## [1] 0.1851852
```

h. Overall, which approach seems to give the best results on this data?

In conclusion, it is evident that the radial basis kernel approach gives the best results on this data. Furthermore, this is due to the fact that the radial kernel approach produces the least amount of misclassification error for the train and test data.