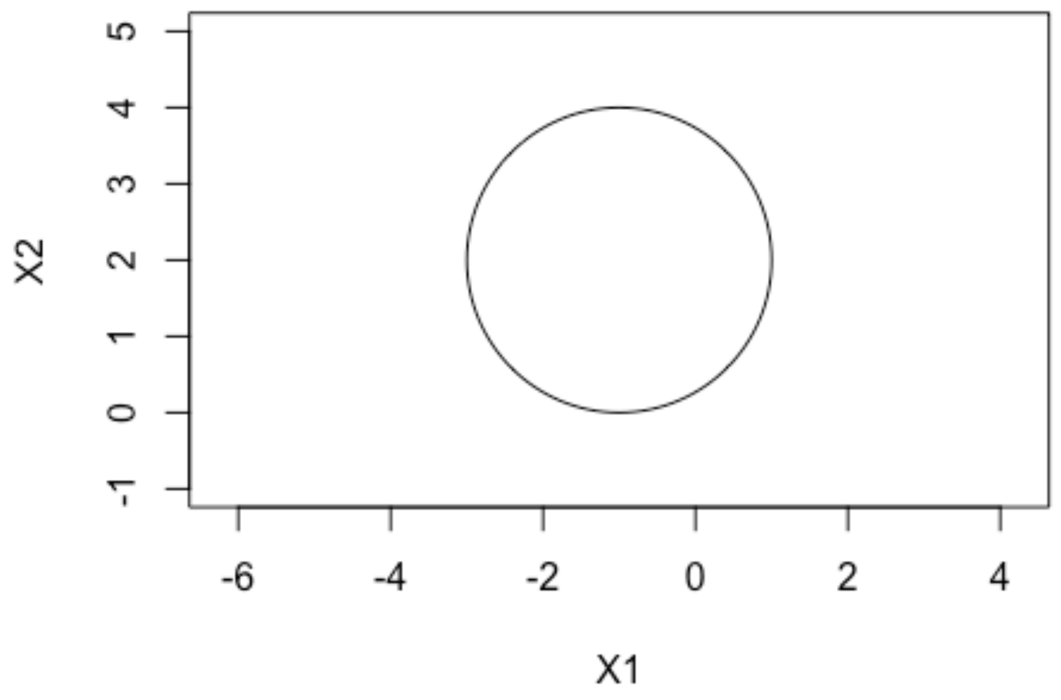
Assignment_12_Shihan.R

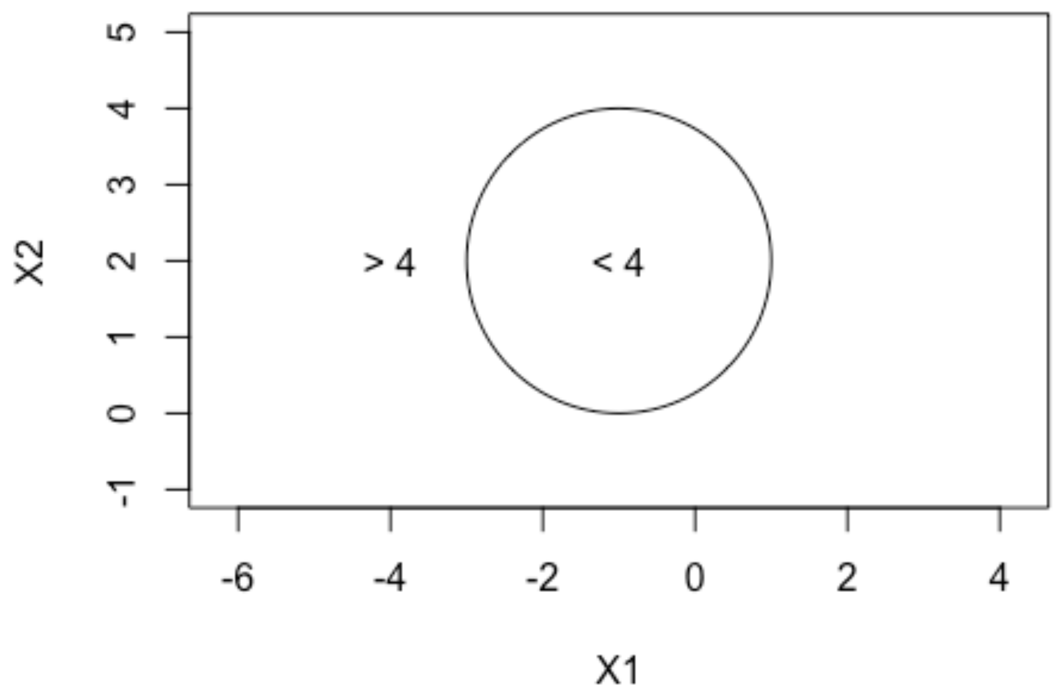
sshihan

Sun Apr 17 10:14:45 2016

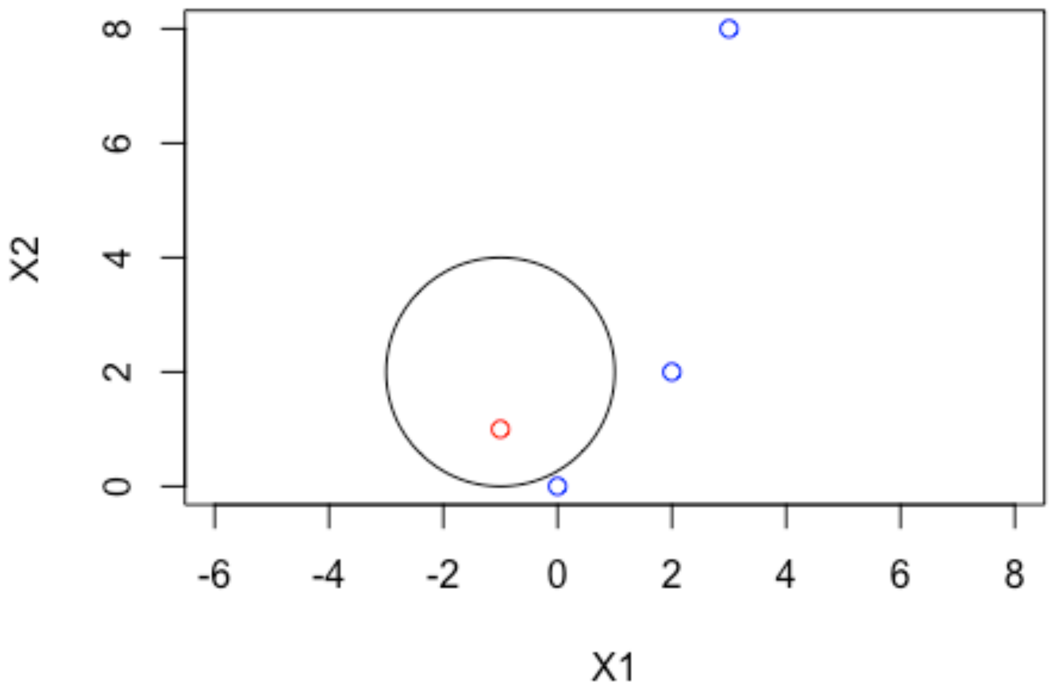
```
# # Assignment 12
# # Shaheed Shihan
# # Modern Applied Statistics
#
# 2. We have seen that in p = 2 dimensions, a linear decision boundary
# takes the form β0+β1X1+β2X2 = 0.We now investigate a non-linear
# decision boundary.
# (a) Sketch the curve
# (1 + X1)2 + (2 − X2)2 = 4.

plot(NA, NA, type = "n", xlim = c(-4, 2), ylim = c(-1, 5), asp = 1, xlab = "X
1", ylab = "X2")
symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)
```



```r
# (b) On your sketch, indicate the set of points for which
# (1 + X1)2 + (2 − X2)2 > 4,
# as well as the set of points for which
# (1 + X1)2 + (2 − X2)2 ≤ 4.

plot(NA, NA, type = "n", xlim = c(-4, 2), ylim = c(-1, 5), asp = 1, xlab = "X
1", ylab = "X2")
symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)
text(c(-1), c(2), "< 4")
text(c(-4), c(2), "> 4")
```



```r
# (c) Suppose that a classifier assigns an observation to the blue class
# if
# (1 + X1)2 + (2 − X2)2 > 4,
# and to the red class otherwise. To what class is the observation
# (0, 0) classified? (−1, 1)? (2, 2)? (3, 8)?

plot(c(0, -1, 2, 3), c(0, 1, 2, 8), col = c("blue", "red", "blue", "blue"),
     type = "p", asp = 1, xlab = "X1", ylab = "X2")
symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)
```



```
# (d) Argue that while the decision boundary in (c) is not linear in
# terms of X1 and X2, it is linear in terms of X1, X2
# 1 , X2, and
# X2
```

$$(1+X_1)^2+(2-X_2)^2=4$$
$$(X_1)^2+(X_2)^2+2X_1-4X_2+1=0$$

which is linear.

```
# 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)
library(e1071)

data("Auto")
#head(Auto)
attach(Auto)
```

```
mileage <- as.factor(ifelse(mpg > median(mpg), 1,0))
Auto <- cbind(Auto, mileage)
#head(Auto)
#str(Auto)

# (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)
tunelinear <- tune(svm,mileage~.,data=Auto,kernel="linear",ranges=list(cost=c
(0.001, 0.01, 0.1, 1,5,10,100)))
summary(tunelinear)
```

```
##
## 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-03 0.09442308 0.03837365
## 2 1e-02 0.07403846 0.05471525
## 3 1e-01 0.03826923 0.05148114
## 4 1e+00 0.01275641 0.01344780
## 5 5e+00 0.01782051 0.01229997
## 6 1e+01 0.02038462 0.01074682
## 7 1e+02 0.03820513 0.01773427
```

```
bestmod1 <- tunelinear$best.model
summary(bestmod1)
```

```
##
## Call:
## best.tune(method = svm, train.x = mileage ~ ., data = Auto, ranges = list(
cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.003205128
```

```
##
## Number of Support Vectors:  56
##
##  ( 26 30 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
#
# (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)
tunerad <-tune(svm, mileage~., data=Auto, kernel="radial", ranges=list(cost=c
(0.1,1,10,100,1000),gamma=c(0.5,1,2,3,4)))

summary(tunerad)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.5
##
## - best performance: 0.04852564
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-01   0.5 0.07916667 0.05201159
## 2  1e+00   0.5 0.05365385 0.05470590
## 3  1e+01   0.5 0.04852564 0.04597747
## 4  1e+02   0.5 0.04852564 0.04597747
## 5  1e+03   0.5 0.04852564 0.04597747
## 6  1e-01   1.0 0.56115385 0.04344202
## 7  1e+00   1.0 0.06634615 0.06187383
## 8  1e+01   1.0 0.06128205 0.06186124
## 9  1e+02   1.0 0.06128205 0.06186124
## 10 1e+03   1.0 0.06128205 0.06186124
## 11 1e-01   2.0 0.56115385 0.04344202
## 12 1e+00   2.0 0.12756410 0.05916990
## 13 1e+01   2.0 0.11730769 0.05277475
## 14 1e+02   2.0 0.11730769 0.05277475
## 15 1e+03   2.0 0.11730769 0.05277475
## 16 1e-01   3.0 0.56115385 0.04344202
## 17 1e+00   3.0 0.37256410 0.14111555
## 18 1e+01   3.0 0.35205128 0.14165290
## 19 1e+02   3.0 0.35205128 0.14165290
## 20 1e+03   3.0 0.35205128 0.14165290
## 21 1e-01   4.0 0.56115385 0.04344202
## 22 1e+00   4.0 0.48198718 0.04342861
## 23 1e+01   4.0 0.47435897 0.05241275
## 24 1e+02   4.0 0.47435897 0.05241275
## 25 1e+03   4.0 0.47435897 0.05241275
```

```
bestmod2 <- tunerad$best.model
summary(bestmod2)
```

```
##
## Call:
## best.tune(method = svm, train.x = mileage ~ ., data = Auto, ranges = list(
cost = c(0.1,
##     1, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.5
##
## Number of Support Vectors:  259
##
##  ( 127 132 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

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

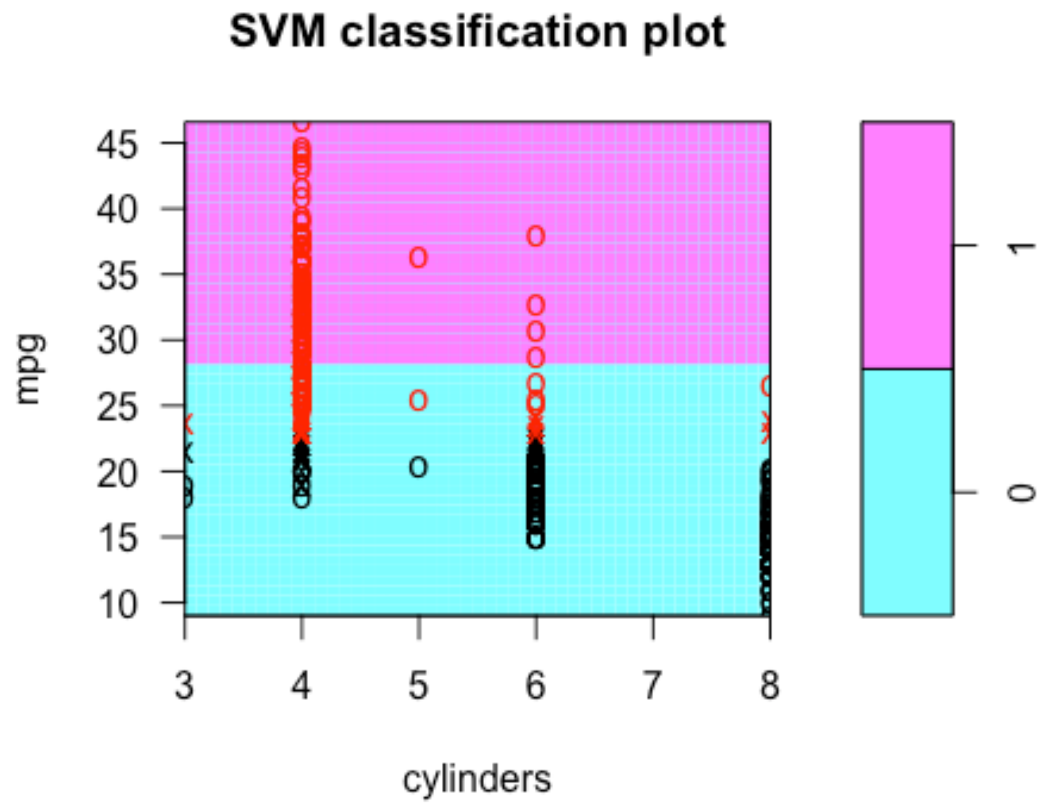
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      2
##
```

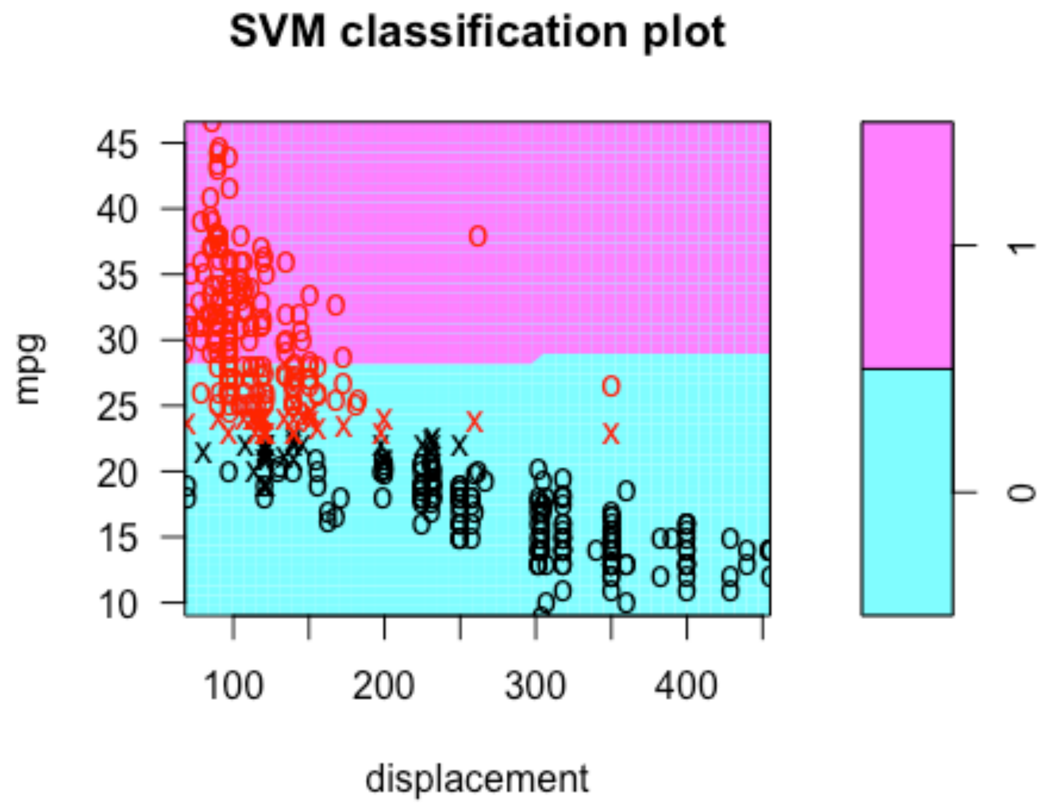
```
## - best performance: 0.3013462
##
## - Detailed performance results:
##      cost degree     error dispersion
## 1  1e-02      2 0.5611538 0.04344202
## 2  1e-01      2 0.5611538 0.04344202
## 3  1e+00      2 0.5611538 0.04344202
## 4  5e+00      2 0.5611538 0.04344202
## 5  1e+01      2 0.5382051 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
```

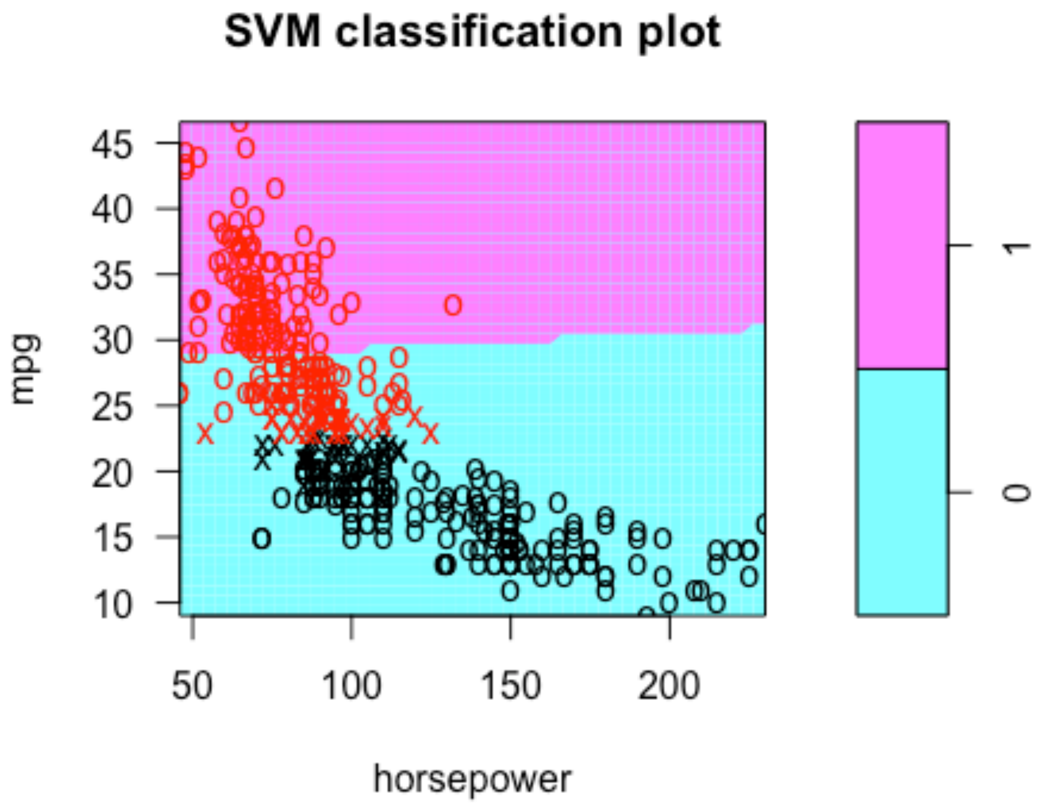
```
bestmod3 <- tunepol$best.model
summary(bestmod3)
```

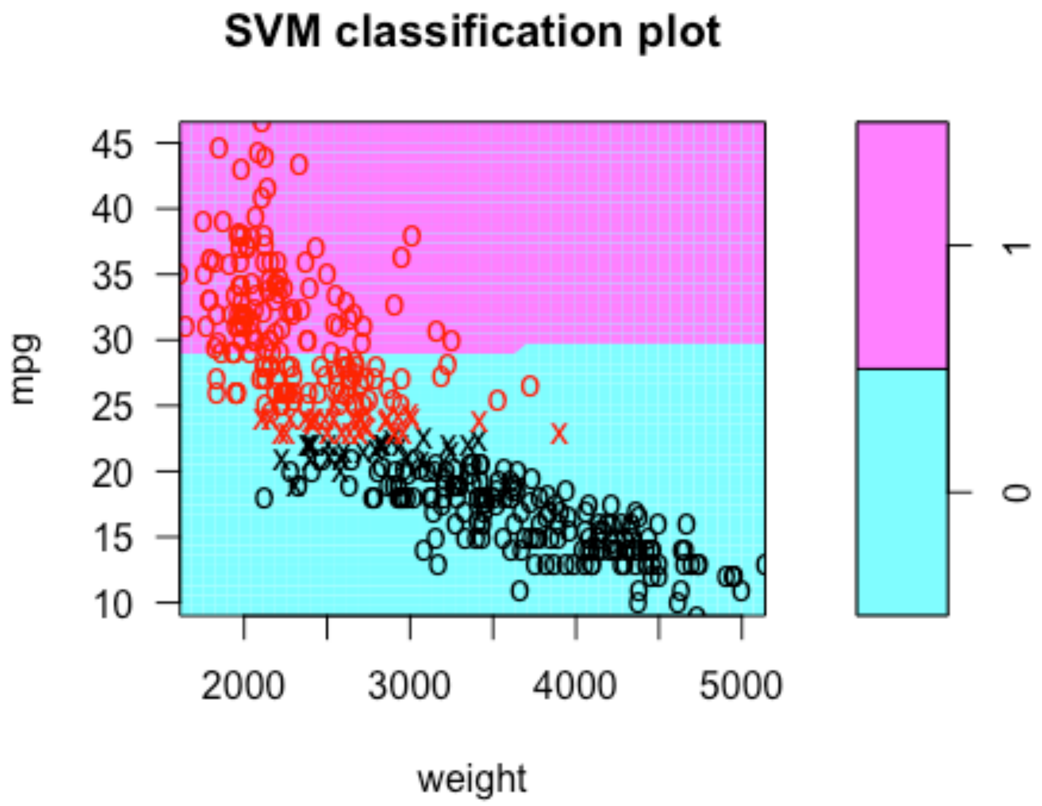
```
##
## Call:
## best.tune(method = svm, train.x = mileage ~ ., data = Auto, ranges = list(
cost = c(0.01,
##     0.1, 1, 5, 10, 100), degree = c(2, 3, 4)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  100
##      degree:  2
##       gamma:  0.003205128
##      coef.0:  0
##
## Number of Support Vectors:  355
##
##  ( 177 178 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```r
# (d) Make some plots to back up your assertions in (b) and (c).
# Hint: In the lab, we used the plot() function for svm objects
# only in cases with p = 2. When p > 2, you can use the plot()
# function to create plots displaying pairs of variables at a time.
# Essentially, instead of typing
# > plot(svmfit , dat)
# where svmfit contains your fitted model and dat is a data frame
# containing your data, you can type
# > plot(svmfit , dat , x1~x4)
# in order to plot just the first and fourth variables. However, you
# must replace x1 and x4 with the correct variable names. To find
# out more, type ?plot.svm.

fit <- svm(mileage~., data = Auto, kernel = "linear", cost = 1)
fit2 <- svm(mileage~., data = Auto, kernel = "polynomial",cost = 100, degree
= 2)
fit3 <- svm(mileage~. , data = Auto, kernel = "radial", cost = 10, gamma = 0.
5)

plotp <-  function(f){
  for (name in names(Auto)[!(names(Auto) %in% c("mpg", "mileage","name"))]) {
    plot(f, Auto, as.formula(paste("mpg~", name, sep="")))
  }
}
plotp(fit)
```

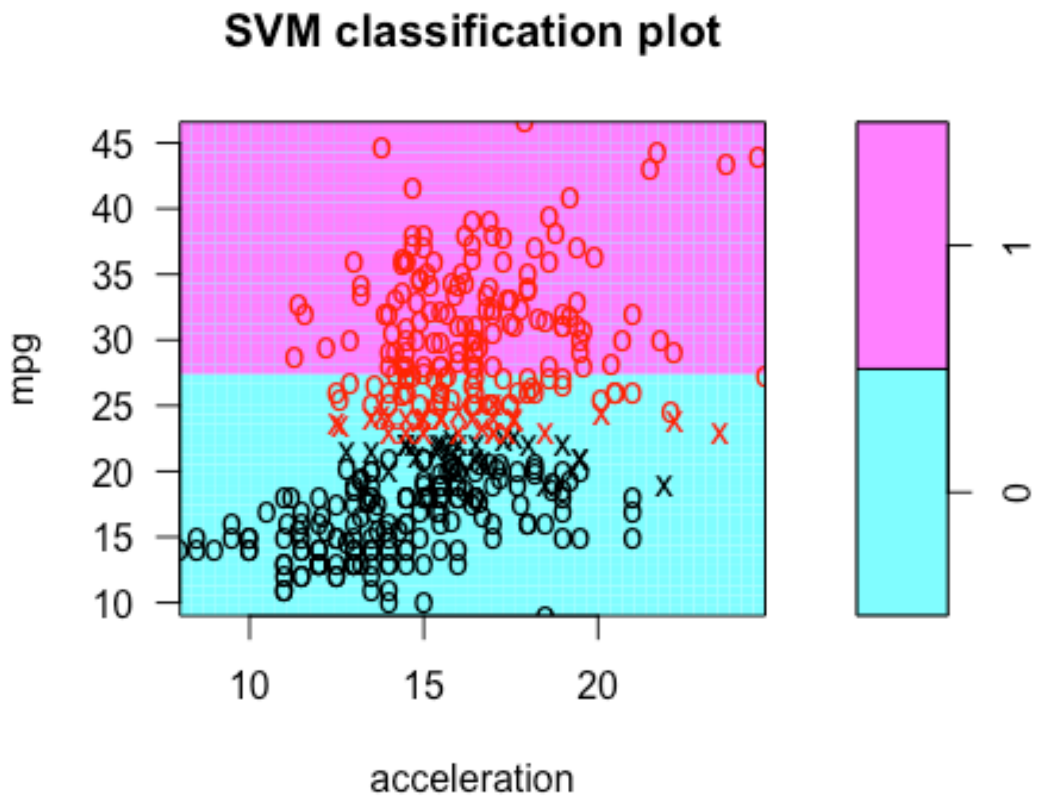

SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0

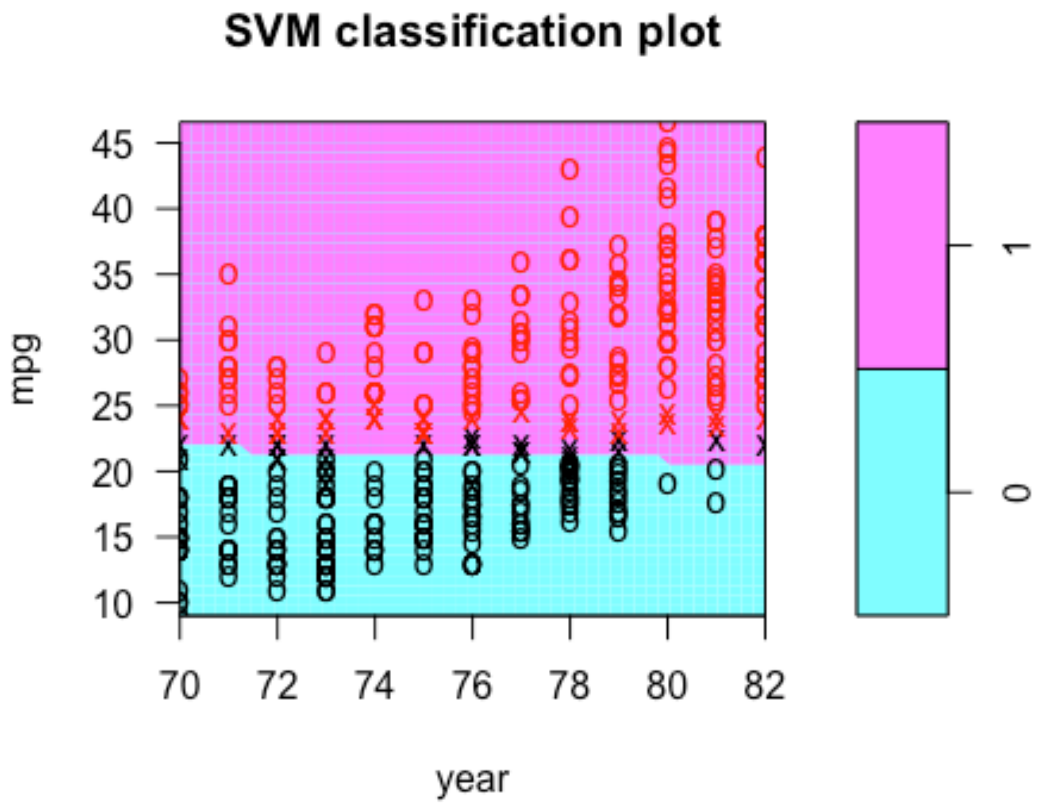

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
200
300
400
1
0

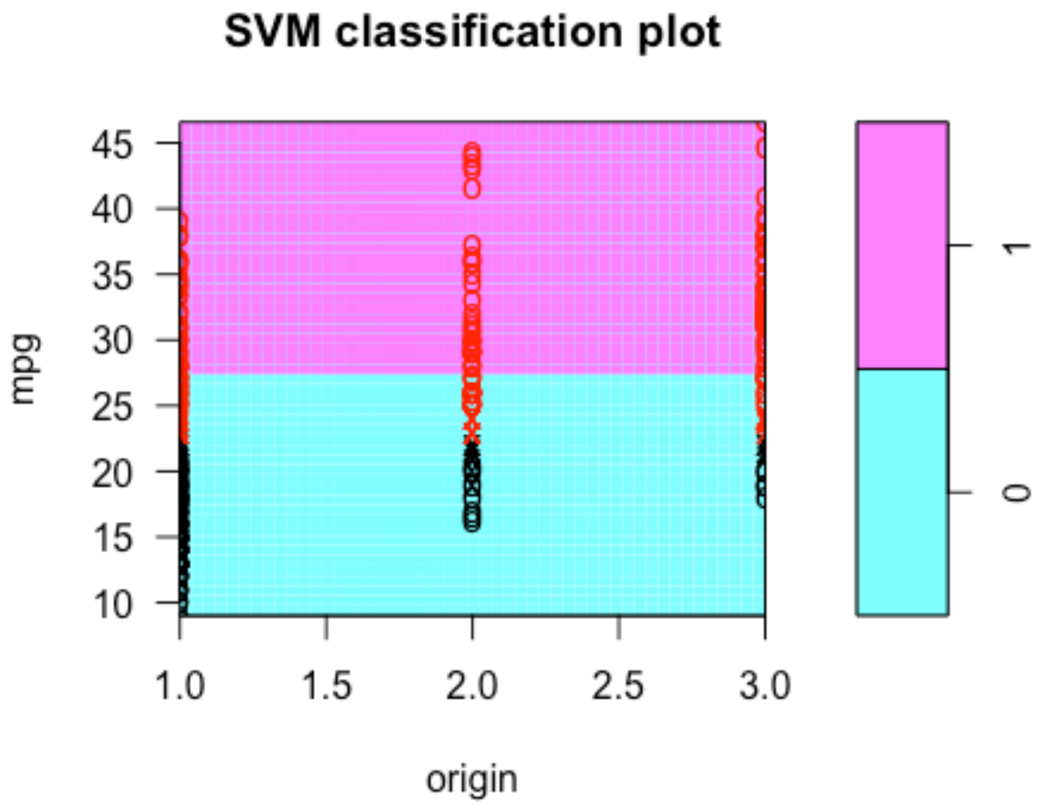

SVM classification plot
mpg
horsepower
45
40
35
30
25
20
15
10
50
100
150
200
1
0



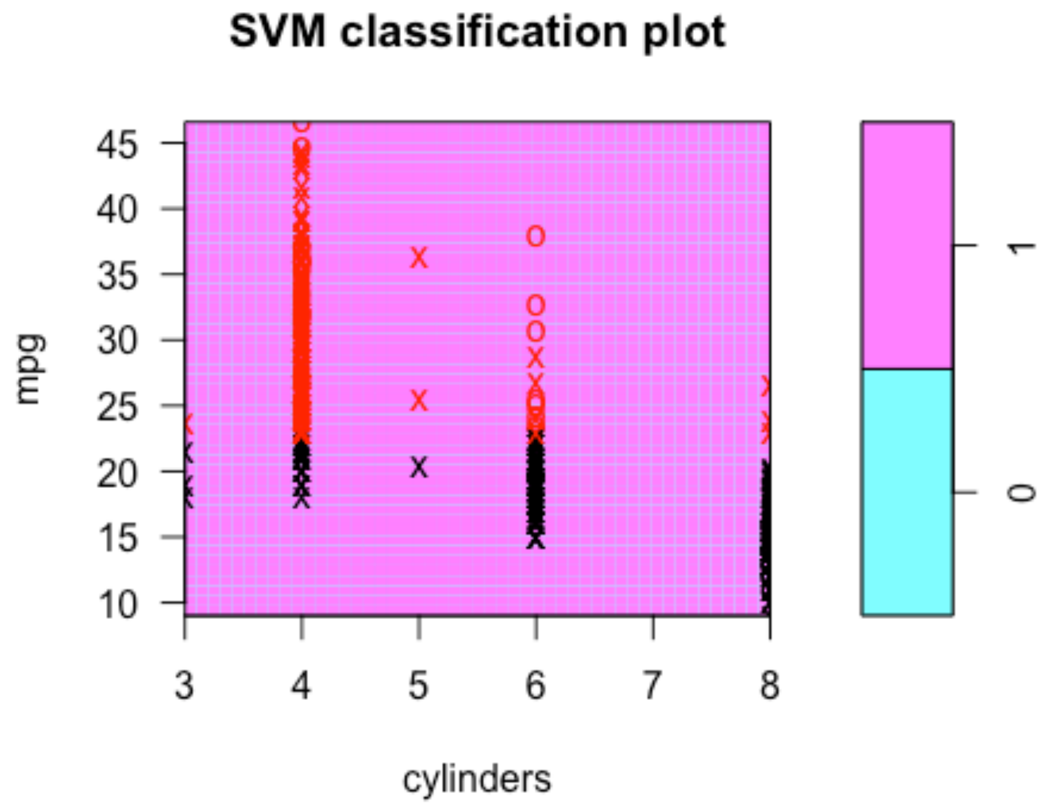
SVM classification plot
mpg
weight
45
40
35
30
25
20
15
10
2000
3000
4000
5000
1
0

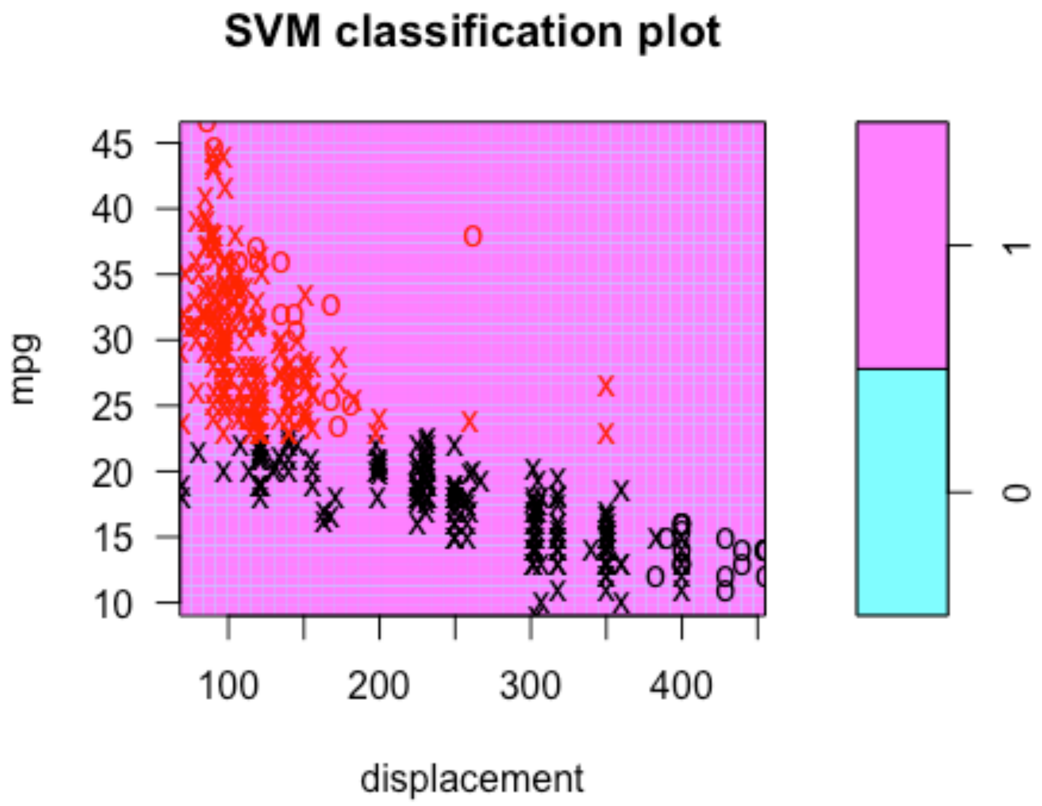

SVM classification plot
mpg
acceleration
10
15
20
25
30
35
40
45
1
0

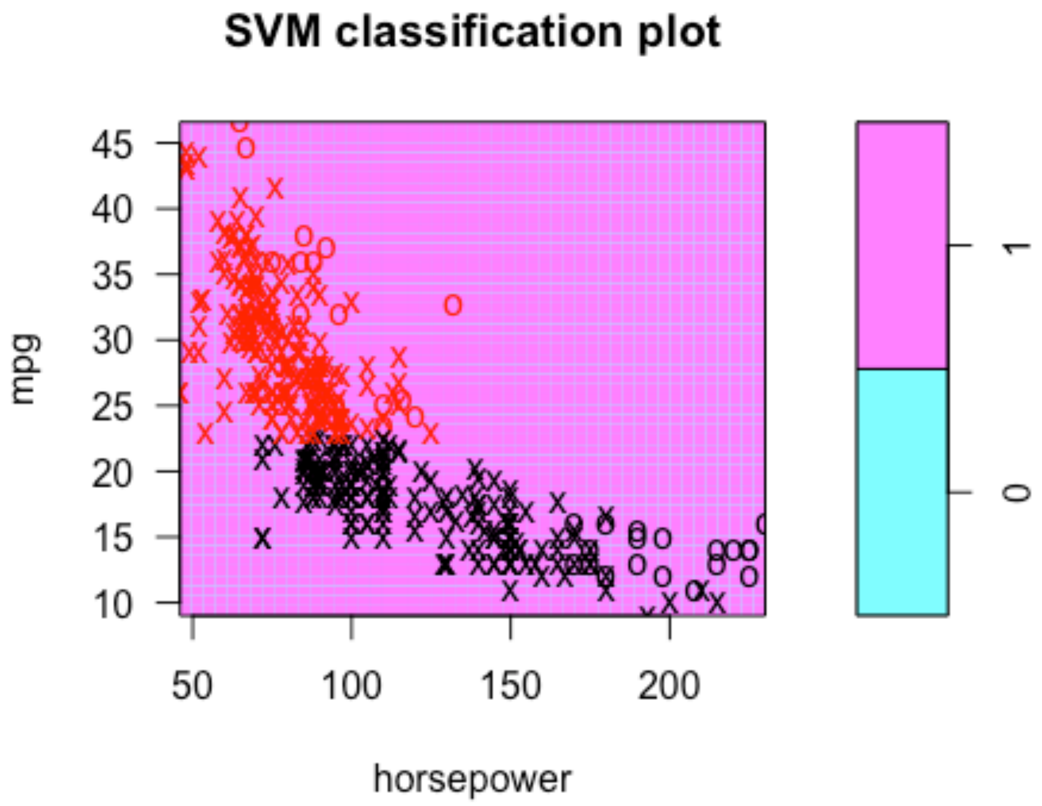

SVM classification plot
mpg
year
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
1
0

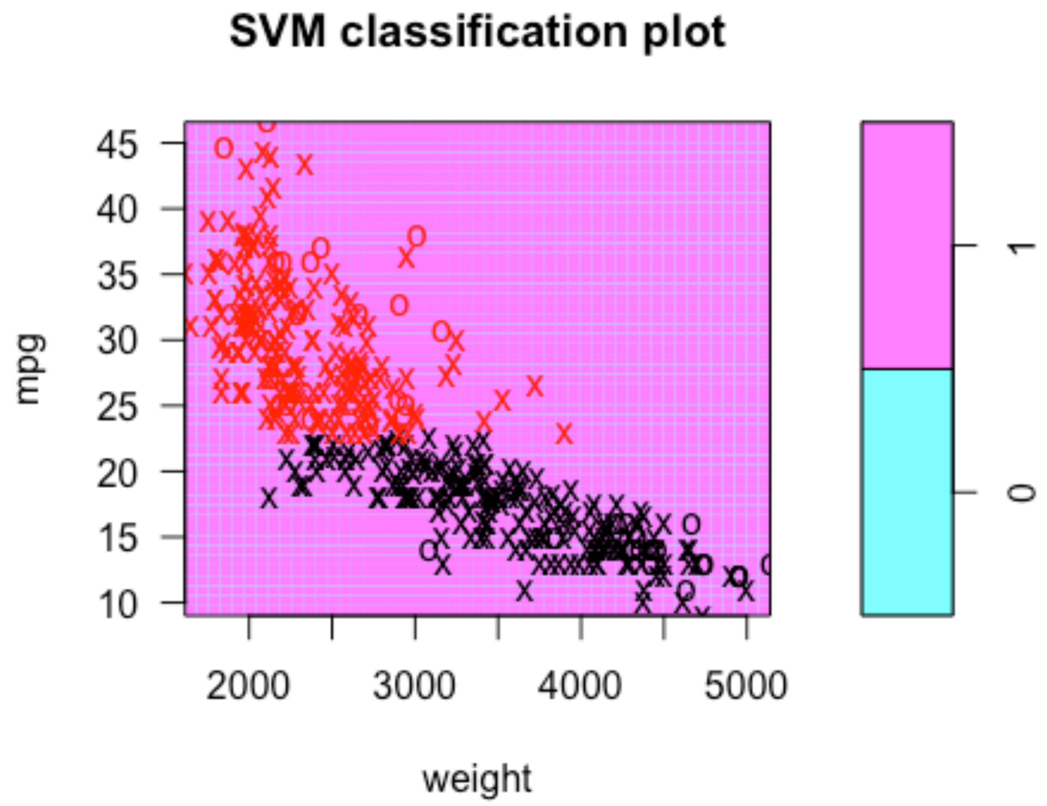

SVM classification plot
mpg
origin
45
40
35
30
25
20
15
10
1.0
1.5
2.0
2.5
3.0
1
0

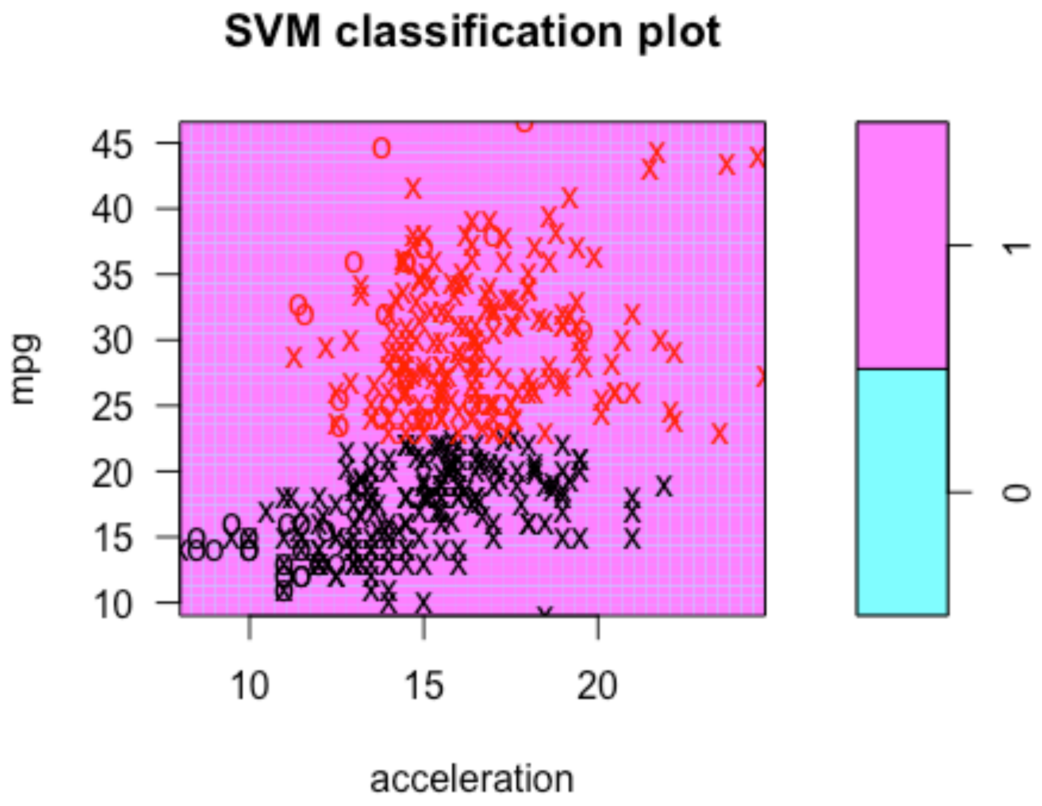
```r
plotp(fit2)
```

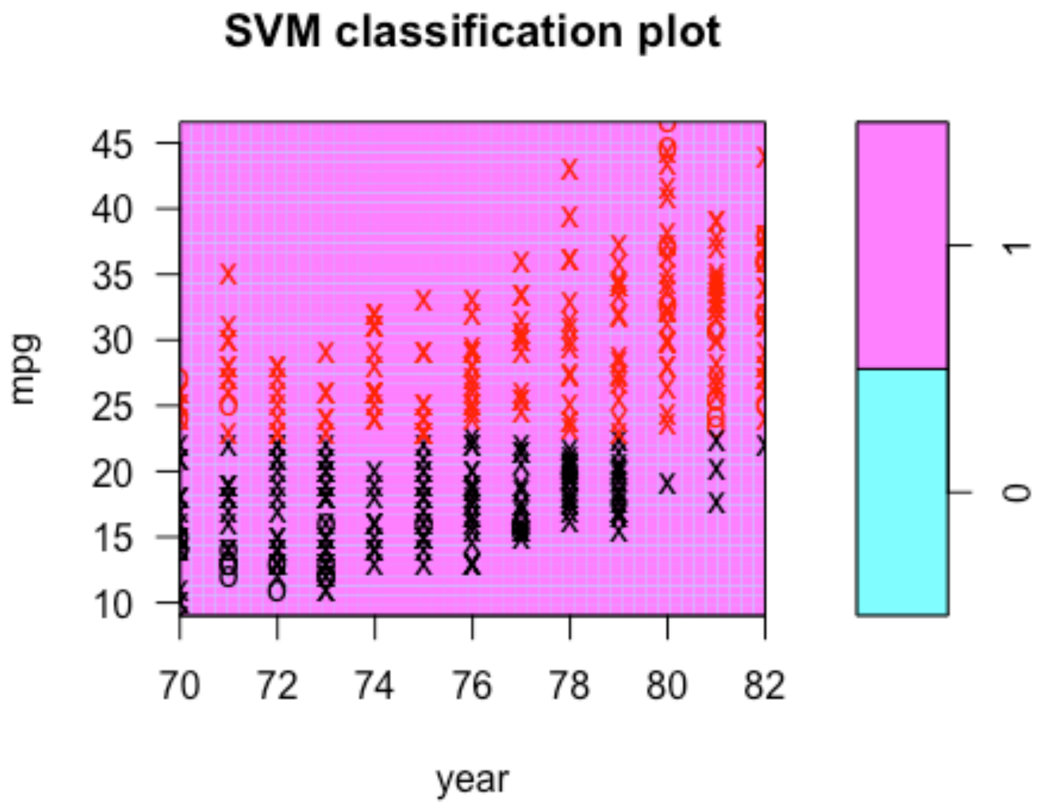

SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0

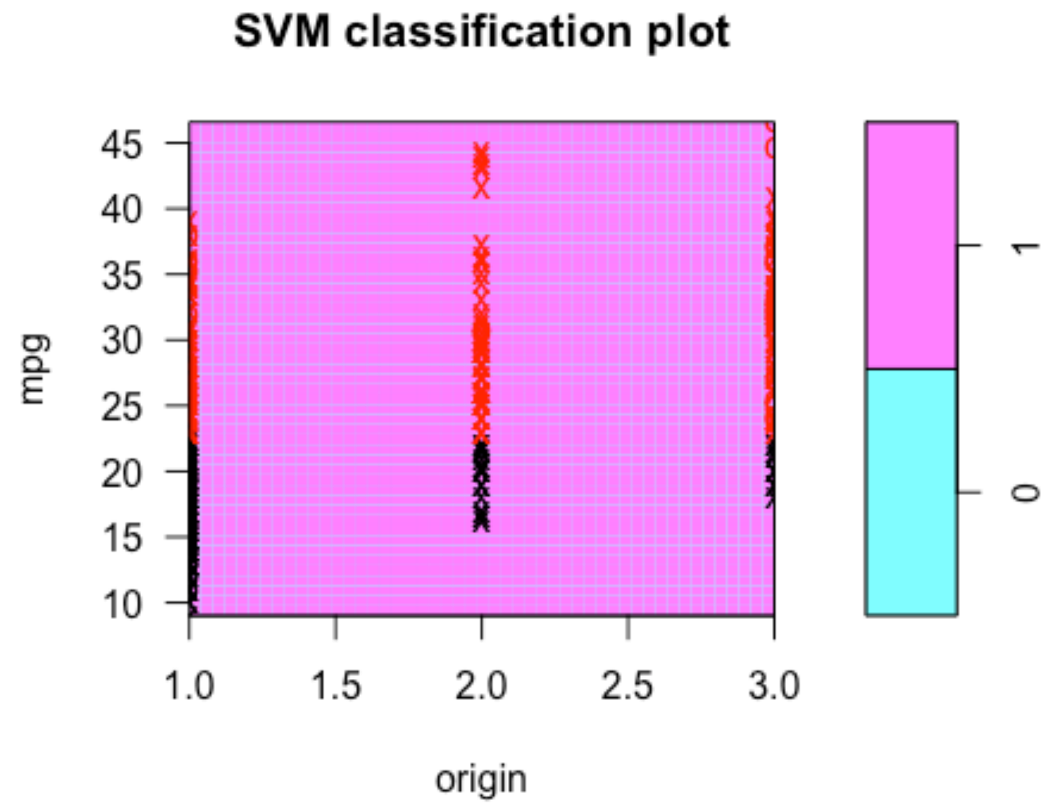

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
200
300
400
1
0


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

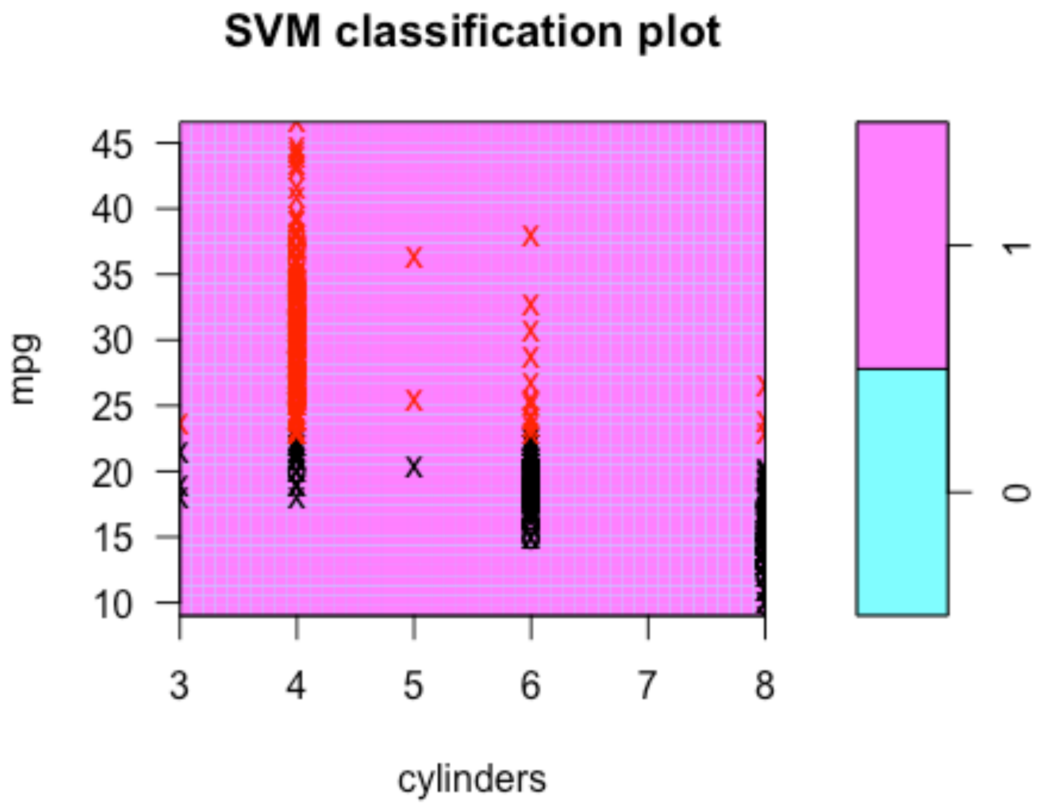

SVM classification plot
mpg
45
40
35
30
25
20
15
10
2000
3000
4000
5000
weight
1
0

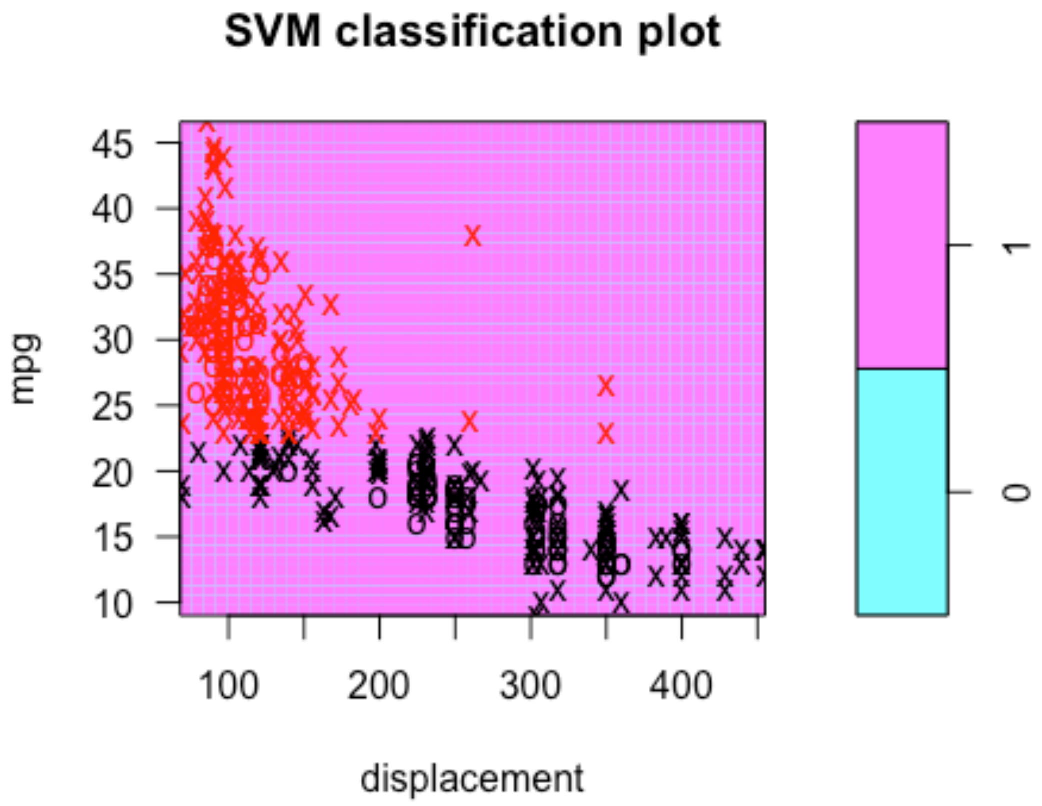

SVM classification plot
mpg
acceleration
10
15
20
25
30
35
40
45
1
0

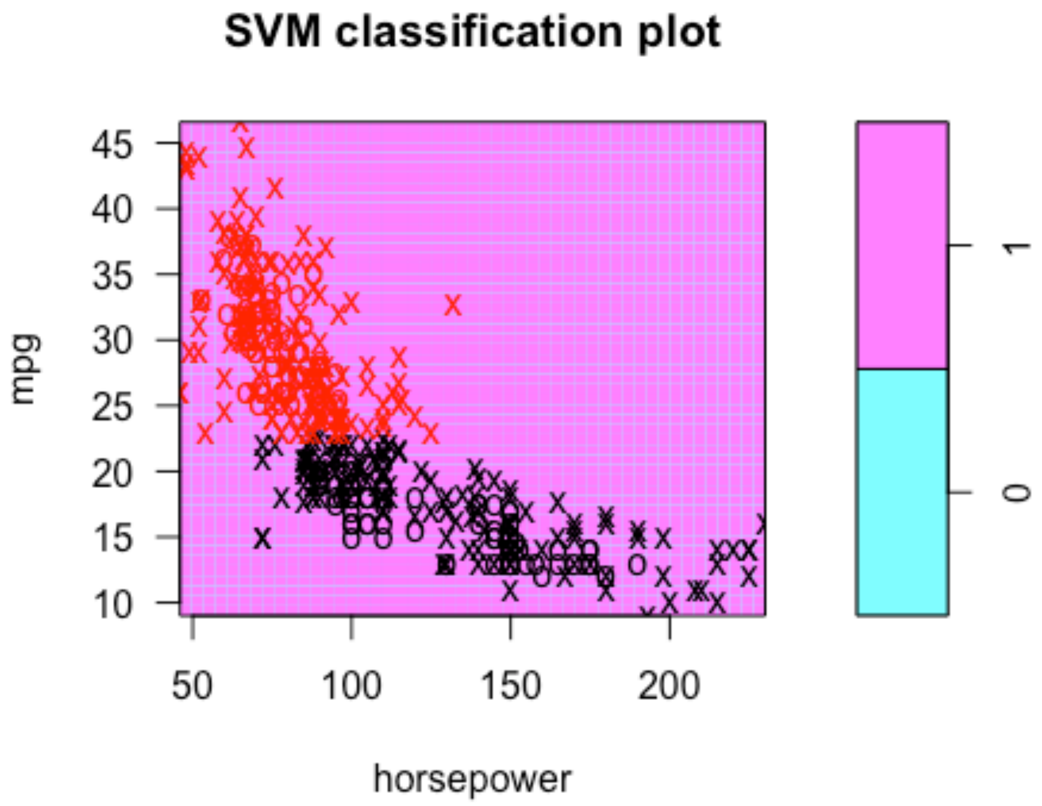

SVM classification plot
mpg
year
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
1
0

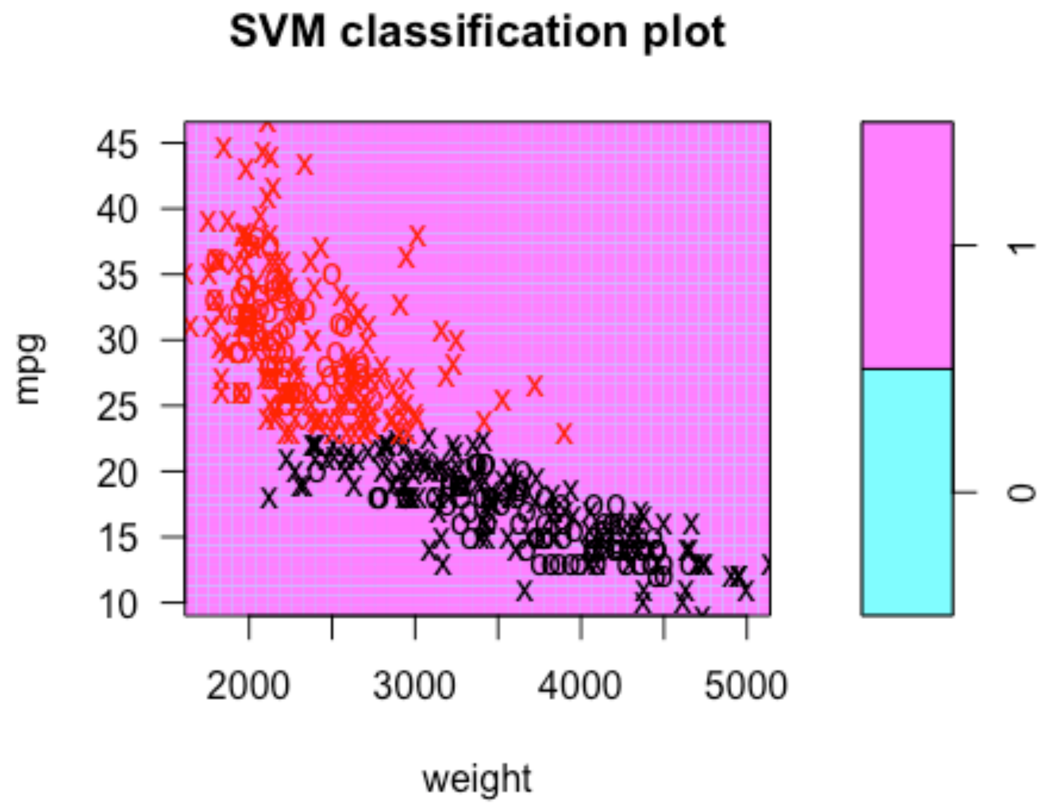

SVM classification plot
mpg
origin
45
40
35
30
25
20
15
10
1.0
1.5
2.0
2.5
3.0
1
0

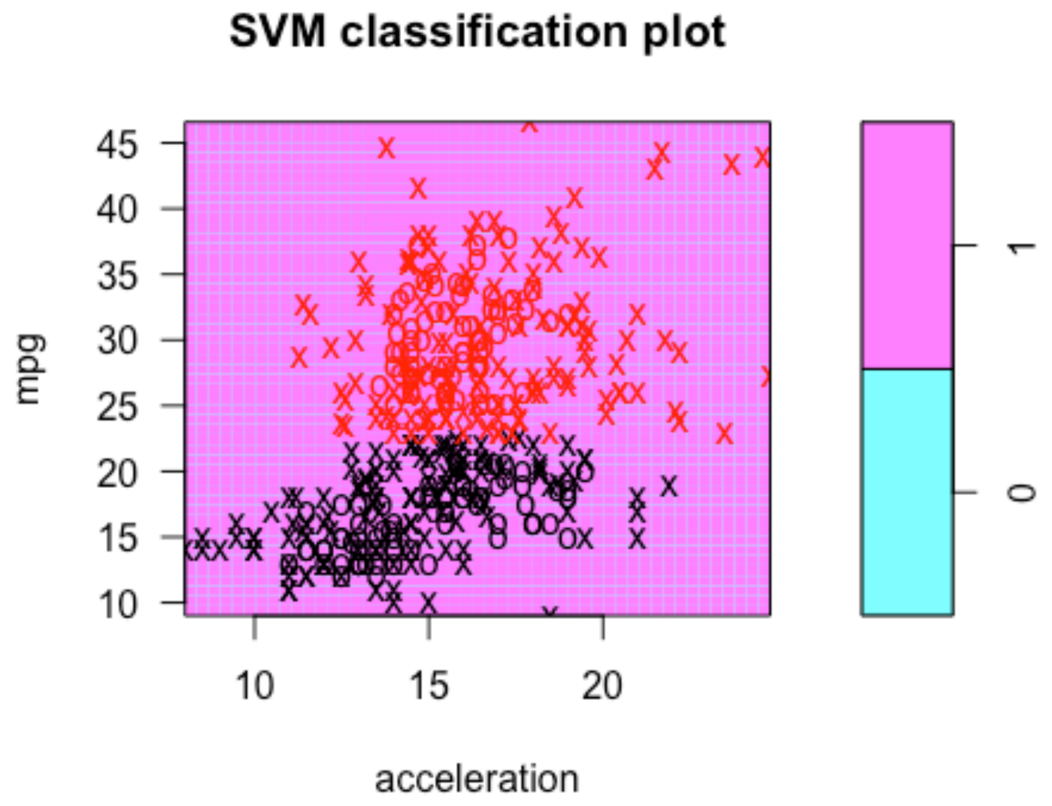
```r
plotp(fit3)
```

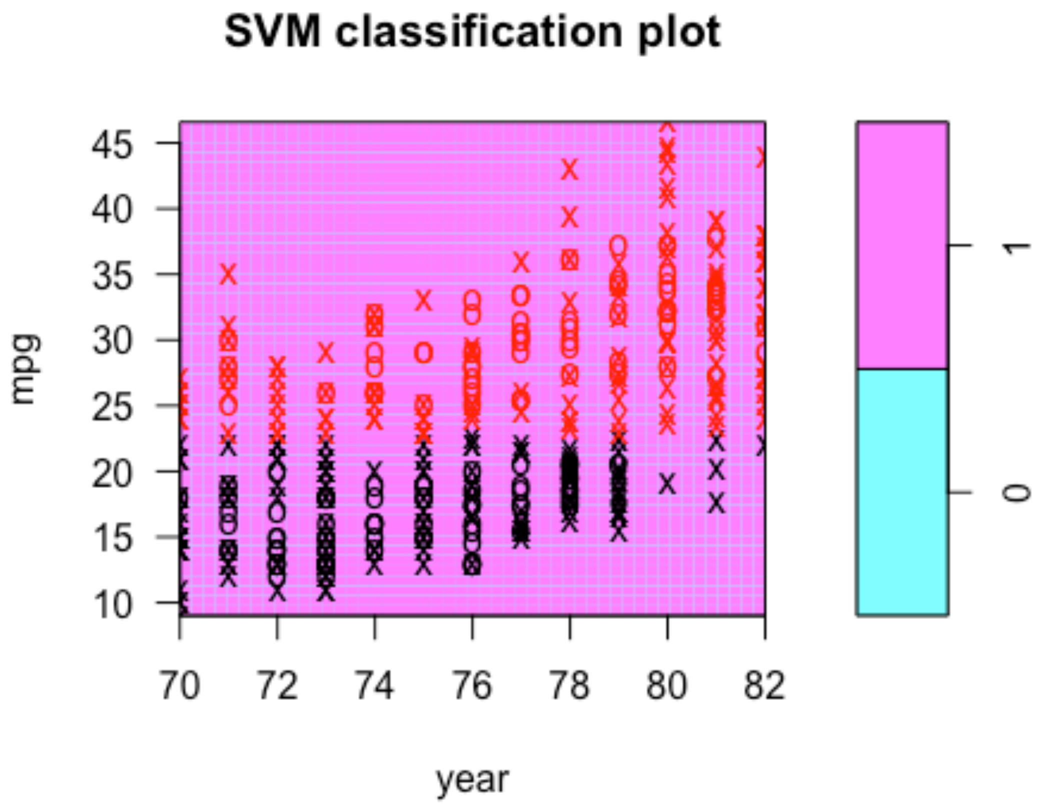

SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0

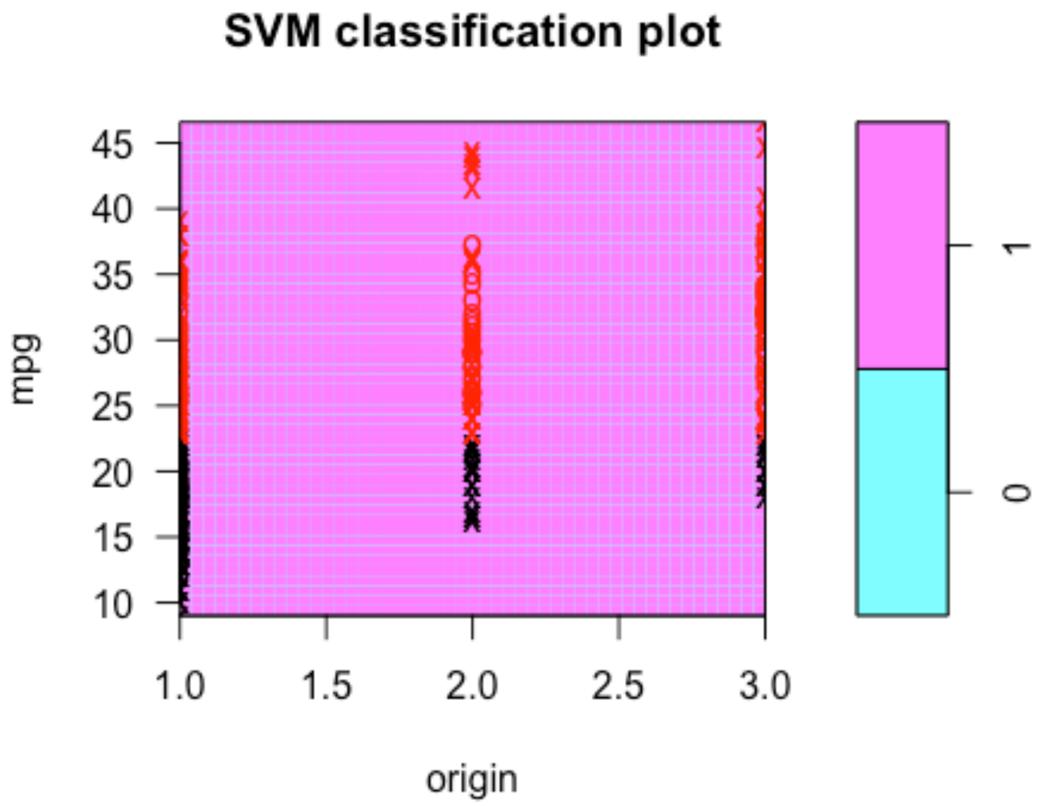

SVM classification plot
mpg
45
40
35
30
25
20
15
10
100
200
300
400
displacement
1
0


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


SVM classification plot
mpg
weight
45
40
35
30
25
20
15
10
2000
3000
4000
5000
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



```
detach(Auto)

# 8. This problem involves the OJ data set which is part of the ISLR
# package.
# 372 9. Support Vector Machines
# (a) Create a training set containing a random sample of 800
# observations, and a test set containing the remaining
# observations.
attach(OJ)
#head(OJ)
#dim(OJ)
train <- sample(nrow(OJ), 800)
trainOJ <- OJ[train, ]
testOJ <- 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.
set.seed(4)
```

```r
linearsvm <- svm(Purchase~. , data = trainOJ, kernel = "linear", cost = 0.01)
summary(linearsvm)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = trainOJ, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  448
##
##  ( 224 224 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
# At a cost of 0.01, 448 points out of 800 were used
# for creating the support vector. 224 of these belong to MM level and the ot
her 224 to the CH level.

# (c) What are the training and test error rates?
predtrain <- predict(linearsvm, trainOJ)
t <- table(predtrain, trainOJ$Purchase)
1-sum(diag(t))/800
```

```
## [1] 0.17
```

```r
##test set
predtest <- predict(linearsvm, testOJ)
t1 <- table(predtest,testOJ$Purchase)
1-sum(diag(t1))/270
```

```
## [1] 0.1518519
```

```r
# (d) Use the tune() function to select an optimal cost. Consider values
# in the range 0.01 to 10.
set.seed(3)
tunelinear1 <- tune(svm,Purchase~. , data=trainOJ, kernel="linear",ranges=lis
t(cost=c(0.01, 0.05, 0.1, 0.25, 0.5,0.75,

1,5,10)))
summary(tunelinear1)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.1725
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.17625 0.03458584
## 2  0.05 0.17250 0.03717451
## 3  0.10 0.17250 0.03717451
## 4  0.25 0.17500 0.04124790
## 5  0.50 0.17625 0.03928617
## 6  0.75 0.17625 0.04101575
## 7  1.00 0.17750 0.03987829
## 8  5.00 0.17625 0.03557562
## 9 10.00 0.18000 0.03917553
```

```
# (e) Compute the training and test error rates using this new value
# for cost.

tunedsvm <- svm(Purchase~. , data = trainOJ, kernel = "linear", cost = 10)
pred1 <- predict(tunedsvm, trainOJ)
t <- table(pred1, trainOJ$Purchase)
1-sum(diag(t))/800
```

```
## [1] 0.1625
```

```
tunedsvm <- svm(Purchase~. , data = testOJ, kernel = "linear", cost = 10)
pred1 <- predict(tunedsvm, testOJ)
t <- table(pred1, testOJ$Purchase)
1-sum(diag(t))/270
```

```
## [1] 0.1259259
```

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


set.seed(4)
radialsvm <- svm(Purchase~. , data = trainOJ, kernel = "radial", cost = 0.01)
summary(radialsvm)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = trainOJ, kernel = "radial",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  653
##
##  ( 325 328 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# At a cost of 0.01, 653 points out of 800 were used
# for creating the support vector. 328 of these belong to MM level and the ot
her 325 to the CH level.

#training and test error rates?
pred2 <- predict(radialsvm, trainOJ)
t <- table(pred2, trainOJ$Purchase)
1-sum(diag(t))/800
```

```
## [1] 0.40625
```

```
##test set
predtest <- predict(radialsvm, testOJ)
t1 <- table(predtest,testOJ$Purchase)
1-sum(diag(t1))/270
```

```
## [1] 0.3407407
```

```
set.seed(3)
tuneradial1 <- tune(svm,Purchase~. , data=trainOJ, kernel="radial",ranges=lis
t(cost=c(0.01, 0.05, 0.1, 0.25, 0.5,0.75,

1,5,10)))
summary(tuneradial1)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
```

```
##
## - best performance: 0.17375
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.40625 0.05408648
## 2  0.05 0.21250 0.03952847
## 3  0.10 0.18500 0.04556741
## 4  0.25 0.18375 0.04896498
## 5  0.50 0.17875 0.04168749
## 6  0.75 0.17750 0.04158325
## 7  1.00 0.17375 0.04059026
## 8  5.00 0.20000 0.04677072
## 9 10.00 0.19500 0.04571956
```

```
tunedsvm <- svm(Purchase~. , data = trainOJ, kernel = "radial", cost = 1)
pred1 <- predict(tunedsvm, trainOJ)
t <- table(pred1, trainOJ$Purchase)
1-sum(diag(t))/800
```

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

```
pred1 <- predict(tunedsvm, testOJ)
t <- table(pred1, testOJ$Purchase)
1-sum(diag(t))/270
```

```
## [1] 0.1518519
```

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

set.seed(5)
polysvm <- svm(Purchase~. , data = trainOJ, kernel = "polynomial", degree = 2
)
summary(polysvm)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = trainOJ, 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:  463
##
```

```
##  ( 229 234 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# At a cost of 1 and degree of 2, 463 points out of 800 were used
# for creating the support vector. 234 of these belong to MM level and the ot
her 229 to the CH level.

#training and test error rates?
pred4 <- predict(polysvm, trainOJ)
t <- table(pred4, trainOJ$Purchase)
1-sum(diag(t))/800
```

```
## [1] 0.19
```

```
##test set
predtest <- predict(polysvm, testOJ)
t1 <- table(predtest,testOJ$Purchase)
1-sum(diag(t1))/270
```

```
## [1] 0.1666667
```

```
set.seed(3)
tunepoly1 <- tune(svm,Purchase~. , data=trainOJ, kernel="polynomial",degree =
2,ranges=list(cost=c(0.01, 0.05, 0.1, 0.25, 0.5,0.75,

1,5,10)))
summary(tunepoly1)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1825
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.40500 0.05749396
## 2  0.05 0.34625 0.05104804
## 3  0.10 0.33250 0.05407043
## 4  0.25 0.22875 0.03537988
## 5  0.50 0.21500 0.03574602
## 6  0.75 0.20875 0.03283481
## 7  1.00 0.20625 0.03448530
## 8  5.00 0.18875 0.03793727
## 9 10.00 0.18250 0.03238227
```

```
tunedsvm <- svm(Purchase~. , data = trainOJ, kernel = "polynomial", cost = 10
, degree = 2)
pred1 <- predict(tunedsvm, trainOJ)
t <- table(pred1, trainOJ$Purchase)
1-sum(diag(t))/800
```

```
## [1] 0.15625
```

```
pred1 <- predict(tunedsvm, testOJ)
t <- table(pred1, testOJ$Purchase)
1-sum(diag(t))/270
```

```
## [1] 0.1481481
```

```
# (h) Overall, which approach seems to give the best results on this data?
# My lowest test error comes from the linear kernel but overall they all seem
to perform quite well
# with error rates around 16%.
```