ML-p4

Vincent Jin

2023-03-08

Homework 04

Vincent Jin

Person I worked with: Jing Wang

Support Vector Machines ISL: 9.7. Problems 7, 8 ((a)-(e) only)

Tree-based Methods ISL: 8.4. Problem 10

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Chapter 9 Problem 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)

auto <- Auto

mpg_median <- median(auto$mpg)

auto <- auto %>%
  mutate(mpg_bi = as.factor(ifelse(mpg > mpg_median, 1, 0)))
```

(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.

```
library(e1071)
set.seed(1)
tune.out <- tune(svm, mpg_bi ~ ., data = auto,kernel = "linear", ranges = list(cost = c(0.001, 0.01, 0.
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.01025641
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.09442308 0.04519425
## 2 1e-02 0.07653846 0.03617137
## 3 1e-01 0.04596154 0.03378238
## 4 1e+00 0.01025641 0.01792836
## 5 5e+00 0.02051282 0.02648194
## 6 1e+01 0.02051282 0.02648194
## 7 1e+02 0.03076923 0.03151981
```

Answer

Since cost = 1.0 the error is the lowest (0.0715), cost = 1 fits the best for linear kernel.

(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.out1 <- tune(svm, mpg_bi ~ ., data = auto, kernel = "polynomial", ranges = list(cost = c(0.01, 0.1
tune.out2 <- tune(svm, mpg_bi ~ ., data = auto, kernel = "radial", ranges = list(cost = c(0.01, 0.1, 1,
summary(tune.out1)
```

```
##
## 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 0.5511538 0.04366593
## 2  1e-01      2 0.5511538 0.04366593
## 3  1e+00      2 0.5511538 0.04366593
## 4  5e+00      2 0.5511538 0.04366593
## 5  1e+01      2 0.5130128 0.08963366
## 6  1e+02      2 0.3013462 0.09961961
## 7  1e-02      3 0.5511538 0.04366593
## 8  1e-01      3 0.5511538 0.04366593
## 9  1e+00      3 0.5511538 0.04366593
## 10 5e+00      3 0.5511538 0.04366593
## 11 1e+01      3 0.5511538 0.04366593
## 12 1e+02      3 0.3446154 0.09821588
## 13 1e-02      4 0.5511538 0.04366593
## 14 1e-01      4 0.5511538 0.04366593
## 15 1e+00      4 0.5511538 0.04366593
## 16 5e+00      4 0.5511538 0.04366593
## 17 1e+01      4 0.5511538 0.04366593
## 18 1e+02      4 0.5511538 0.04366593
```

```
summary(tune.out2)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100  0.01
##
## - best performance: 0.01019231
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-02 1e-02 0.60192308 0.06346118
## 2  1e-01 1e-02 0.08653846 0.06488131
## 3  1e+00 1e-02 0.07134615 0.04769894
## 4  5e+00 1e-02 0.04846154 0.03905899
## 5  1e+01 1e-02 0.02294872 0.02534336
## 6  1e+02 1e-02 0.01019231 0.01786828
## 7  1e-02 1e-01 0.23974359 0.06938360
## 8  1e-01 1e-01 0.07891026 0.05147085
## 9  1e+00 1e-01 0.05096154 0.03995812
## 10 5e+00 1e-01 0.02301282 0.03069264
## 11 1e+01 1e-01 0.02294872 0.02807826
## 12 1e+02 1e-01 0.03064103 0.02651089
## 13 1e-02 1e+00 0.60192308 0.06346118
## 14 1e-01 1e+00 0.60192308 0.06346118
## 15 1e+00 1e+00 0.06365385 0.04845299
```

```
## 16 5e+00 1e+00 0.06121795 0.04387918
## 17 1e+01 1e+00 0.06121795 0.04387918
## 18 1e+02 1e+00 0.06121795 0.04387918
## 19 1e-02 5e+00 0.60192308 0.06346118
## 20 1e-01 5e+00 0.60192308 0.06346118
## 21 1e+00 5e+00 0.52814103 0.08413728
## 22 5e+00 5e+00 0.52814103 0.08238251
## 23 1e+01 5e+00 0.52814103 0.08238251
## 24 1e+02 5e+00 0.52814103 0.08238251
## 25 1e-02 1e+01 0.60192308 0.06346118
## 26 1e-01 1e+01 0.60192308 0.06346118
## 27 1e+00 1e+01 0.55615385 0.07526477
## 28 5e+00 1e+01 0.55358974 0.07343728
## 29 1e+01 1e+01 0.55358974 0.07343728
## 30 1e+02 1e+01 0.55358974 0.07343728
## 31 1e-02 1e+02 0.60192308 0.06346118
## 32 1e-01 1e+02 0.60192308 0.06346118
## 33 1e+00 1e+02 0.60192308 0.06346118
## 34 5e+00 1e+02 0.60192308 0.06346118
## 35 1e+01 1e+02 0.60192308 0.06346118
## 36 1e+02 1e+02 0.60192308 0.06346118
```

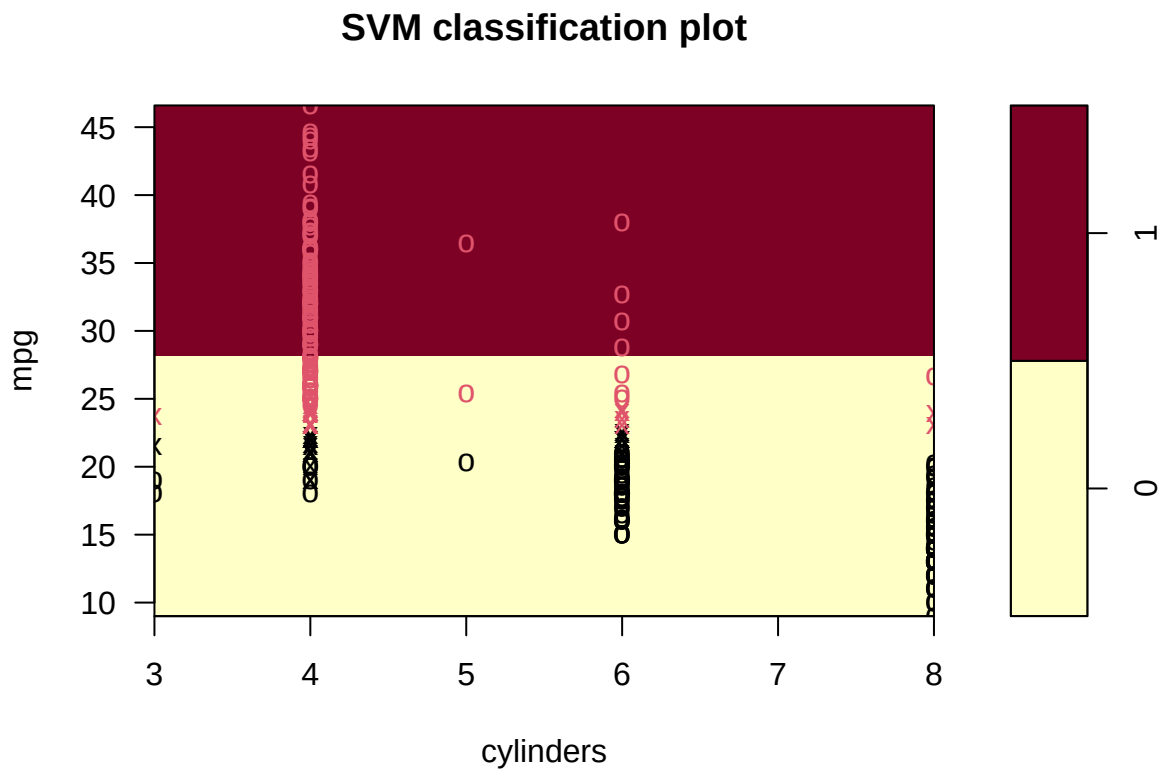
Answer

Using SVMs with polynomial basis kernel, cost = 100 with degree of 2 fits the best as suggested by lowest error. For radial, it was also cost = 100 with gamma = 0.01.

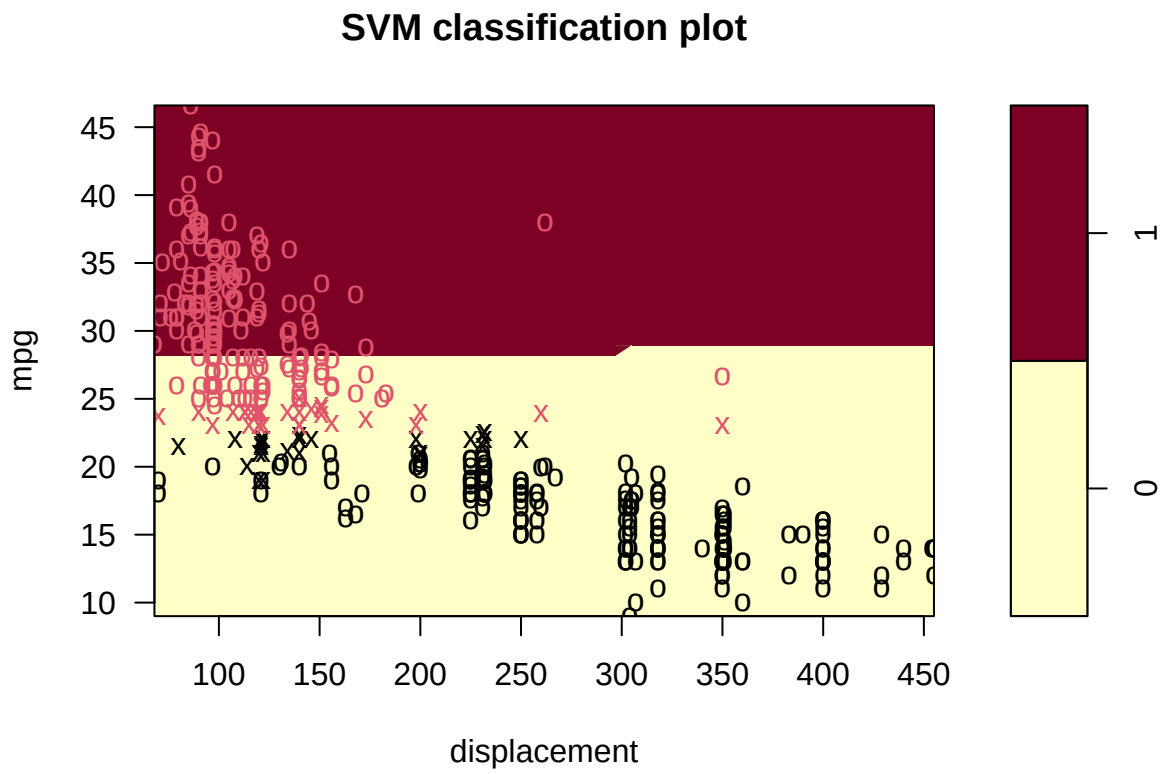
(d) Make some plots to back up your assertions in (b) and (c).

```
svmlinear <- svm(mpg_bi ~ ., data = auto, kernel = "linear", cost = 1)
svmpoly <- svm(mpg_bi ~ ., data = auto, kernel = "polynomial", cost = 100, degree = 2)
svmradial <- svm(mpg_bi ~ ., data = auto, kernel = "radial", cost = 100, gamma = 0.01)
plotpairs <- function(fit) {
    for (name in names(auto)[!(names(auto) %in% c("mpg", "mpg_bi", "name"))]) {
      temp = as.formula(paste("mpg~", name, sep = ""))
      print(temp)
      plot(fit, auto, temp)
    }
}
print(plotpairs(svmlinear))
```

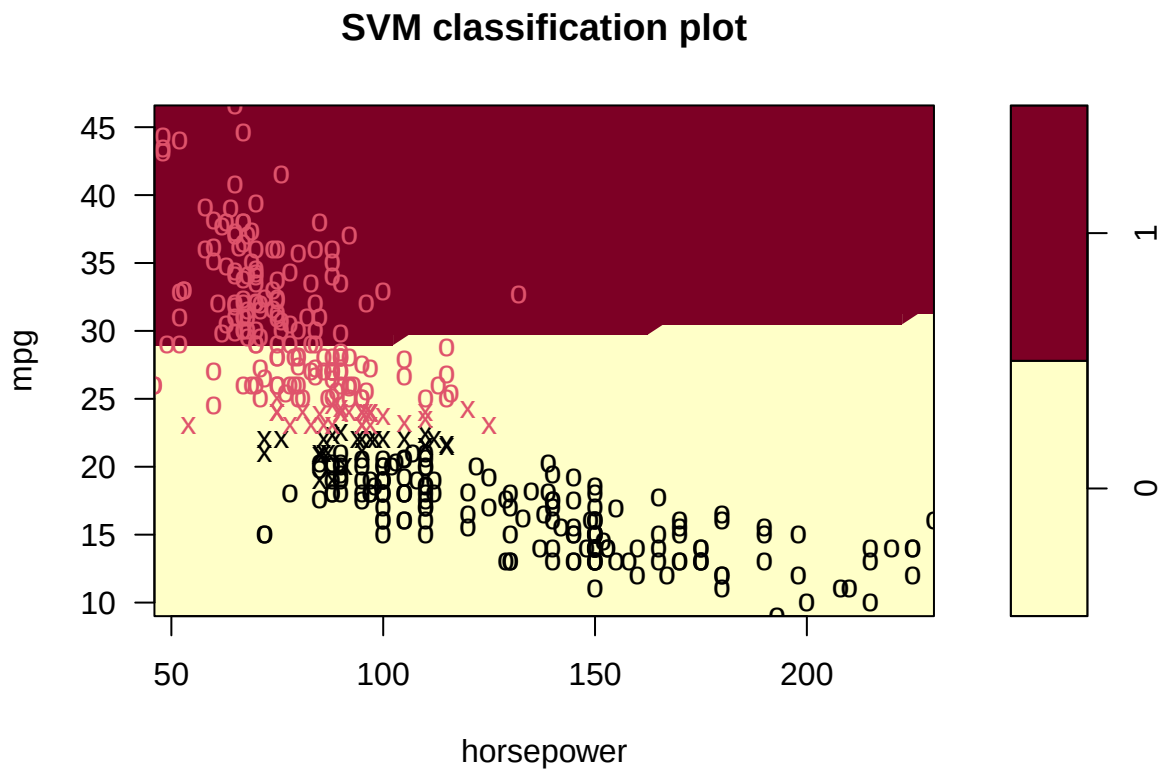
```
## mpg ~ cylinders
## <environment: 0x0000000015a7f2d8>
```



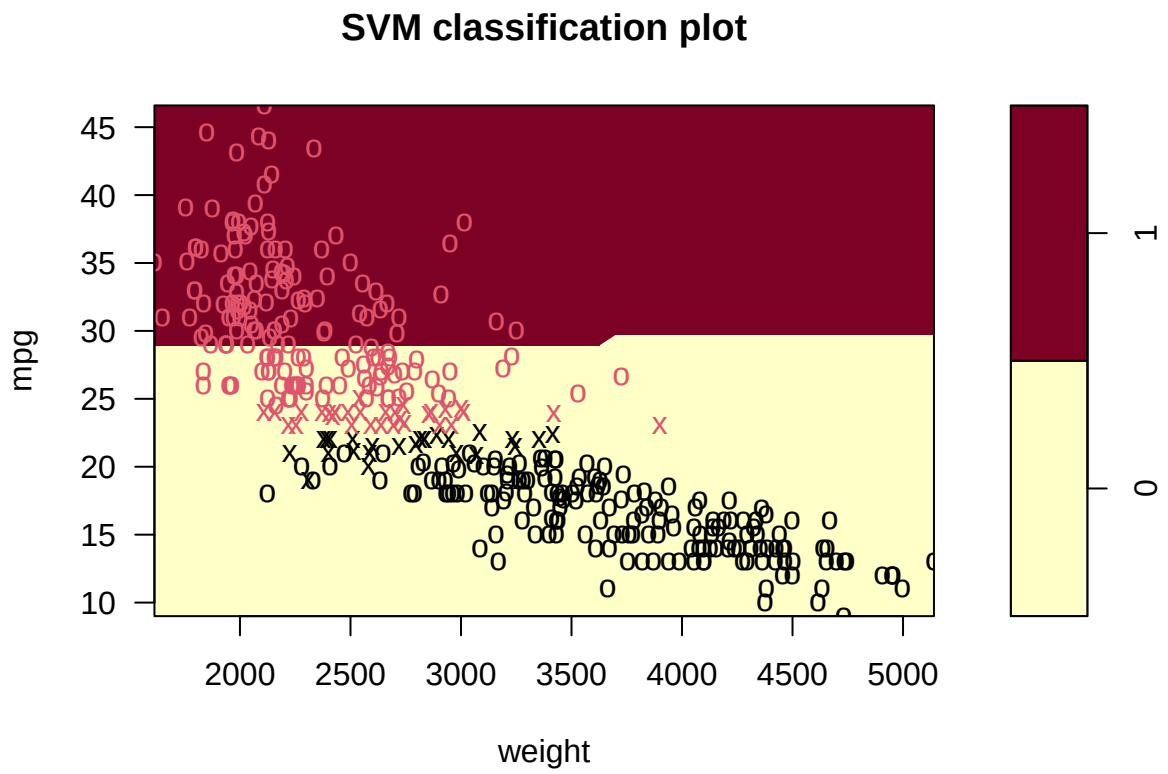

```
## mpg ~ displacement
## <environment: 0x0000000015a7f2d8>
```



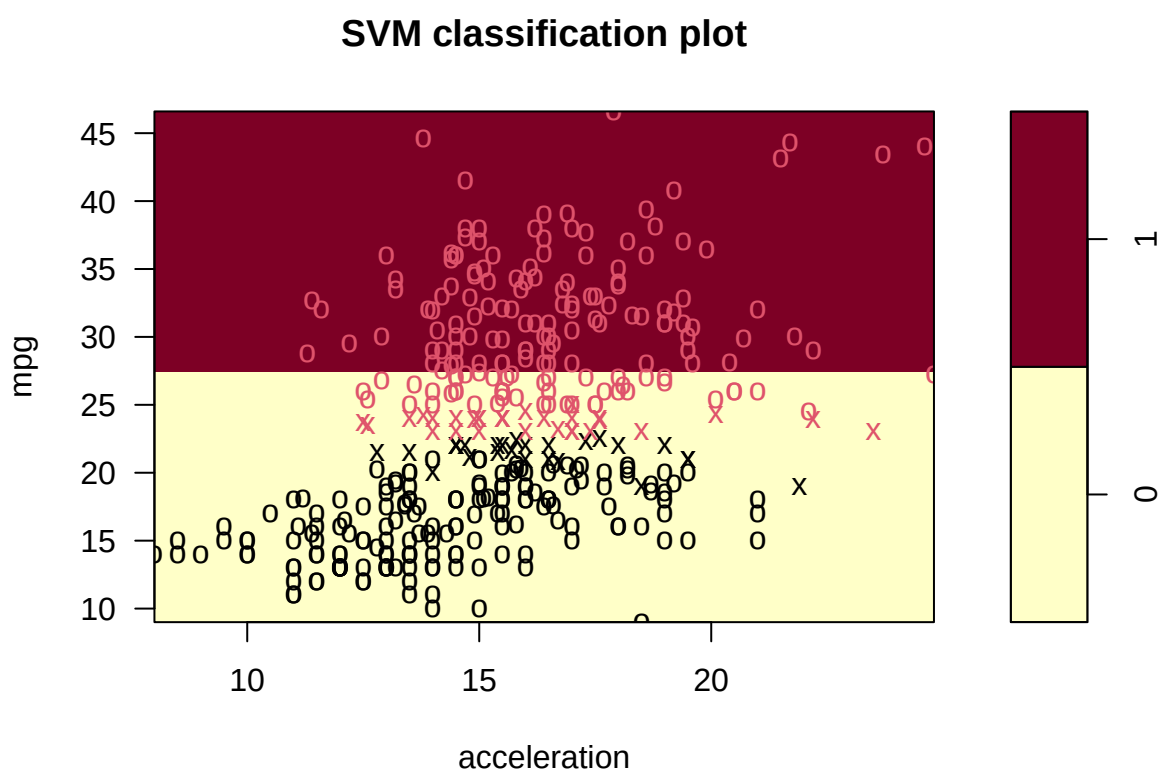

```
## mpg ~ horsepower
## <environment: 0x0000000015a7f2d8>
```



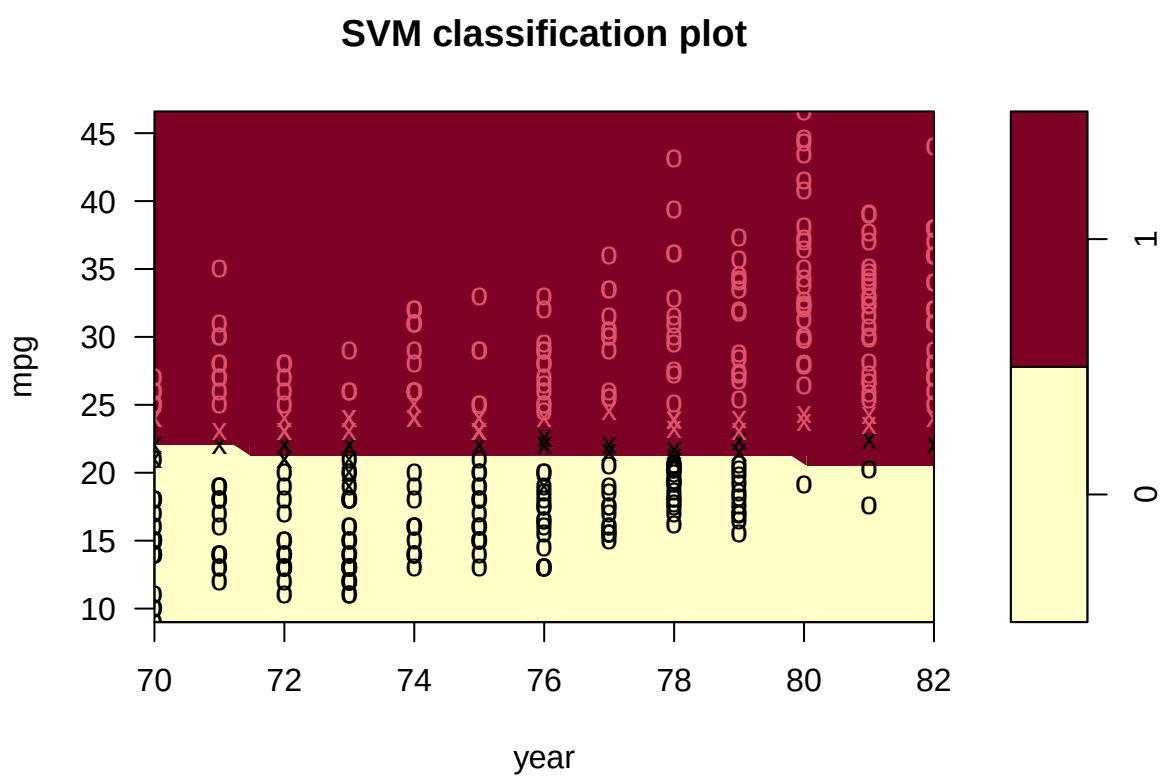

```
## mpg ~ weight
## <environment: 0x0000000015a7f2d8>
```



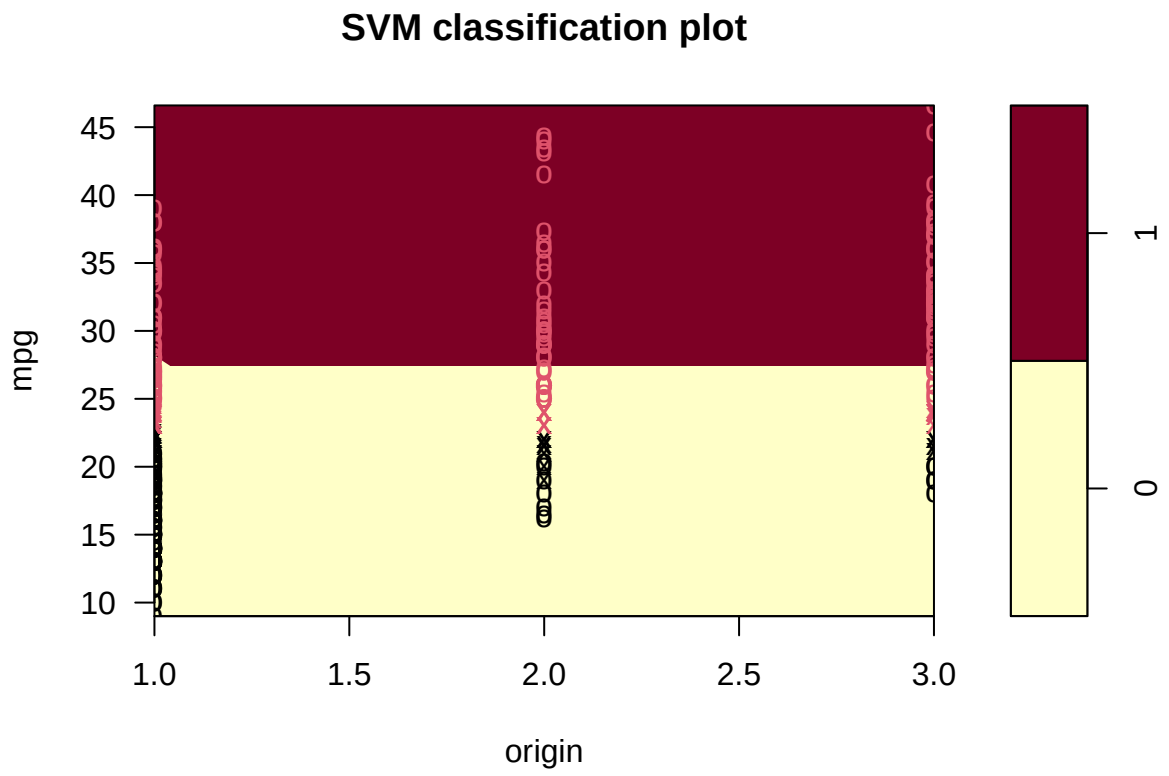

```
## mpg ~ acceleration
## <environment: 0x0000000015a7f2d8>
```




```
## mpg ~ year
## <environment: 0x0000000015a7f2d8>
```



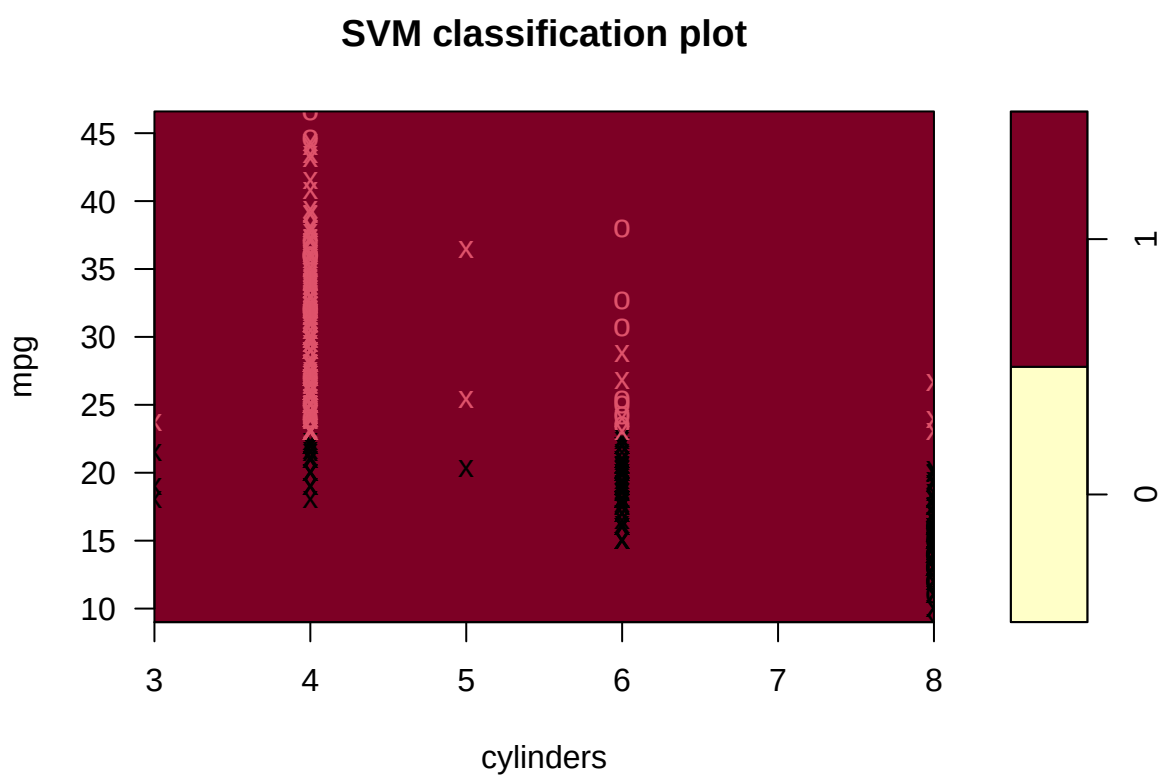

```
## mpg ~ origin
## <environment: 0x0000000015a7f2d8>
```



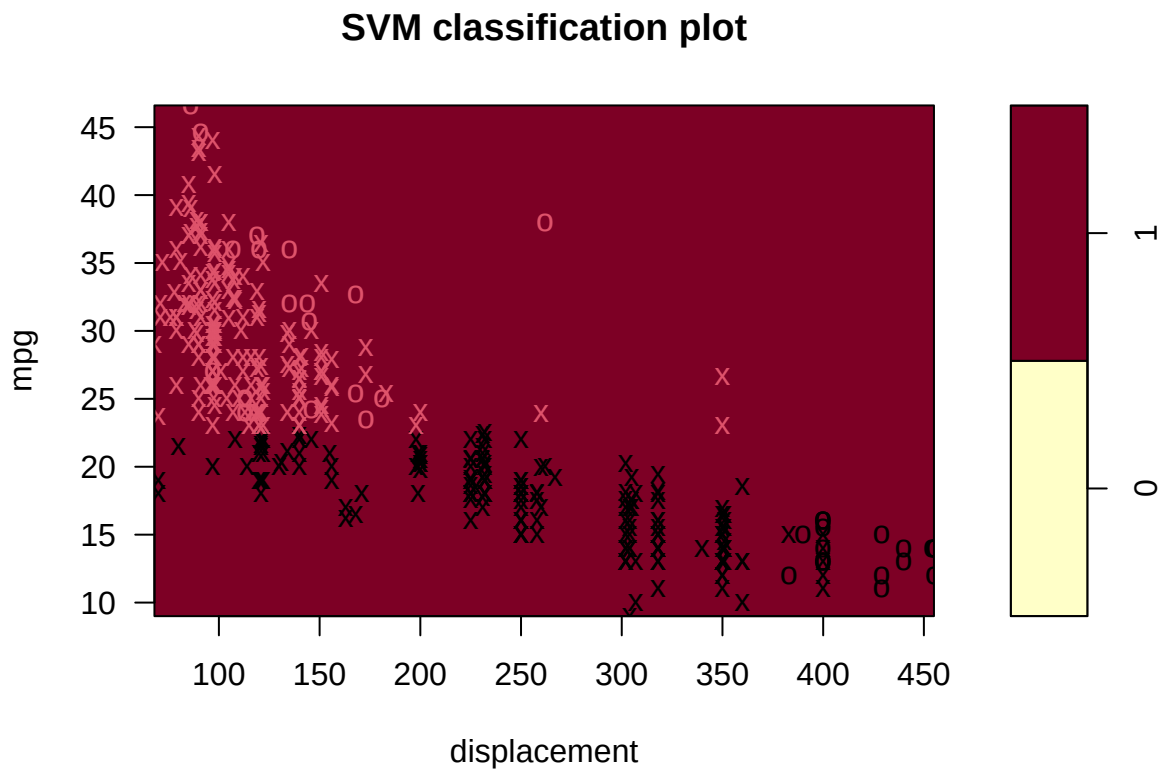

```
## NULL
```

```
print(plotpairs(svmpoly))
```

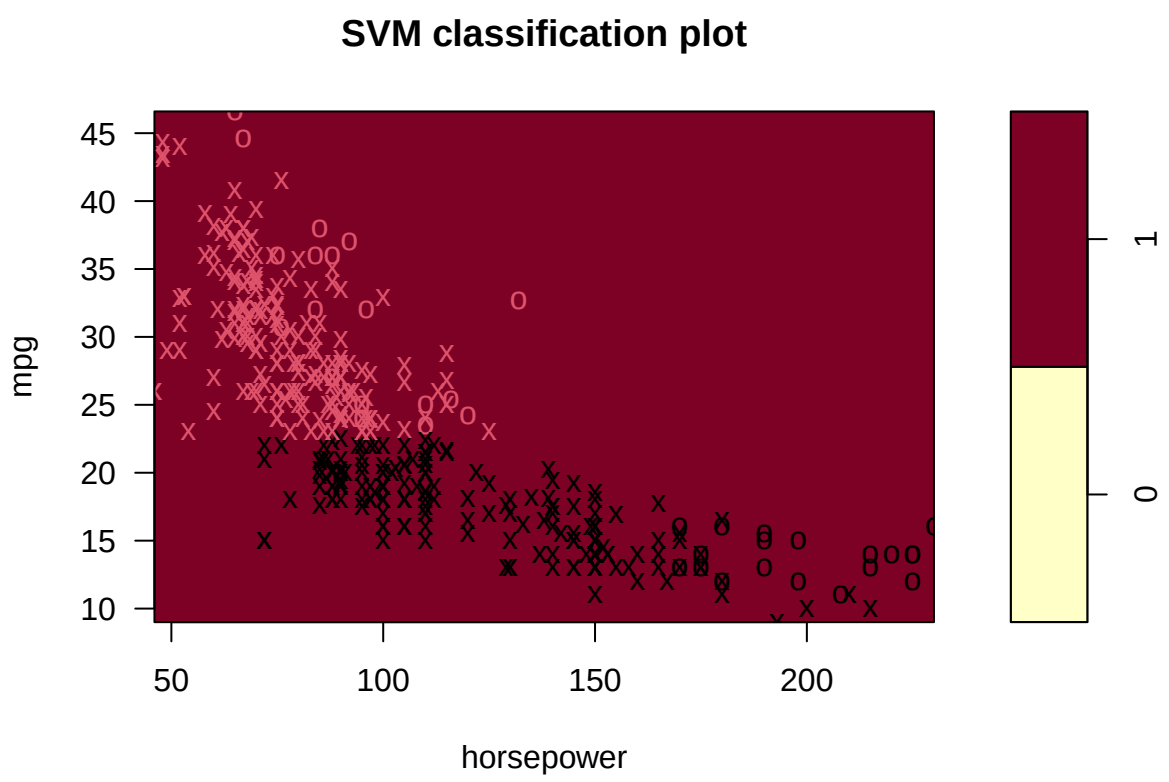
```
## mpg ~ cylinders
## <environment: 0x0000000021eeff80>
```



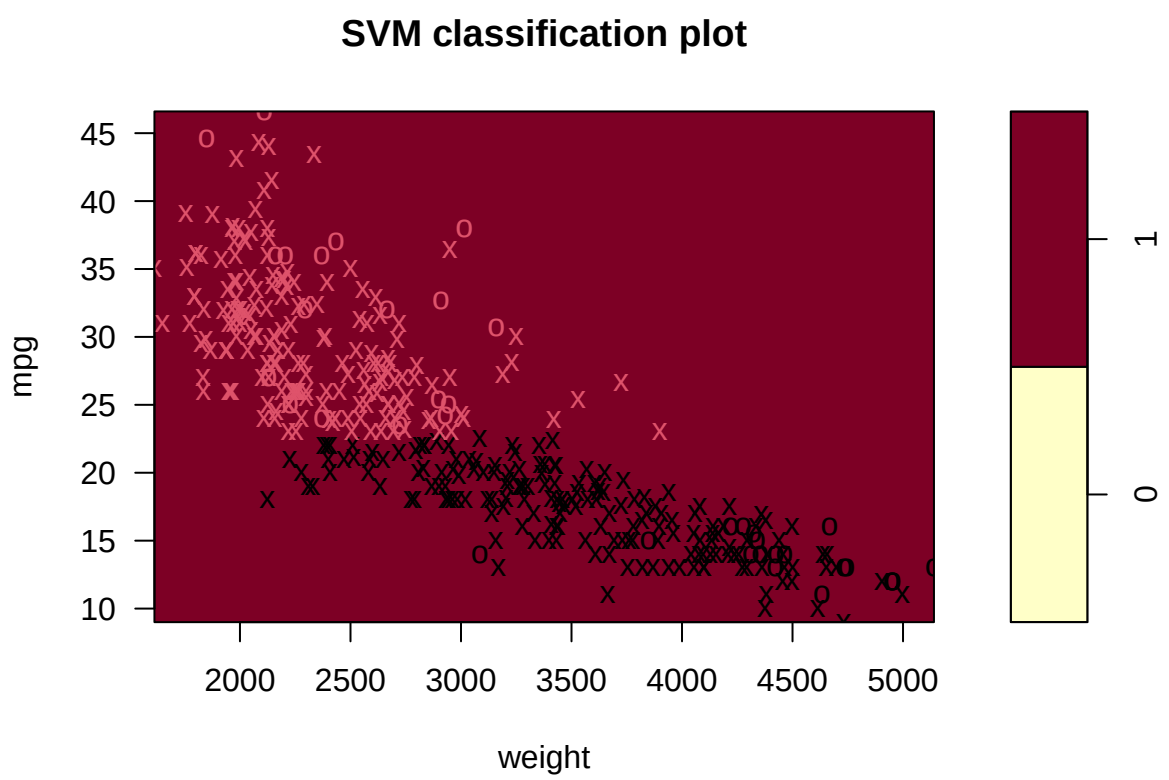

```
## mpg ~ displacement
## <environment: 0x0000000021eeff80>
```



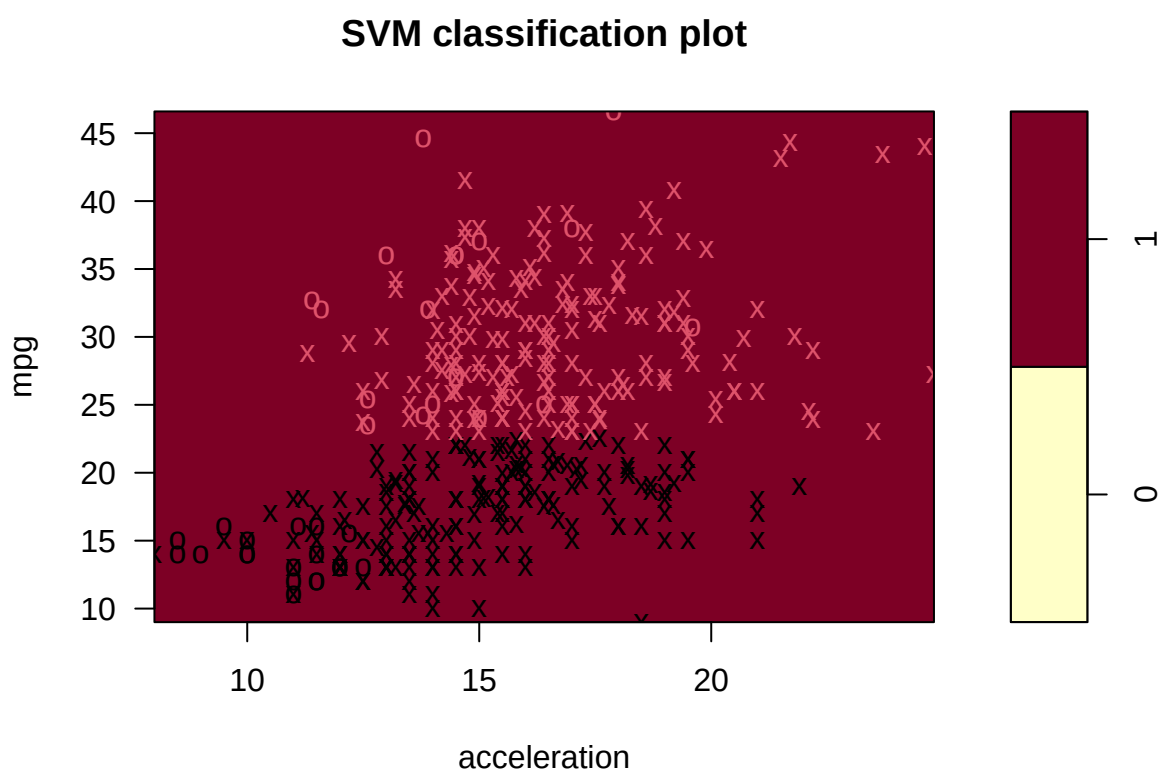

```
## mpg ~ horsepower
## <environment: 0x0000000021eeff80>
```



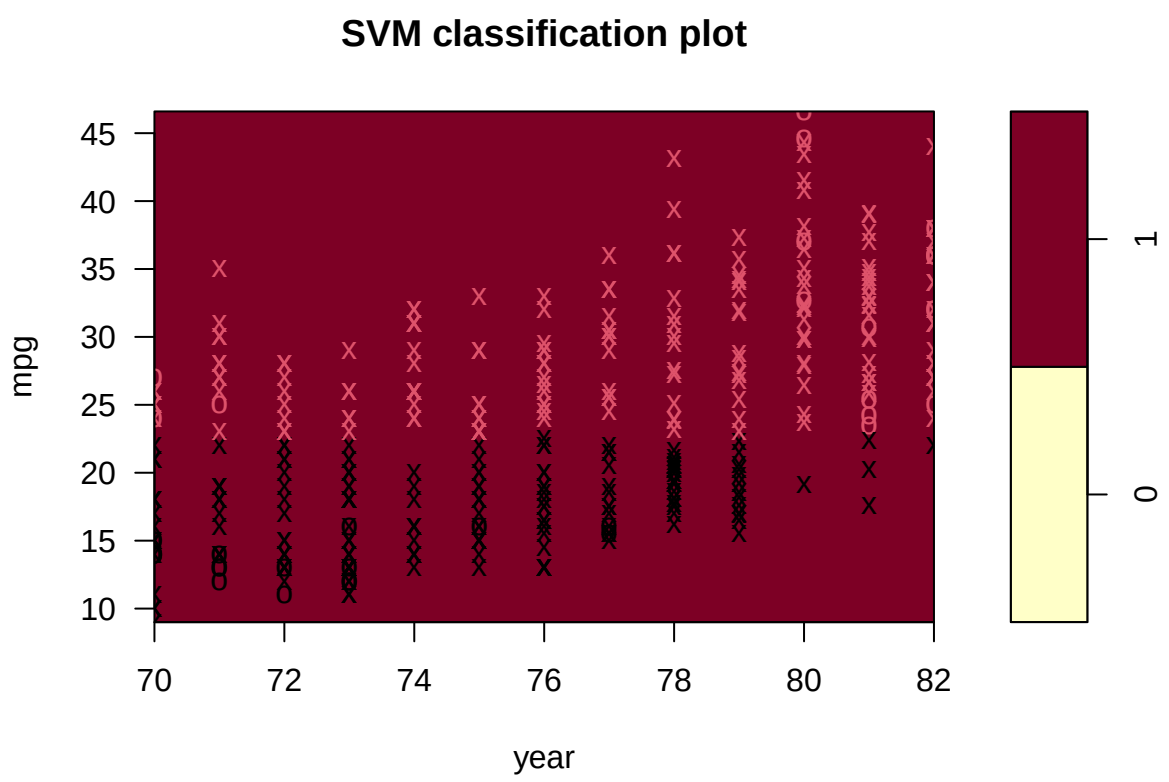

```
## mpg ~ weight
## <environment: 0x0000000021eeff80>
```



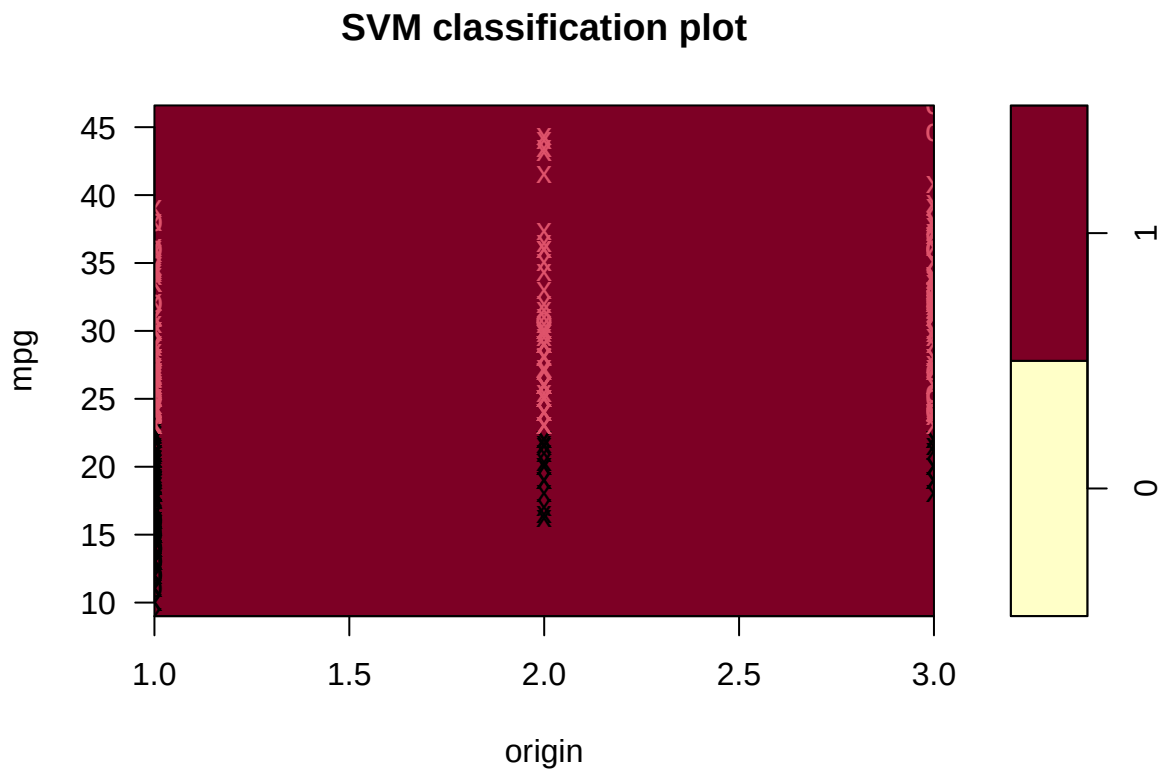

```
## mpg ~ acceleration
## <environment: 0x0000000021eeff80>
```




```
## mpg ~ year
## <environment: 0x0000000021eeff80>
```



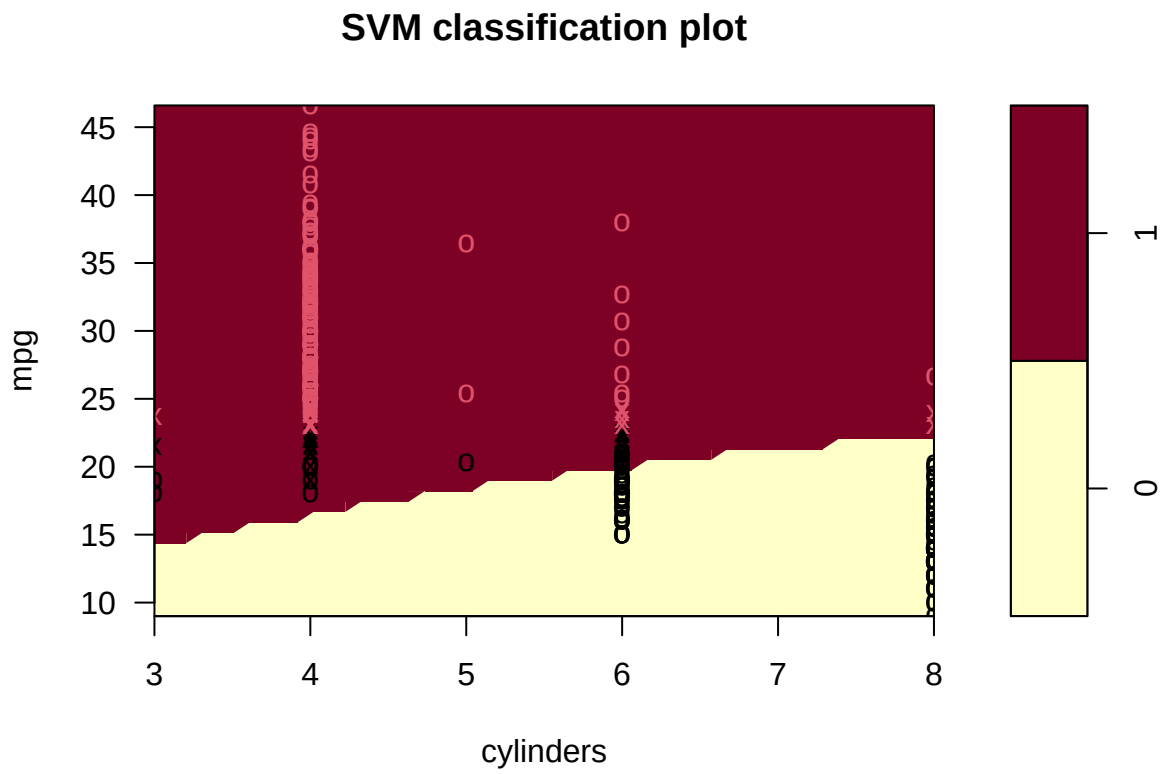

```
## mpg ~ origin
## <environment: 0x0000000021eeff80>
```



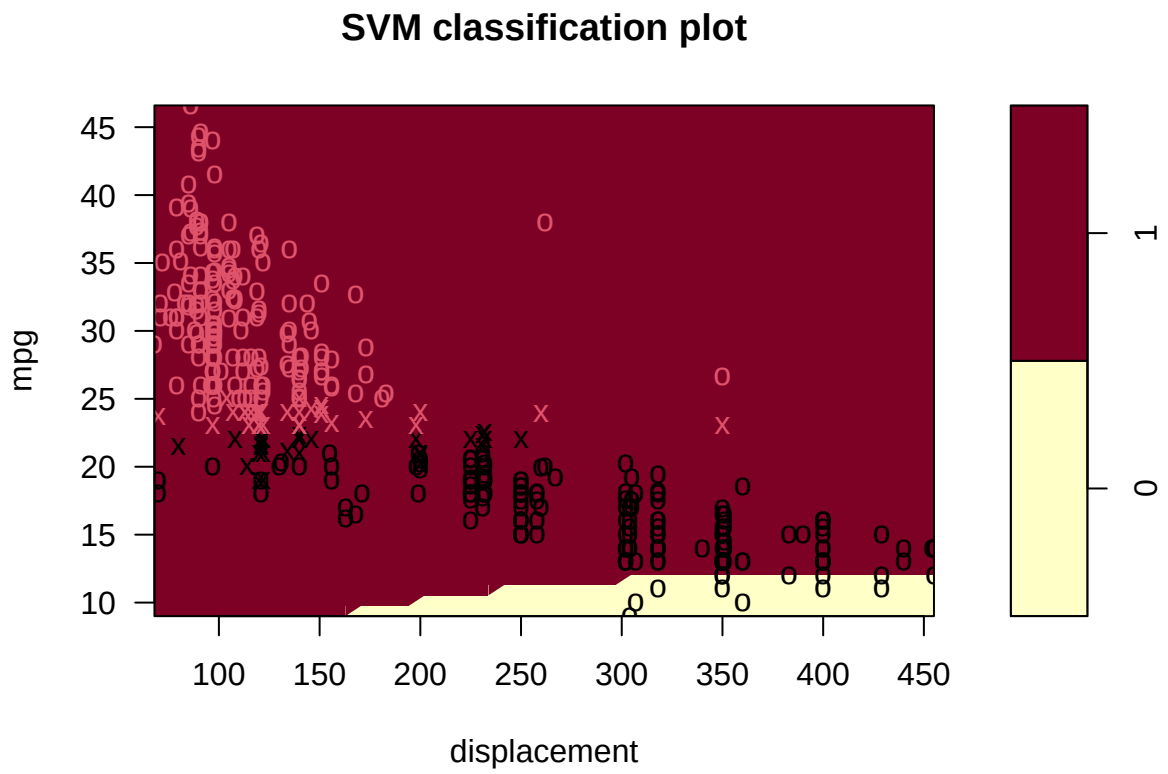

```
## NULL
```

```
print(plotpairs(svmradial))
```

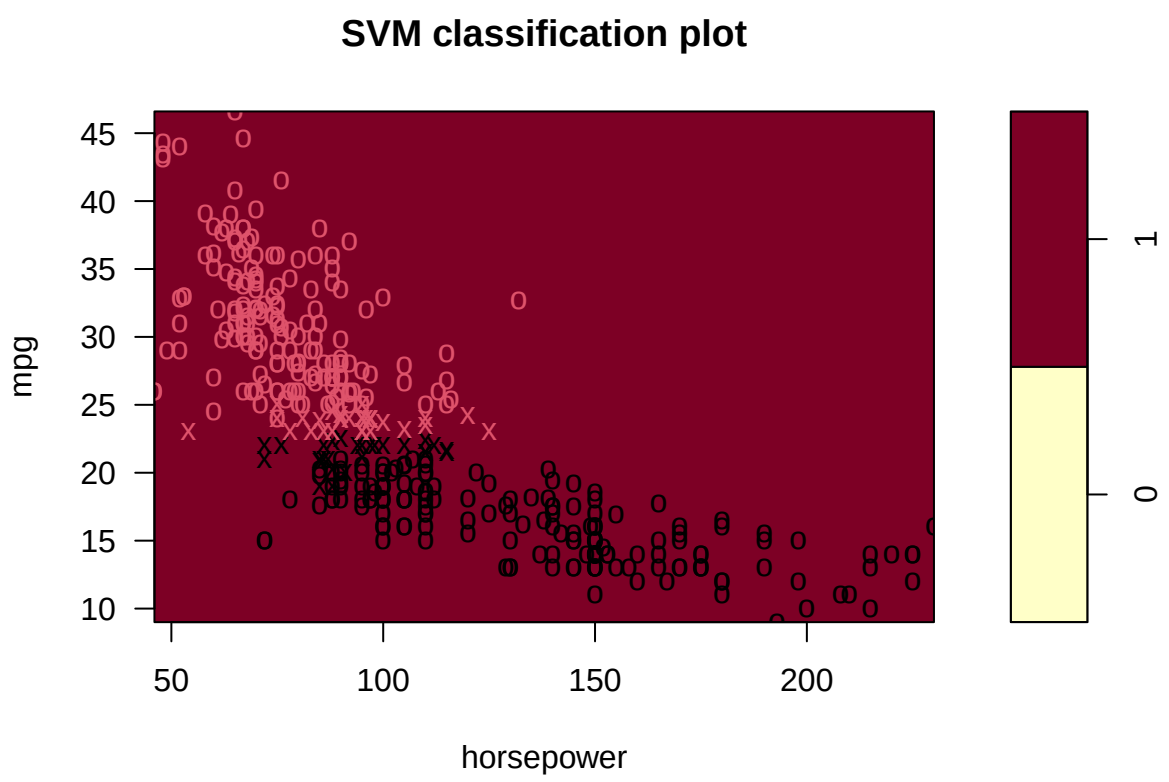
```
## mpg ~ cylinders
## <environment: 0x0000000022829fb8>
```



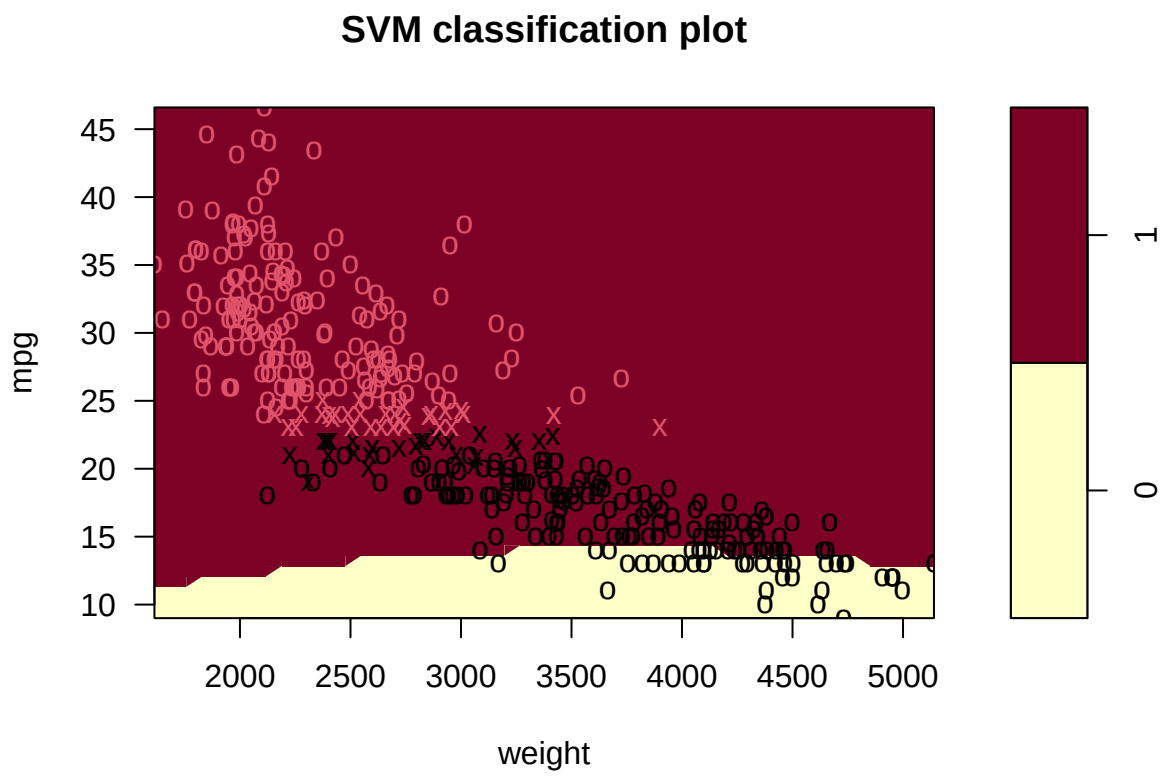

```
## mpg ~ displacement
## <environment: 0x0000000022829fb8>
```



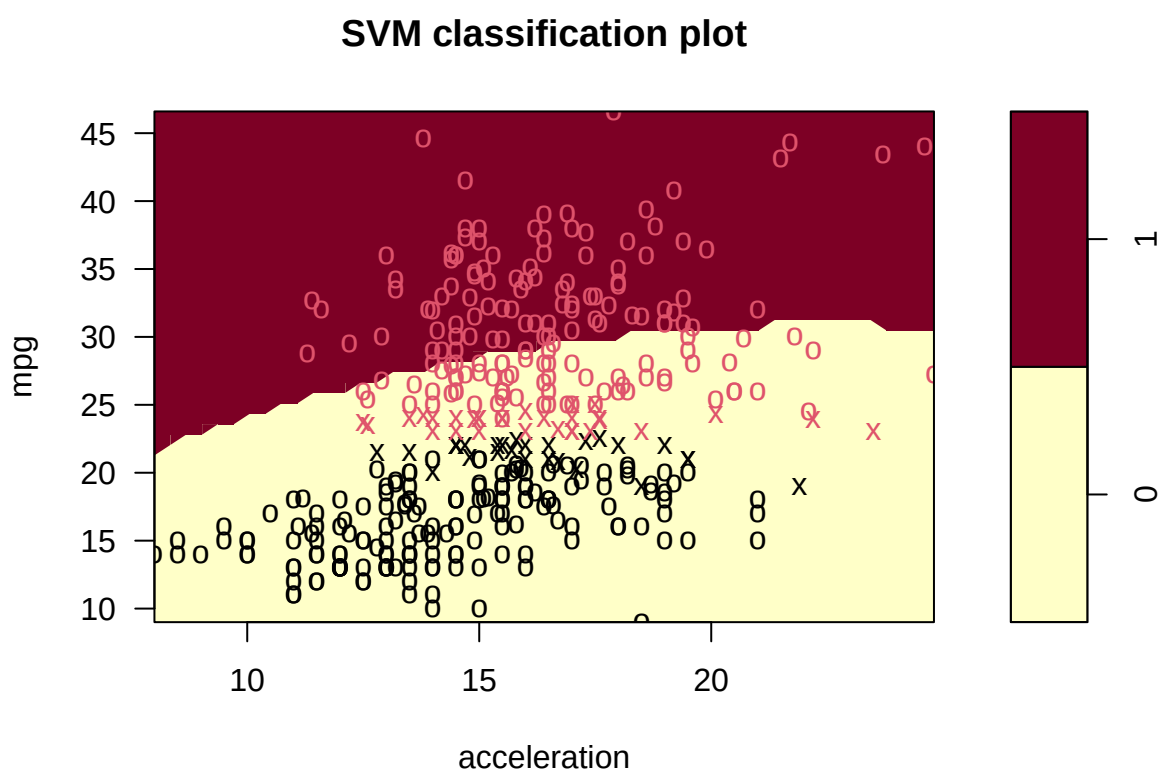

```
## mpg ~ horsepower
## <environment: 0x0000000022829fb8>
```



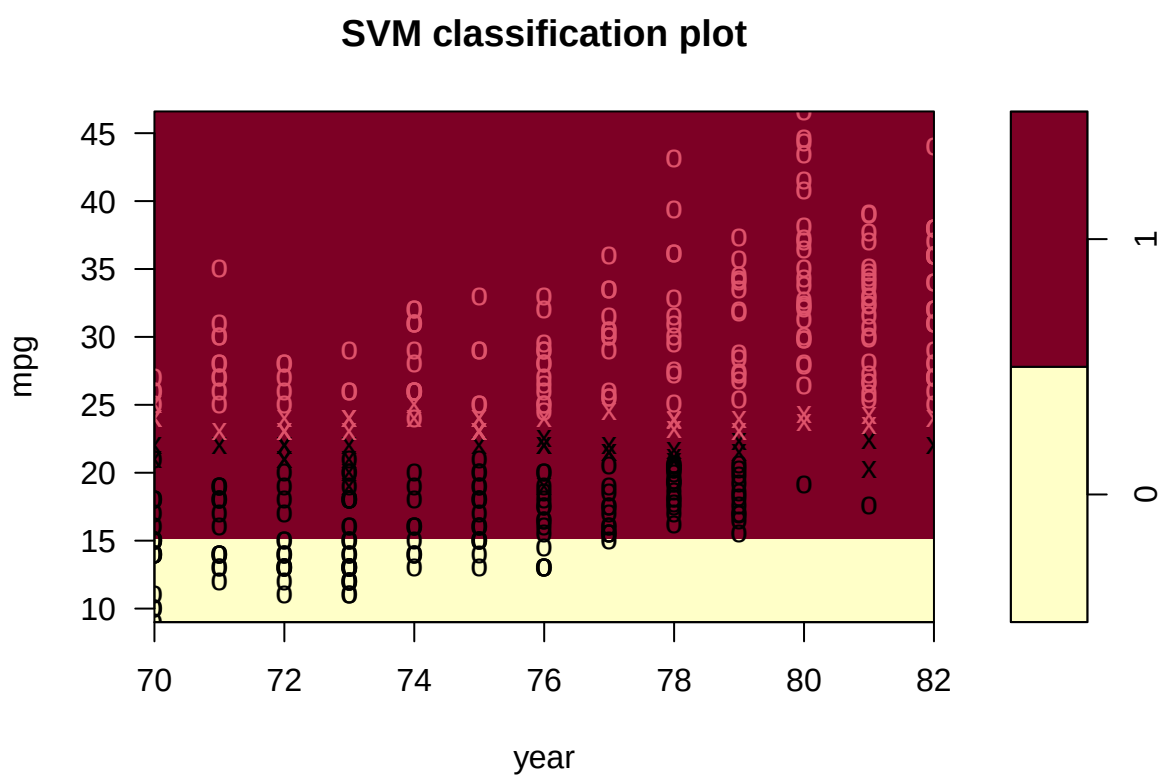

```
## mpg ~ weight
## <environment: 0x0000000022829fb8>
```



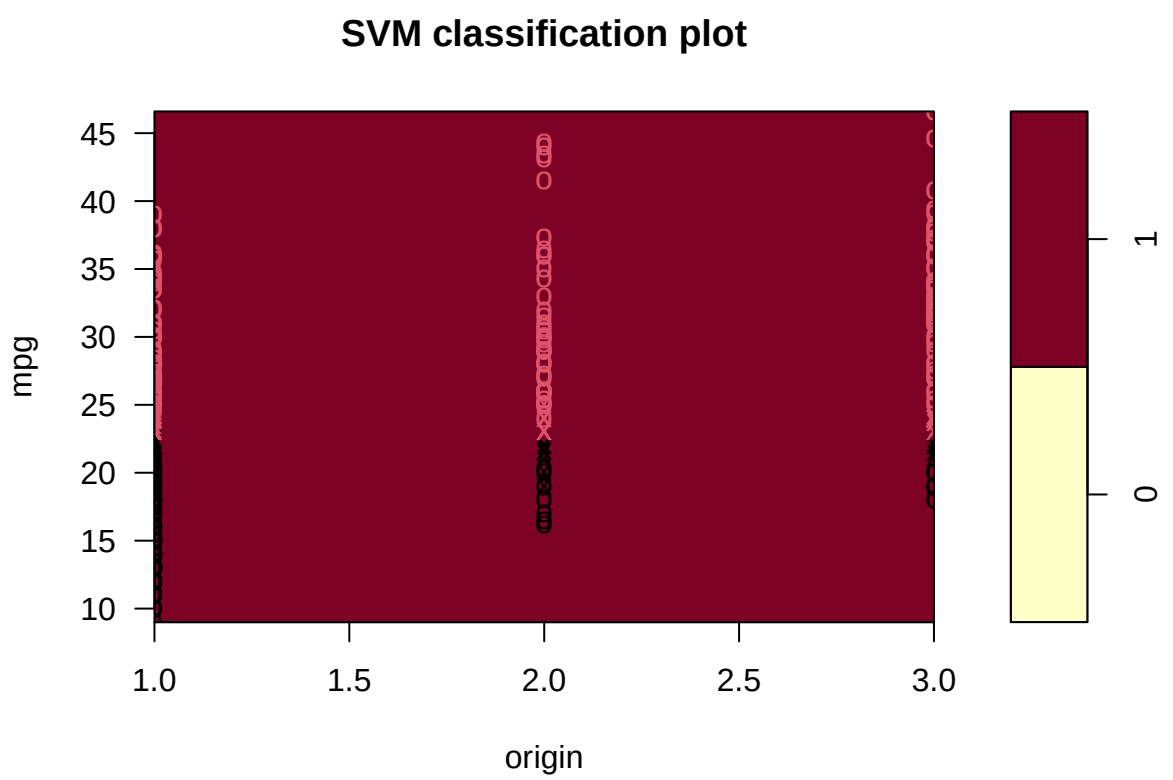

```
## mpg ~ acceleration
## <environment: 0x0000000022829fb8>
```




```
## mpg ~ year
## <environment: 0x0000000022829fb8>
```




```
## mpg ~ origin
## <environment: 0x0000000022829fb8>
```




```
## NULL
```

Chapter 9 Problem 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_helper <- sample(nrow(OJ), 800)
train <- OJ[train_helper, ]
test <- OJ[-train_helper, ]
```

(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.

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  435
##
##  ( 219 216 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Answer

435 support vectors were created, 216 belong to MM level and 219 belong to CH.

(c) What are the training and test error rates?

```
pred <- predict(svmlinear, train)
table(train$Purchase, pred)
```

```
##     pred
##       CH  MM
##   CH 420  65
##   MM  75 240
```

```
pred1 <- predict(svmlinear, test)
table(test$Purchase, pred1)
```

```
##     pred1
##       CH  MM
##   CH 153  15
##   MM  33  69
```

```
print(round((65 + 75) / (420 + 65 + 75 + 240) * 100, 2))
```

```
## [1] 17.5
```

```
print(round(((33 + 15) / (153 + 15 + 33 + 69) * 100), 2))
```

```
## [1] 17.78
```

Answer

The train error rate is 17.5%. The test error rate is 17.78%.

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

```
set.seed(1)
tune.out3 <- tune(svm, Purchase ~ ., data = train, lernel = "linear", ranges = list(cost = 10^seq(-2, 1
summary(tune.out3)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##       cost
##  0.5623413
##
## - best performance: 0.16875
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39375 0.04007372
## 2   0.01778279 0.39375 0.04007372
## 3   0.03162278 0.35750 0.05927806
## 4   0.05623413 0.19500 0.02443813
## 5   0.10000000 0.18625 0.02853482
## 6   0.17782794 0.18250 0.03291403
## 7   0.31622777 0.17875 0.03230175
## 8   0.56234133 0.16875 0.02651650
## 9   1.00000000 0.17125 0.02128673
## 10  1.77827941 0.17625 0.02079162
## 11  3.16227766 0.17750 0.02266912
## 12  5.62341325 0.18000 0.02220485
## 13 10.00000000 0.18625 0.02853482
```

Answer

The error rate suggested that the optimal cost is 0.56.

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

```
svm.linear <- svm(Purchase ~ ., kernel = "linear", data = train, cost = tune.out3$best.parameters$cost)
pred2 <- predict(svm.linear, train)
table(train$Purchase, pred2)
```

```
##     pred2
##       CH  MM
##   CH 424  61
##   MM  71 244
```

```
pred3 <- predict(svm.linear, test)
table(test$Purchase, pred3)
```

```
##     pred3
##       CH  MM
##   CH 155  13
##   MM  29  73
```

```
print(round(((61 + 71) / (424 + 61 + 71 +244) * 100), 2))
```

```
## [1] 16.5
```

```
print(round(((13 + 29) / (155 + 13 + 29 + 73) * 100), 2))
```

```
## [1] 15.56
```

Answer

Using the new value of cost, the training error rate was 16.5% and test error rate was 15.56%

Chapter 8 Problem 10

We now use boosting to predict Salary in the Hitters data set.

(a) Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

```
hitters <- Hitters %>%
  filter(!is.na(Salary)) %>%
  mutate(Salary = log(Salary))
```

(b) Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

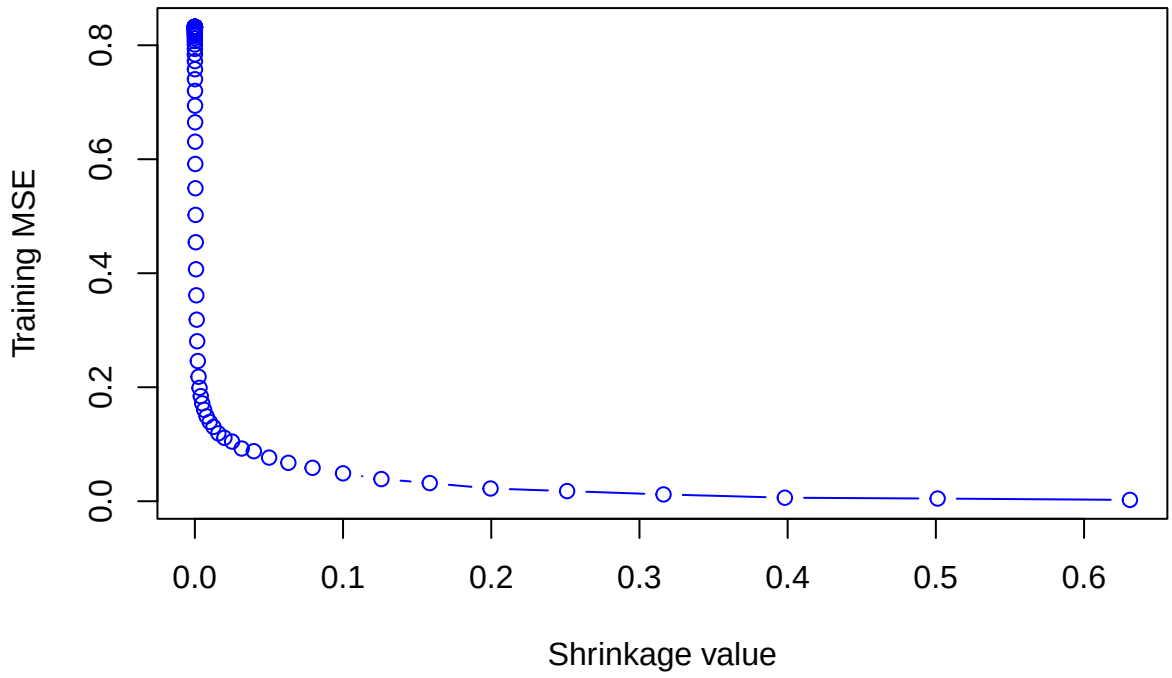
```
train_helper <- 1:200
train <- hitters[train_helper,]
test <- hitters[-train_helper,]
```

(c) Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter lambda. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

```
library(gbm)
```

```
## Loaded gbm 2.1.8
```

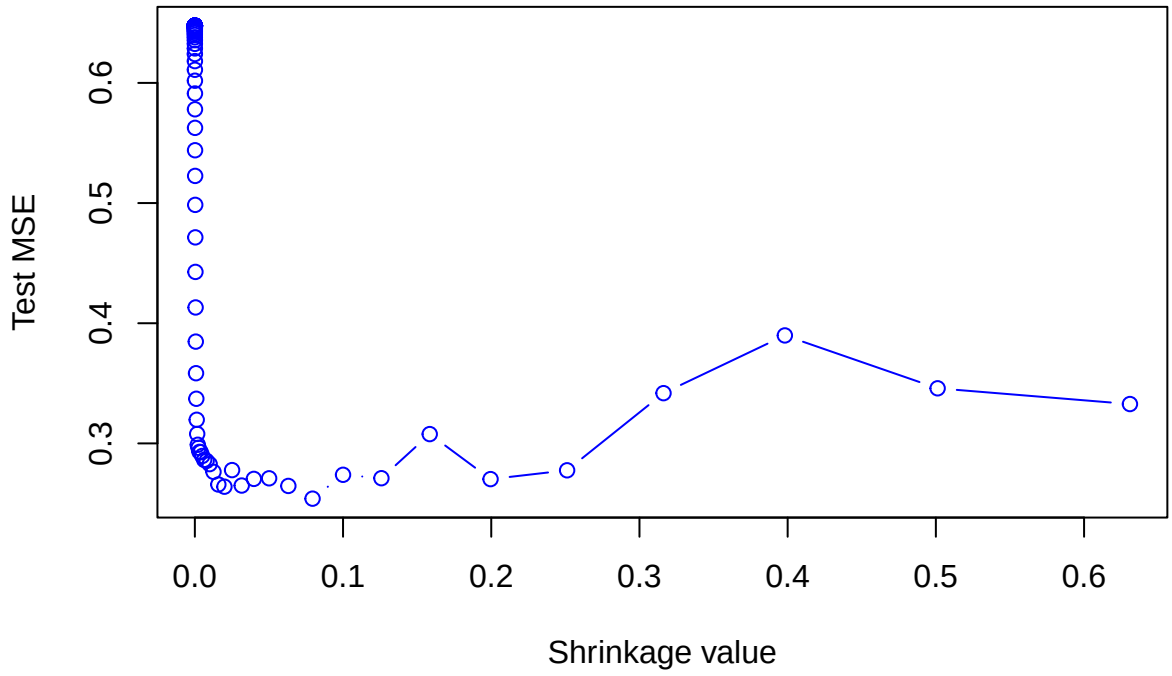
```r
set.seed(1)

lambda_helper <- seq(-10, -0.2, by = 0.1)
lambda <- 10 ^ lambda_helper
error_train <- rep(NA, length(lambda))
for (i in 1:length(lambda)) {
    boost <- gbm(Salary ~ ., data = train, distribution = "gaussian", n.trees = 1000, shrinkage = lambda
    pred <- predict(boost, train, n.trees = 1000)
    error_train[i] <- mean((pred - train$Salary)^2)
}

plot(lambda, error_train, type = "b", xlab = "Shrinkage value", ylab = "Training MSE",
    col = "blue")
```



(d) Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

```r
set.seed(1)
error_test <- rep(NA, length(lambda))
for (i in 1:length(lambda)) {
    boost <- gbm(Salary ~ ., data = train, distribution = "gaussian", n.trees = 1000, shrinkage = lambda
    pred <- predict(boost, test, n.trees = 1000)
    error_test[i] <- mean((pred - test$Salary)^2)
```

```
}
plot(lambda, error_test, type = "b", xlab = "Shrinkage value", ylab = "Test MSE",col = "blue")
```



(e) Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-6
```

```
lm.fit <- lm(Salary ~ .,data = train)
pred4 <- predict(lm.fit, test)
round(mean((pred4 - test$Salary) ^ 2) * 100, 2)
```

```
## [1] 49.18
```

```
train_model<- model.matrix(Salary ~ ., data = train)
test_model <- model.matrix(Salary ~ ., data = test)
salary <- train$Salary
glmnet.fit <- glmnet(train_model, salary, alpha = 0)
pred5 <- predict(glmnet.fit, s = 0.01, newx = test_model)
round(mean((pred5 - test$Salary) ^ 2) * 100 , 2)
```

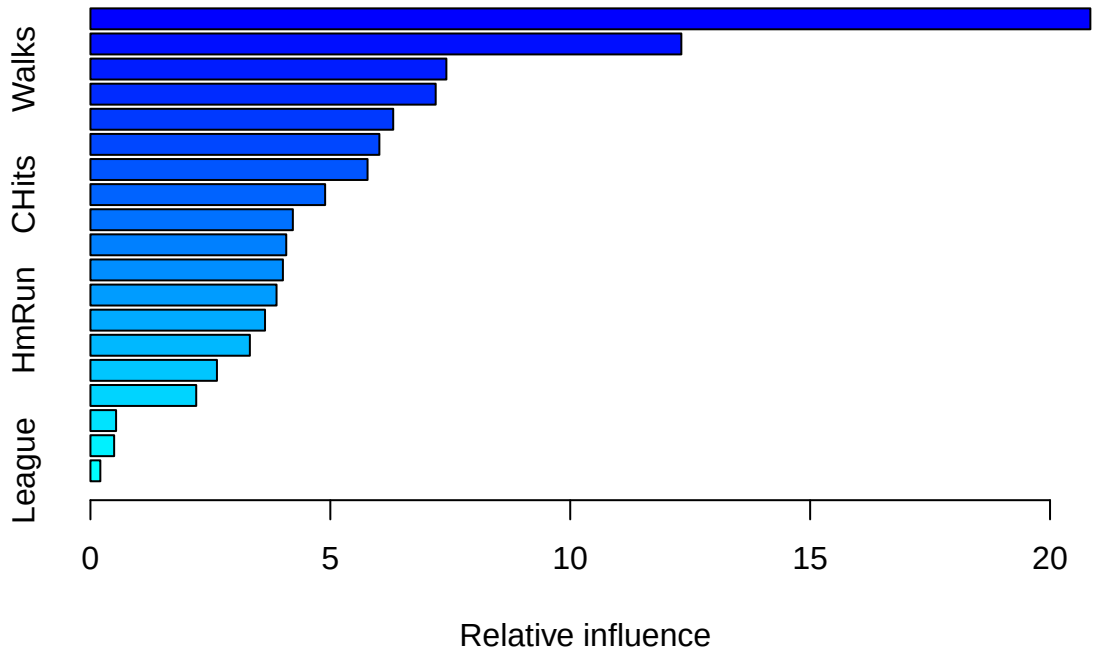
```
## [1] 45.7
```

Answer

The test error for boosting was lower.

(f) Which variables appear to be the most important predictors in the boosted model?

```
boost <- gbm(Salary ~., data = train, distribution = "gaussian", n.trees = 1000, shrinkage = lambda[whi
summary(boost)
```




```
##                 var    rel.inf
## CAtBat       CAtBat 20.8404970
## CRBI           CRBI 12.3158959
```

```
## Walks         Walks  7.4186037
## PutOuts     PutOuts  7.1958539
## Years         Years  6.3104535
## CWalks       CWalks  6.0221656
## CHmRun       CHmRun  5.7759763
## CHits         CHits  4.8914360
## AtBat         AtBat  4.2187460
## RBI             RBI  4.0812410
## Hits           Hits  4.0117255
## Assists     Assists  3.8786634
## HmRun         HmRun  3.6386178
## CRuns         CRuns  3.3230296
## Errors       Errors  2.6369128
## Runs           Runs  2.2048386
## Division   Division  0.5347342
## NewLeague NewLeague  0.4943540
## League       League  0.2062551
```

Answer

Variable of CAtBat seems to be the most important predictor.

(g) Now apply bagging to the training set. What is the test set MSE for this approach?

```
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(1)
bagging <- randomForest(Salary ~ ., data = train, mtry = 19, ntree = 500)
pred6 <- predict(bagging, newdata = test)
round(mean((pred6 - test$Salary) ^ 2) * 100, 2)
```

```
## [1] 22.99
```

Answer

The test MSE for bagging approach is 22.99%.