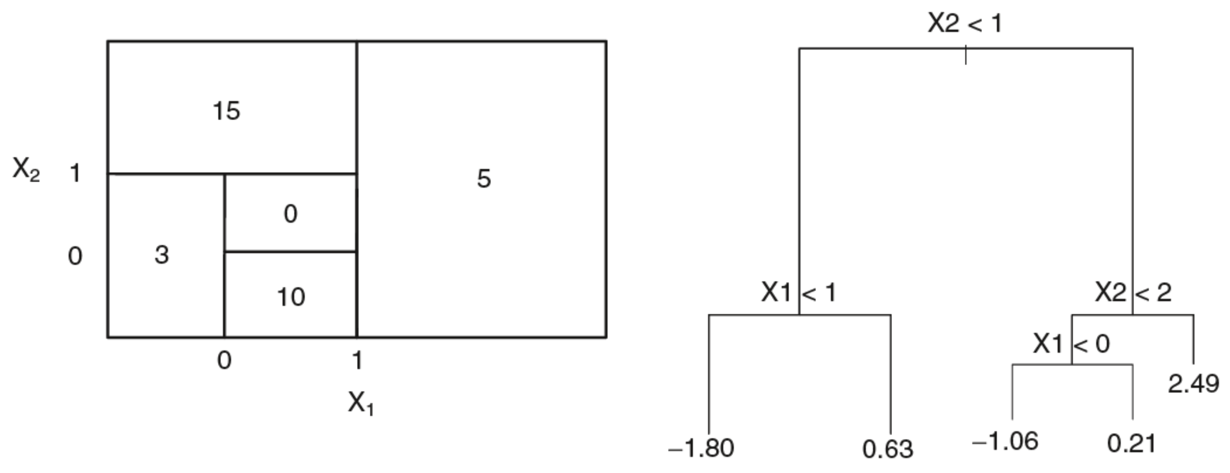
BDA/IDAR Coursework 2 - Solutions

1. Decision Trees (10% | 20%)

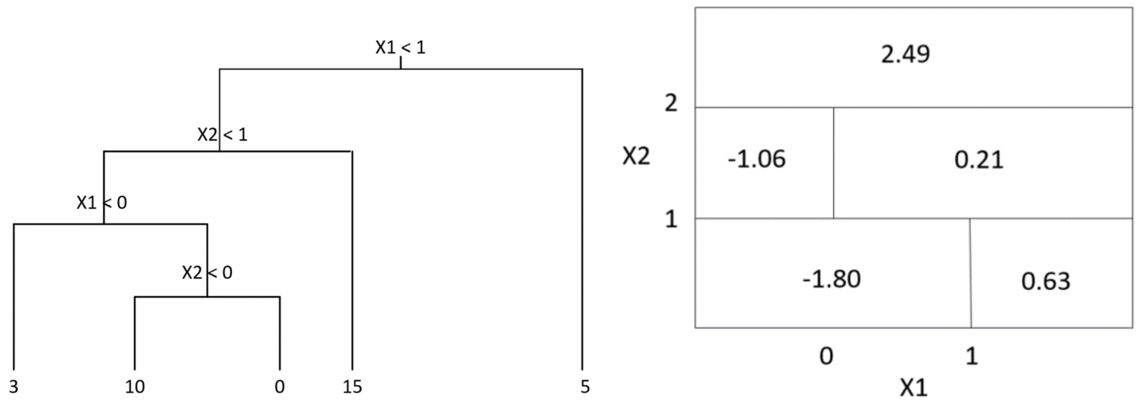
Marking scheme: 5% | 10% each.

This question relates to the following figure.



(a) Sketch the tree corresponding to the partition of the predictor space illustrated in the left-hand panel of the figure above. The numbers inside the boxes indicate the mean of Y within each region.

Solution: See the plot below on the left hand side.



(b) Create a diagram similar to the left-hand panel of the figure, using the tree illustrated in the right-hand panel of the same figure. You should divide up the predictor space into the correct regions, and indicate the mean for each region.

Solution: See the plot above on the right hand side.

2. Regression Trees (15% | 20%)

Marking scheme: 3% | 4% each.

In the lab, a classification tree was applied to the `Carseats` data set after converting `Sales` into a qualitative response variable. Now we will seek to predict `Sales` using regression trees and related approaches, treating the response as a quantitative variable.

(a) Split the data set into a training set and a test set.

Solution:

```
library(ISLR)

set.seed(1)
train <- sample(nrow(Carseats), nrow(Carseats)/2)

Carseats.train <- Carseats[train,]
Carseats.test <- Carseats[-train, ]
sales.test <- Carseats.test$Sales
```

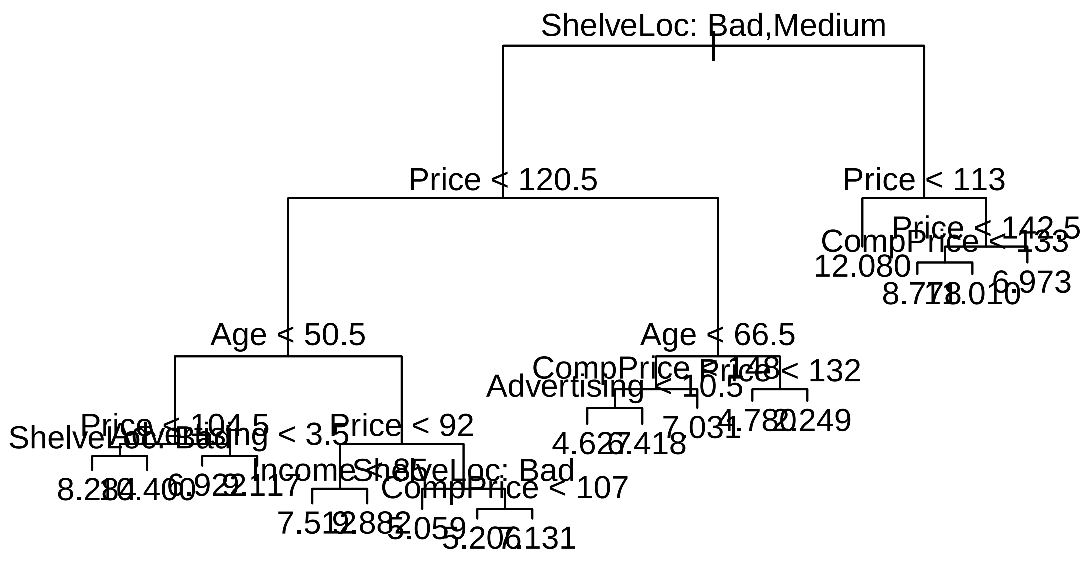
(b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

Solution:

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.4.4
```

```
tree.carseats.train <- tree(Sales ~ ., data = Carseats.train)
# or
# tree.carseats.train <- tree(Sales ~ ., data = Carseats, subset = train)
plot(tree.carseats.train)
text(tree.carseats.train, pretty = 0)
```



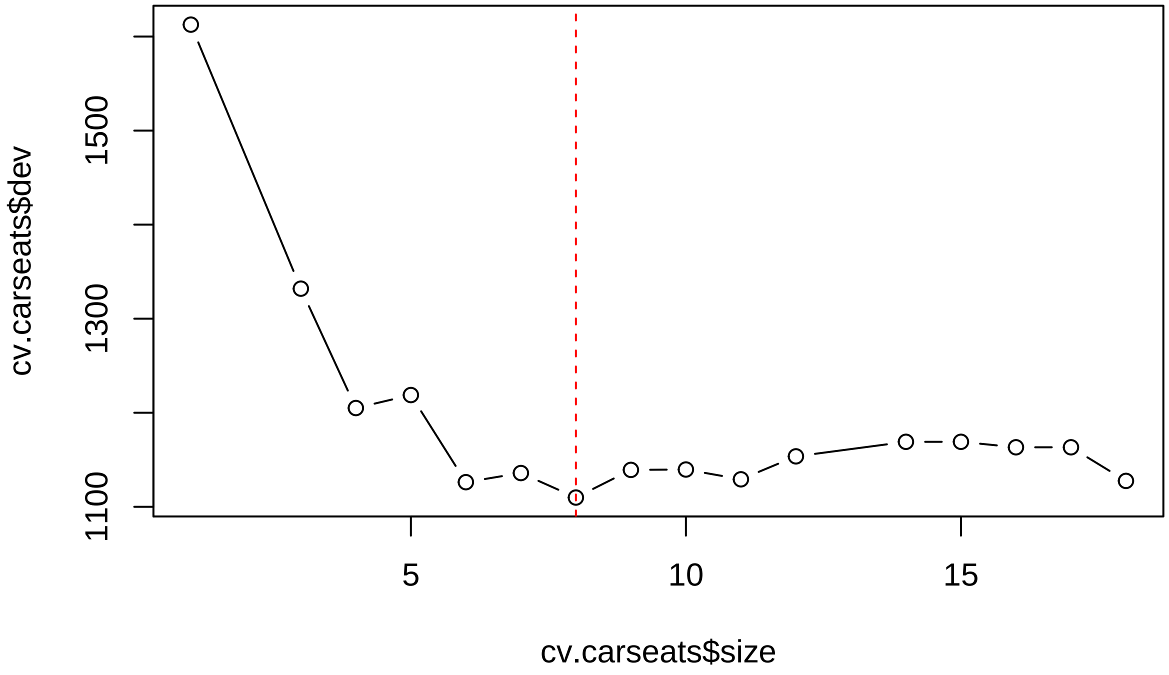
```
sales.tree.pred <- predict(tree.carseats.train, Carseats.test)

# The test MSE is:
mean((sales.test - sales.tree.pred)^2)
```

```
## [1] 4.148897
```

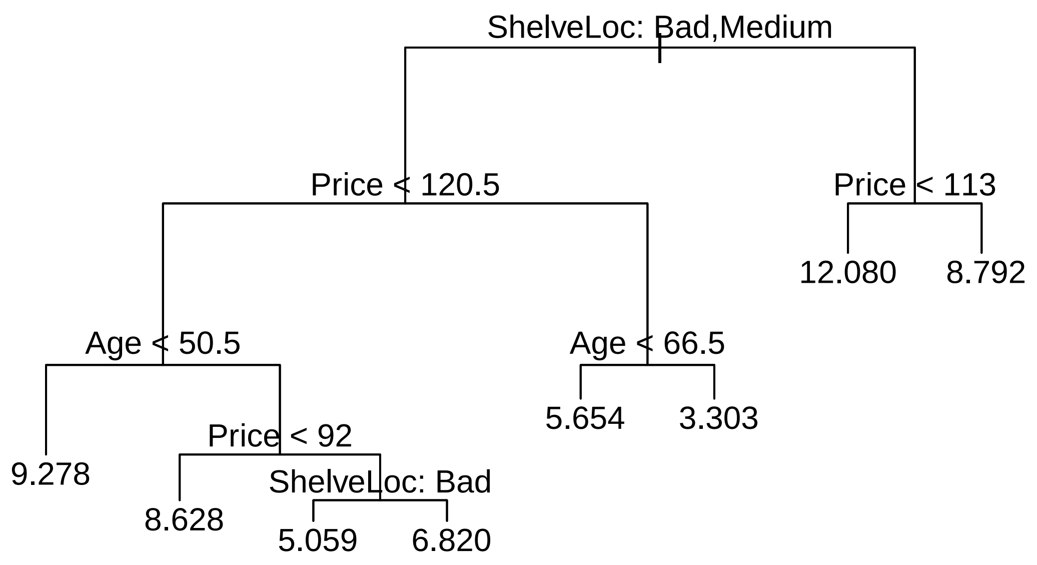
(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

Solution:

```r
library(tree)
tree.carseats.train <- tree(Sales ~ ., data = Carseats.train)

set.seed(1)
cv.carseats <- cv.tree(tree.carseats.train)
# here it shouldn't have FUN=prune.misclass,
# which indicates that we want the classification error rate to
# guide the cross-validation and pruning process,
# rather than the default for the cv.tree() function, which is deviance.

plot(cv.carseats$size, cv.carseats$dev, type = 'b')
#The best size is 8
abline(v = 8, col = "red", lty = 2)
```



```r
prune.carseats <- prune.tree(tree.carseats.train, best = 8)
plot(prune.carseats)
text(prune.carseats, pretty = 0)
```



```r
sales.prune.pred <- predict(prune.carseats, newdata = Carseats.test)
# The test MSE is
mean((sales.test - sales.prune.pred)^2)
```

```
## [1] 5.09085
```

Pruning the tree does not improve the test MSE.

(d) Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the `importance()` function to determine which variables are most important.

Solution:

```r
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```r
set.seed(1)
bag.carseats <- randomForest(Sales ~ .,
                             data = Carseats.train,
                             mtry = ncol(Carseats)-1,
                             ntree = 500,
                             importance = T)

sales.bag.pred <- predict(bag.carseats, Carseats.test)
mean((sales.test - sales.bag.pred)^2)
```

```
## [1] 2.614642
```

```r
#This is a big improvement comparing to the tree model.

importance(bag.carseats)
```

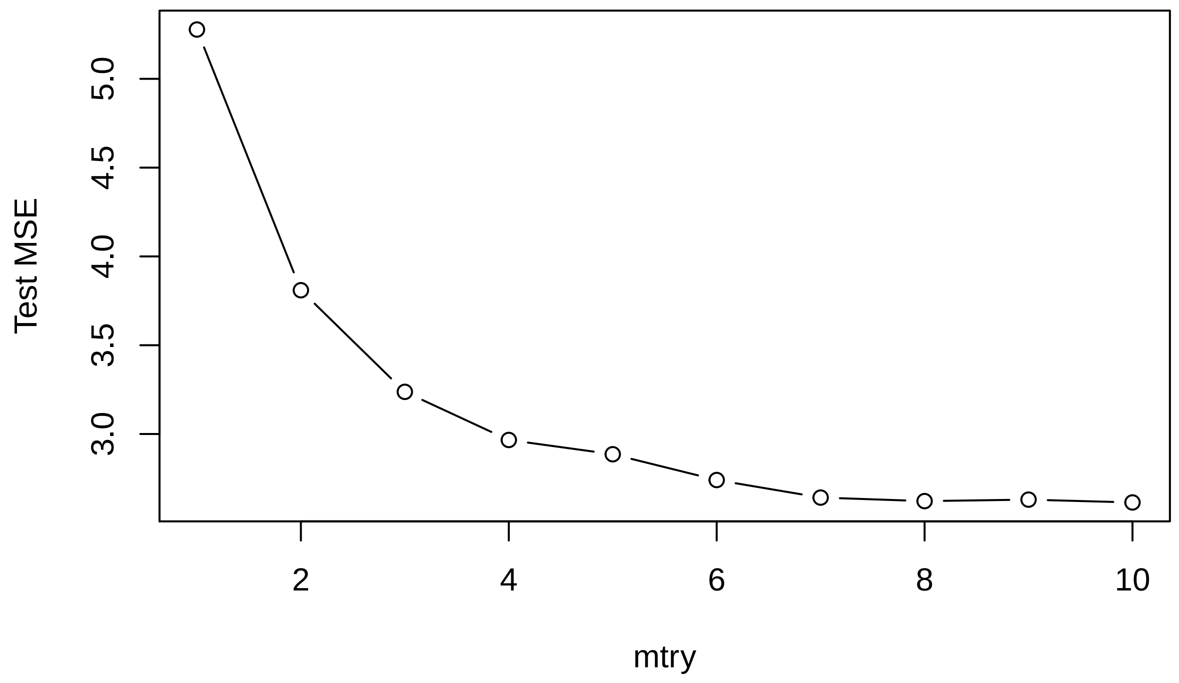
```
##               %IncMSE IncNodePurity
## CompPrice   16.4714051    126.605047
## Income       4.0561872     78.821925
## Advertising 16.2730251    122.793232
## Population   0.7711188     62.796112
## Price       54.5571815    512.940454
## ShelveLoc   42.4486118    320.749734
## Age         20.5369414    184.804253
## Education    2.7755968     42.427788
## Urban       -2.3962157      8.583232
## US           7.2258536     17.605661
```

`Price`, `ShelveLoc` and `Age` are the three most importance factors.

(e) Use random forests to analyze this data. What test MSE do you obtain? Use the `importance()` function to determine which variables are most important. Describe the effect of `m`, the number of variables considered at each split, on the test MSE obtained.

Solution:

```r
#library(randomForest)

testMSE <- rep(0,10)
for (i in 1:10){
  set.seed(1)
  rf.carseats <- randomForest(Sales ~ .,
                              data = Carseats.train,
                              mtry = i,
                              ntree = 500,
                              importance = T)

  sales.rf.pred <- predict(rf.carseats, Carseats.test)
  testMSE[i] <- mean((sales.test - sales.rf.pred)^2)
}
plot(testMSE, type = 'b', xlab = "mtry", ylab = "Test MSE")
```



```r
which.min(testMSE)
```

```
## [1] 10
```

```
#This is a big improvement comparing to the tree model.

rf.carseats.best <- randomForest(Sales ~ .,
                                 data = Carseats.train,
                                 mtry = which.min(testMSE),
                                 ntree = 500,
                                 importance = T)

importance(rf.carseats.best)
```

```
##               %IncMSE IncNodePurity
## CompPrice   15.468317    130.277912
## Income       4.387115     78.616037
## Advertising 16.419399    126.871241
## Population   2.229225     61.905565
## Price       57.416406    512.861624
## ShelveLoc   44.298953    324.030870
## Age         21.950214    191.800070
## Education    3.911940     42.315330
## Urban       -1.715083      7.642252
## US           5.695216     14.487266
```

The best random forest model is the same as the bagging model.

3. Classification Trees (15% | 20%)

Marking scheme: MSc: (i)(j)(k) 1% each, the rest 1.5% each | BSc: (j)(k) 1% each, the rest 2% each.

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.

Solution:

```
library(ISLR)

set.seed(50)
train <- sample(nrow(OJ), 800)
OJ.train <- OJ[train, ]
OJ.test <- OJ[-train, ]
Purchase.test <- OJ.test$Purchase
```

(b) Fit a tree to the training data, with `Purchase` as the response and the other variables as predictors. Use the `summary()` function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

Solution:

```
library(tree)

tree.oj.train <- tree(Purchase ~., data = OJ, subset = train)
summary(tree.oj.train)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ, subset = train)
## Variables actually used in tree construction:
## [1] "LoyalCH"     "PriceDiff"   "SpecialCH"   "SalePriceMM"
## Number of terminal nodes:  9
## Residual mean deviance:  0.6983 = 552.4 / 791
## Misclassification error rate: 0.1562 = 125 / 800
```

Four variables `LoyalCH`, `PriceDiff`, `SpecialCH` and `SalePriceMM` were used in tree construction. The training error rate is 0.1562. There are 9 terminal nodes.

(c) Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

Solution:

```
tree.oj.train
```

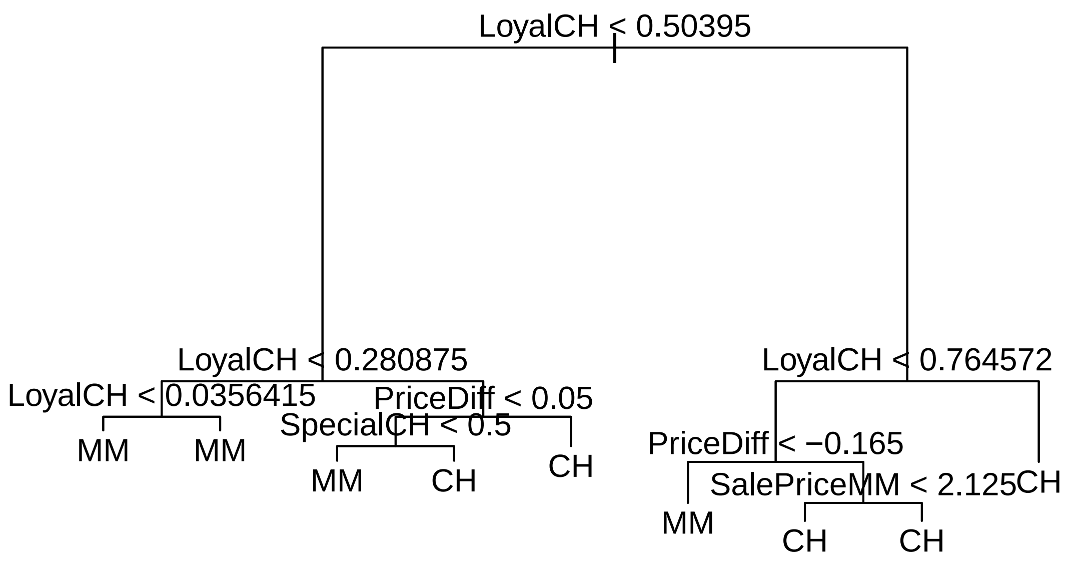
```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1063.000 CH ( 0.61875 0.38125 )
##    2) LoyalCH < 0.50395 334  393.200 MM ( 0.27545 0.72455 )
##      4) LoyalCH < 0.280875 162  128.700 MM ( 0.13580 0.86420 )
##        8) LoyalCH < 0.0356415 54    9.959 MM ( 0.01852 0.98148 ) *
##        9) LoyalCH > 0.0356415 108  106.400 MM ( 0.19444 0.80556 ) *
##      5) LoyalCH > 0.280875 172  232.500 MM ( 0.40698 0.59302 )
##       10) PriceDiff < 0.05 71   67.600 MM ( 0.18310 0.81690 )
##         20) SpecialCH < 0.5 56   33.700 MM ( 0.08929 0.91071 ) *
##         21) SpecialCH > 0.5 15   20.730 CH ( 0.53333 0.46667 ) *
##       11) PriceDiff > 0.05 101  138.300 CH ( 0.56436 0.43564 ) *
##    3) LoyalCH > 0.50395 466  369.200 CH ( 0.86481 0.13519 )
##      6) LoyalCH < 0.764572 200  239.000 CH ( 0.71500 0.28500 )
##       12) PriceDiff < -0.165 34   39.300 MM ( 0.26471 0.73529 ) *
##       13) PriceDiff > -0.165 166  162.800 CH ( 0.80723 0.19277 )
##         26) SalePriceMM < 2.125 96  115.900 CH ( 0.70833 0.29167 ) *
##         27) SalePriceMM > 2.125 70   30.660 CH ( 0.94286 0.05714 ) *
##      7) LoyalCH > 0.764572 266   57.360 CH ( 0.97744 0.02256 ) *
```

The node labeled `5)` says that the splitting variable at this node is `LoyalCH`, with condition `LoyalCH > 0.280875`. There are 172 observations in this subtree. The deviance in this subtree is 232.5. The prediction at this node is `Purchase = MM`, based on the prediction that 0.40698 of the observations in this subtree have `CH` and 0.59302 have `MM`. The majority vote concludes it to be `MM`. This is not a leaf node.

(d) Create a plot of the tree, and interpret the results.

Solution:

```
plot(tree.oj.train)
text(tree.oj.train, pretty = 0)
```



LoyalCH is the most important predictor, as the first two levels are solely dependent on loyalCH. When `loyalCH < 0.280875`, it will be MM; when `loyalCH > 0.764572` it will be CH. For the other cases, we also need to look at PriceDiff, SpecialCH or SalePriceMM.

(e) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

Solution:

```
Purchase.pred <- predict(tree.oj.train, OJ.test, type = "class")
table(predict = Purchase.pred, truth = Purchase.test)
```

```
##        truth
## predict  CH  MM
##      CH 142  33
##      MM  16  79
```

```
#Test error rate is:
mean(Purchase.pred !=  Purchase.test)
```

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

(f) Apply the cv.tree() function to the training set in order to determine the optimal tree size.

Solution:

```
set.seed(3)
cv.oj <- cv.tree(tree.oj.train, FUN = prune.misclass)
cv.oj
```

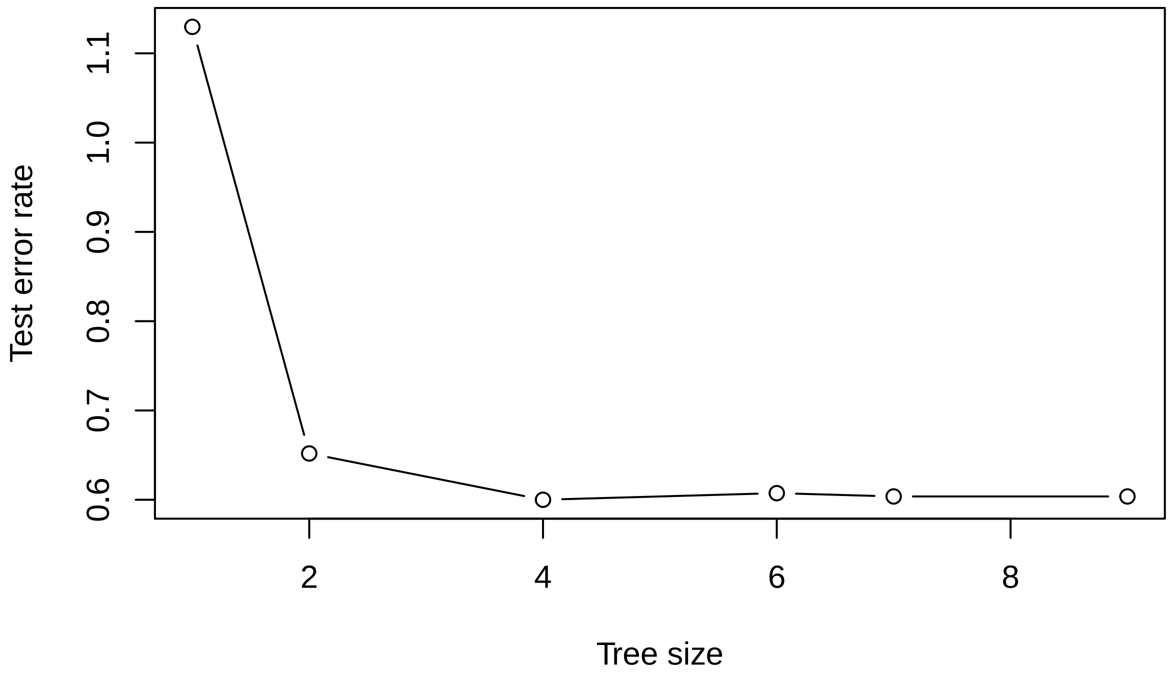
```
## $size
## [1] 9 7 6 4 2 1
##
## $dev
## [1] 163 163 164 162 176 305
##
## $k
## [1]  -Inf   0.0   1.0   6.5   8.0 150.0
##
## $method
## [1] "misclass"
```

```
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

(g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

Solution:

```
plot(cv.oj$size, cv.oj$dev/nrow(OJ.test),
     xlab = "Tree size", ylab = "Test error rate", type = "b")
```



(h) Which tree size corresponds to the lowest cross-validated classification error rate?

Solution: From the result, the best model is when there are 4 leaves.

(i) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

Solution:

```
prune.oj <- prune.misclass(tree.oj.train, best = 4)
```

(j) Compare the training error rates between the pruned and unpruned trees. Which is higher?

Solution:

```
summary(prune.oj)
```

```
##
## Classification tree:
## snip.tree(tree = tree.oj.train, nodes = c(13L, 2L))
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  4
## Residual mean deviance:  0.8199 = 652.6 / 796
## Misclassification error rate: 0.1737 = 139 / 800
```

The training error rate of the unpruned tree `tree.oj.train` is 0.1562, and that of the pruned tree `prune.oj` is 0.1737.

(k) Compare the test error rates between the pruned and unpruned trees. Which is higher?

Solution:

```
Purchase.prune.pred <- predict(prune.oj, OJ.test, type = "class")
mean(Purchase.prune.pred != Purchase.test)
```

```
## [1] 0.2185185
```

The test error rate of the unpruned tree `tree.oj.train` is 0.1815 and that of the pruned tree `prune.oj` is 0.2185.

4. SVM (15% | 20%)

Marking scheme: MSc: (a) 3%, the rest 4% each | BSc: 4% each.

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.

Solution:

```
data("Auto")
n <- nrow(Auto)
Auto$mpg01 <- rep(0, n)
Auto$mpg01[Auto$mpg >= median(Auto$mpg)] <- 1
Auto$mpg01 <- as.factor(Auto$mpg01)
Auto$mpg <- NULL  #remove the column of mpg
head(Auto)
```

```
##   cylinders displacement horsepower weight acceleration year origin
## 1         8          307        130   3504         12.0   70      1
## 2         8          350        165   3693         11.5   70      1
## 3         8          318        150   3436         11.0   70      1
## 4         8          304        150   3433         12.0   70      1
## 5         8          302        140   3449         10.5   70      1
## 6         8          429        198   4341         10.0   70      1
##                        name mpg01
## 1 chevrolet chevelle malibu     0
## 2         buick skylark 320     0
## 3        plymouth satellite     0
## 4             amc rebel sst     0
## 5               ford torino     0
## 6          ford galaxie 500     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.

Solution:

```
library(e1071)
set.seed(100)
tune.out.linear <- tune(svm, mpg01 ~ ., data = Auto,
                        kernel = "linear",
                        ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 50, 100)))
summary(tune.out.linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.08942308
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.12762821 0.07353560
## 2 1e-02 0.08942308 0.04419023
## 3 1e-01 0.09961538 0.05349840
## 4 1e+00 0.10448718 0.05563850
## 5 5e+00 0.11730769 0.05547413
## 6 1e+01 0.12230769 0.05599171
## 7 5e+01 0.13000000 0.05012658
## 8 1e+02 0.12230769 0.04903630
```

The best parameter is `cost = 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.

Solution:

```
library(e1071)
set.seed(100)
tune.out.radial <- tune(svm, mpg01 ~ ., data = Auto,
                        kernel = "radial",
                        ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 50, 100),
                                      gamma = c(0.1,0.5,1,2,3,4))
)
summary(tune.out.radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.1
##
## - best performance: 0.07160256
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-03   0.1 0.55115385 0.04842542
## 2  1e-02   0.1 0.21173077 0.10654149
## 3  1e-01   0.1 0.09198718 0.04412820
## 4  1e+00   0.1 0.08429487 0.04381671
## 5  5e+00   0.1 0.07410256 0.04908785
## 6  1e+01   0.1 0.07160256 0.04504246
## 7  5e+01   0.1 0.09205128 0.05172547
## 8  1e+02   0.1 0.09961538 0.05204577
## 9  1e-03   0.5 0.55115385 0.04842542
## 10 1e-02   0.5 0.55115385 0.04842542
## 11 1e-01   0.5 0.08942308 0.04581353
## 12 1e+00   0.5 0.07916667 0.04101813
## 13 5e+00   0.5 0.07666667 0.03646679
## 14 1e+01   0.5 0.08179487 0.03610441
## 15 5e+01   0.5 0.08692308 0.04070746
## 16 1e+02   0.5 0.08692308 0.04070746
## 17 1e-03   1.0 0.55115385 0.04842542
## 18 1e-02   1.0 0.55115385 0.04842542
## 19 1e-01   1.0 0.55115385 0.04842542
## 20 1e+00   1.0 0.07929487 0.05064964
## 21 5e+00   1.0 0.08435897 0.04551954
## 22 1e+01   1.0 0.08179487 0.04345042
## 23 5e+01   1.0 0.08179487 0.04345042
## 24 1e+02   1.0 0.08179487 0.04345042
## 25 1e-03   2.0 0.55115385 0.04842542
## 26 1e-02   2.0 0.55115385 0.04842542
## 27 1e-01   2.0 0.55115385 0.04842542
## 28 1e+00   2.0 0.12282051 0.07841106
## 29 5e+00   2.0 0.12538462 0.07715255
## 30 1e+01   2.0 0.12538462 0.07715255
## 31 5e+01   2.0 0.12538462 0.07715255
## 32 1e+02   2.0 0.12538462 0.07715255
## 33 1e-03   3.0 0.55115385 0.04842542
## 34 1e-02   3.0 0.55115385 0.04842542
## 35 1e-01   3.0 0.55115385 0.04842542
## 36 1e+00   3.0 0.34237179 0.17305805
## 37 5e+00   3.0 0.33224359 0.17921735
## 38 1e+01   3.0 0.33224359 0.17921735
## 39 5e+01   3.0 0.33224359 0.17921735
## 40 1e+02   3.0 0.33224359 0.17921735
## 41 1e-03   4.0 0.55115385 0.04842542
## 42 1e-02   4.0 0.55115385 0.04842542
## 43 1e-01   4.0 0.55115385 0.04842542
## 44 1e+00   4.0 0.48230769 0.04420155
## 45 5e+00   4.0 0.47205128 0.05380221
## 46 1e+01   4.0 0.47205128 0.05380221
## 47 5e+01   4.0 0.47205128 0.05380221
## 48 1e+02   4.0 0.47205128 0.05380221
```

```r
set.seed(100)
tune.out.poly <- tune(svm, mpg01 ~ ., data = Auto,
                      kernel = "polynomial",
                      ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 50, 100),
                                    degree = c(2,3,4,5,6))
)
```

```r
summary(tune.out.poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      2
##
## - best performance: 0.3164103
##
## - Detailed performance results:
##     cost degree     error dispersion
## 1  1e-03      2 0.5511538 0.04842542
## 2  1e-02      2 0.5511538 0.04842542
## 3  1e-01      2 0.5511538 0.04842542
## 4  1e+00      2 0.5511538 0.04842542
## 5  5e+00      2 0.5511538 0.04842542
## 6  1e+01      2 0.4921795 0.11110158
## 7  5e+01      2 0.3621154 0.07702181
## 8  1e+02      2 0.3164103 0.06544408
## 9  1e-03      3 0.5511538 0.04842542
## 10 1e-02      3 0.5511538 0.04842542
## 11 1e-01      3 0.5511538 0.04842542
## 12 1e+00      3 0.5511538 0.04842542
## 13 5e+00      3 0.5511538 0.04842542
## 14 1e+01      3 0.5511538 0.04842542
## 15 5e+01      3 0.4537179 0.14737606
## 16 1e+02      3 0.3977564 0.09638019
## 17 1e-03      4 0.5511538 0.04842542
## 18 1e-02      4 0.5511538 0.04842542
## 19 1e-01      4 0.5511538 0.04842542
## 20 1e+00      4 0.5511538 0.04842542
## 21 5e+00      4 0.5511538 0.04842542
## 22 1e+01      4 0.5511538 0.04842542
## 23 5e+01      4 0.5511538 0.04842542
## 24 1e+02      4 0.5511538 0.04842542
## 25 1e-03      5 0.5511538 0.04842542
## 26 1e-02      5 0.5511538 0.04842542
## 27 1e-01      5 0.5511538 0.04842542
## 28 1e+00      5 0.5511538 0.04842542
## 29 5e+00      5 0.5511538 0.04842542
## 30 1e+01      5 0.5511538 0.04842542
## 31 5e+01      5 0.5511538 0.04842542
## 32 1e+02      5 0.5511538 0.04842542
## 33 1e-03      6 0.5511538 0.04842542
## 34 1e-02      6 0.5511538 0.04842542
## 35 1e-01      6 0.5511538 0.04842542
## 36 1e+00      6 0.5511538 0.04842542
## 37 5e+00      6 0.5511538 0.04842542
## 38 1e+01      6 0.5511538 0.04842542
## 39 5e+01      6 0.5511538 0.04842542
## 40 1e+02      6 0.5511538 0.04842542
```

The best parameters for the radial kernel model are `cost = 1` and `gamma = 0.5`, while those for the polynomial kernel model are `cost = 100` and `degree = 3`.

(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)
```

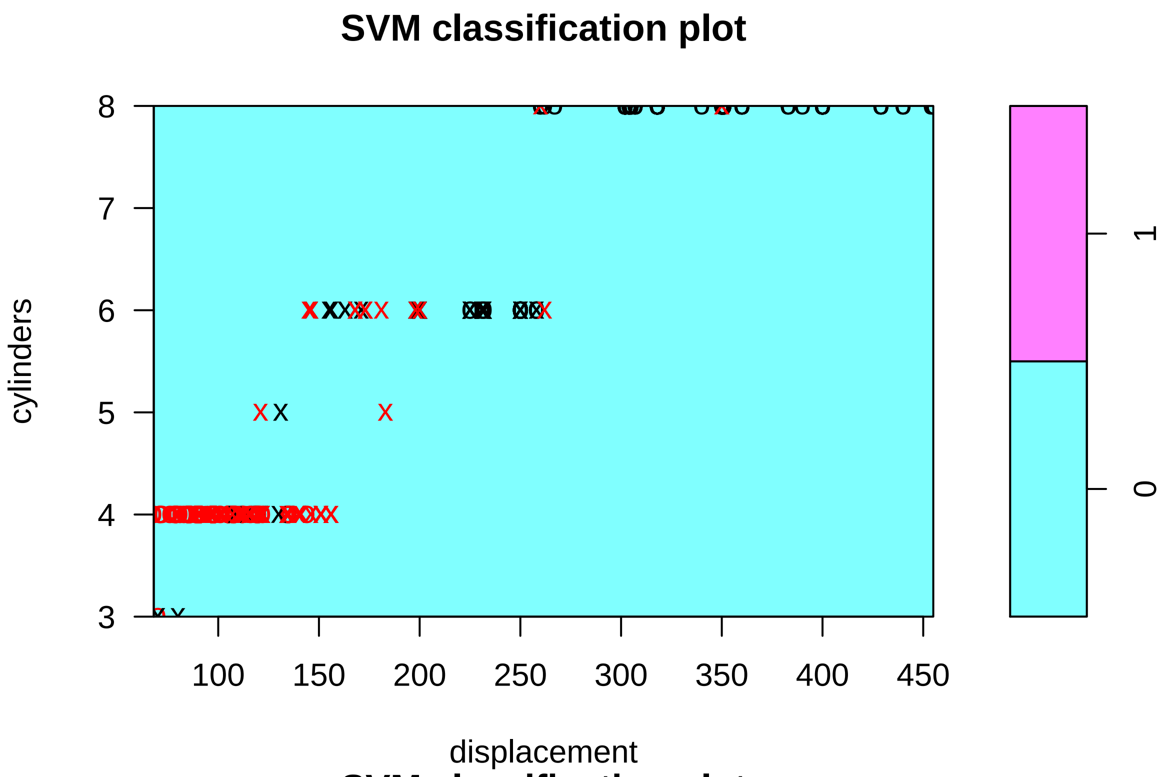
where `svmfit` contains your fitted model and `dat` is a data frame containing your data, you can type

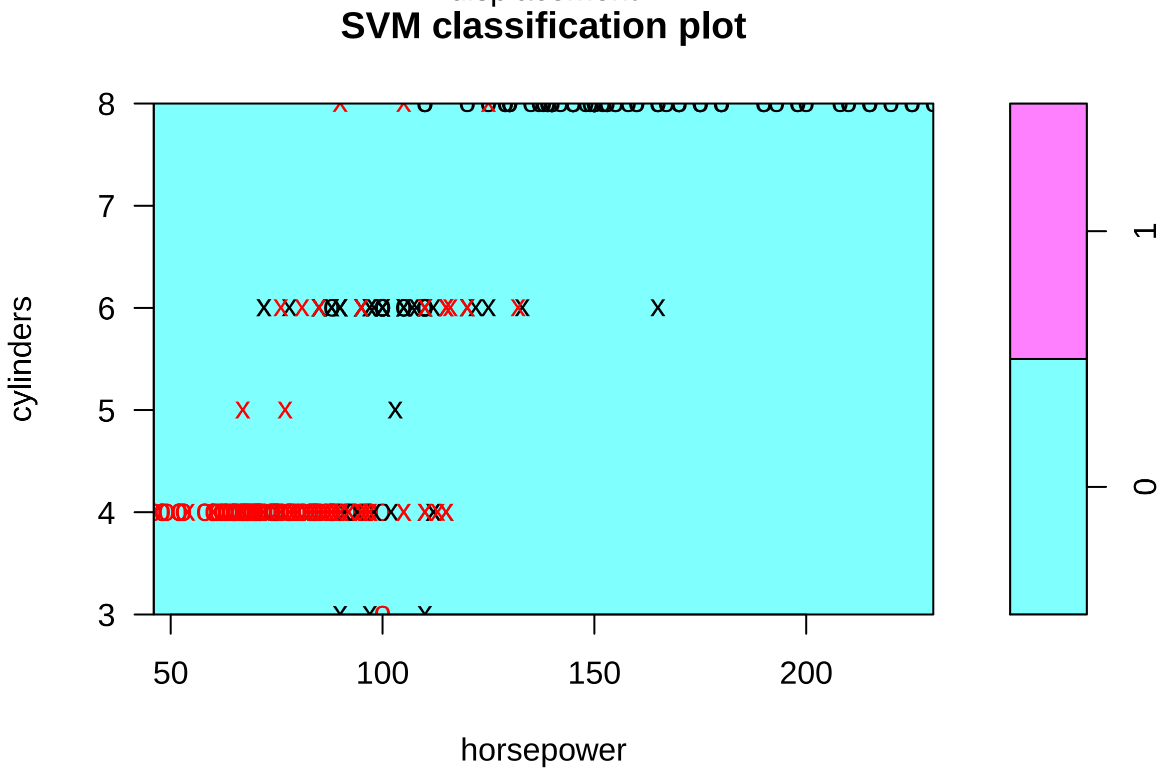
```
plot(svmfit , dat , x1~x4)
```

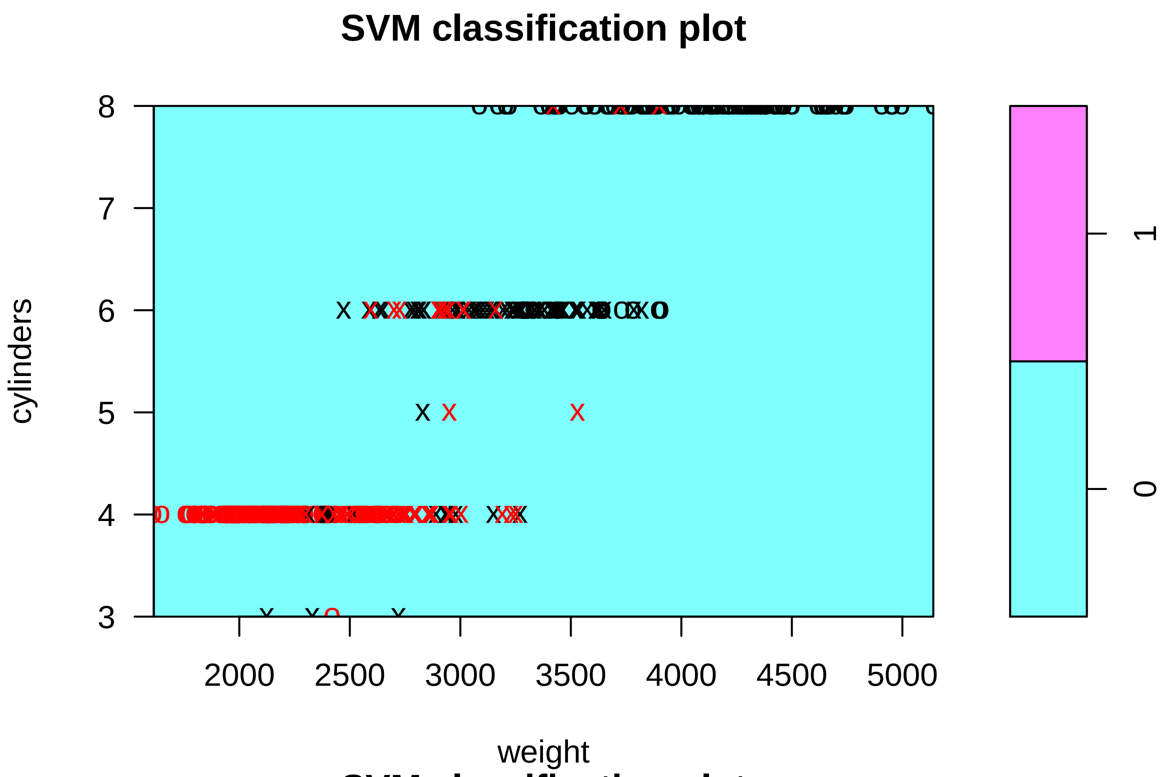
in order to plot just the first and fourth variables. However, you must replace `x1` and `x4` with the correct variable names. To find out more, type `?plot.svm`.

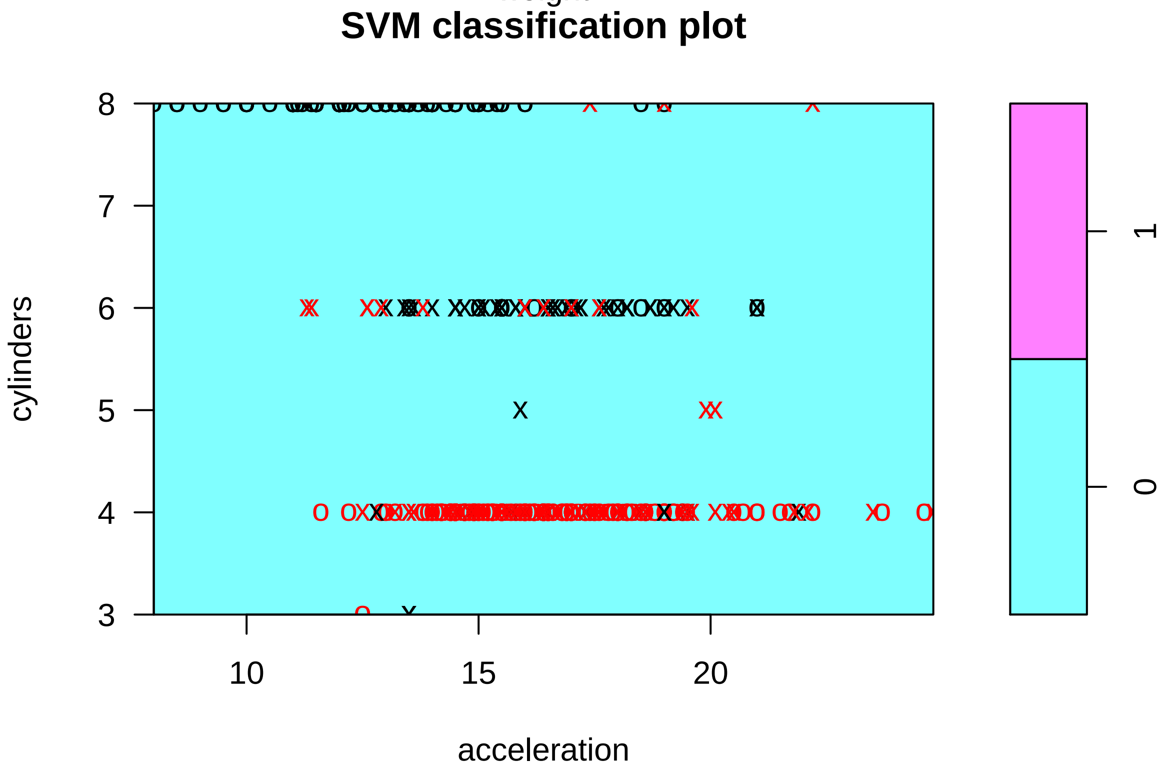
Solution:

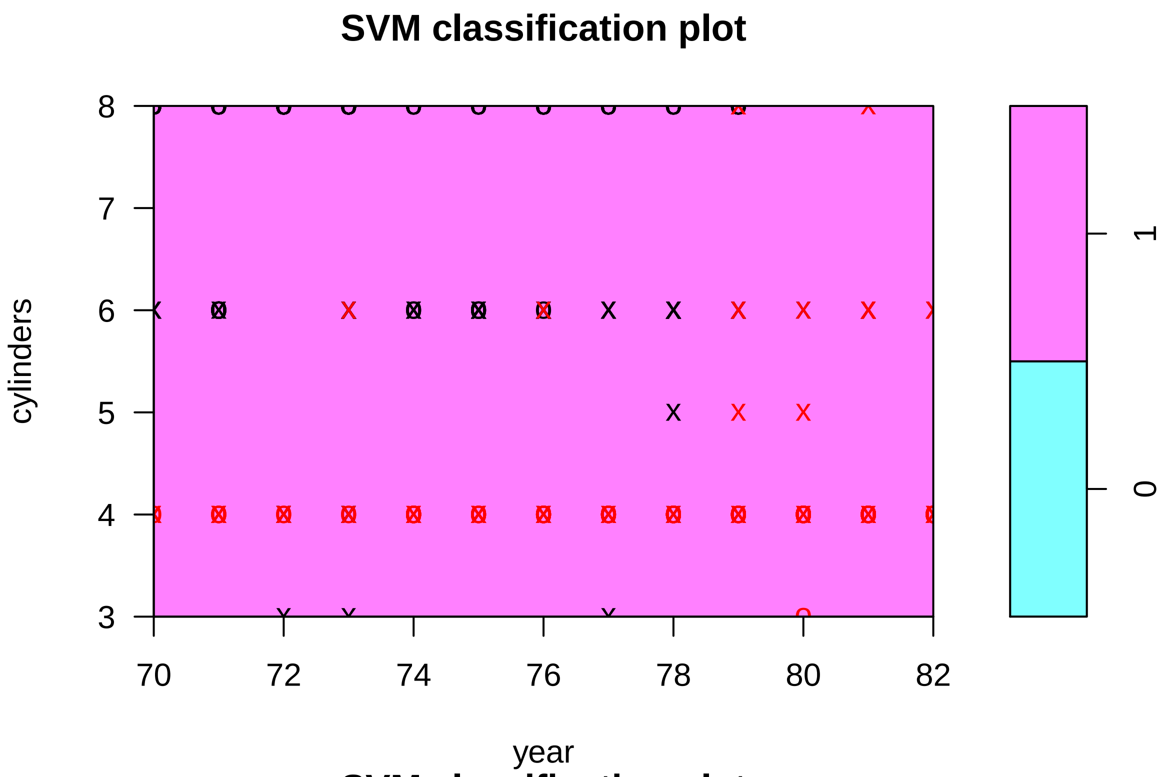
```
# names(Auto)
# [1] "cylinders"    "displacement" "horsepower"   "weight"       "acceleration"
# [6] "year"         "origin"       "name"         "mpg01"

# In the function below, i and j are the indices of names(Auto)
# We exclude "name" and "mpg01", as "name" is a categorical predictor and "mpg01" is the response. That

# The nested for-loop builds up pairs from predictors in names(Auto). The condition (i<j) gurantees tha

# paste0(names(Auto)[i],"~", names(Auto)[j])) function concatenate the string names(Auto)[i] with "~" a

# as.formula() function turns a string to a formula.

plotpairs <- function(svmfit){
  for (i in 1:7){
    for (j in 1:7){
      if (i<j){
        plot(svmfit, Auto, as.formula(paste0(names(Auto)[i],"~", names(Auto)[j])))
      }
    }
  }
}


#Linear kernel
plotpairs(tune.out.linear$best.model)

#Radial kernel
plotpairs(tune.out.radial$best.model)

#Polynomial kernel
plotpairs(tune.out.poly$best.model)
```

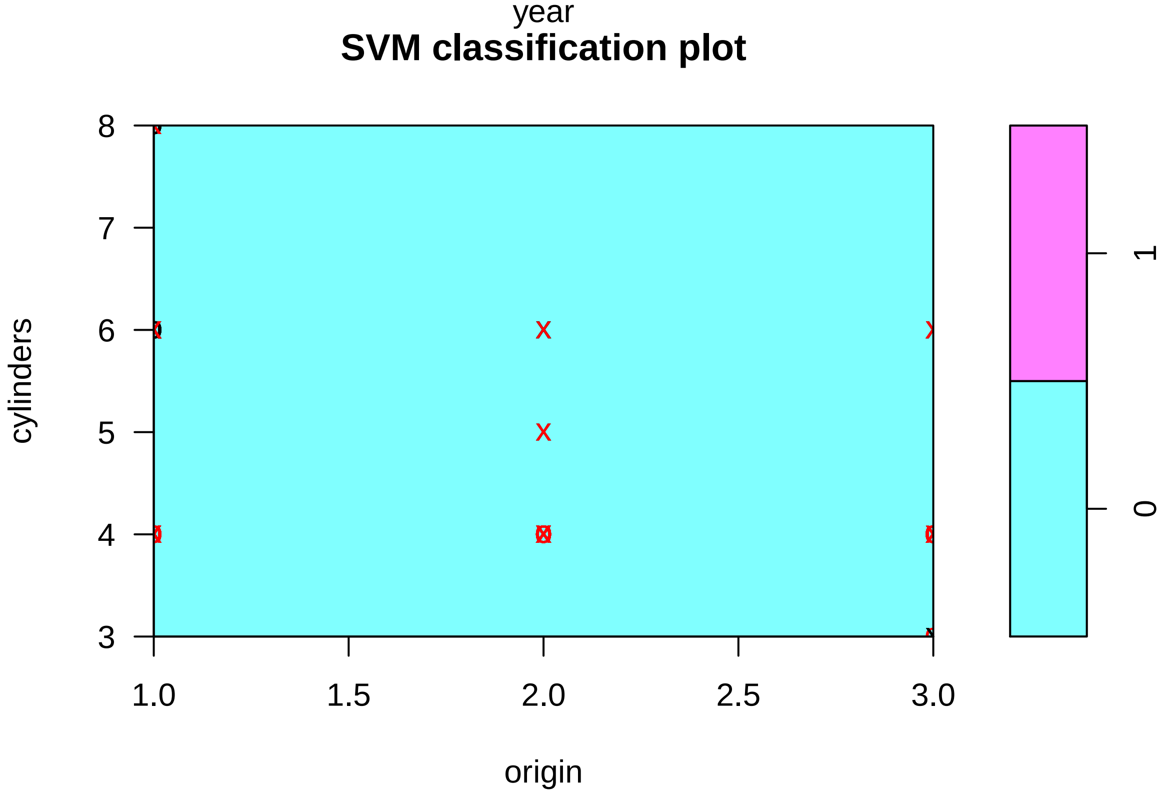

SVM classification plot
cylinders
8
7
6
5
4
3
100
150
200
250
300
350
400
450
displacement
1
0

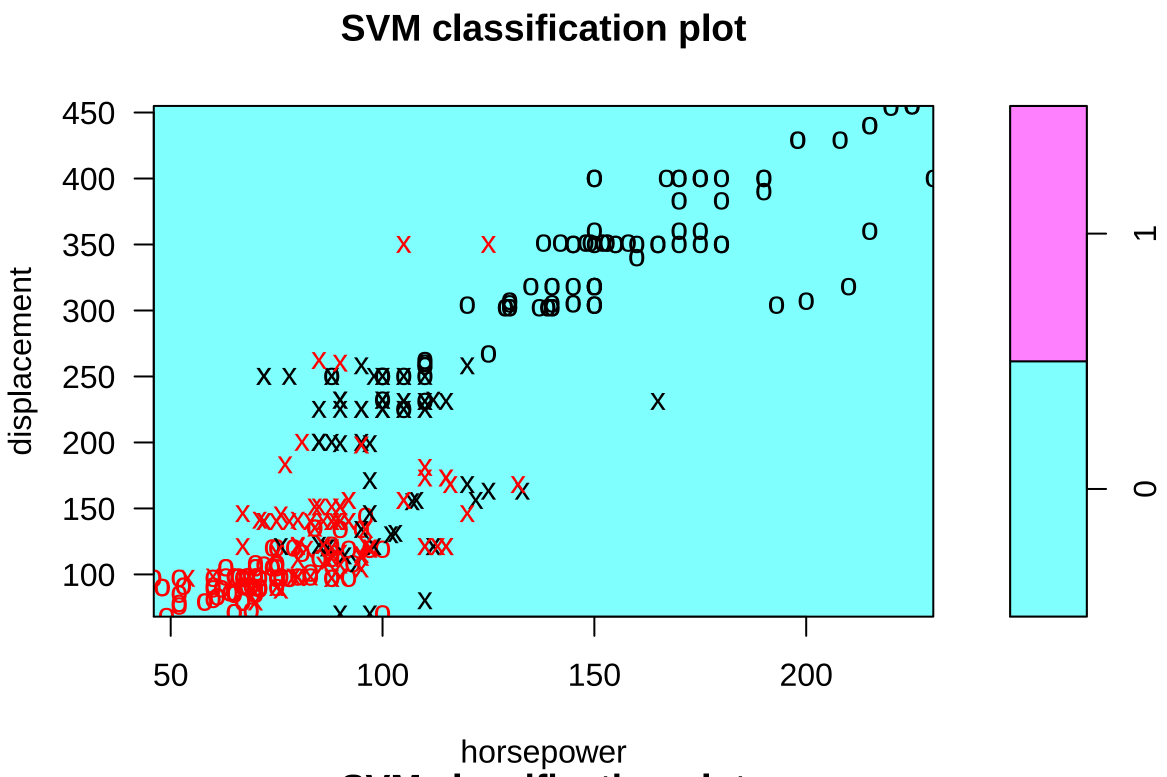

SVM classification plot
cylinders
8
7
6
5
4
3
50
100
150
200
horsepower
1
0


SVM classification plot
cylinders
weight
8
7
6
5
4
3
2000
2500
3000
3500
4000
4500
5000
1
0

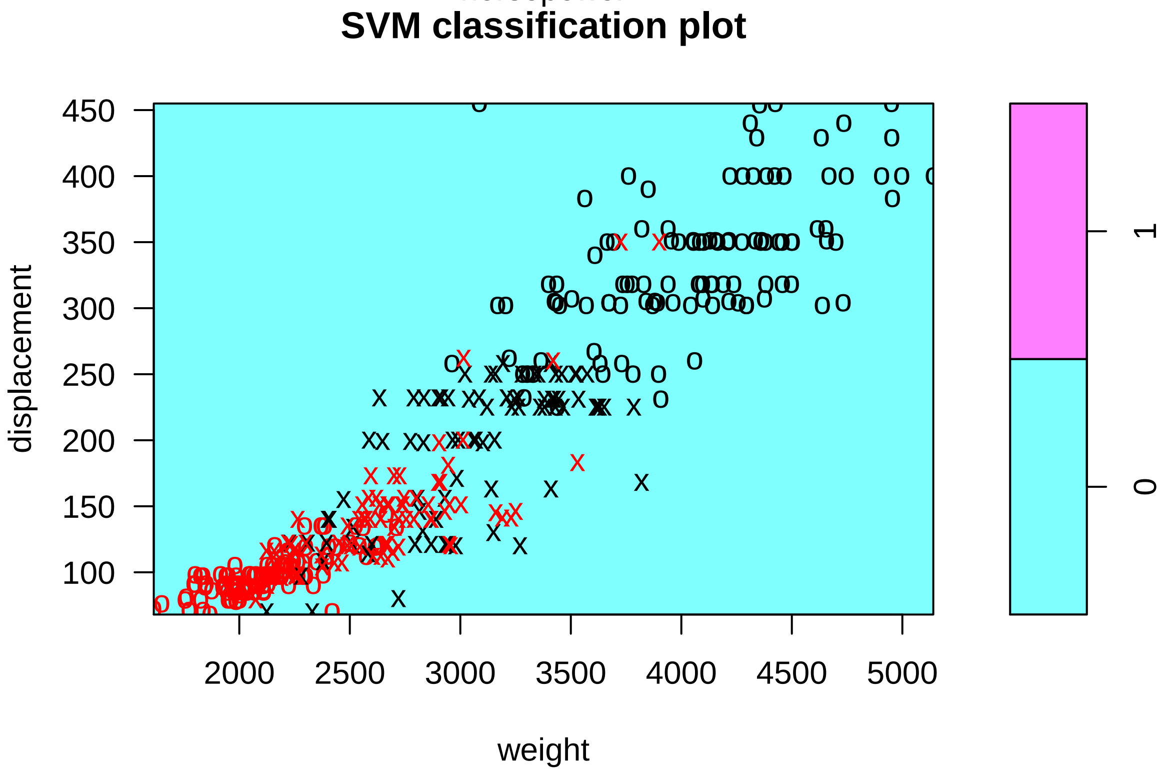

SVM classification plot
cylinders
acceleration
8
7
6
5
4
3
10
15
20
1
0


SVM classification plot
cylinders
year
70
72
74
76
78
80
82
3
4
5
6
7
8
0
1

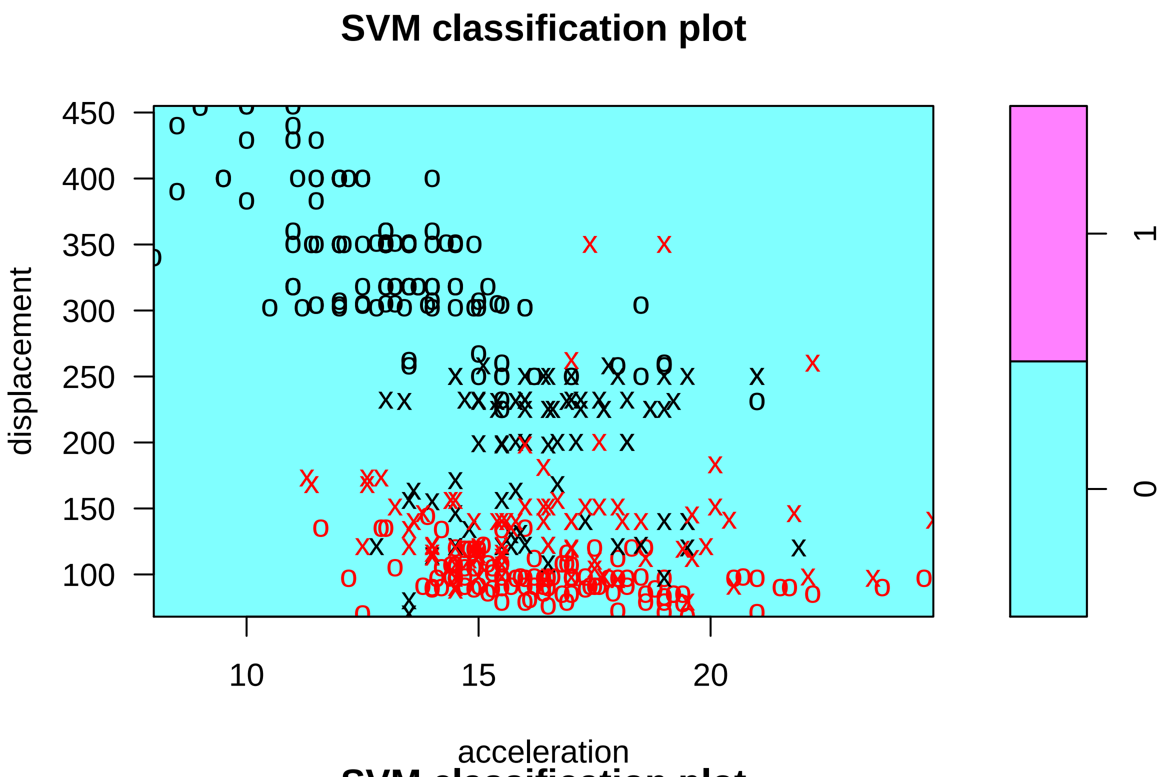

SVM classification plot
cylinders
origin
1.0
1.5
2.0
2.5
3.0
3
4
5
6
7
8
0
1

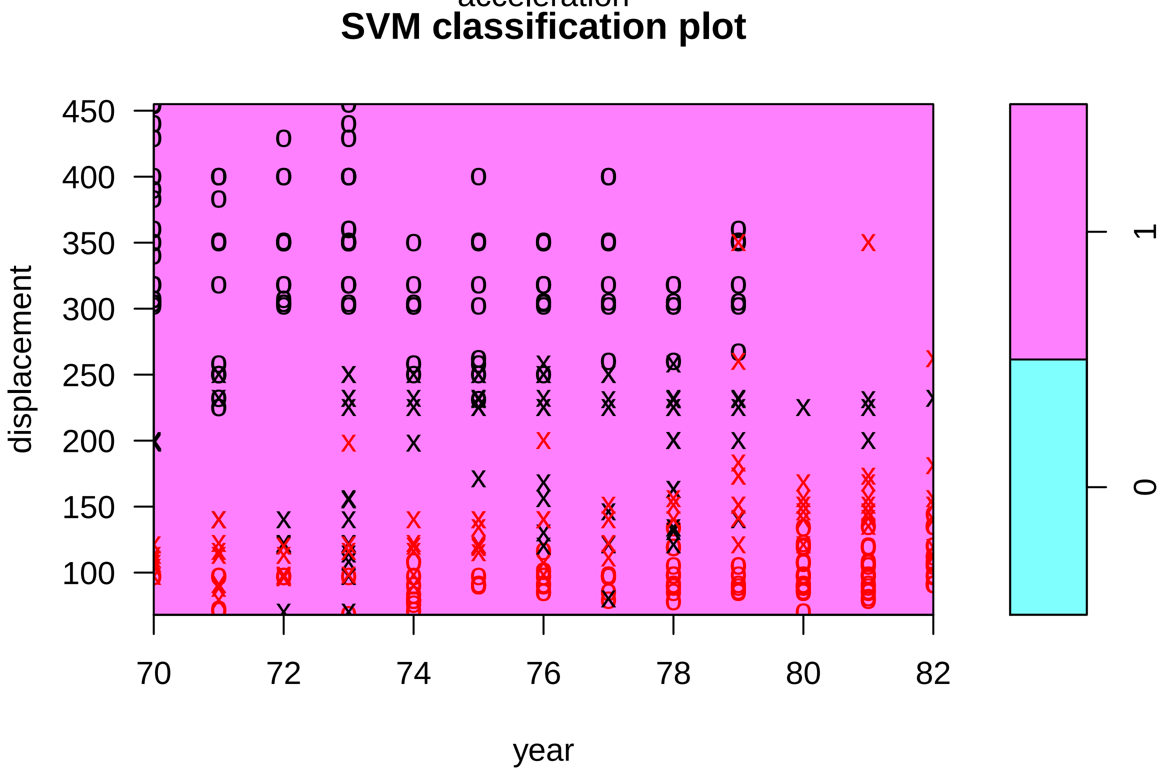


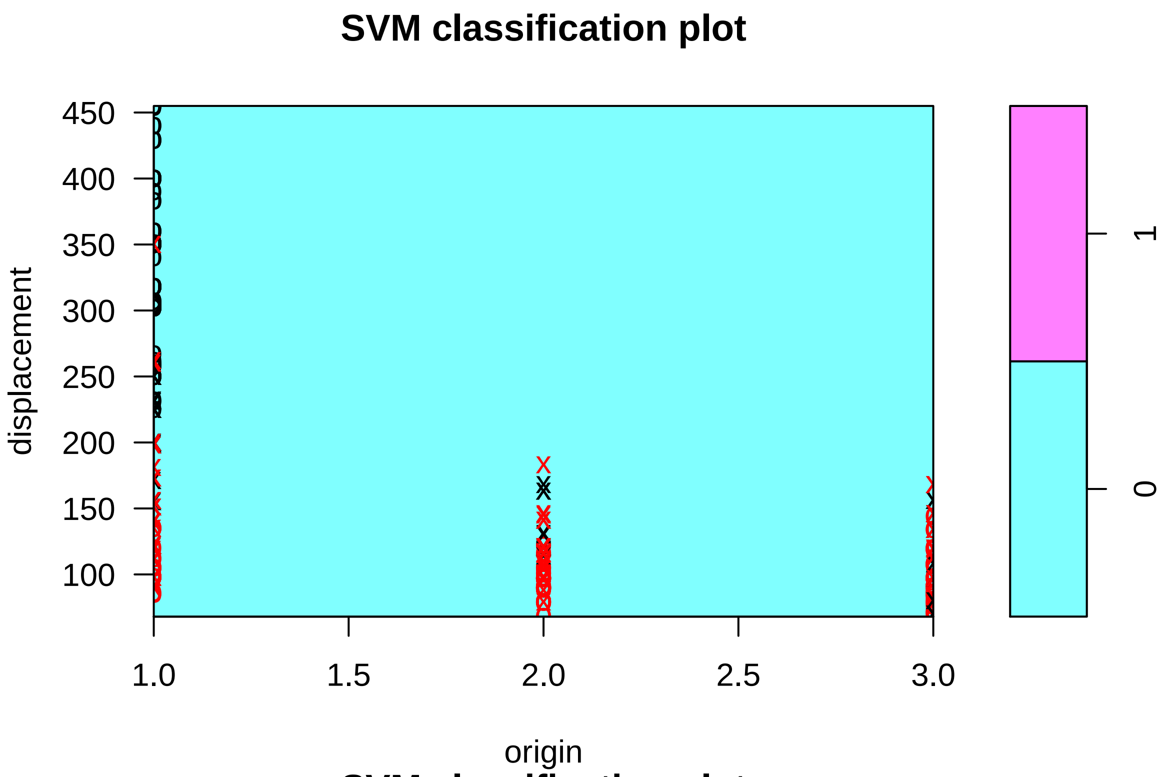
SVM classification plot
displacement
horsepower
450
400
350
300
250
200
150
100
50
100
150
200
1
0

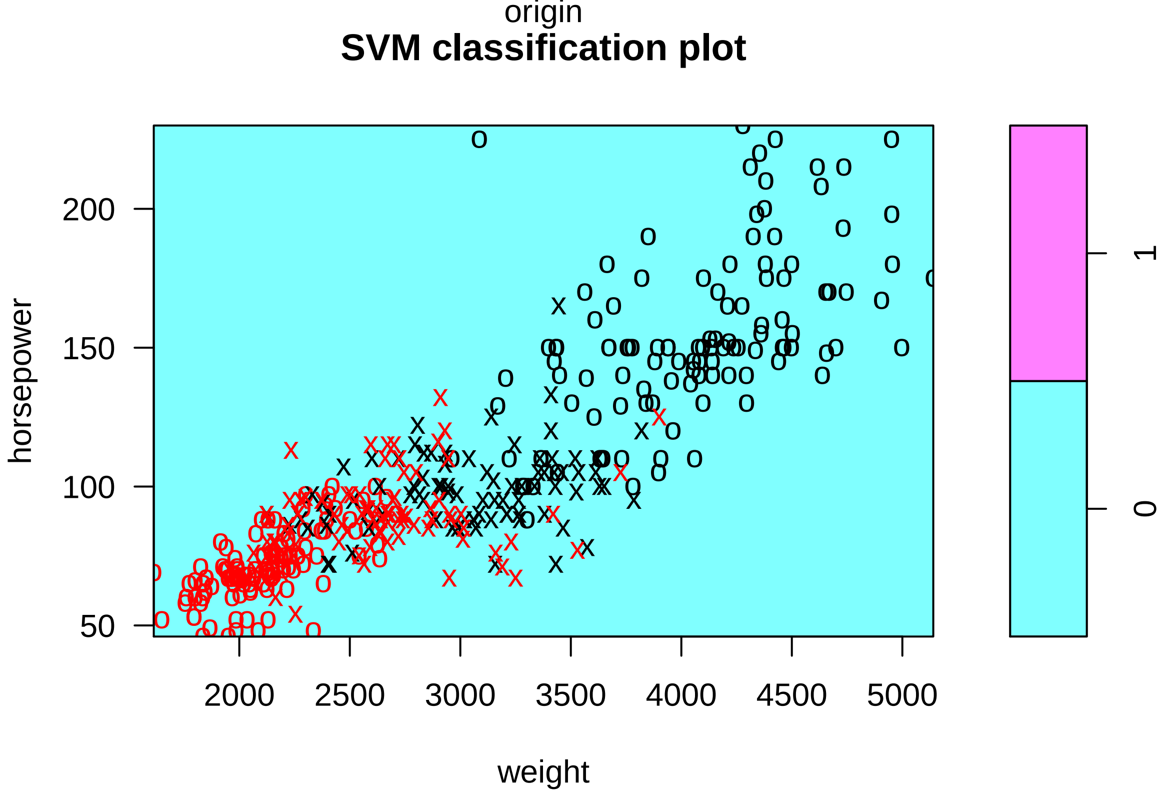


SVM classification plot
displacement
weight
450
400
350
300
250
200
150
100
2000
2500
3000
3500
4000
4500
5000
1
0

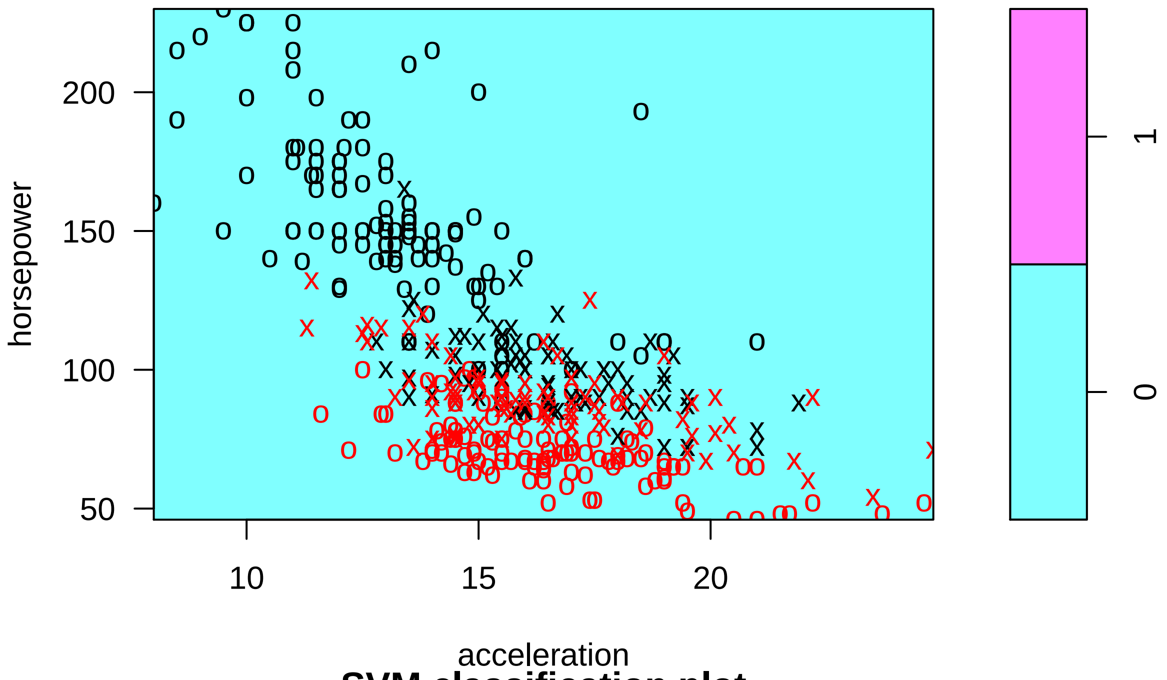

SVM classification plot
displacement
acceleration
450
400
350
300
250
200
150
100
10
15
20
1
0


SVM classification plot
displacement
year
450
400
350
300
250
200
150
100
70
72
74
76
78
80
82
1
0

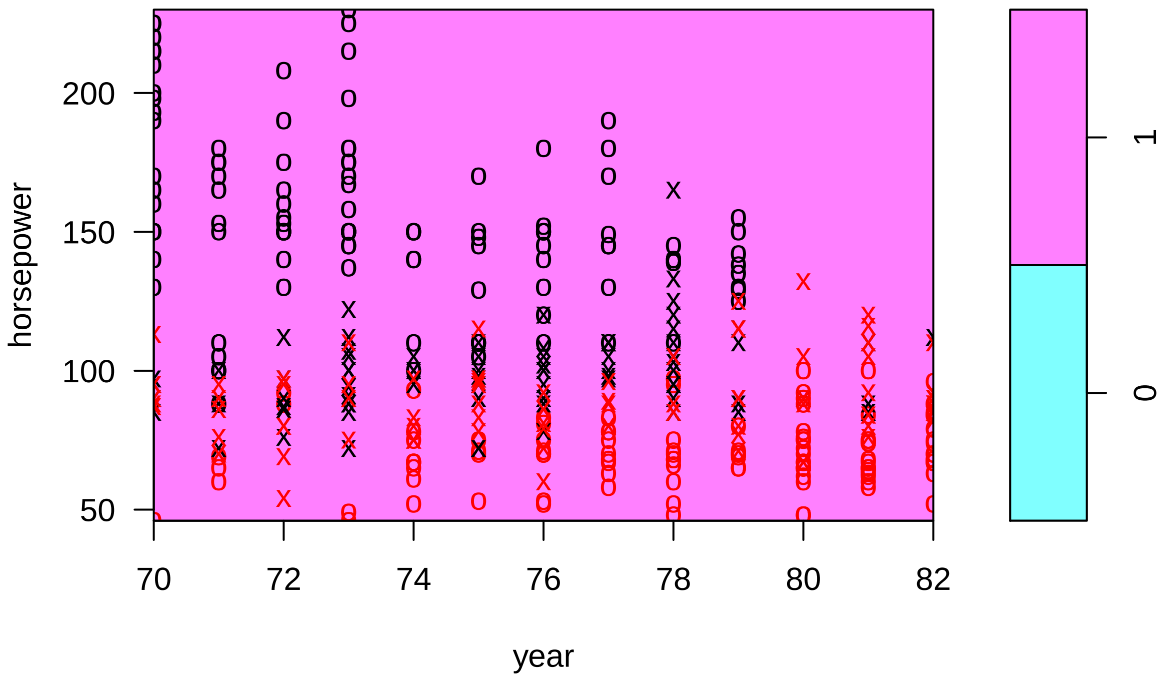

SVM classification plot
displacement
origin
450
400
350
300
250
200
150
100
1.0
1.5
2.0
2.5
3.0
1
0

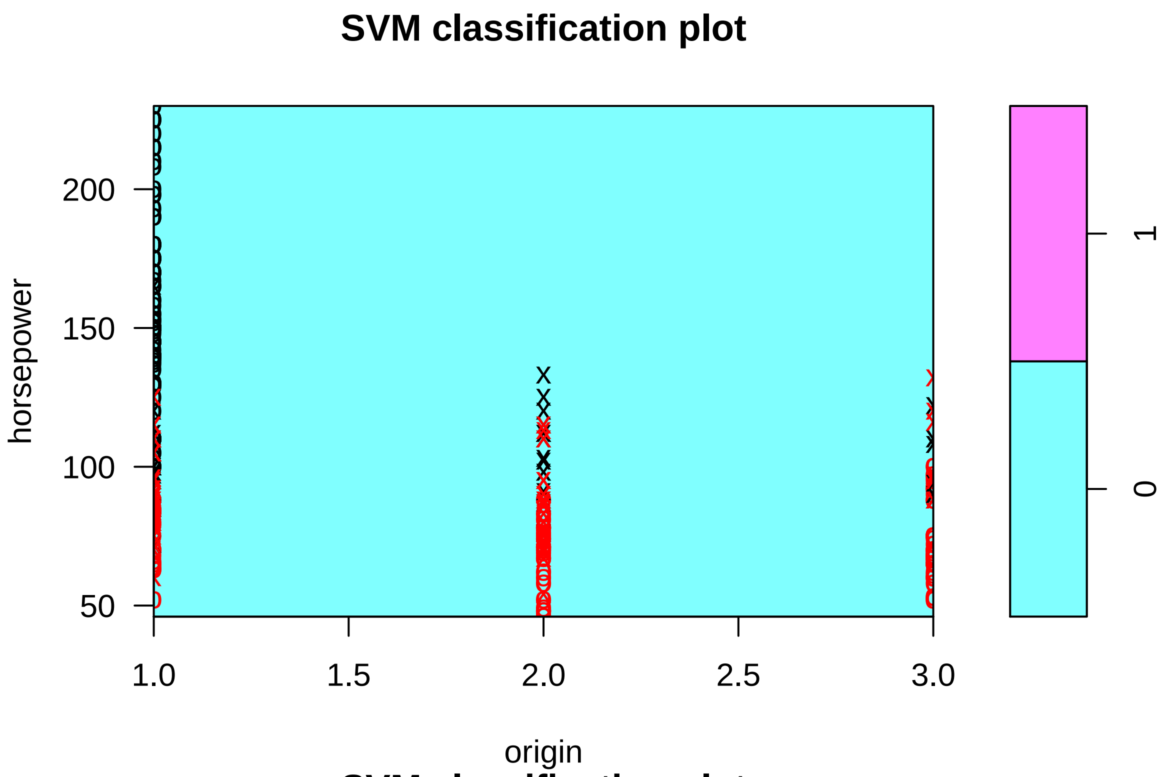

SVM classification plot
horsepower
weight
200
150
100
50
2000
2500
3000
3500
4000
4500
5000
1
0

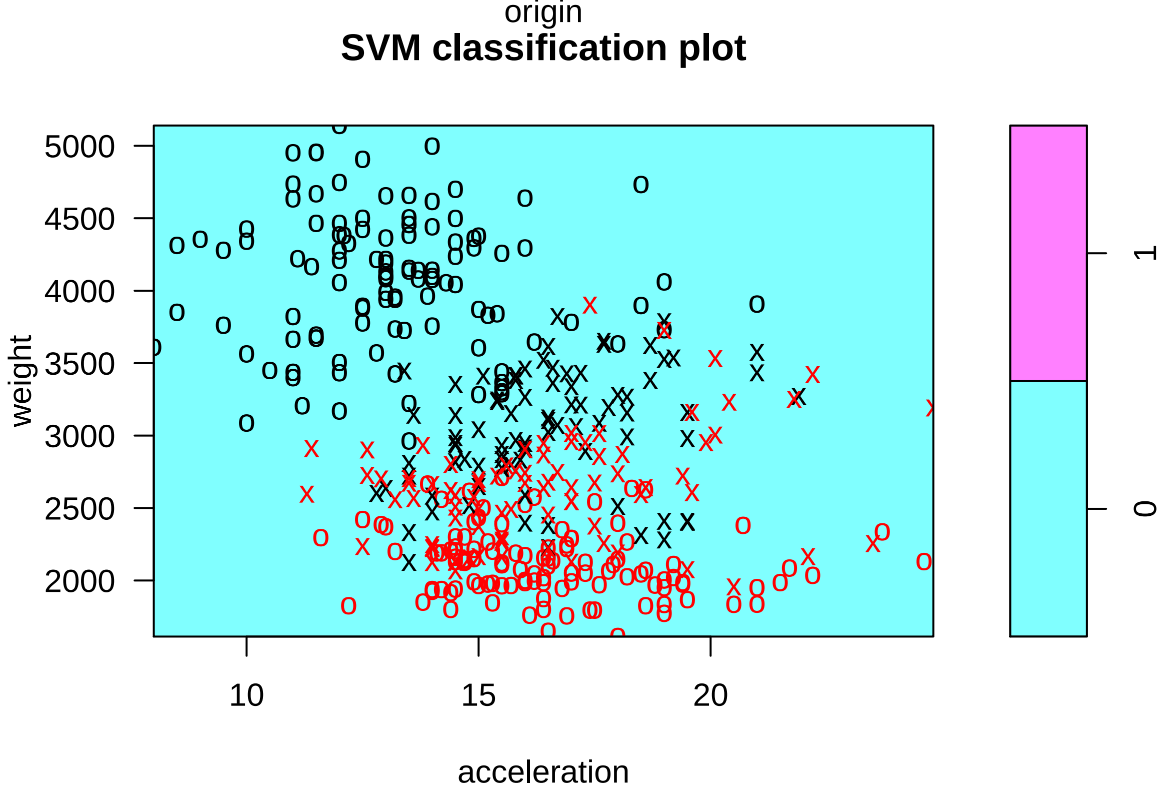
SVM classification plot

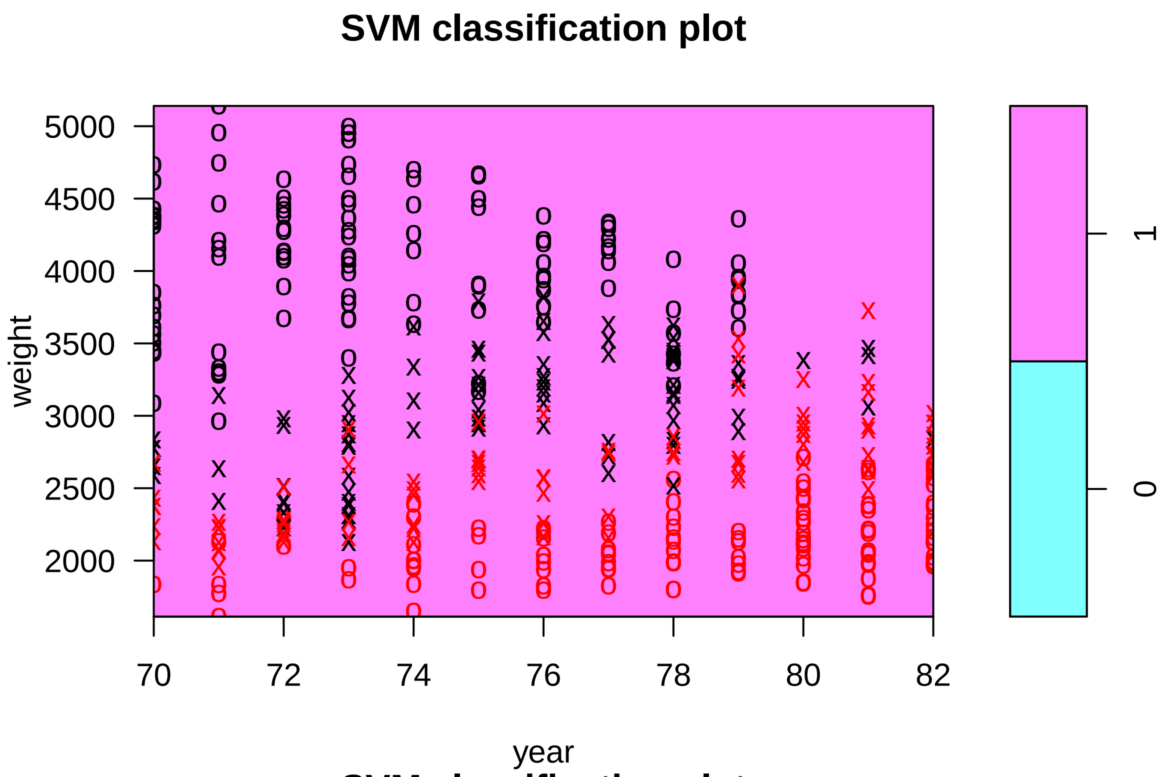


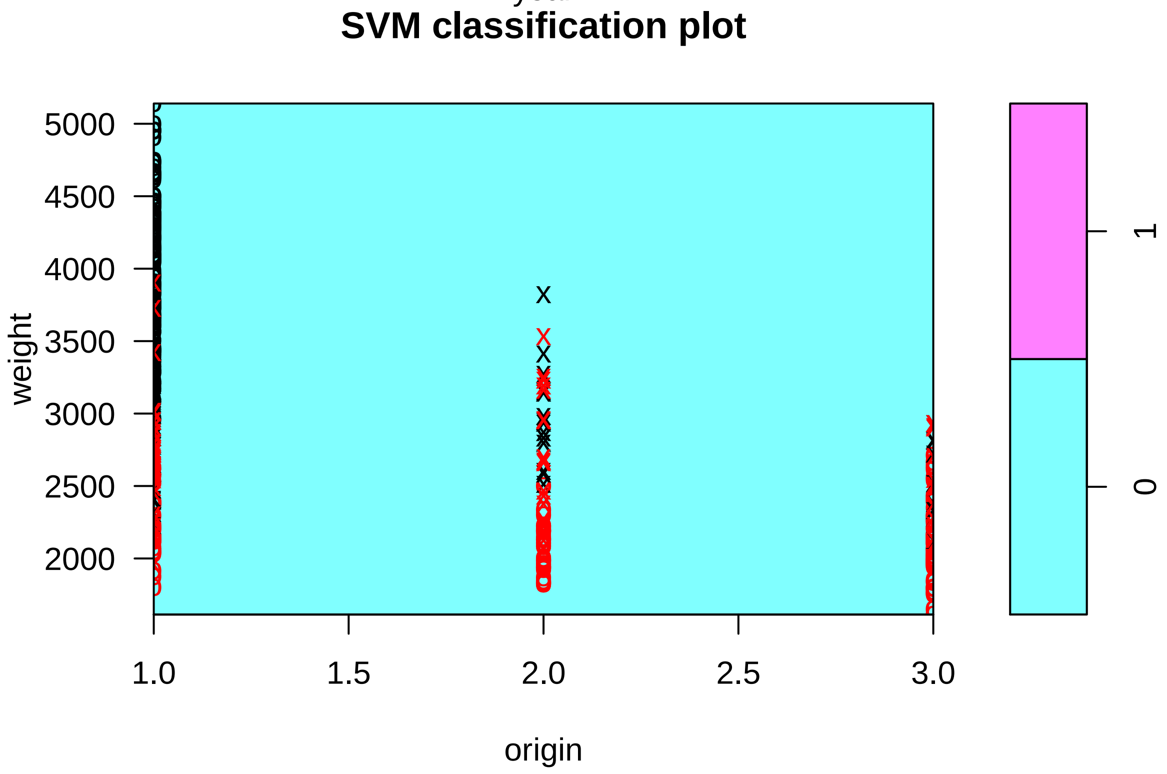
SVM classification plot



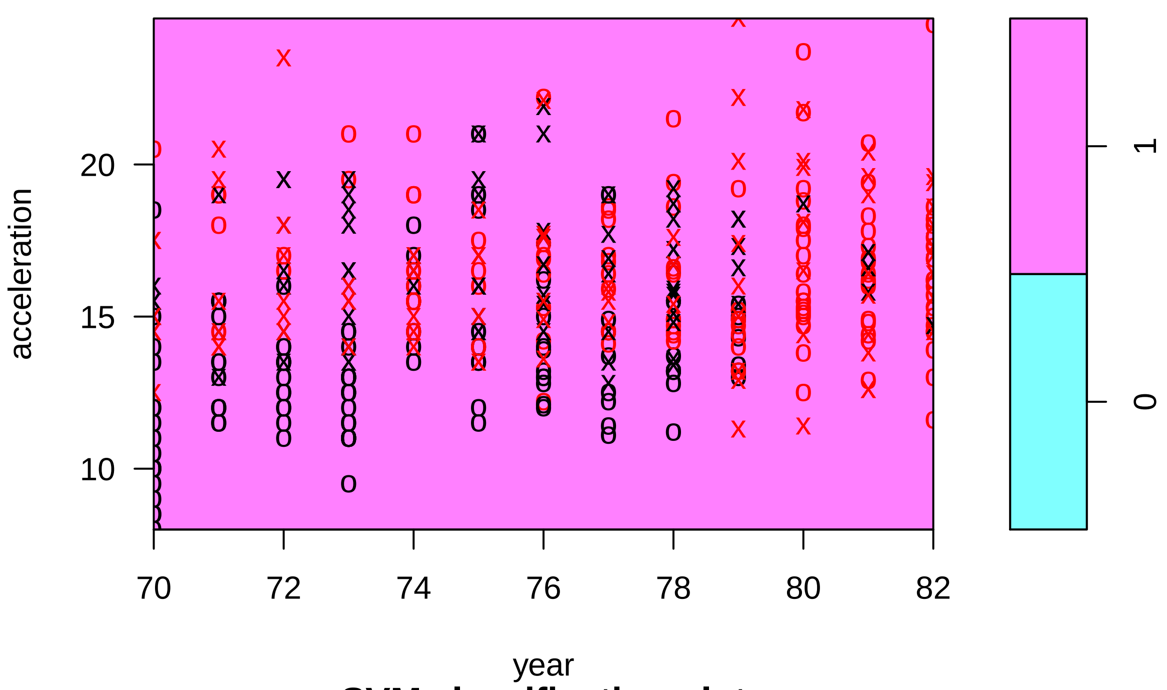

SVM classification plot
horsepower
200
150
100
50
1.0
1.5
2.0
2.5
3.0
origin
1
0


SVM classification plot
weight
5000
4500
4000
3500
3000
2500
2000
10
15
20
acceleration
1
0

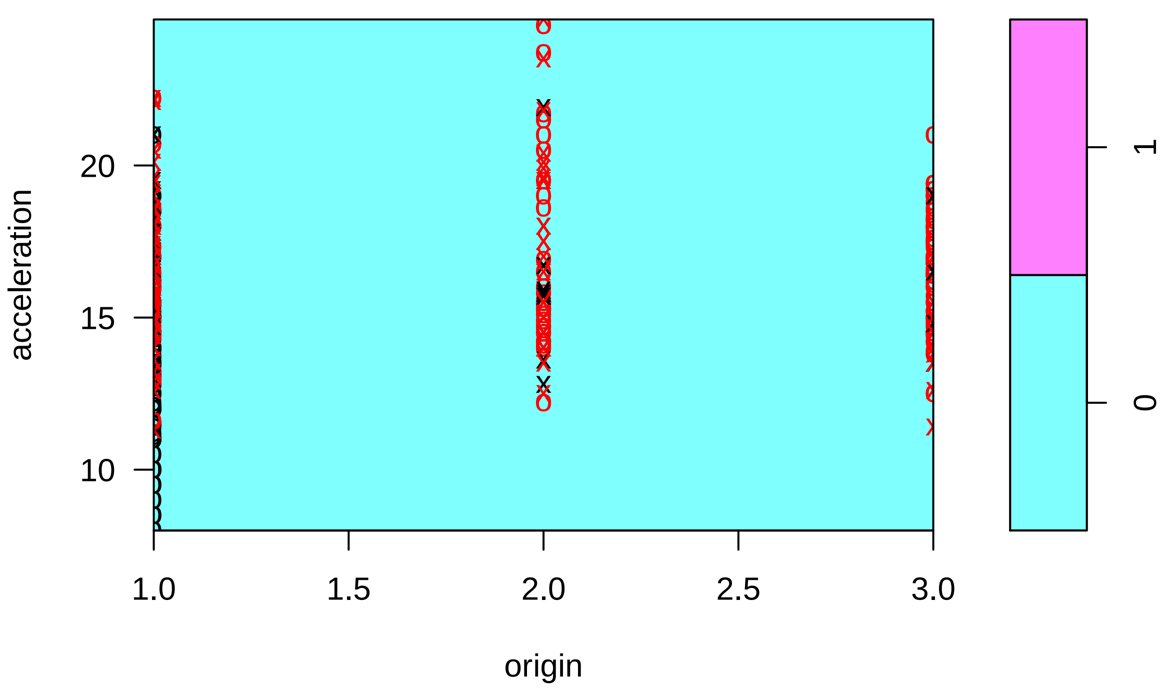

SVM classification plot
weight
year
5000
4500
4000
3500
3000
2500
2000
70
72
74
76
78
80
82
1
0

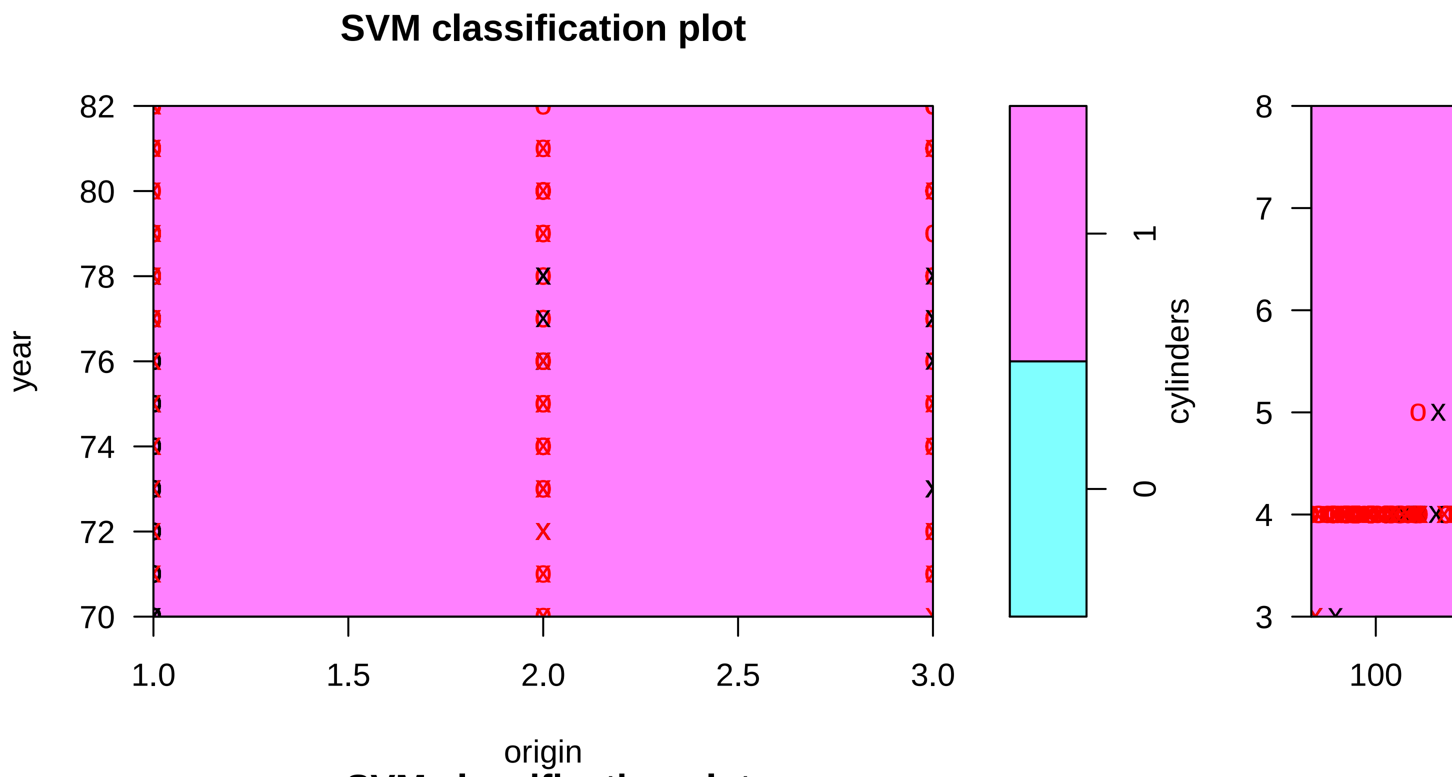

SVM classification plot
weight
origin
5000
4500
4000
3500
3000
2500
2000
1.0
1.5
2.0
2.5
3.0
1
0

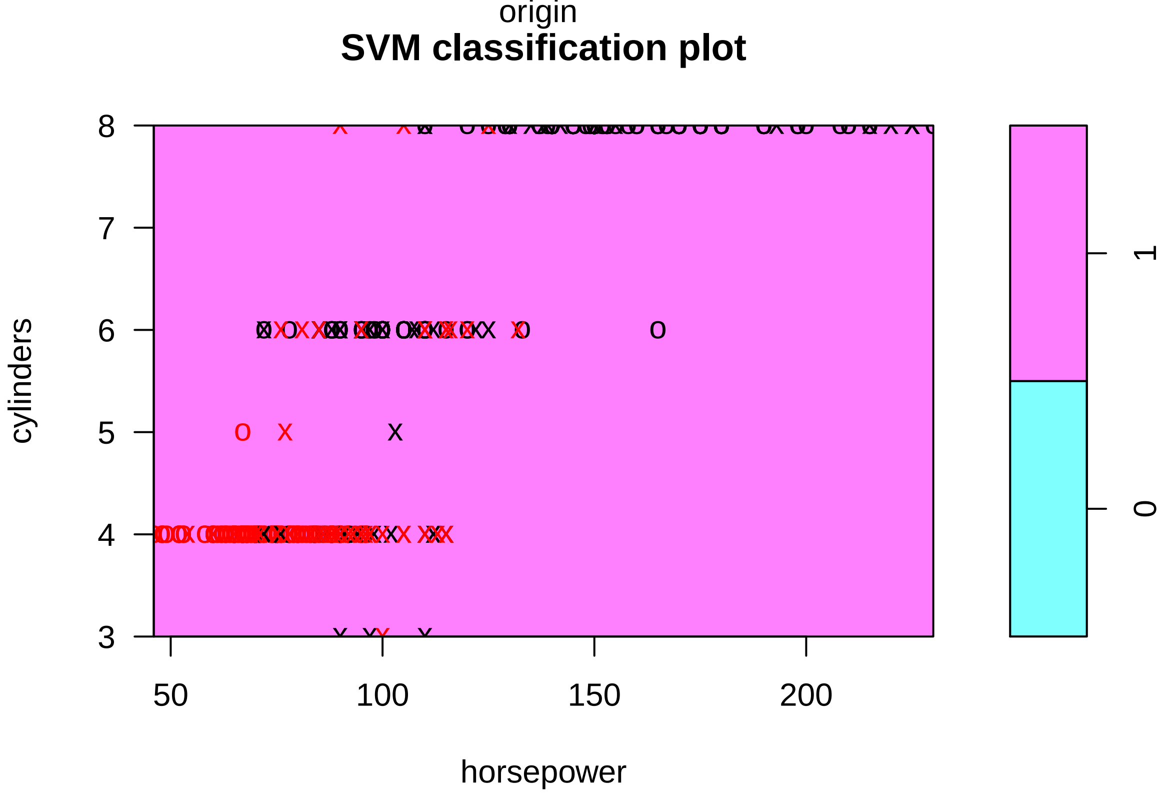
SVM classification plot

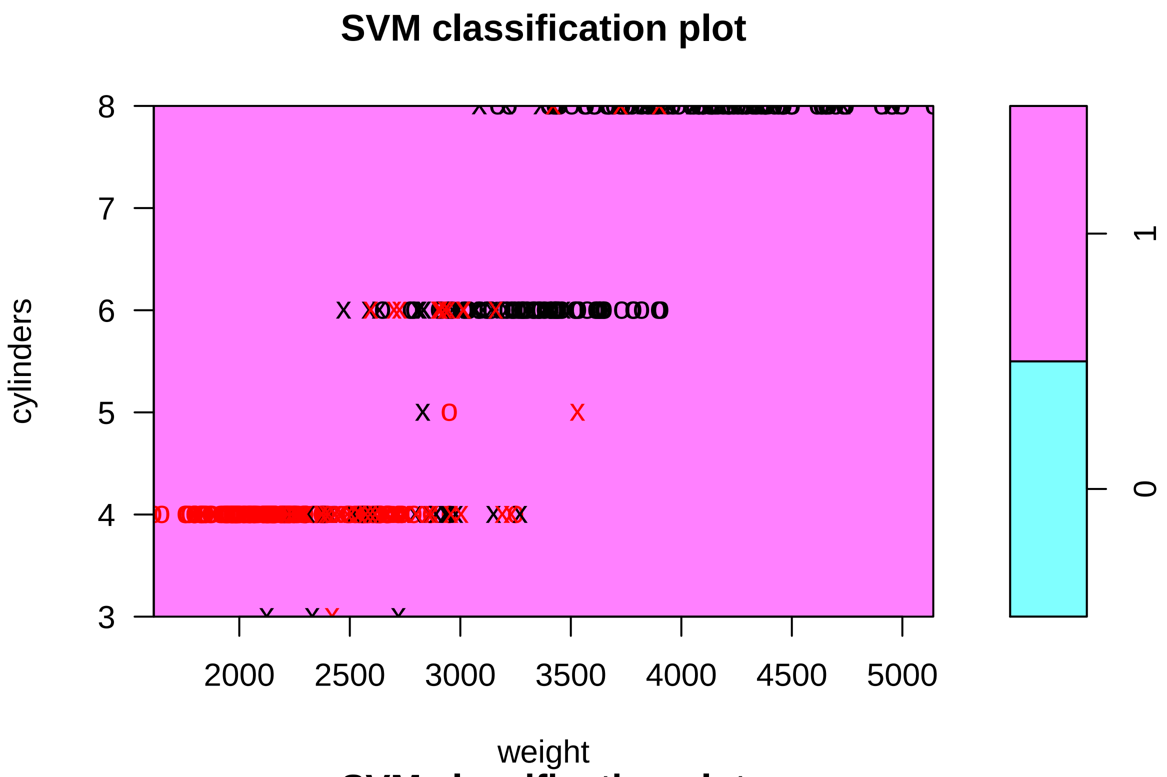


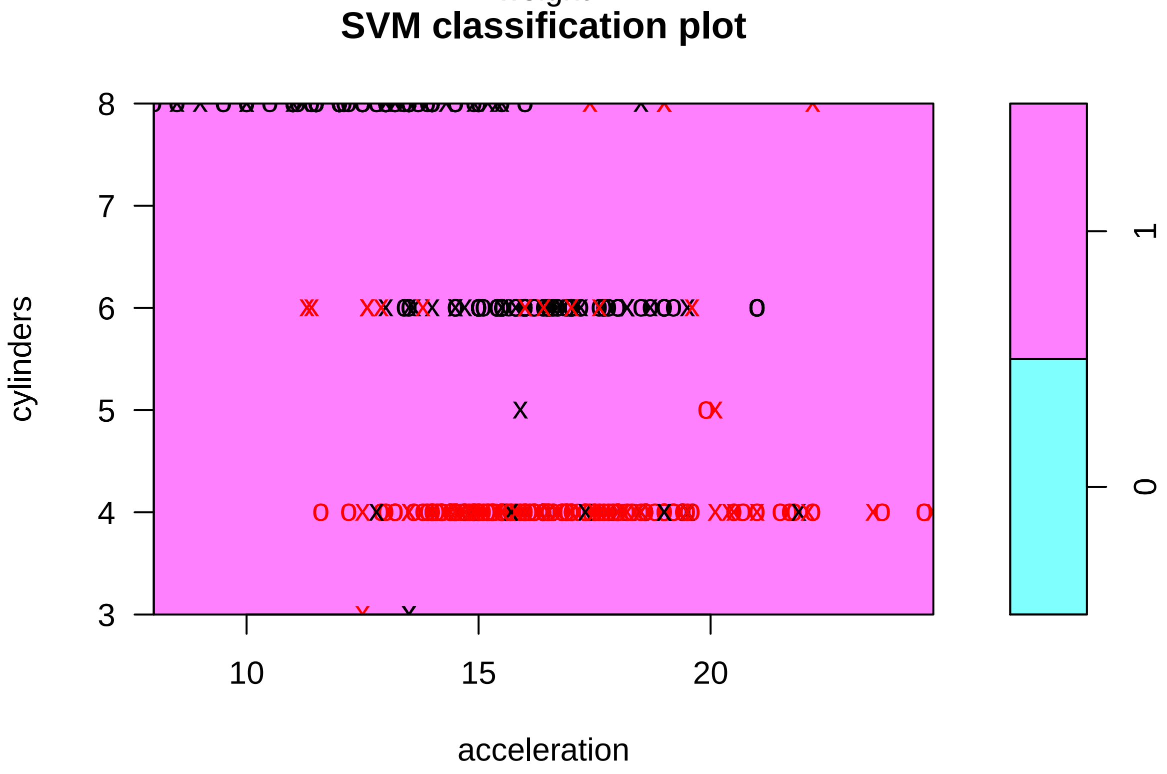
SVM classification plot

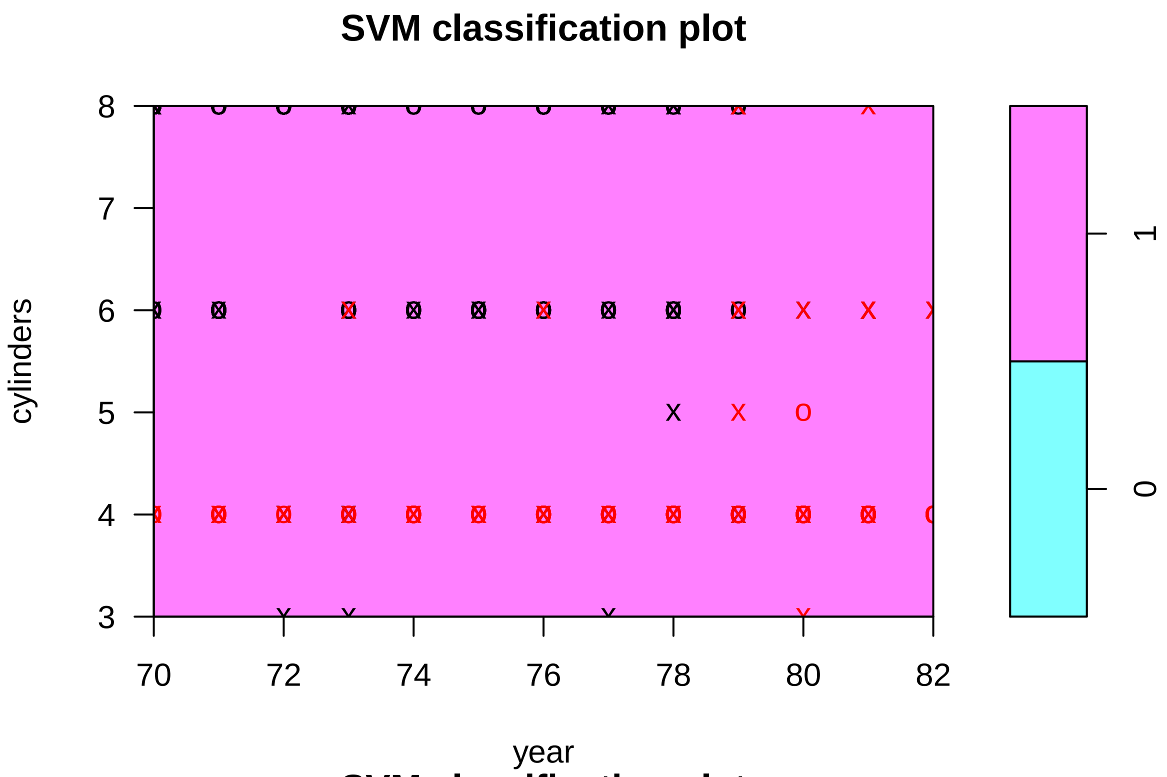


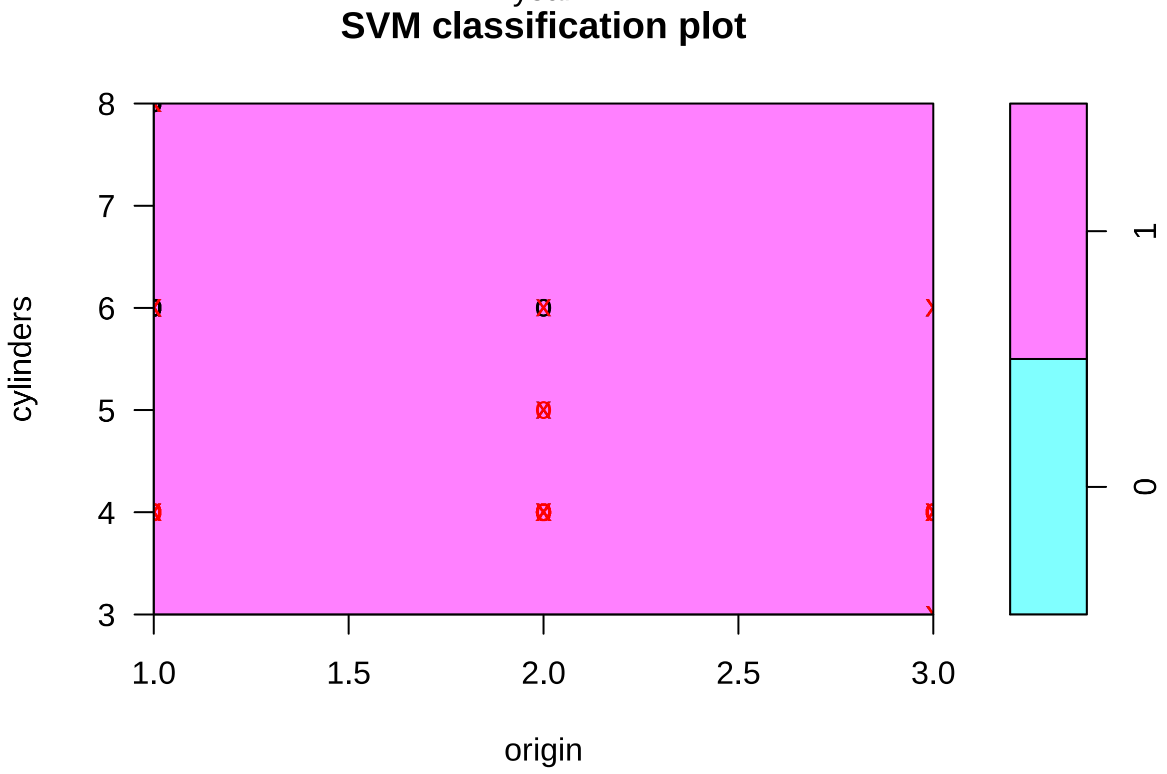

SVM classification plot
year
82
80
78
76
74
72
70
1.0
1.5
2.0
2.5
3.0
origin
1
0
cylinders
8
7
6
5
4
3
100

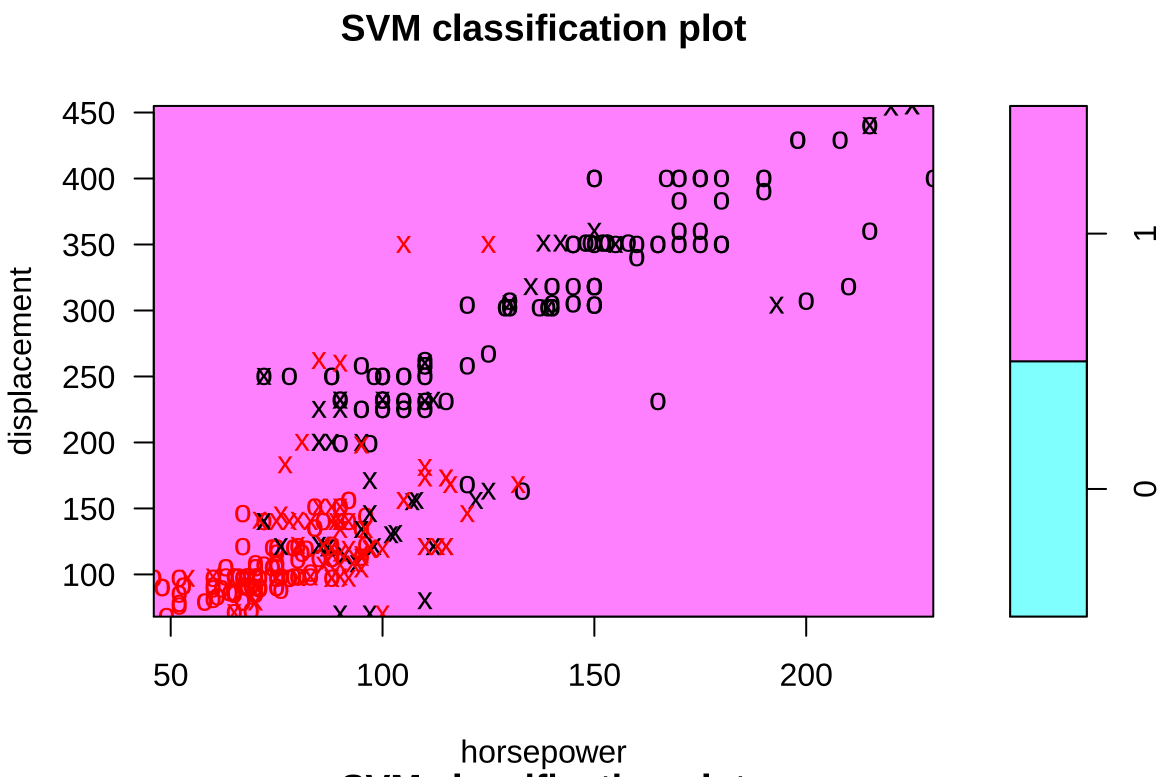

SVM classification plot
cylinders
8
7
6
5
4
3
50
100
150
200
horsepower
1
0

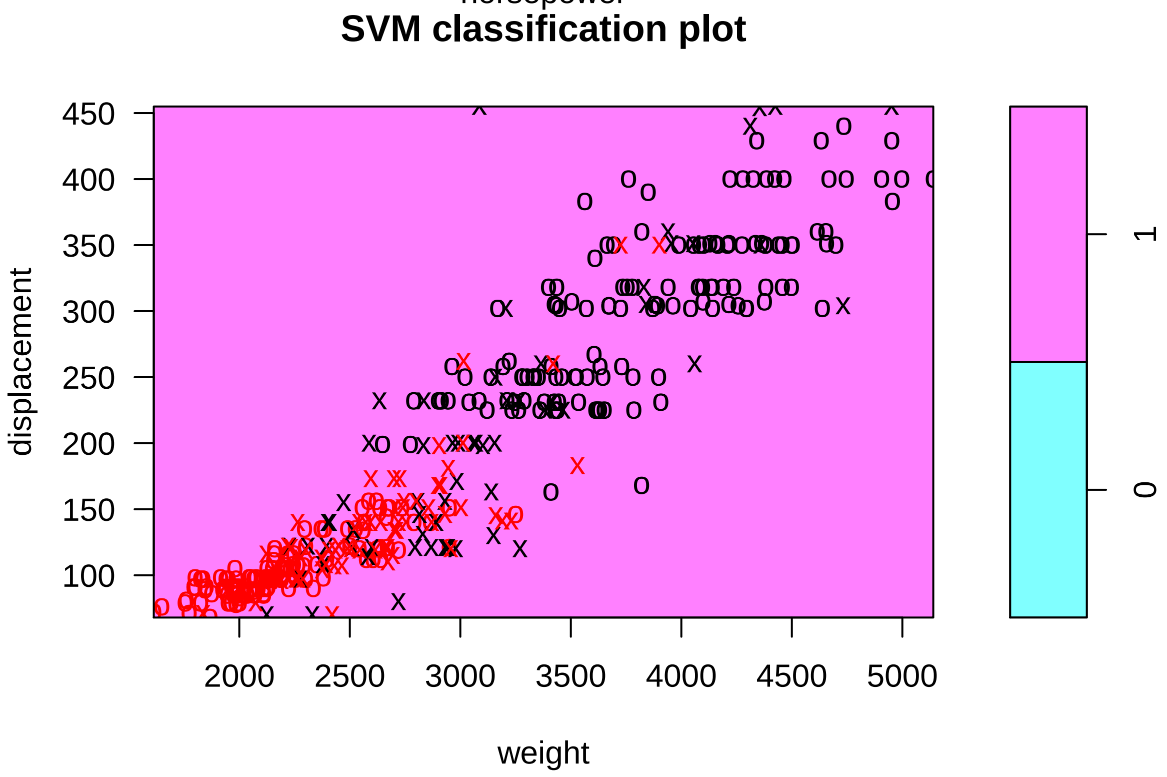

SVM classification plot
cylinders
weight
8
7
6
5
4
3
2000
2500
3000
3500
4000
4500
5000
1
0

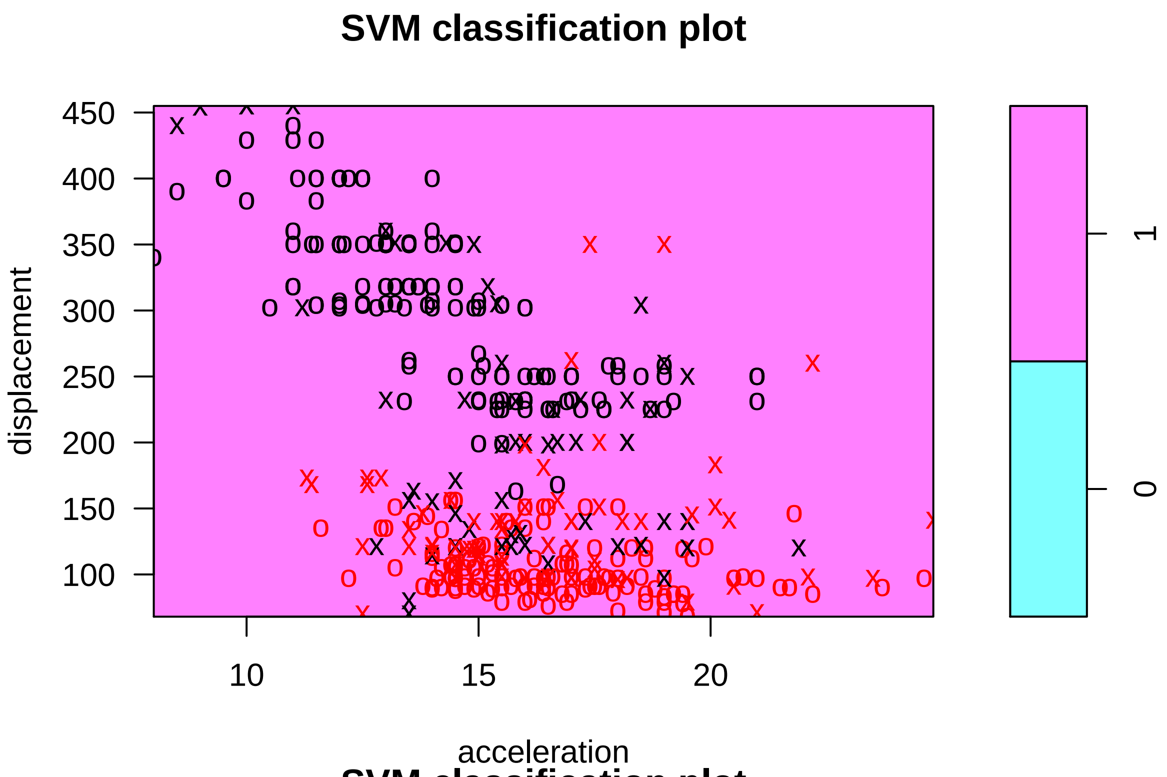

SVM classification plot
cylinders
acceleration
8
7
6
5
4
3
10
15
20
1
0

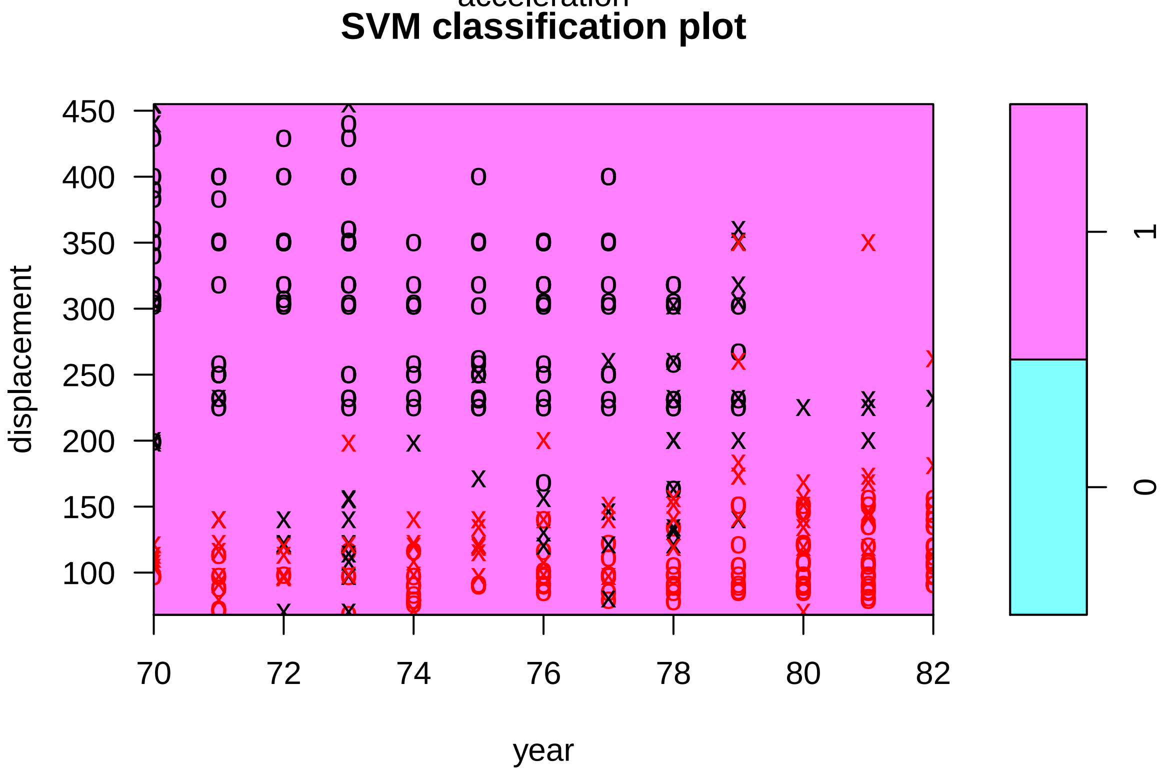

SVM classification plot
cylinders
year
0
1

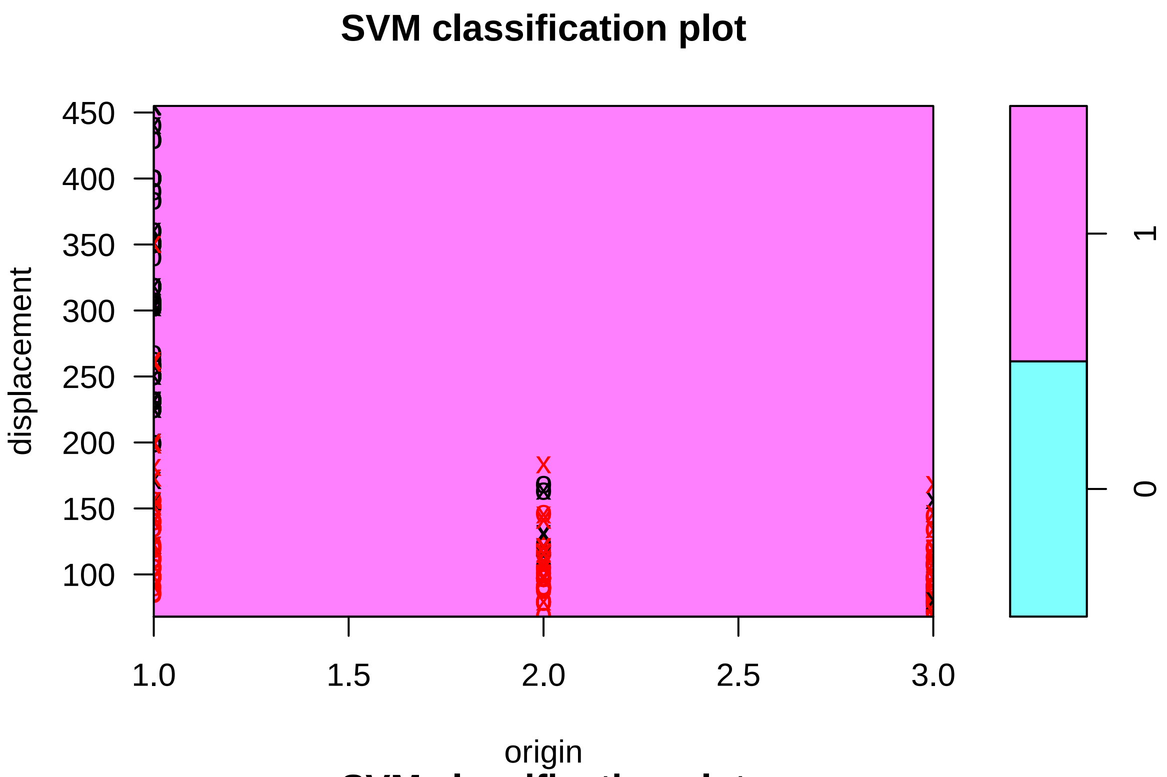

SVM classification plot
cylinders
origin
0
1

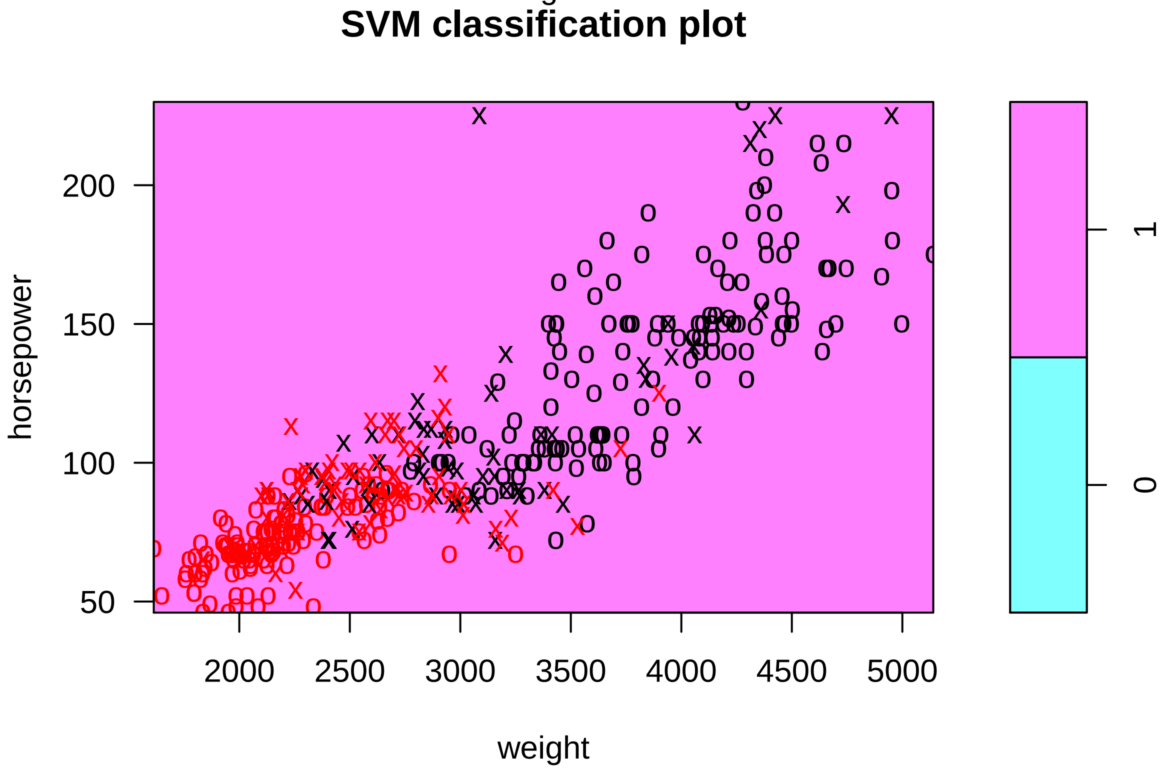

SVM classification plot
displacement
horsepower
450
400
350
300
250
200
150
100
50
100
150
200
1
0

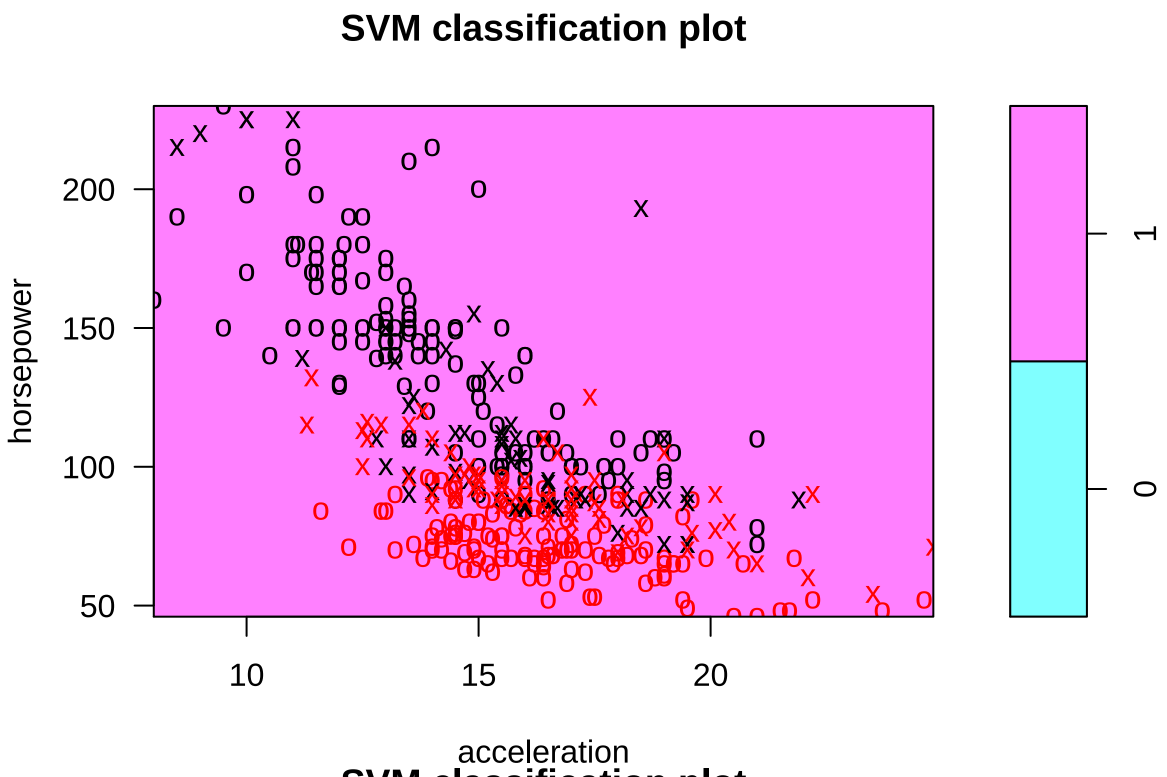

SVM classification plot
displacement
weight
450
400
350
300
250
200
150
100
2000
2500
3000
3500
4000
4500
5000
1
0

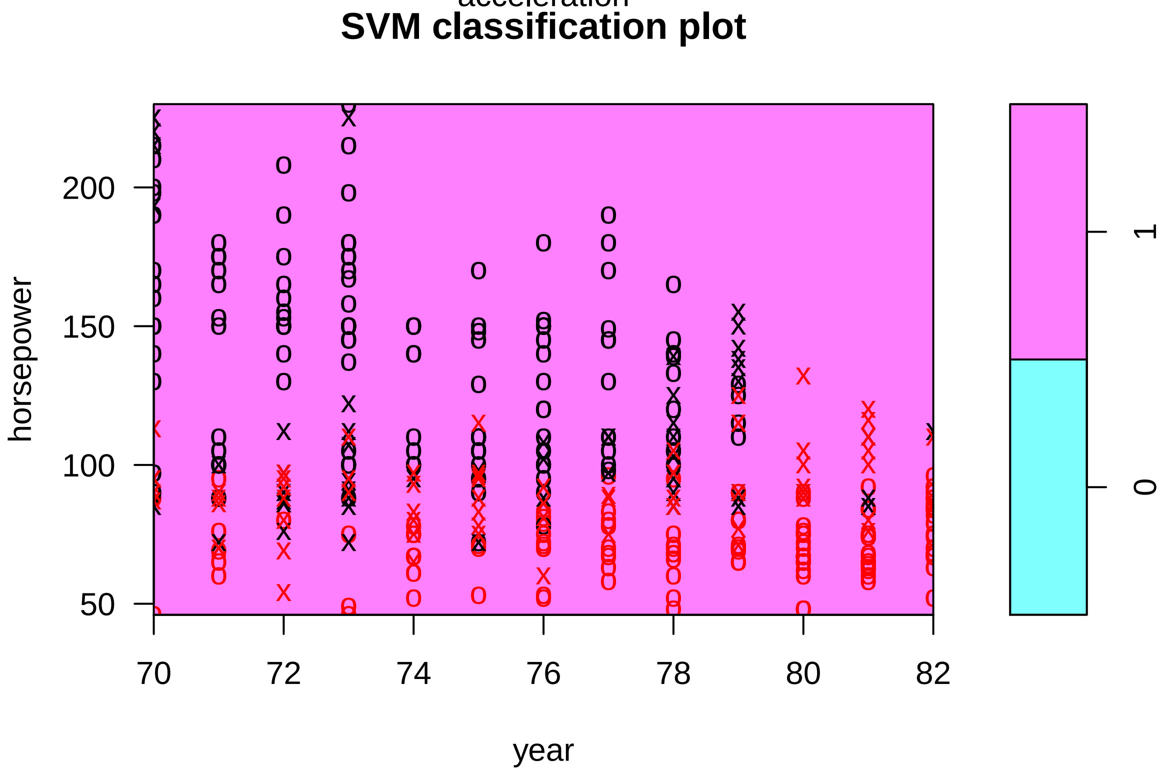

SVM classification plot
displacement
450
400
350
300
250
200
150
100
10
15
20
acceleration
1
0

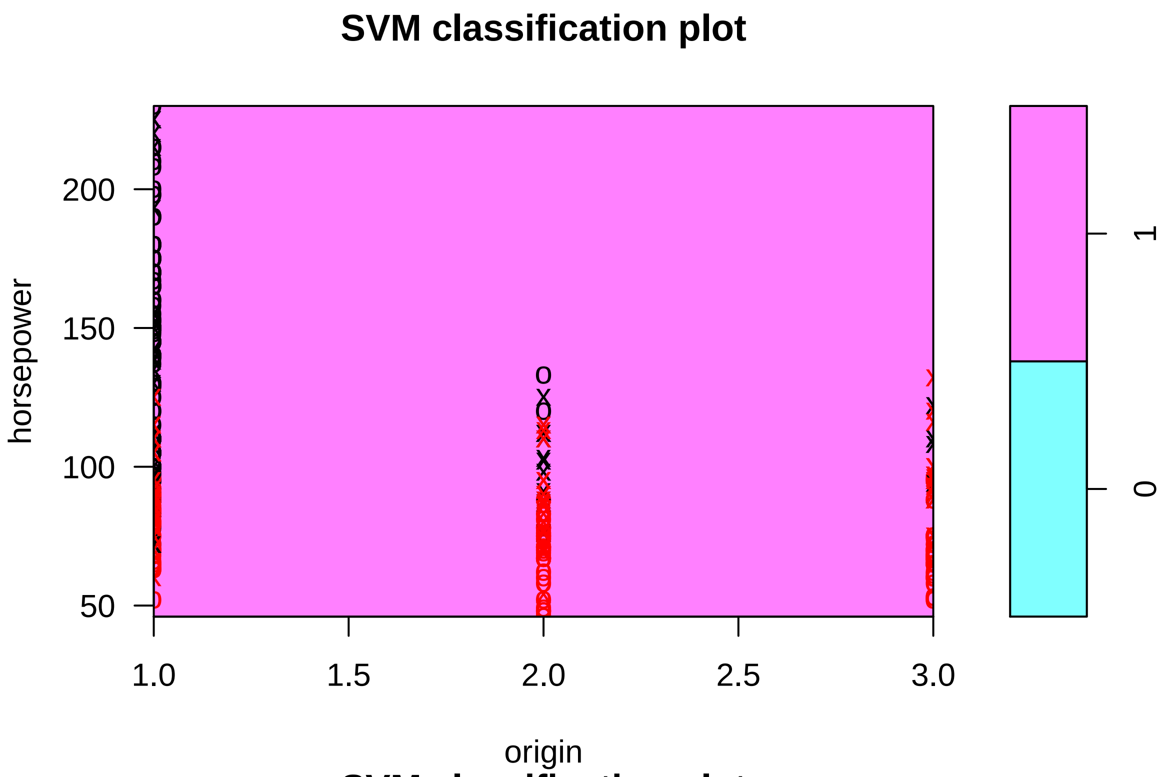

SVM classification plot
displacement
450
400
350
300
250
200
150
100
70
72
74
76
78
80
82
year
1
0

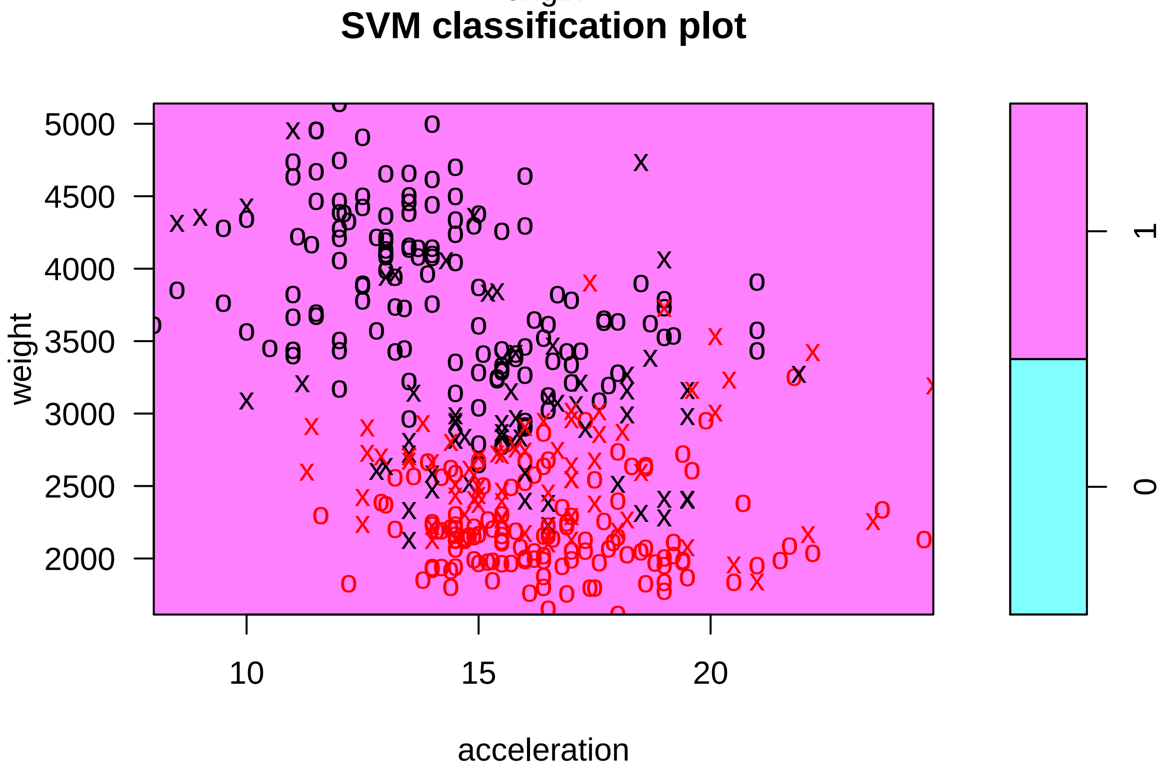

SVM classification plot
displacement
450
400
350
300
250
200
150
100
1.0
1.5
2.0
2.5
3.0
origin
1
0

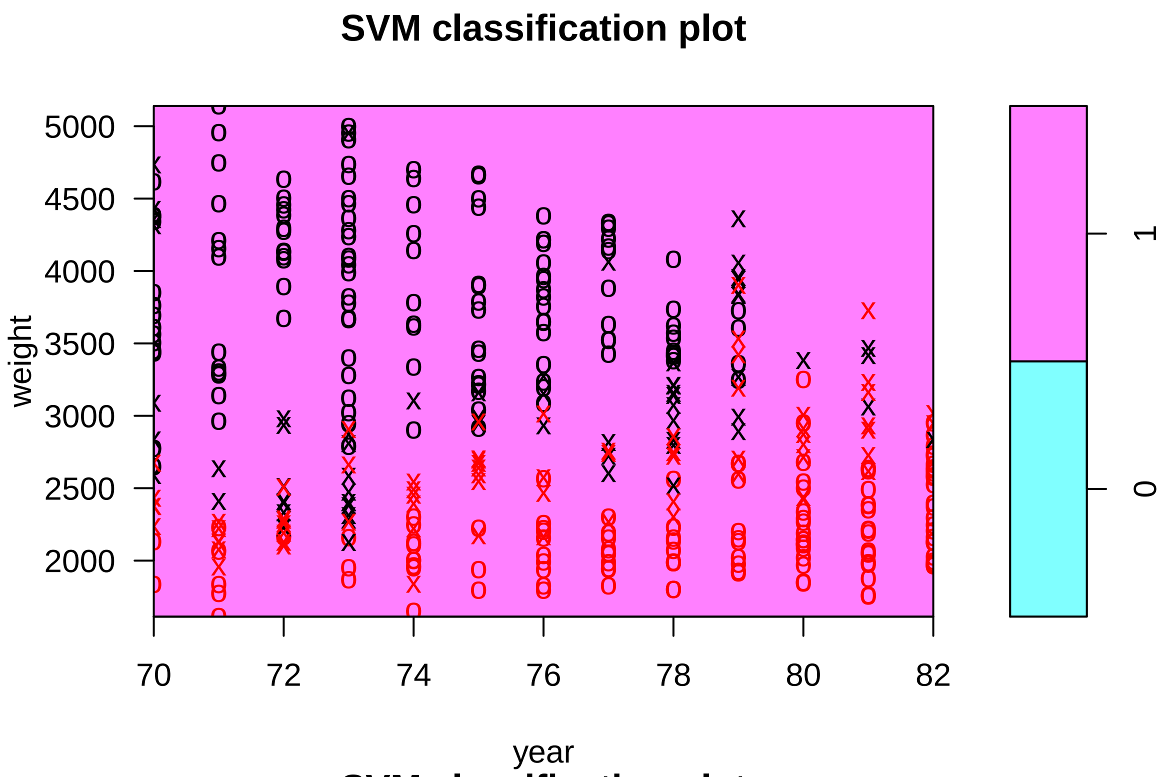
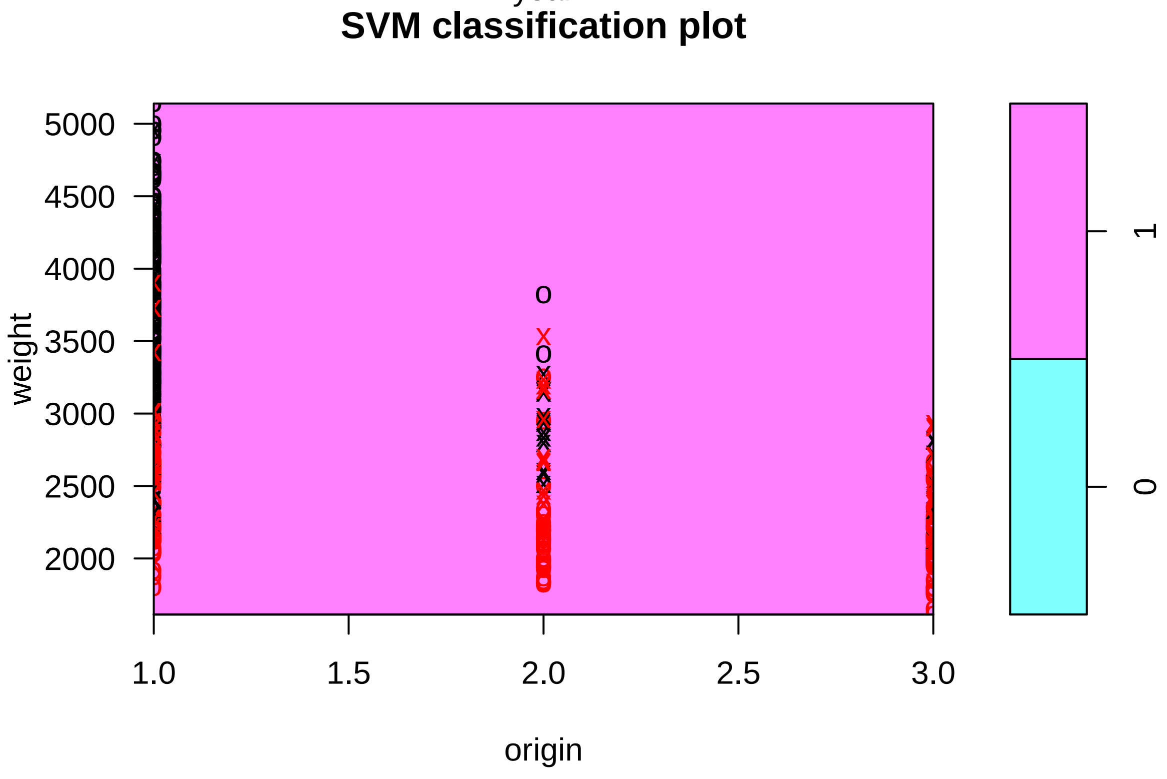

SVM classification plot
horsepower
200
150
100
50
2000
2500
3000
3500
4000
4500
5000
weight
1
0

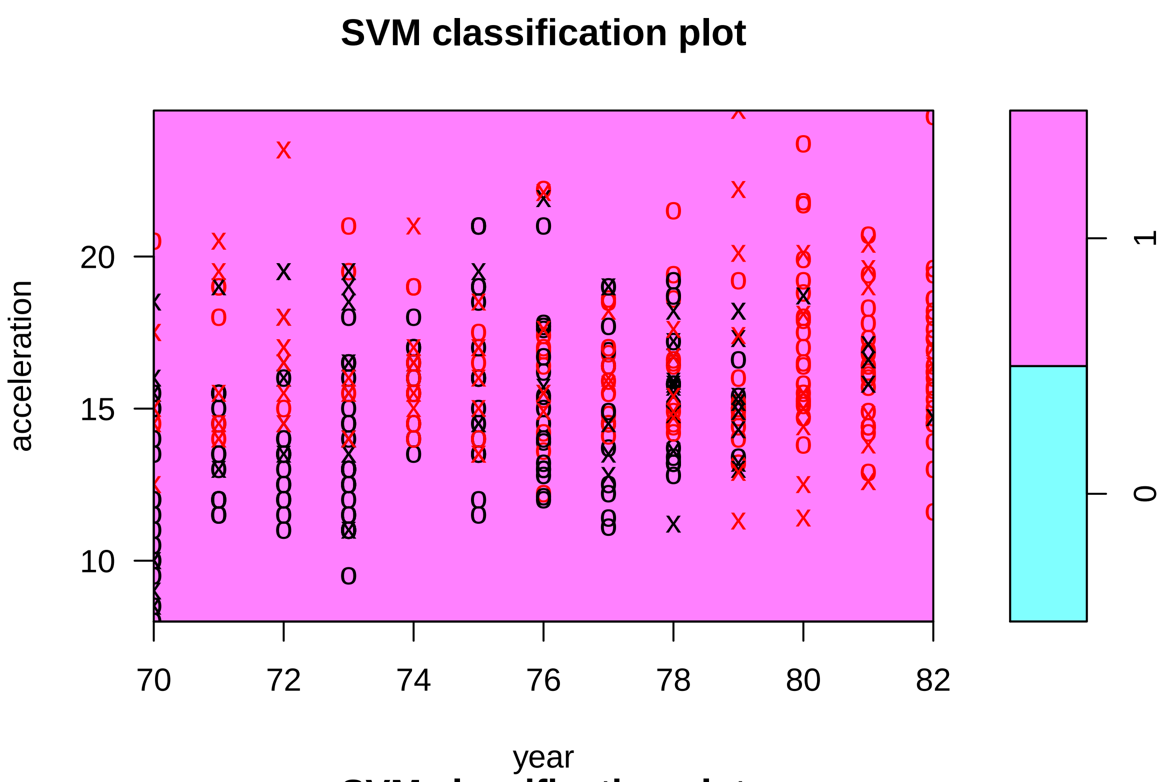

SVM classification plot
horsepower
50
100
150
200
10
15
20
acceleration
0
1

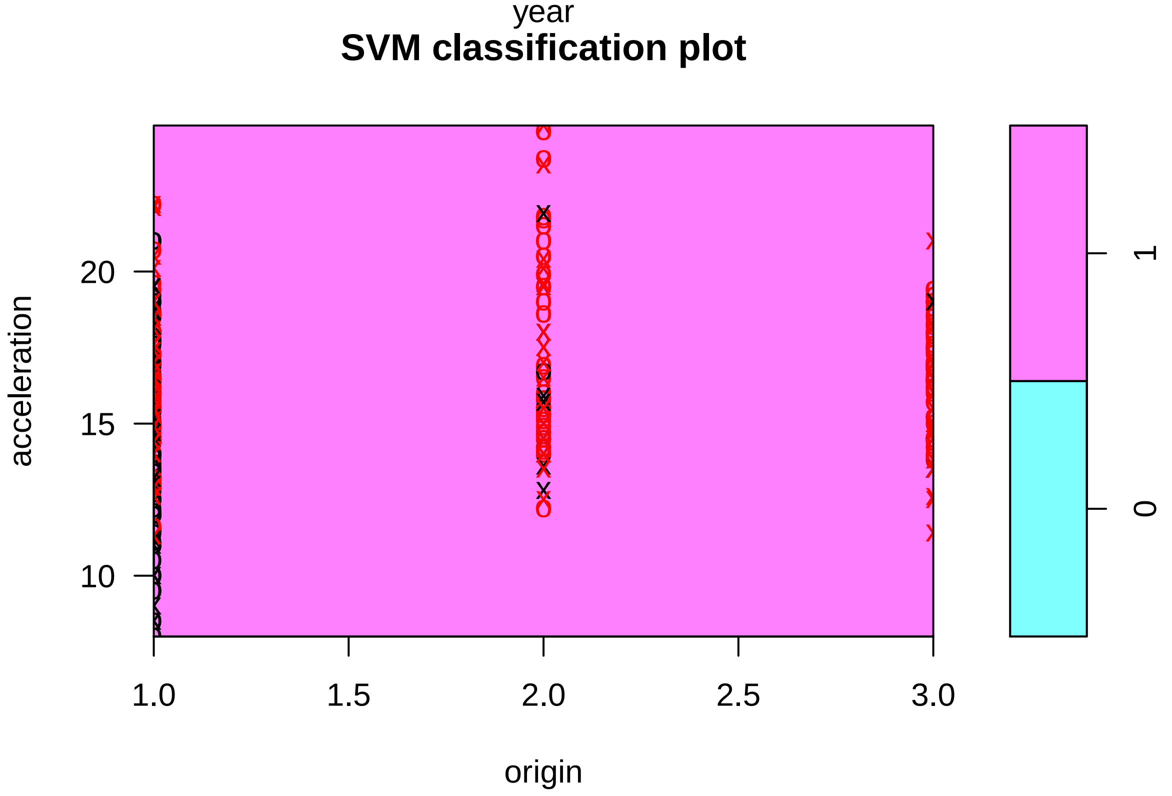

SVM classification plot
horsepower
50
100
150
200
70
72
74
76
78
80
82
year
0
1

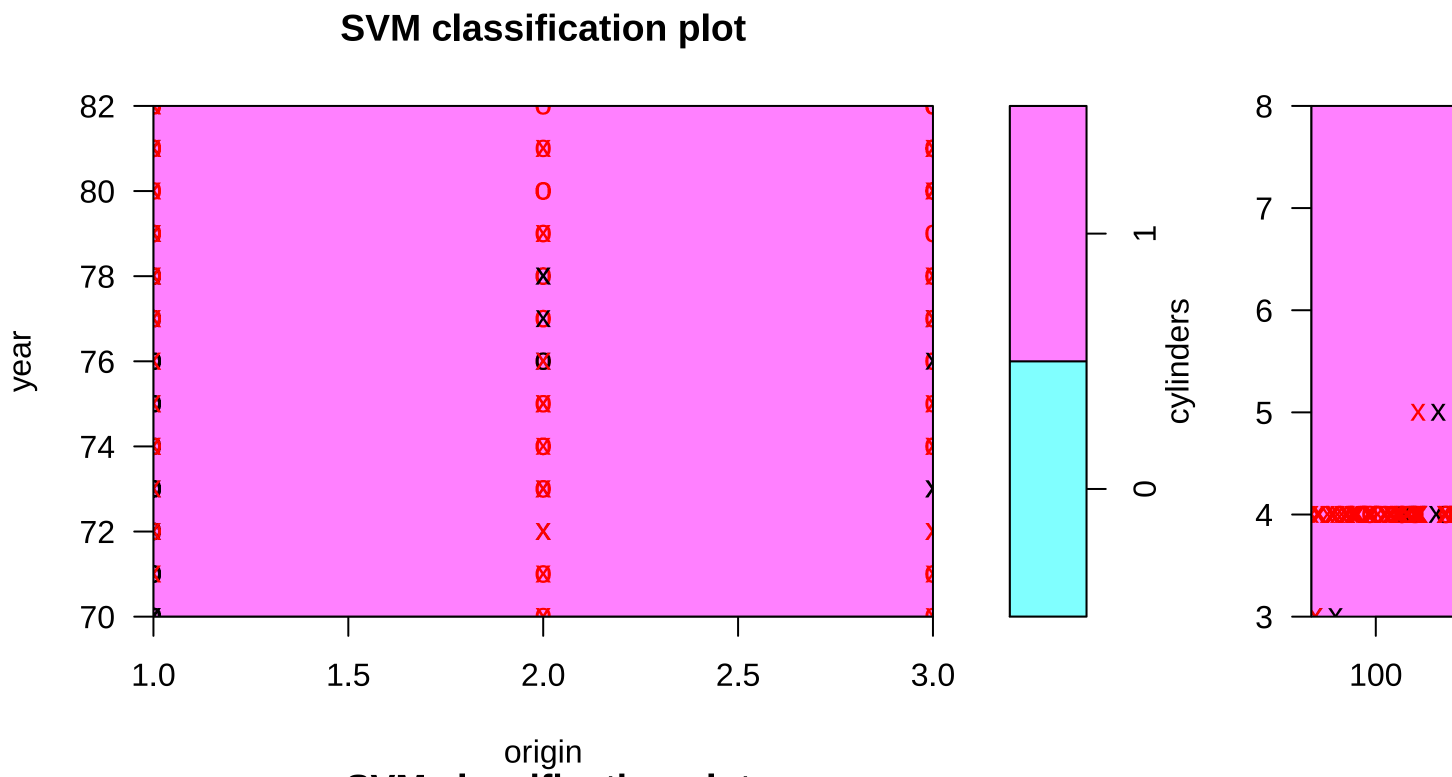

SVM classification plot
horsepower
origin
200
150
100
50
1.0
1.5
2.0
2.5
3.0
1
0

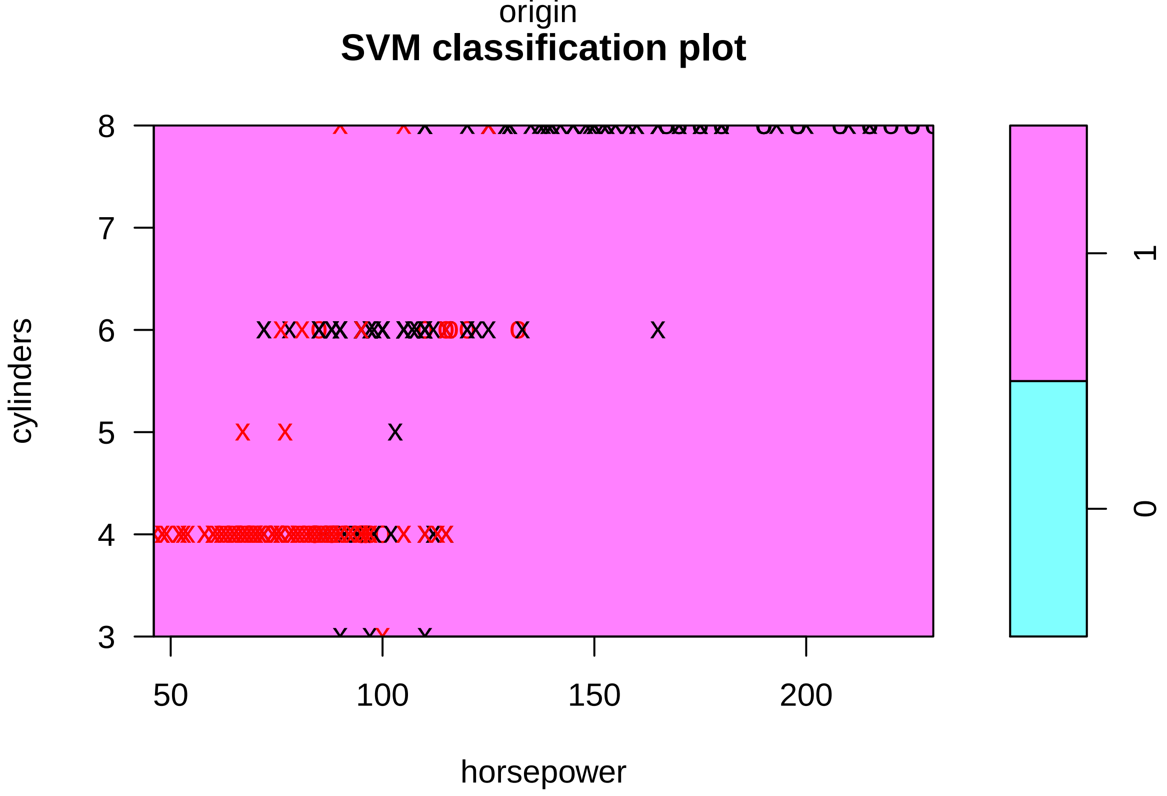

SVM classification plot
weight
acceleration
5000
4500
4000
3500
3000
2500
2000
10
15
20
1
0

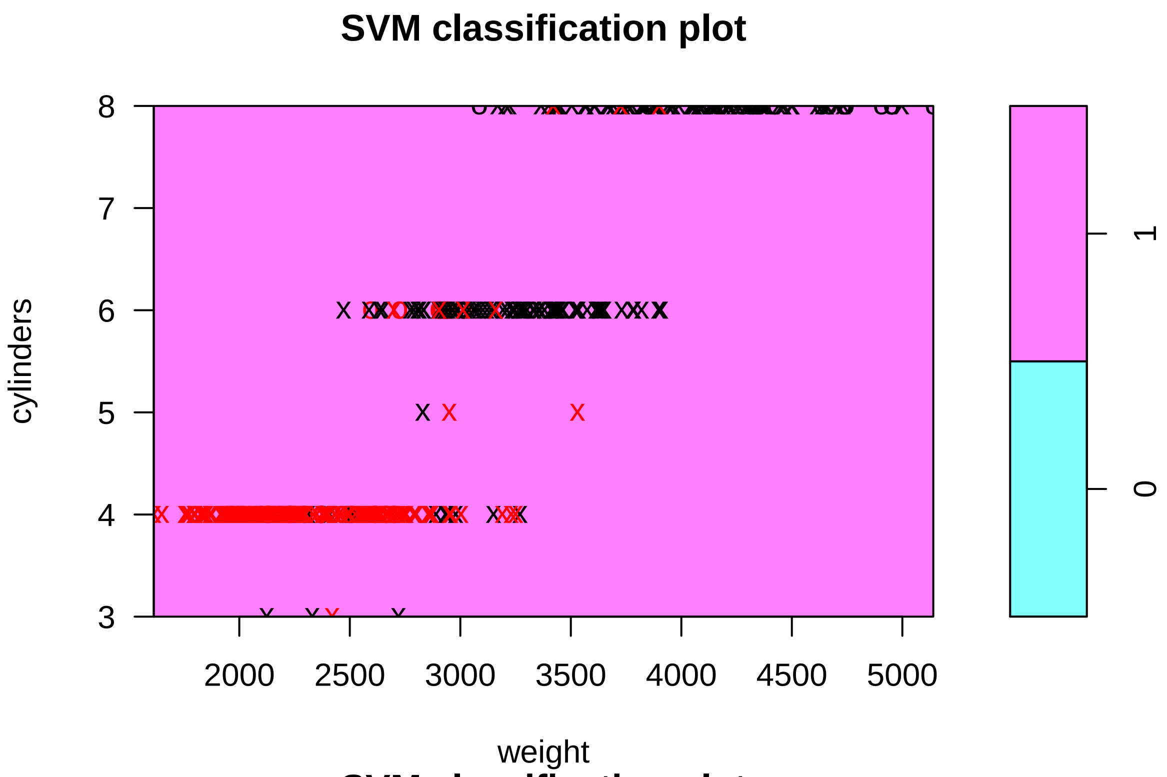

SVM classification plot
weight
year
1
0

SVM classification plot
weight
origin
1
0

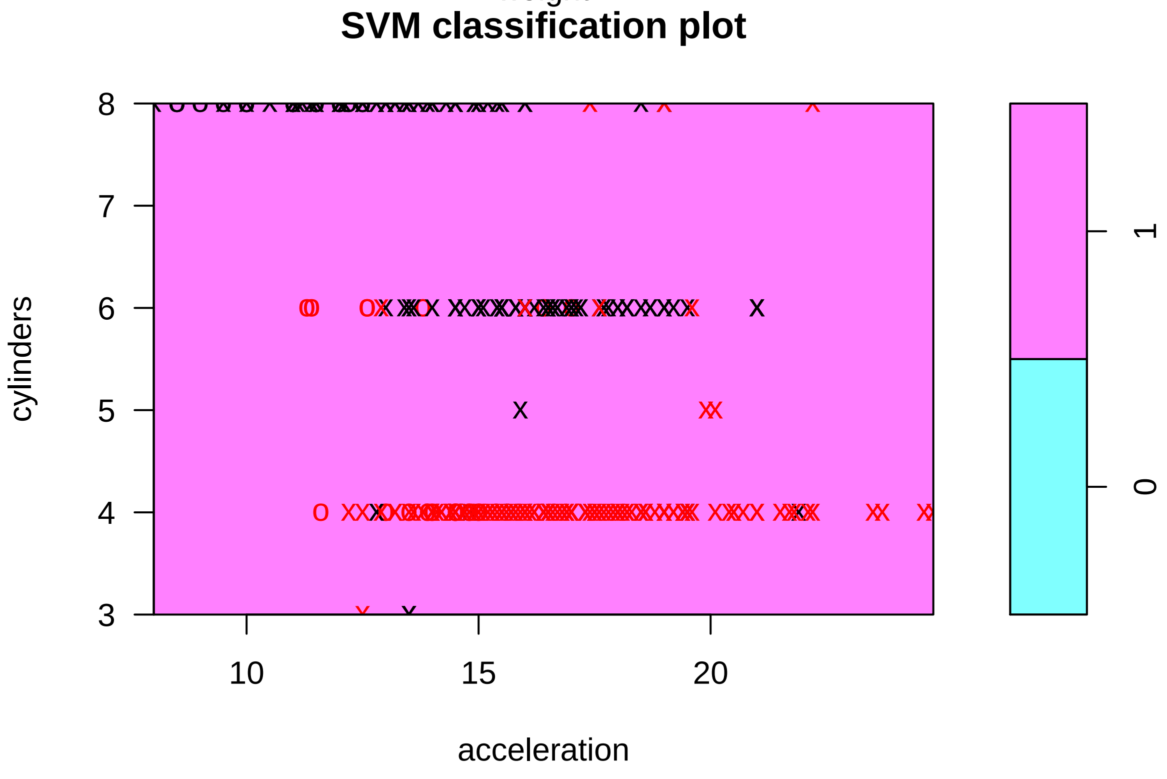

SVM classification plot
acceleration
year
20
15
10
70
72
74
76
78
80
82
1
0

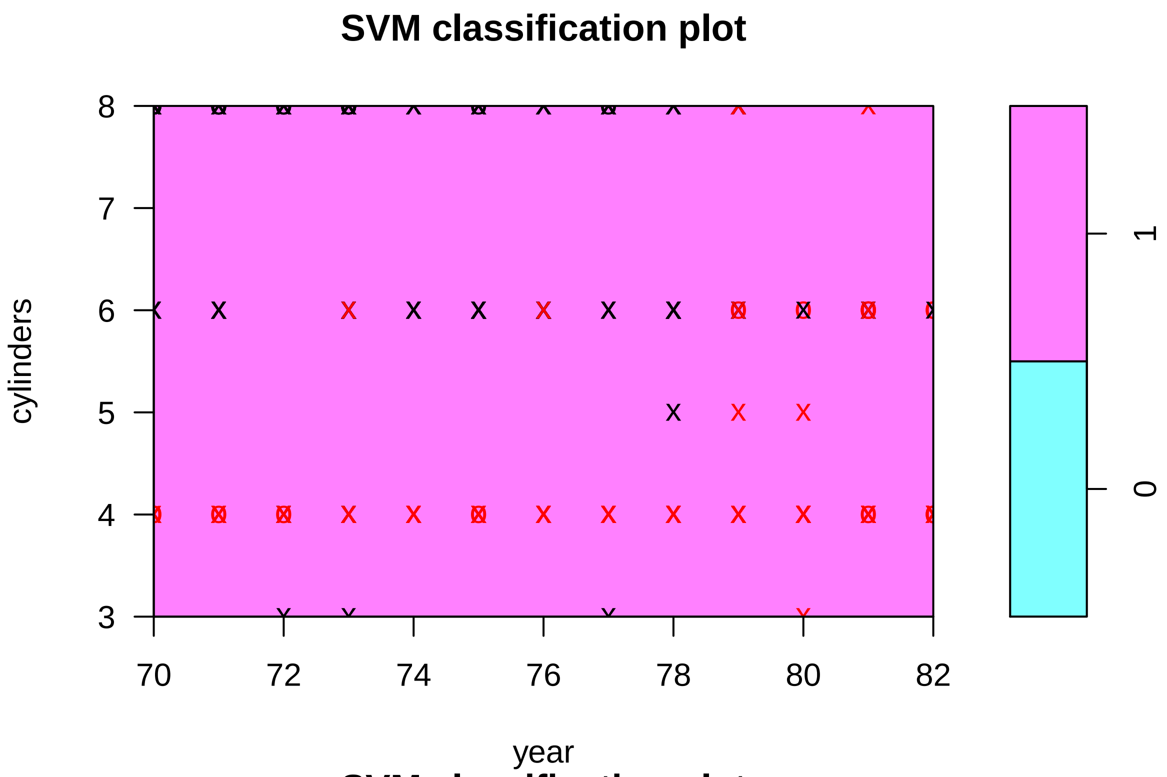

SVM classification plot
acceleration
origin
20
15
10
1.0
1.5
2.0
2.5
3.0
1
0

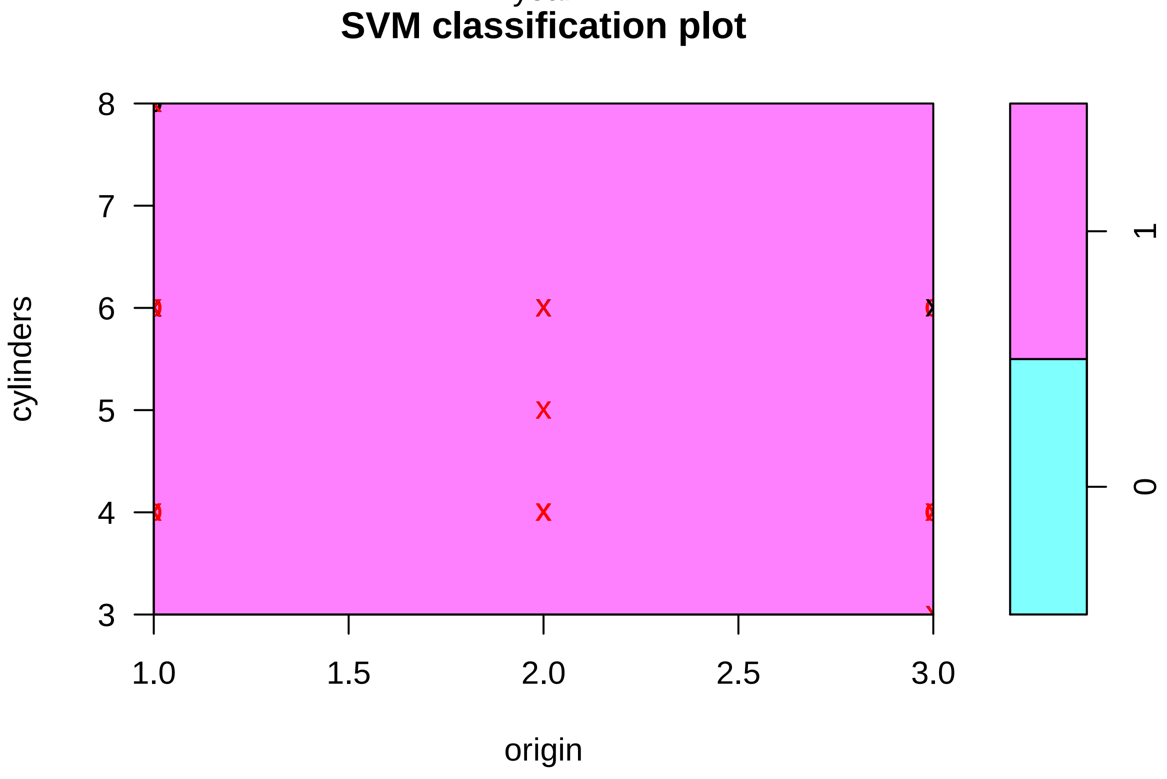

SVM classification plot
year
origin
cylinders
0
1
82
80
78
76
74
72
70
1.0
1.5
2.0
2.5
3.0
8
7
6
5
4
3
100

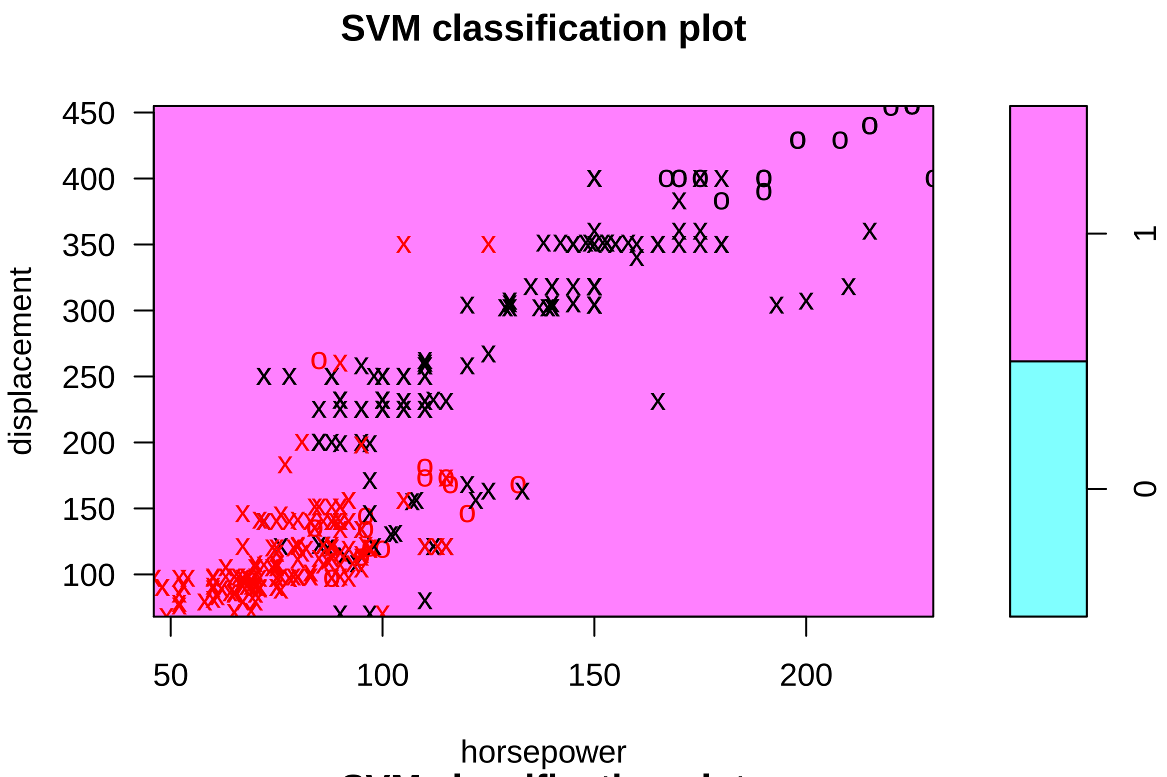

SVM classification plot
cylinders
horsepower
8
7
6
5
4
3
50
100
150
200
0
1

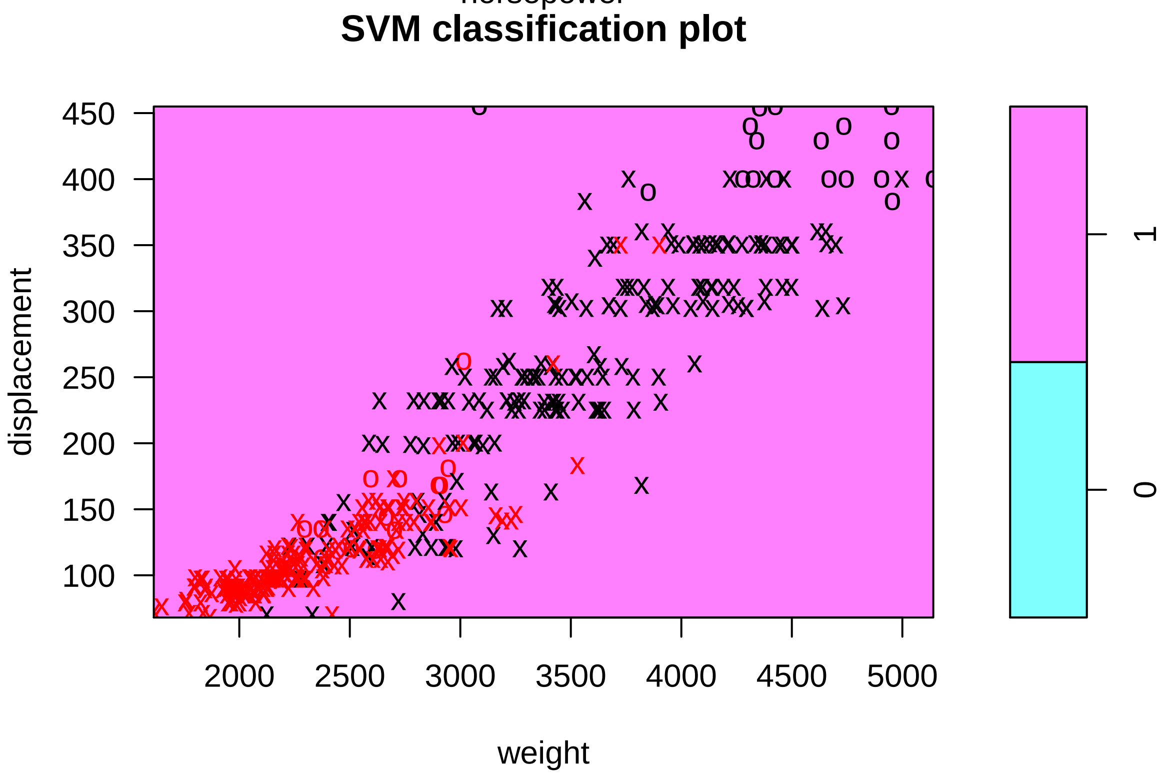

SVM classification plot
cylinders
weight
8
7
6
5
4
3
2000
2500
3000
3500
4000
4500
5000
1
0

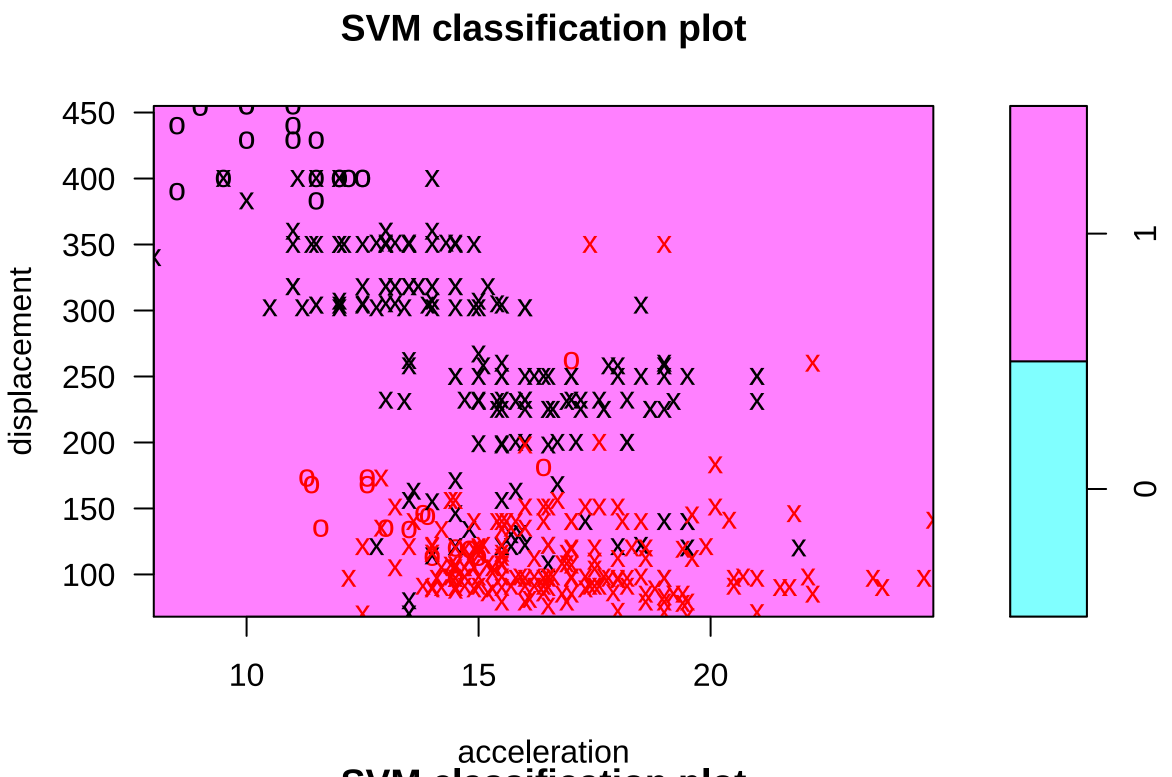

SVM classification plot
cylinders
acceleration
8
7
6
5
4
3
10
15
20
1
0

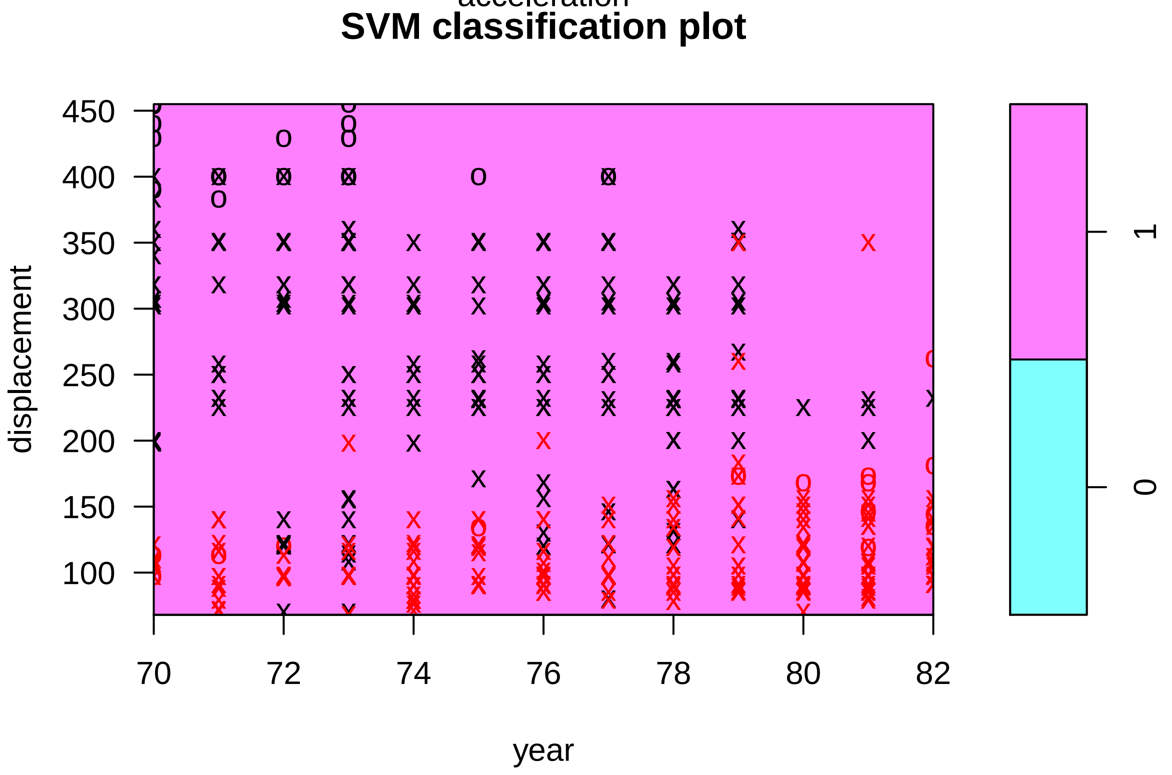

SVM classification plot
cylinders
year
8
7
6
5
4
3
70
72
74
76
78
80
82
1
0

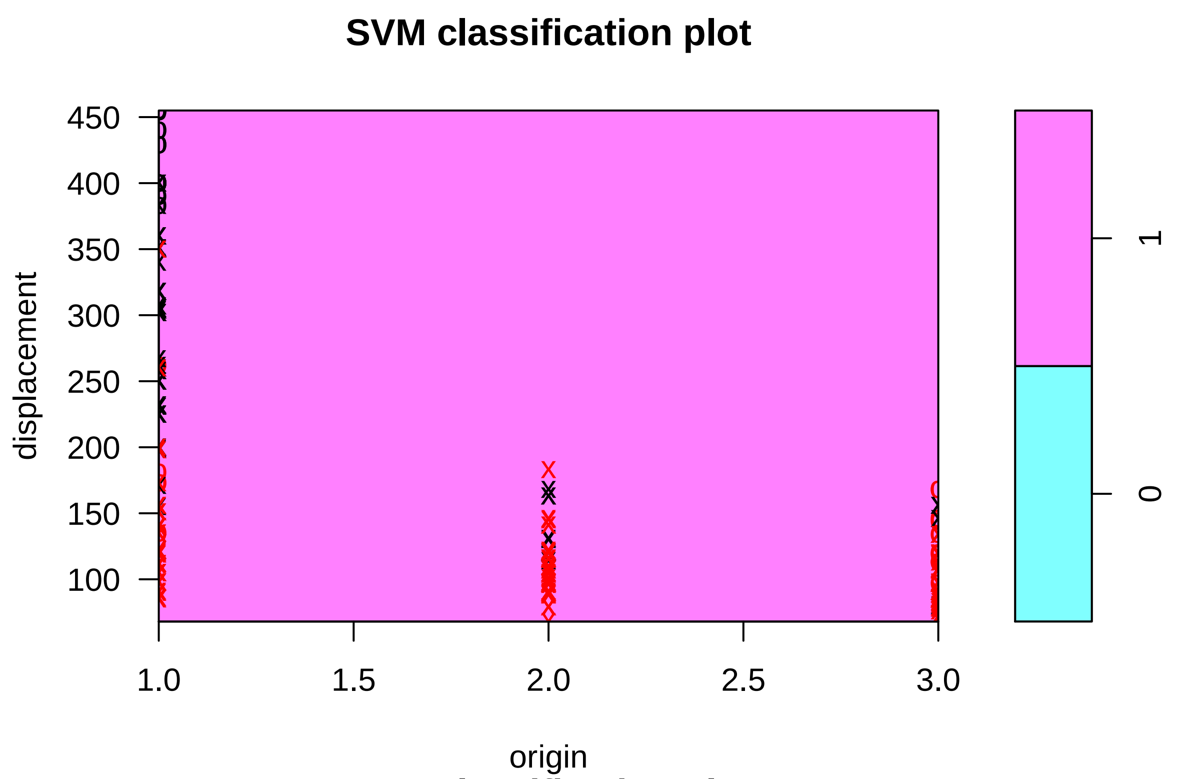

SVM classification plot
cylinders
origin
8
7
6
5
4
3
1.0
1.5
2.0
2.5
3.0
1
0

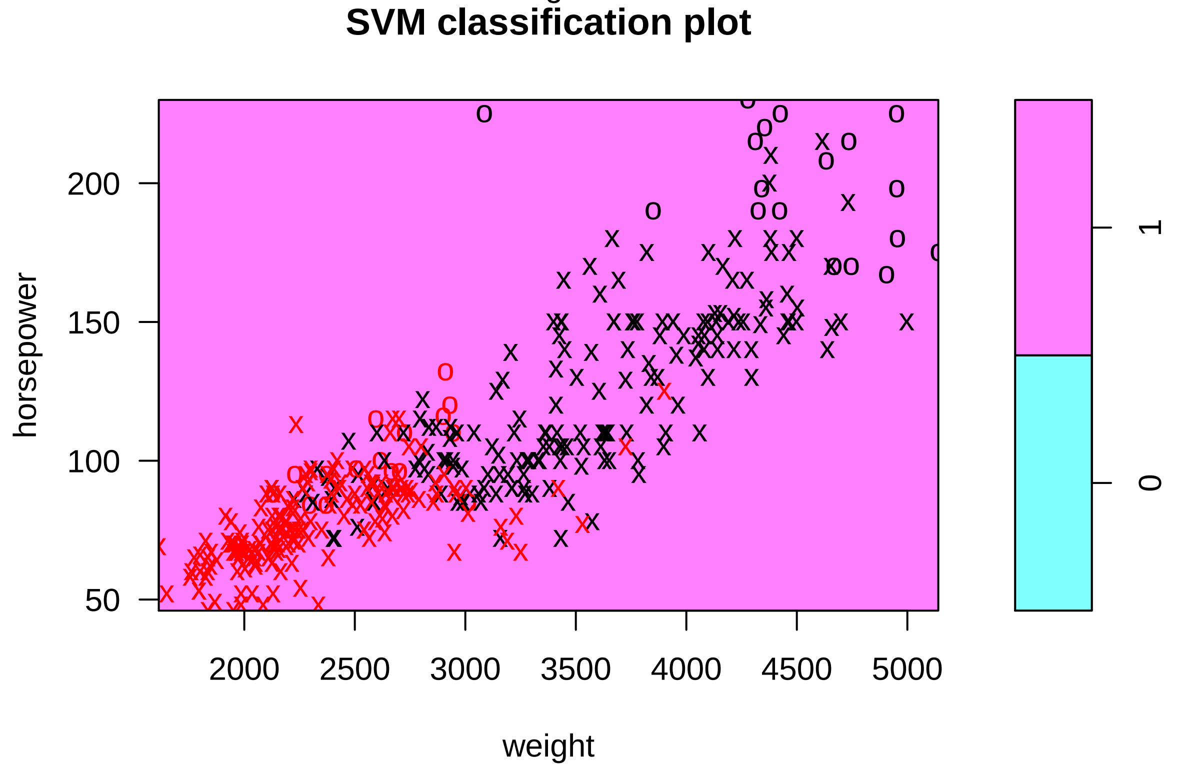

SVM classification plot
displacement
horsepower
450
400
350
300
250
200
150
100
50
100
150
200
1
0

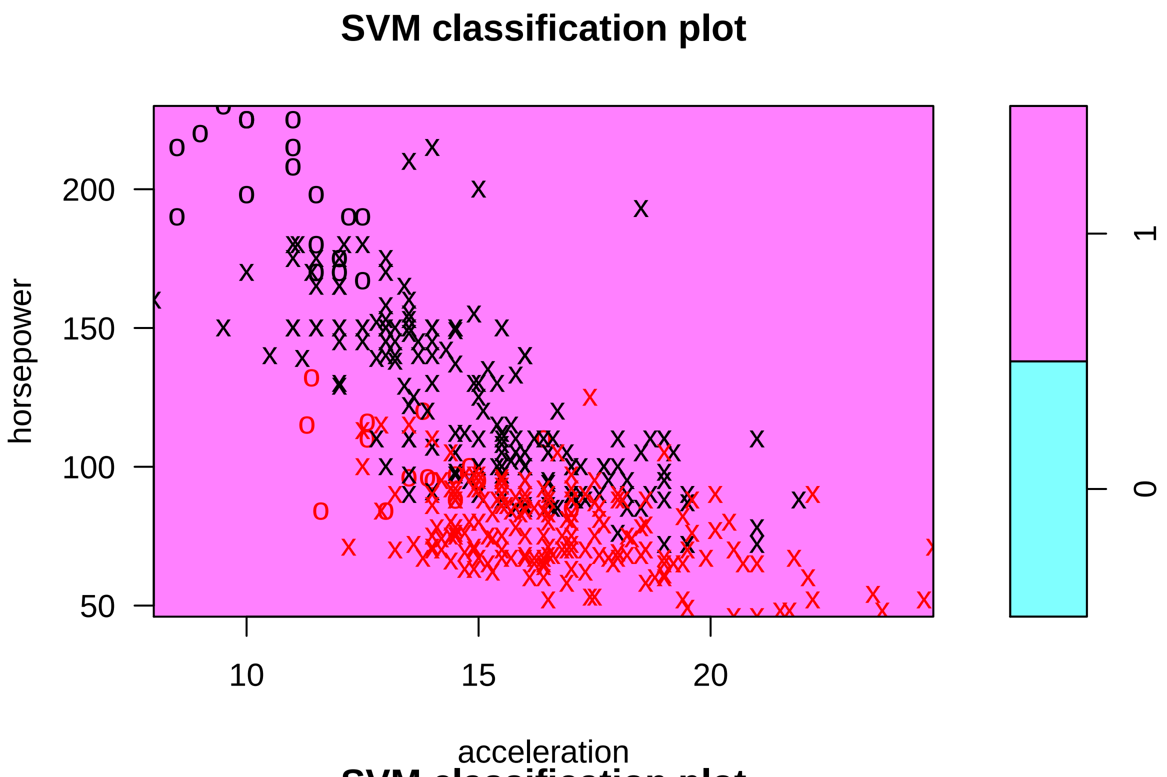

SVM classification plot
displacement
weight
450
400
350
300
250
200
150
100
2000
2500
3000
3500
4000
4500
5000
1
0

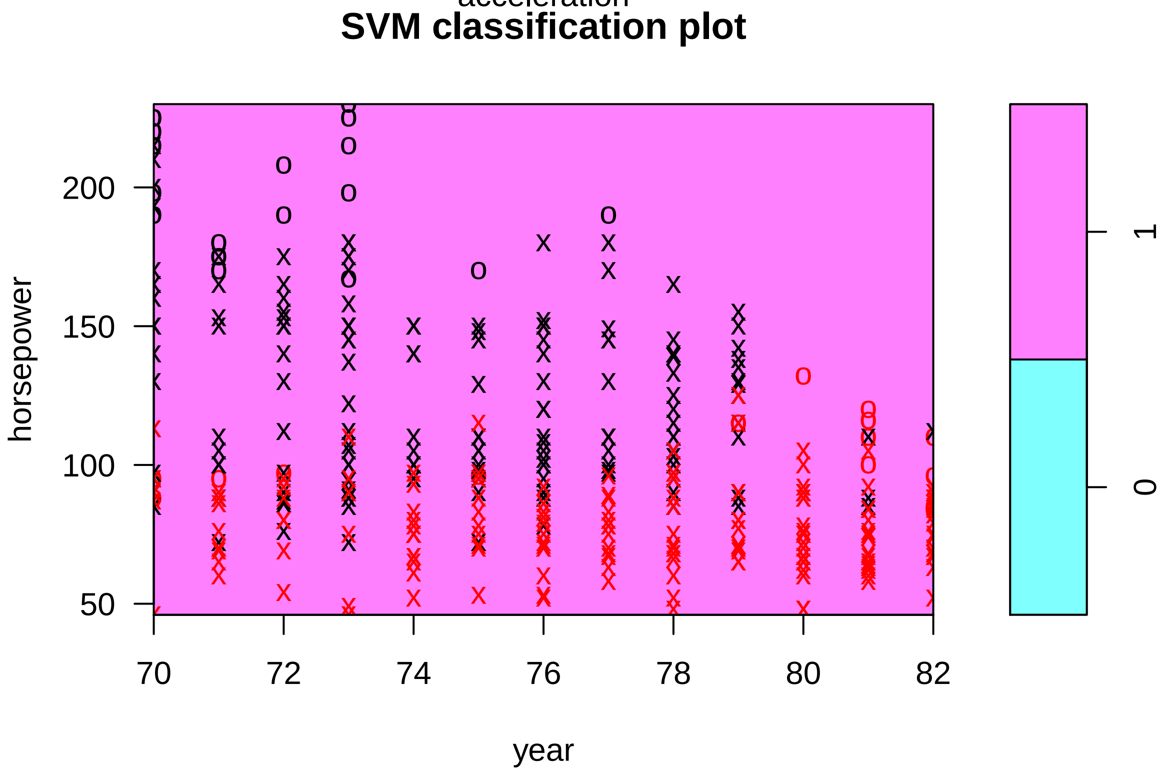

SVM classification plot
displacement
acceleration
450
400
350
300
250
200
150
100
10
15
20
1
0

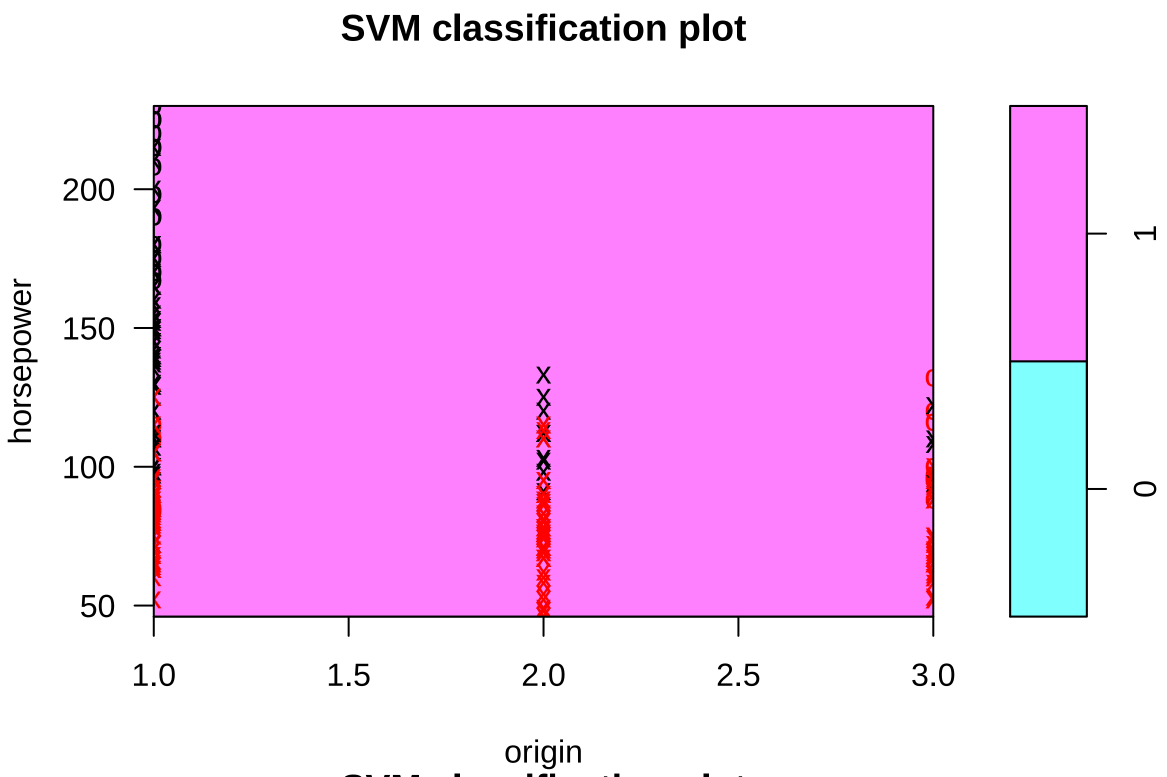

SVM classification plot
displacement
year
450
400
350
300
250
200
150
100
70
72
74
76
78
80
82
1
0

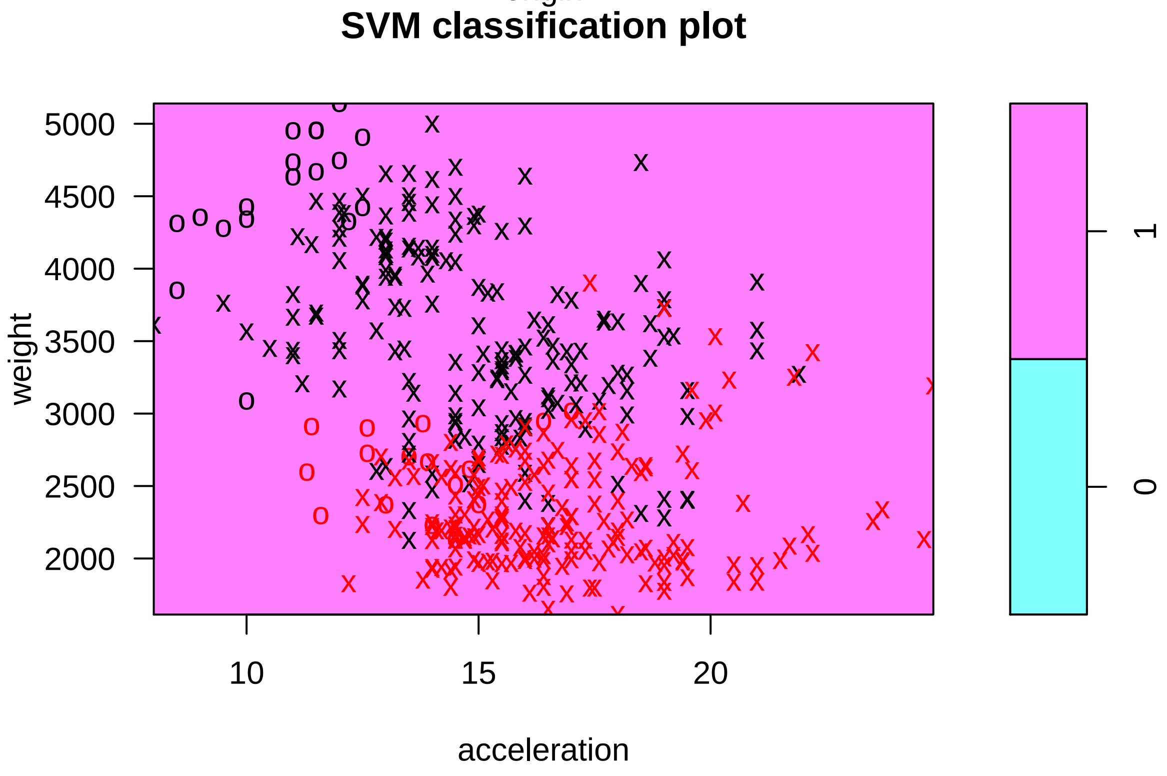

SVM classification plot
displacement
origin
450
400
350
300
250
200
150
100
1.0
1.5
2.0
2.5
3.0
1
0

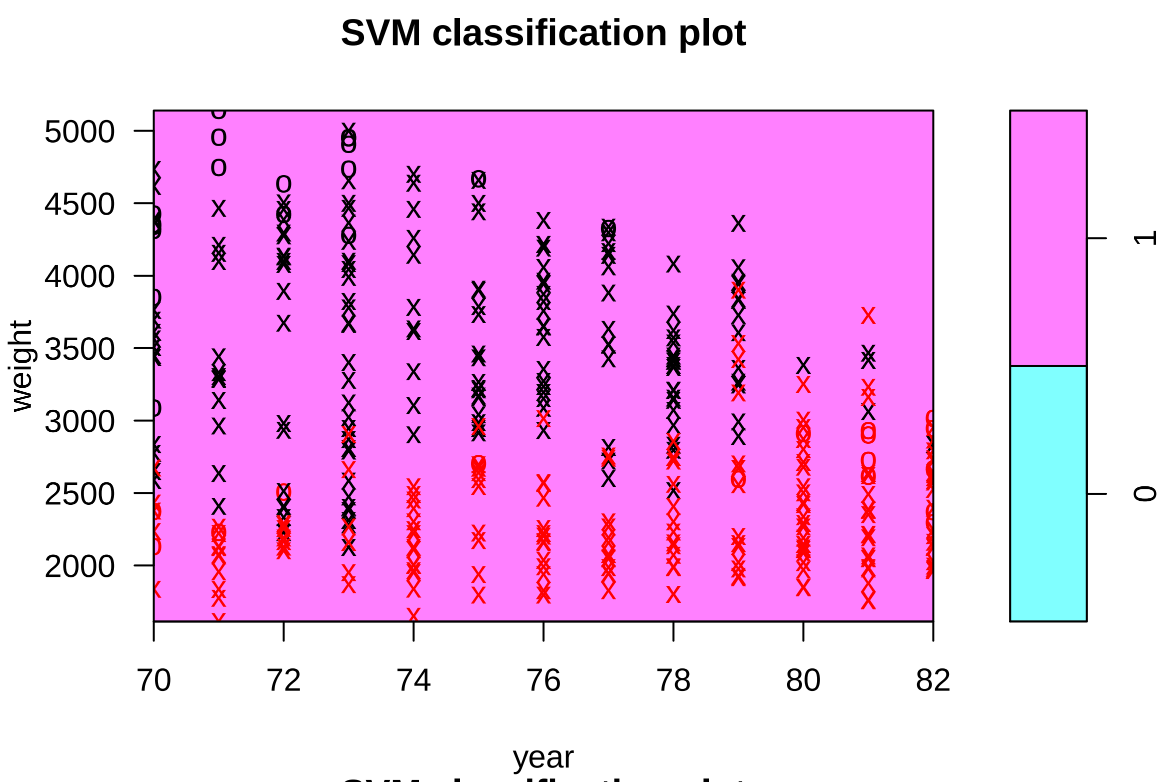

SVM classification plot
horsepower
weight
200
150
100
50
2000
2500
3000
3500
4000
4500
5000
1
0

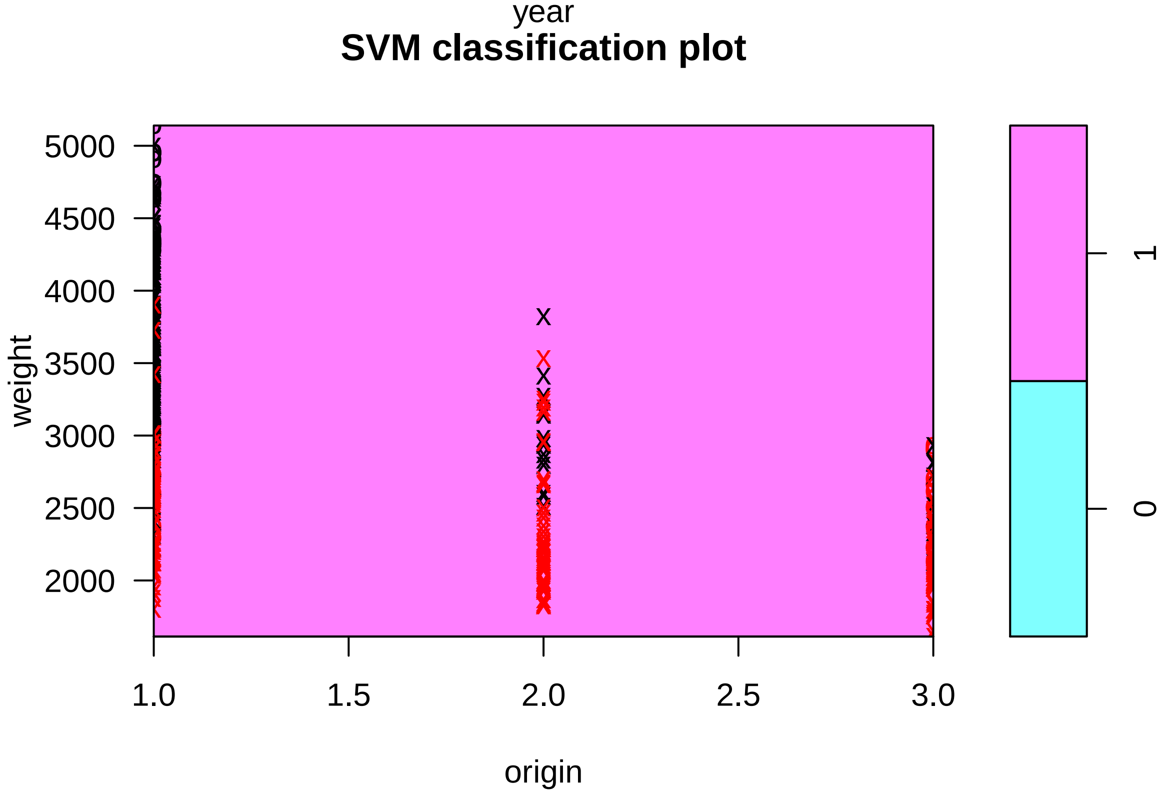

SVM classification plot
horsepower
acceleration
200
150
100
50
10
15
20
1
0

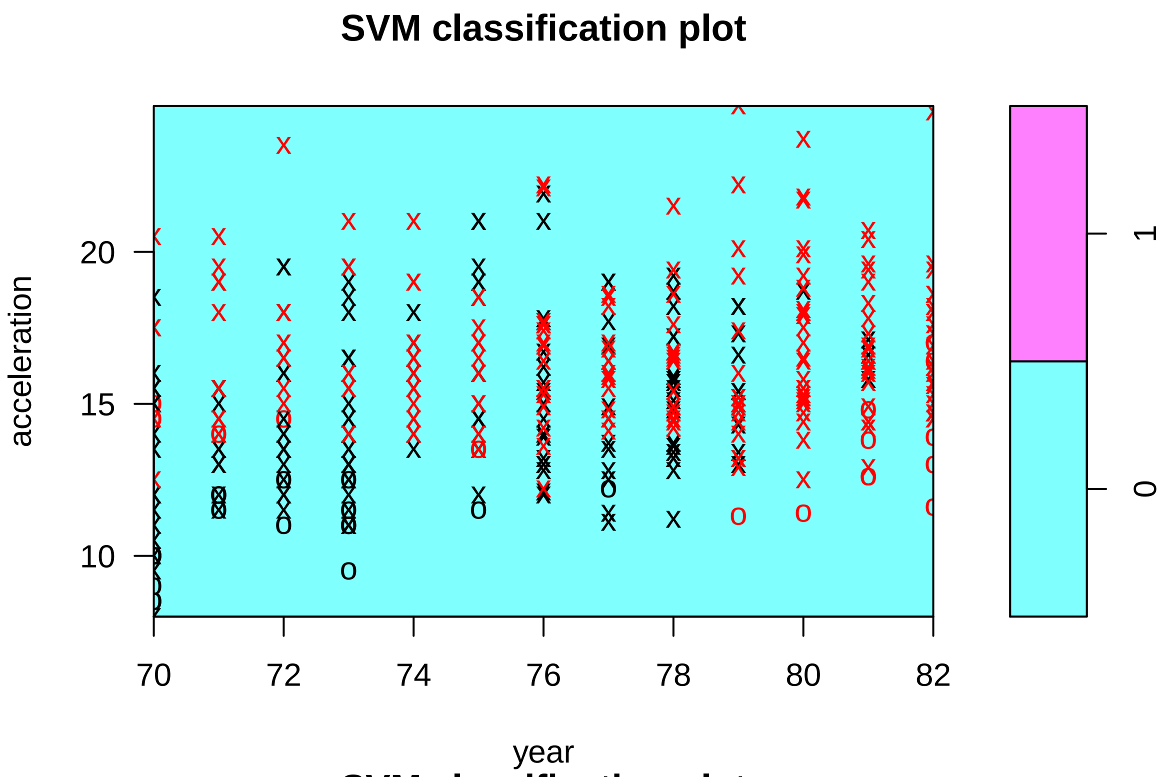

SVM classification plot
horsepower
year
200
150
100
50
70
72
74
76
78
80
82
1
0

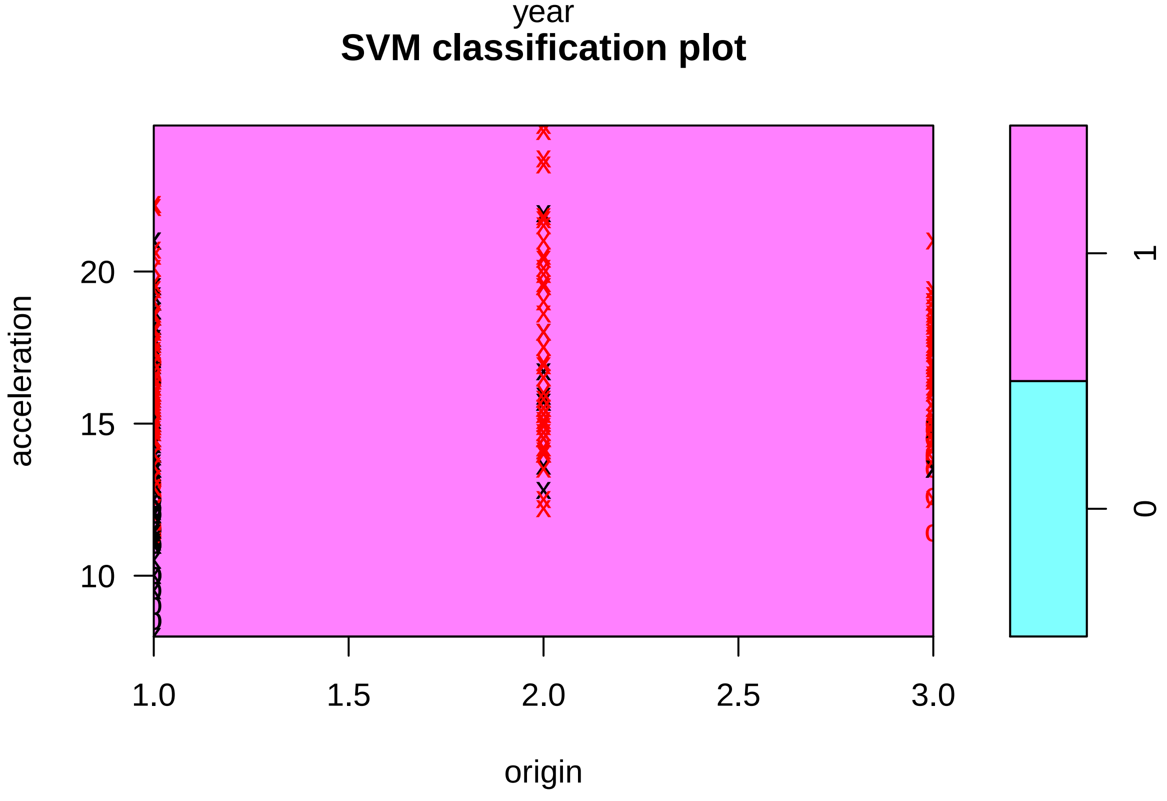

SVM classification plot
horsepower
origin
200
150
100
50
1.0
1.5
2.0
2.5
3.0
1
0

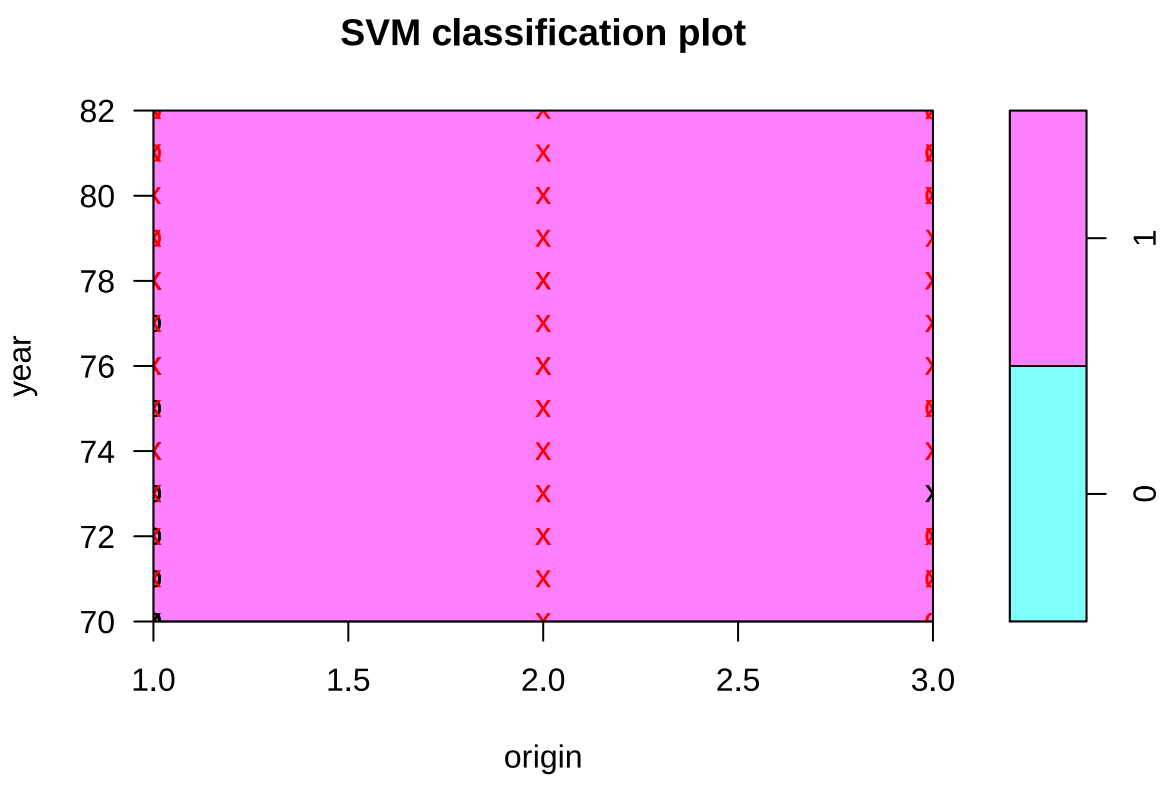

SVM classification plot
weight
acceleration
5000
4500
4000
3500
3000
2500
2000
10
15
20
1
0


SVM classification plot
weight
5000
4500
4000
3500
3000
2500
2000
70
72
74
76
78
80
82
year
1
0


SVM classification plot
weight
5000
4500
4000
3500
3000
2500
2000
1.0
1.5
2.0
2.5
3.0
origin
1
0


SVM classification plot
acceleration
20
15
10
70
72
74
76
78
80
82
year
1
0


SVM classification plot
acceleration
20
15
10
1.0
1.5
2.0
2.5
3.0
origin
1
0



5. SVM (15% | 0%)

Marking scheme: MSc only. (h) 1%, the rest 2% each.

Here we explore the maximal margin classifier on a toy data set.

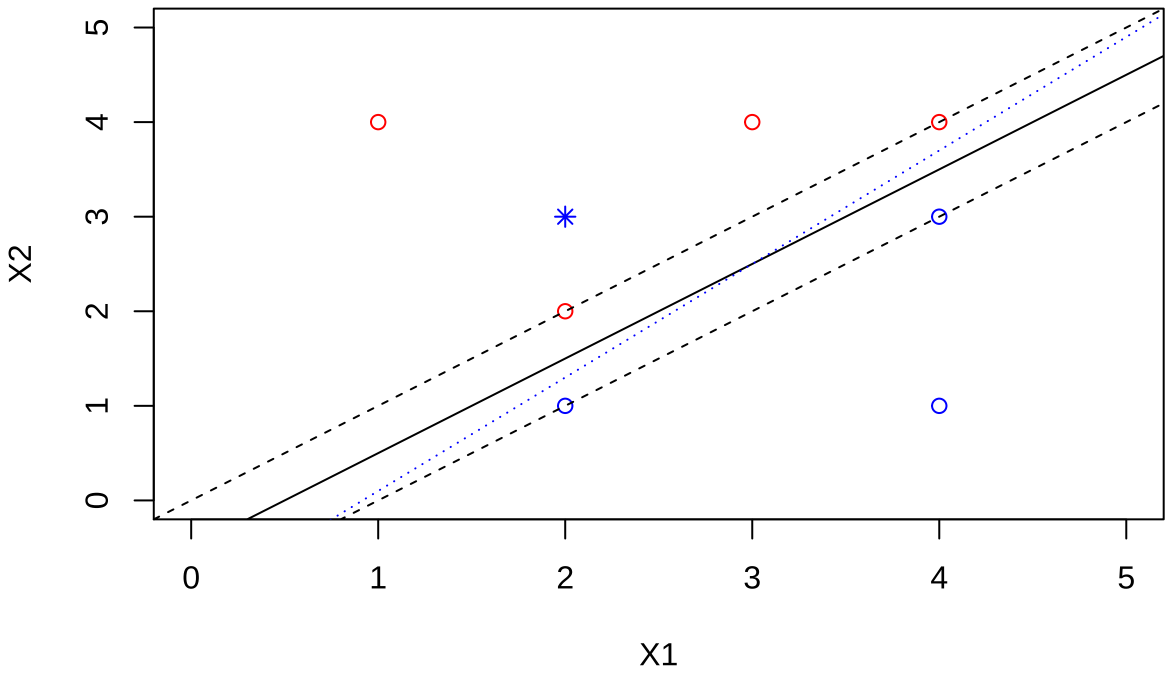
(a) We are given $n = 7$ observations in $p = 2$ dimensions. For each observation, there is an associated class label. Sketch the observations.

Obs.	X_1	X_2	Y
1	3	4	Red
2	2	2	Red
3	4	4	Red
4	1	4	Red
5	2	1	Blue
6	4	3	Blue
7	4	1	Blue

Solution:

```
X1 <- c(3,2,4,1,2,4,4)
X2 <- c(4,2,4,4,1,3,1)
Y <- c("R","R","R","R","B","B","B")
df <- data.frame(X1,X2,Y)
```

```r
plot(X1, X2,
     col = ifelse(Y=="R", 2, 4),
     ylim = c(0,5),
     xlim = c(0,5))

#Solution to (d)
abline(a=-0.5, b=1)  # This is the separating line
abline(a=0, b=1, lty=2)  #These two are the margin lines.
abline(a=-1, b=1, lty=2)

#Solution to (g)
abline(a=-1.1, b=1.2, lty = 3, col = 4)

#Solution to (h)
points(x=2, y=3, col = 4, pch = 8)
```



(b) Sketch the optimal separating hyperplane, and provide the equation for this hyperplane of the following form.

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

Solution: From the plot, it can be concluded that the hyperplane will go through the following two points: (2, 1.5) and (4, 3.5). The line that goes through these two points is of the form

$$\frac{1}{2} - X_1 + X_2 = 0$$

(c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$, and classify to Blue otherwise." Provide the values for β_0, β_1, and β_2.

Solution:

Classify to Red if $\frac{1}{2} - X_1 + X_2 > 0$ and classify to Blue otherwise.

(d) On your sketch, indicate the margin for the maximal margin hyperplane.

Solution:

```r
#abline(a=-0.5, b=1)
#abline(a=0, b=1, lty=2)
#abline(a=-1, b=1, lty=2)
```

(e) Indicate the support vectors for the maximal margin classifier.

Solution:

The support vectors for the MMC are $\#2 : (2, 2), \#3 : (4, 4), \#5 : (2, 1) and \#6 : (4, 3)$.

(f) Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

Solution: #7 is not a support vector and is away from the margin. so a slight movement wouldn't affect the maximal margin hyperplane.

(g) Sketch a hyperplane that is *not* the optimal separating hyperplane, and provide the equation for this hyperplane.

Solution:

For instance $1.2X_1 - X_2 - 1.1 = 0$ is a separating hyperplane, but it is not optimal.

```r
#abline(a=-1.1, b=1.2, lty = 3, col = 4)
```

(h) Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

Solution:

```r
# points(x=2, y=3, col = 4, pch = 8)
```

6. Hierarchical clustering (10% | 20%)

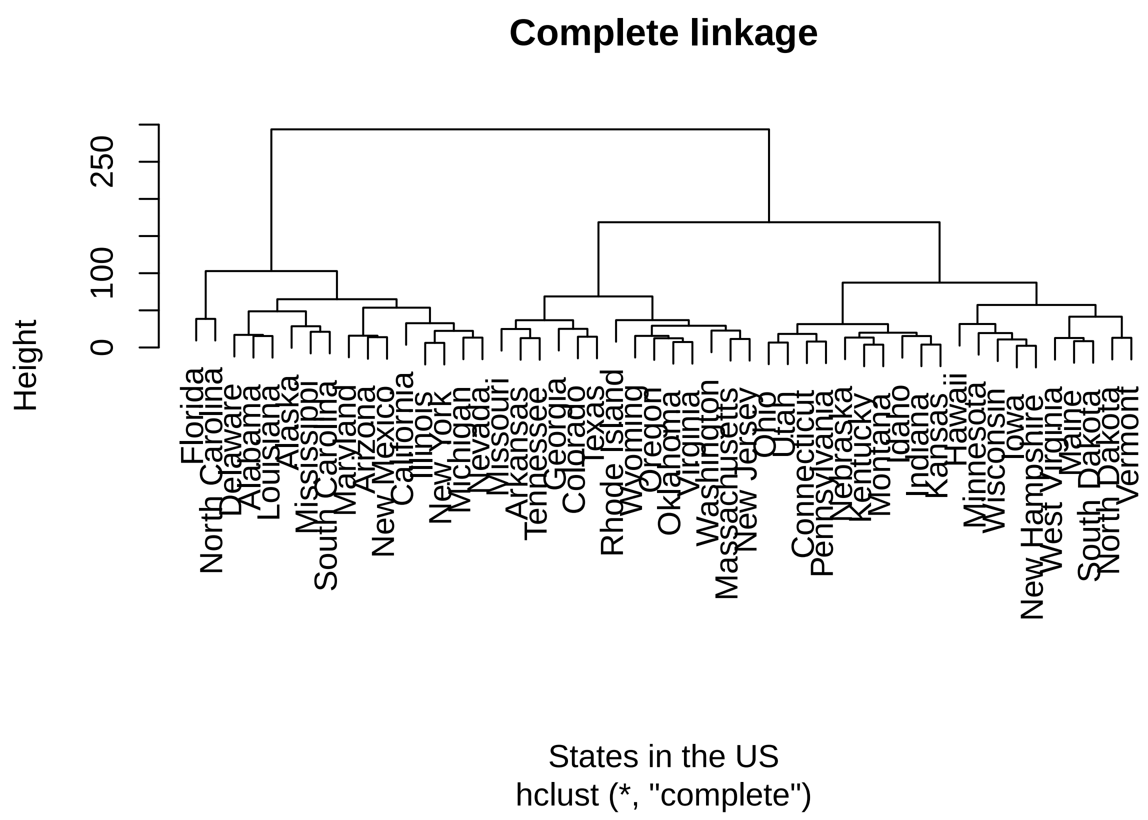
Marking scheme: 2.5% | 5% each.

Consider the `USArrests` data. We will now perform hierarchical clustering on the states.

(a) Using hierarchical clustering with complete linkage and Euclidean distance, cluster the states.

Solution:

```r
library(ISLR)

hc.complete <- hclust(dist(USArrests), method = "complete")
plot(hc.complete, main="Complete linkage", xlab = "States in the US")
```



(b) Cut the dendrogram at a height that results in three distinct clusters. Which states belong to which clusters?

Solution:

```
ctree <- cutree(hc.complete, 3)
table(cutree(hc.complete, 3))
```

```
##
##  1  2  3
## 16 14 20
```

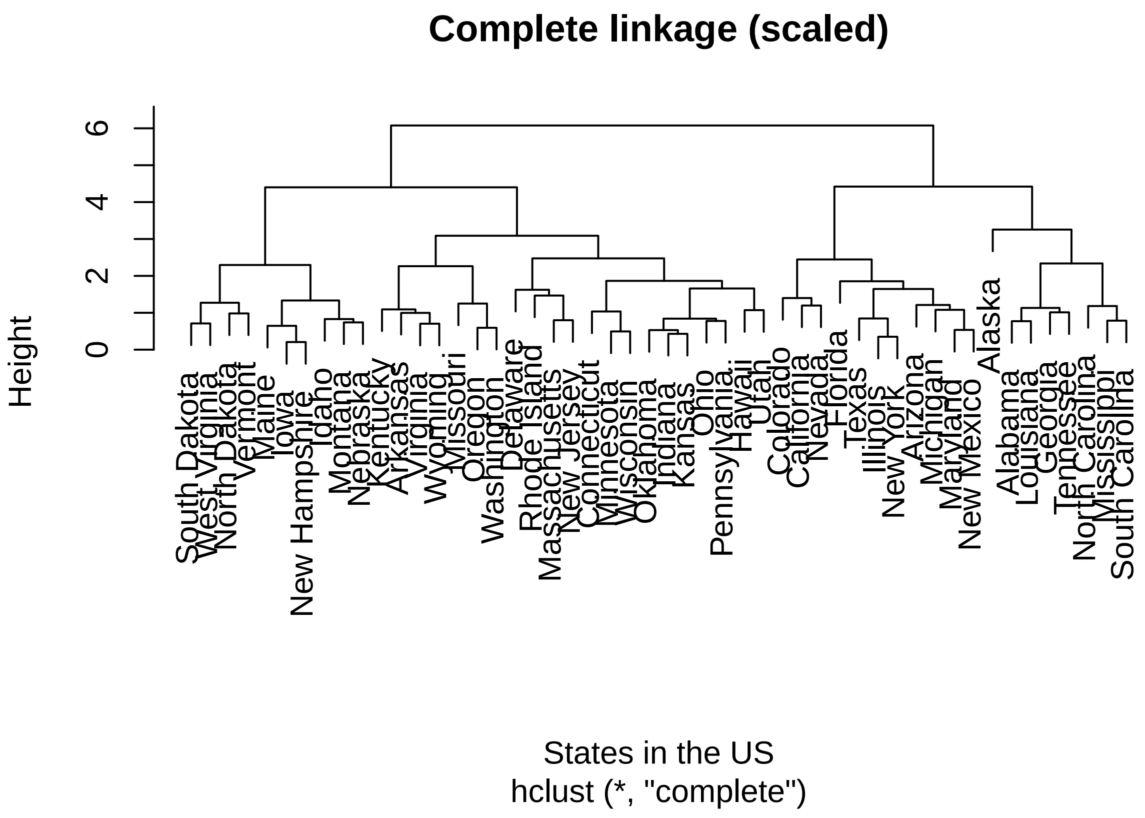
```
# Print which states go into each cluster:
#
for(k in 1:3){
  print(k)
  print(rownames(USArrests)[ctree == k] )
}
```

```
## [1] 1
##  [1] "Alabama"        "Alaska"         "Arizona"        "California"
##  [5] "Delaware"       "Florida"        "Illinois"       "Louisiana"
##  [9] "Maryland"       "Michigan"       "Mississippi"    "Nevada"
## [13] "New Mexico"     "New York"       "North Carolina" "South Carolina"
## [1] 2
##  [1] "Arkansas"      "Colorado"      "Georgia"       "Massachusetts"
##  [5] "Missouri"      "New Jersey"    "Oklahoma"      "Oregon"
##  [9] "Rhode Island"  "Tennessee"     "Texas"         "Virginia"
## [13] "Washington"    "Wyoming"
## [1] 3
##  [1] "Connecticut"   "Hawaii"        "Idaho"         "Indiana"
##  [5] "Iowa"          "Kansas"        "Kentucky"      "Maine"
##  [9] "Minnesota"     "Montana"       "Nebraska"      "New Hampshire"
## [13] "North Dakota"  "Ohio"          "Pennsylvania"  "South Dakota"
## [17] "Utah"          "Vermont"       "West Virginia" "Wisconsin"
```

(c) Hierarchically cluster the states using complete linkage and Euclidean distance, after scaling the variables to have standard deviation one.

Solution:

```r
hc.complete.sc <- hclust(dist(scale(USArrests)), method="complete")
plot(hc.complete.sc, main="Complete linkage (scaled)", xlab = "States in the US")
```



```r
summary(USArrests)
```

```
##      Murder          Assault         UrbanPop          Rape
##  Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30
##  1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07
##  Median : 7.250   Median :159.0   Median :66.00   Median :20.10
##  Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23
##  3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18
##  Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00
```

(d) What effect does scaling the variables have on the hierarchical clustering obtained? In your opinion, should the variables be scaled before the inter-observation dissimilarities are computed? Provide a justification for your answer.

Solution:

```r
ctree.sc <- cutree(hc.complete.sc, 3)

table(ctree.sc)
```

```
## ctree.sc
##  1  2  3
##  8 11 31
```

```r
table(unscaled = ctree, scaled = ctree.sc)
```

```
##         scaled
## unscaled  1  2  3
##        1  6  9  1
##        2  2  2 10
##        3  0  0 20
```

```r
# Print which states go into each cluster:
#
for(k in 1:3){
  print(k)
  print(rownames(USArrests)[ctree.sc == k] )
}
```

```
## [1] 1
## [1] "Alabama"        "Alaska"         "Georgia"        "Louisiana"
## [5] "Mississippi"    "North Carolina" "South Carolina" "Tennessee"
## [1] 2
##  [1] "Arizona"    "California" "Colorado"   "Florida"    "Illinois"
##  [6] "Maryland"   "Michigan"   "Nevada"     "New Mexico" "New York"
## [11] "Texas"
## [1] 3
##  [1] "Arkansas"      "Connecticut"   "Delaware"      "Hawaii"
##  [5] "Idaho"         "Indiana"       "Iowa"          "Kansas"
##  [9] "Kentucky"      "Maine"         "Massachusetts" "Minnesota"
## [13] "Missouri"      "Montana"       "Nebraska"      "New Hampshire"
## [17] "New Jersey"    "North Dakota"  "Ohio"          "Oklahoma"
## [21] "Oregon"        "Pennsylvania"  "Rhode Island"  "South Dakota"
## [25] "Utah"          "Vermont"       "Virginia"      "Washington"
## [29] "West Virginia" "Wisconsin"     "Wyoming"
```

The scaling affects the how states go to different clusters. `UrbanPop` has a different unit (percentages) from the other predictors (number per 100,000). It is recommended that scaling is done in this case.

7. PCA and K-Means Clustering (20% | 0%)

Marking scheme: MSc only. (a) 2% and the rest 3% each.

In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data.

(a) Generate a simulated data set with 20 observations in each of three classes (i.e. 60 observations total), and 50 variables.

Hint: There are a number of functions in `R` that you can use to generate data. One example is the `rnorm()` function; `runif()` is another option. Be sure to add a mean shift to the observations in each class so that there are three distinct classes.

Solution:

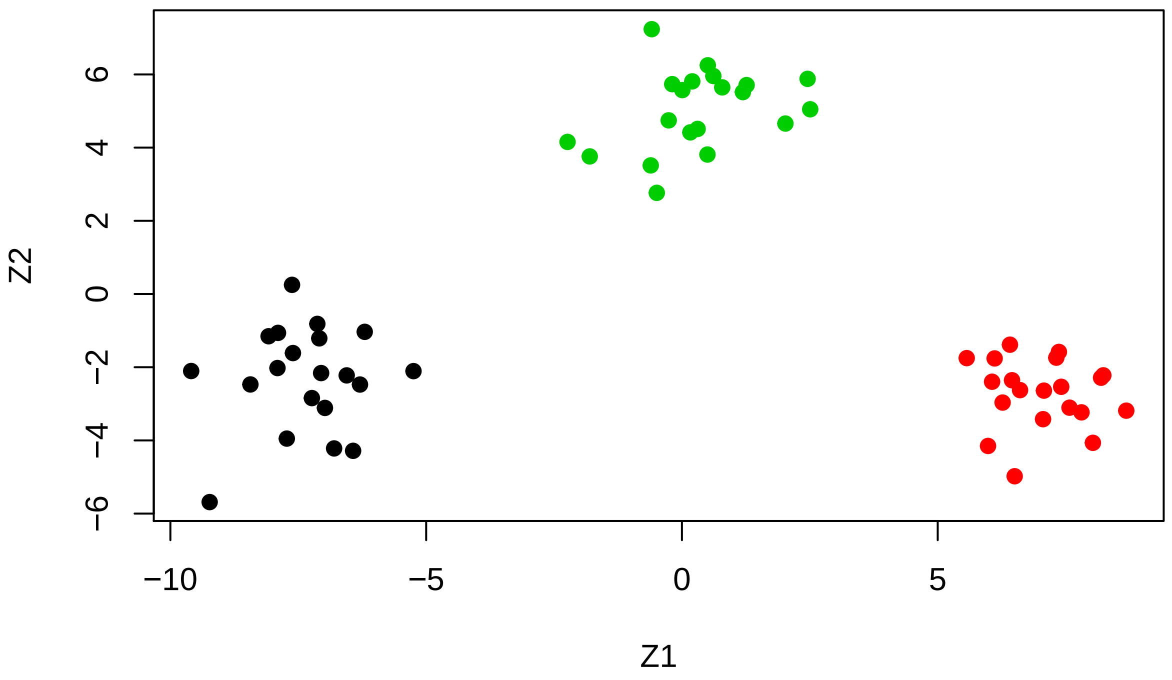
```r
set.seed(100)

# Part (a) generate data:
```

```r
#
K = 3  # the number of classes
n = 20 # the number of samples per class
p = 50 # the number of variables

# Create data for class 1:
X_1 <- matrix(rnorm(n*p), nrow=n, ncol=p)
for(row in 1:n){
  X_1[row,] <- X_1[row,] + rep(1, p)
}

# Create data for class 2:
X_2 <- matrix(rnorm(n*p), nrow=n, ncol=p)
for(row in 1:n){
  X_2[row,] <- X_2[row,] + rep(-1, p)
}

# Create data for class 3:
X_3 <- matrix(rnorm(n*p), nrow=n, ncol=p)
for(row in 1:n){
  X_3[row,] <- X_3[row,] + c(rep(+1, p/2), rep(-1, p/2))
}

X = rbind(X_1, X_2, X_3)
labels = c(rep(1,n), rep(2,n), rep(3,n)) # the "true" labels of the points
```

(b) Perform PCA on the 60 observations and plot the first two principal components' eigenvector. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue on to part (c). If not, then return to part (a) and modify the simulation so that there is greater separation between the three classes. Do not continue to part (c) until the three classes show at least some separation in the first two principal component eigenvectors.

Solution:

```r
#pr.out = prcomp(X, scale=TRUE)
pr.out = prcomp(X, scale=FALSE)
plot(pr.out$x[,1], pr.out$x[,2],
col=labels, xlab="Z1", ylab="Z2", pch=19)
```



(c) Perform K-means clustering of the observations with K = 3. How well do the clusters that you obtained in K-means clustering compare to the true class labels?

Hint: You can use the `table()` function in `R` to compare the true class labels to the class labels obtained by clustering. Be careful how you interpret the results: K-means clustering will arbitrarily number the clusters, so you cannot simply check whether the true class labels and clustering labels are the same.

Solution:

```
km.out.3 = kmeans(X, 3, nstart=100)

table(clustered = km.out.3$cluster, original = labels)
```

```
##          original
## clustered  1  2  3
##         1  0 20  0
##         2  0  0 20
##         3 20  0  0
```

The three classes were perfectly clustered in their original classes.

(d) Perform K-means clustering with K = 2. Describe your results.

Solution:

```
km.out.2 = kmeans(X, 2, nstart=100)

table(clustered = km.out.2$cluster, original = labels)
```

```
##          original
## clustered  1  2  3
##         1 20  0  0
##         2  0 20 20
```

Two of the classes were merged to form a new cluster.

(e) Now perform K-means clustering with K = 4, and describe your results.

Solution:

```
km.out.4 = kmeans(X, 4, nstart=100)

table(clustered = km.out.4$cluster, original = labels)
```

```
##          original
## clustered  1  2  3
##         1  8  0  0
##         2  0  0 20
##         3 12  0  0
##         4  0 20  0
```

One of the clusters has been split up into two clusters.

(f) Now perform K-means clustering with K = 3 on the first two principal components, rather than on the raw data. That is, perform K-means clustering on the 60 × 2 matrix of which the first column is the first principal component's corresponding eigenvector, and the second column is the second principal component's corresponding eigenvector. Comment on the results.

Solution:

```
km.out.pr.3 = kmeans(pr.out$x[,1:2], 3, nstart=100)

table(clustered = km.out.pr.3$cluster, original = labels)
```

```
##          original
## clustered  1  2  3
##         1  0  0 20
##         2 20  0  0
##         3  0 20  0
```

All three classes have been perfectly clustered.

(g) Using the `scale()` function, perform K-means clustering with K = 3 on the data after scaling each variable to have standard deviation one. How do these results compare to those obtained in (b)? Explain.

Solution:

```
km.out.sc.3 <- kmeans(scale(X), 3, nstart = 100)
table(clustered = km.out.sc.3$cluster, original = labels)
```

```
##          original
## clustered  1  2  3
##         1  0 20  0
##         2 20  0  0
##         3  0  0 20
```

Scaling does not affect the clustering.