Touray_Assignment6

Sheikh-Sedat Touray

2023-10-07

This exercise involves the **Auto** dataset.

```
library(ISLR)
?Auto
Auto$origin = as.factor(Auto$origin)
summary(Auto)
```

```
##       mpg          cylinders      displacement     horsepower        weight    
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613  
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225  
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2804  
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978  
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615  
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140  
##                                                                              
##   acceleration        year       origin                  name    
##  Min.   : 8.00   Min.   :70.00   1:245   amc matador       :  5  
##  1st Qu.:13.78   1st Qu.:73.00   2: 68   ford pinto        :  5  
##  Median :15.50   Median :76.00   3: 79   toyota corolla    :  5  
##  Mean   :15.54   Mean   :75.98           amc gremlin       :  4  
##  3rd Qu.:17.02   3rd Qu.:79.00           amc hornet        :  4  
##  Max.   :24.80   Max.   :82.00           chevrolet chevette:  4  
##                                          (Other)           :365  
```

convert origin 1 = American, 2 = European, 3 = Japanese

```
library(plyr)
Auto$origin <- revalue(Auto$origin, c('1' = 'American', '2' = 'European', '3' = 'Japanese')) # renaming
head(Auto)
```

```
##   mpg cylinders displacement horsepower weight acceleration year   origin
## 1  18         8          307        130   3504         12.0   70 American
## 2  15         8          350        165   3693         11.5   70 American
## 3  18         8          318        150   3436         11.0   70 American
## 4  16         8          304        150   3433         12.0   70 American
## 5  17         8          302        140   3449         10.5   70 American
## 6  15         8          429        198   4341         10.0   70 American
##                        name
## 1 chevrolet chevelle malibu
## 2         buick skylark 320
## 3        plymouth satellite
## 4             amc rebel sst
## 5               ford torino
## 6          ford galaxie 500
```

(a) Create a new variable origin2 that takes value **1** if a car is American, and **0** otherwise.

```
origin2 <- factor(ifelse (Auto$origin == "American", 1,0))#creating dummy variables for American and ot
head(origin2)
```

```
## [1] 1 1 1 1 1 1
## Levels: 0 1
```

```
# Now we add the new column (origin2) to the Auto dataset
Auto2 <- cbind(Auto,origin2)
Auto2$origin <- NULL#removes the origin variable
head(Auto2,5)
```

```
##   mpg cylinders displacement horsepower weight acceleration year
## 1  18         8          307        130   3504         12.0   70
## 2  15         8          350        165   3693         11.5   70
## 3  18         8          318        150   3436         11.0   70
## 4  16         8          304        150   3433         12.0   70
## 5  17         8          302        140   3449         10.5   70
##                        name origin2
## 1 chevrolet chevelle malibu       1
## 2         buick skylark 320       1
## 3        plymouth satellite       1
## 4             amc rebel sst       1
## 5               ford torino       1
```

(b) Split Auto data into a training set and a test set, placing approximately 80% and 20% of observations in each set.

```
library(caTools)
set.seed(11)
total_rows <- nrow(Auto2)

# Calculate the number of rows for the training and testing sets
train_rows <- round(0.8 * total_rows)  # 80% for training
test_rows <- total_rows - train_rows  # 20% for testing

# Generate random indices for the training set
train_indices <- sample(1:total_rows, train_rows)

# Create the training and testing datasets
train_data <- Auto2[train_indices, ]
test_data <- Auto2[-train_indices, ]
```

Origin2 must be changed back into factor to avoid error *factor predictors must have at most 32 level*

```
train_data$name <- NULL#removes the name variable
```

(c) Perform Classification Tree on the training data in order to predict origin2. Use cross- validation to prune the tree. Plot the resulting tree. Evaluate performance on the test data. What test error do you obtain?

```
library(tree)
#change this to factors so that it can make a classification tree without having level issues
origin2= factor(train_data$origin2)
tree_origin2 <- tree(origin2~.-origin2,train_data)
```

```r
summary(tree_origin2)#gives summary statistics of tree
```

```
##
## Classification tree:
## tree(formula = origin2 ~ . - origin2, data = train_data)
## Variables actually used in tree construction:
## [1] "displacement" "horsepower"   "weight"       "mpg"          "acceleration"
## [6] "cylinders"
## Number of terminal nodes:  13
## Residual mean deviance:  0.2683 = 80.75 / 301
## Misclassification error rate: 0.04777 = 15 / 314
```

```r
#perform prediction on test data
tree_pred = predict(tree_origin2, test_data, type = "class")

table(prediction = tree_pred, truth = test_data$origin2) #confusion matrix
```

```
##           truth
## prediction  0  1
##          0 23  4
##          1  7 44
```

find below the test terror for my classification tree

```r
mean(tree_pred != test_data$origin2) #test error
```

```
## [1] 0.1410256
```

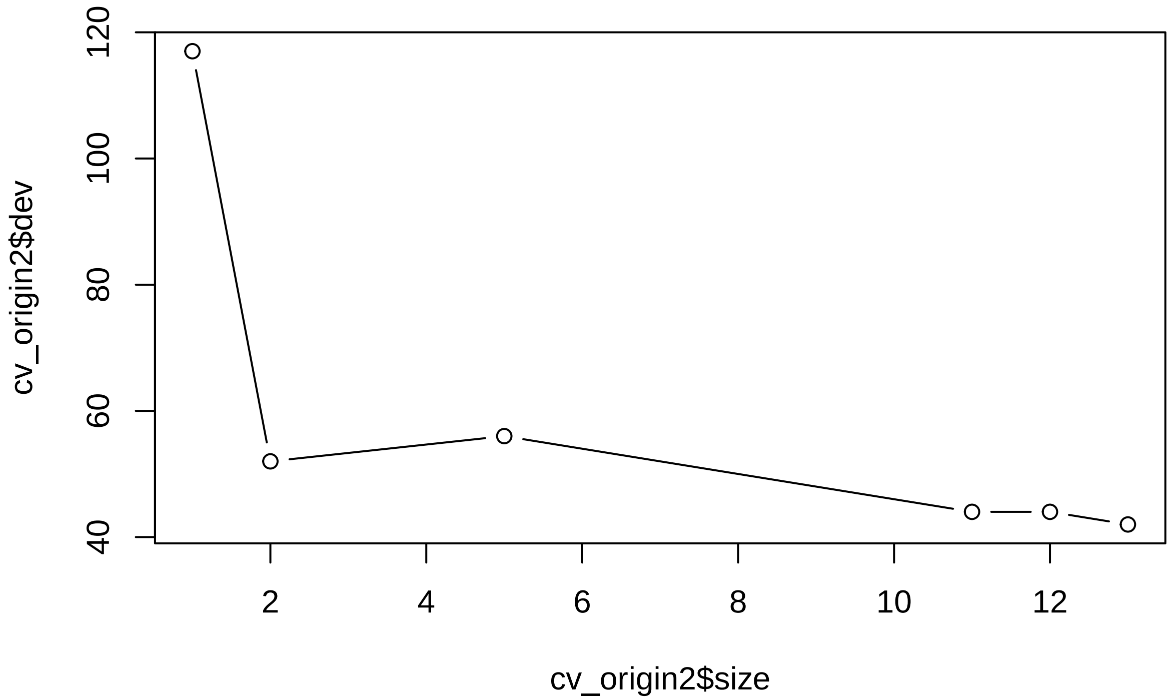
```r
mean(tree_pred == test_data$origin2) #test accuracy
```

```
## [1] 0.8589744
```

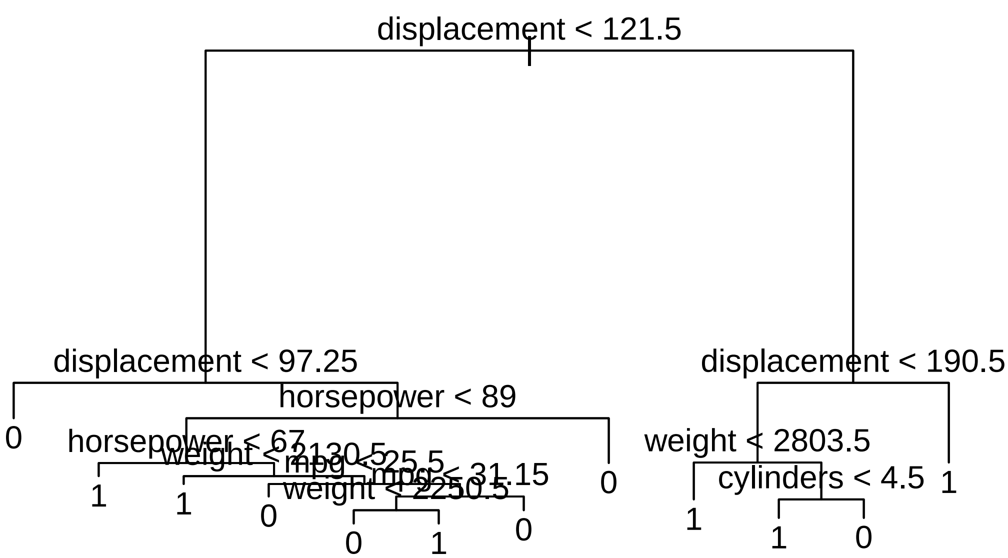
```r
set.seed(1014)
#perform cross validation
cv_origin2 = cv.tree(tree_origin2, FUN = prune.misclass)

cv_origin2
```

```
## $size
## [1] 13 12 11  5  2  1
##
## $dev
## [1]  42  44  44  56  52 117
##
## $k
## [1]      -Inf  0.000000  1.000000  2.500000  3.333333 76.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
#plot the CV
plot(cv_origin2$size, cv_origin2$dev, type = "b")
text(tree_origin2, pretty=TRUE, cex=0.8)
```




```r
#prune tree and plot resulting tree
prune.origin2 <- prune.tree(tree_origin2, best = 12)
plot(prune.origin2)
text(prune.origin2,pretty=TRUE)
```



find below the test terror for my pruned classification tree

```r
#predict test data on pruned tree
prune.pred = predict(prune.origin2, test_data,type="class")
table(prediction=prune.pred,truth=test_data$origin2)
```

```
##            truth
## prediction  0  1
##          0 23  4
##          1  7 44
```

```r
mean(prune.pred != test_data$origin2) #test error
```

```
## [1] 0.1410256
```

(d) Perform Random Forest on the training data in order to predict origin2. Evaluate performance on the test data. What test error do you obtain?

```r
library(randomForest)
```

```
## randomForest 4.7-1.1

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

```r
set.seed(222)
#random forest model
randori <- randomForest(origin2~., train_data, mtry =5, importance = TRUE)
```

```r
#predicting new data on the random forest model
pred.randori <- predict(randori,newdata = test_data)
mean(pred.randori!=test_data$origin2)
```

```
## [1] 0.05128205
```

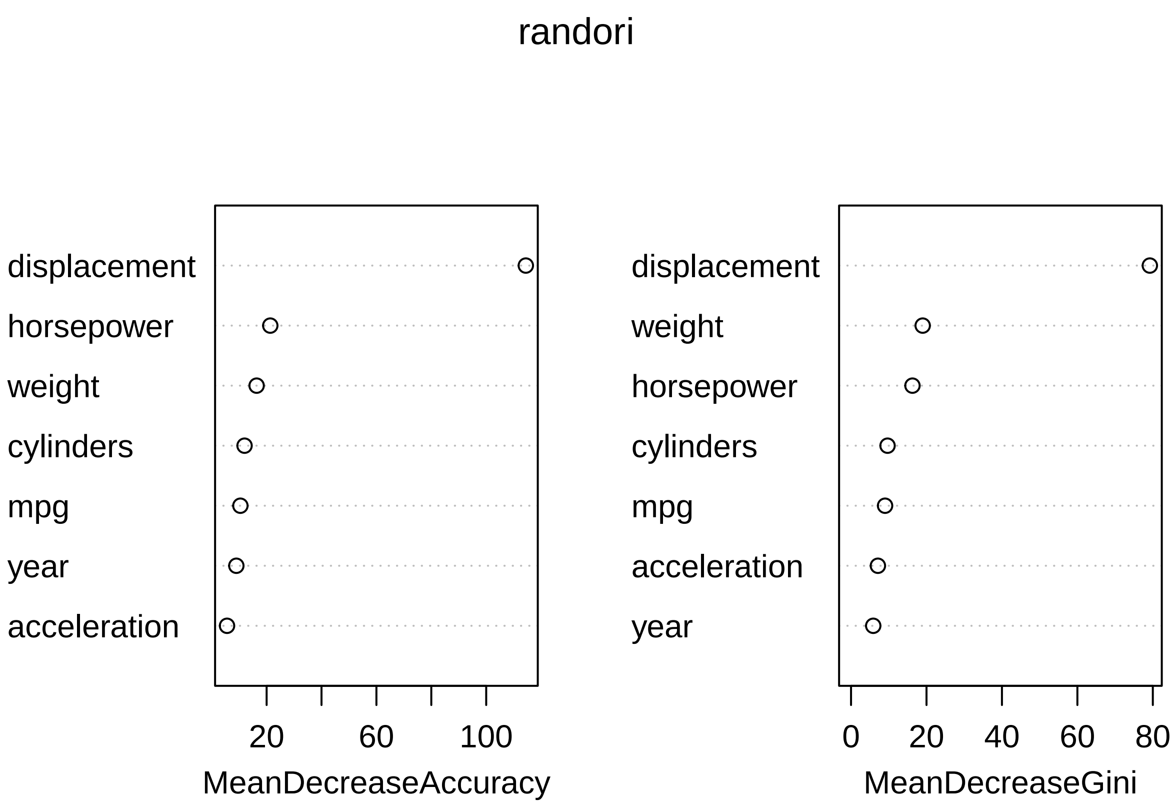
```r
#confusion matrix
table(prediction=pred.randori,truth=test_data$origin2)
```

```
##           truth
## prediction  0  1
##          0 28  2
##          1  2 46
```

```r
#importance of variables
importance(randori)
```

```
##                      0         1 MeanDecreaseAccuracy MeanDecreaseGini
## mpg           3.151496 10.233190            10.443201         9.018948
## cylinders     5.252305 11.035547            11.961998         9.677918
## displacement 61.604081 92.878450           114.436643        79.205701
## horsepower    4.277305 20.895180            21.335553        16.277295
## weight        8.822431 12.276961            16.362796        18.992009
## acceleration  4.795932  2.603146             5.581027         7.116262
## year          3.460030  9.124179             8.970632         5.881521
```

```r
#plot of variables based on importance
varImpPlot(randori)
```



These plots above show important variables and displacement variable shows up as the most important variable.

e) Fit a Support Vector Classifier to the data with various values of cost, in order to predict whether a car is American or not. Report the cross-validation errors associated with different values of this parameter. Comment on your results.

```
library(e1071)
set.seed(433)
#cv on SV classifier
svmlinear =  tune(svm,origin2 ~ ., data = train_data, kernel = "linear", ranges = list(cost = c(0.001, 0
summary(svmlinear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.1147177
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.3730847 0.08899998
## 2 1e-02 0.2167339 0.09116283
## 3 1e+00 0.1180444 0.04847208
## 4 5e+00 0.1147177 0.04833760
```

```
## 5 1e+01 0.1211694 0.04995500
## 6 1e+02 0.1243952 0.05373235
```

The lowest error was achieved by using a cost of 1, which was 0.1147. The highest error was .3731 which was when cost was set to 0.001.

```r
set.seed(821)
# SV classifier model
svmlinear =  svm(origin2 ~ ., data = train_data, kernel = "linear", ranges = list(cost = c(0.001, 0.01,
summary(svmlinear)
```

```
##
## Call:
## svm(formula = origin2 ~ ., data = train_data, kernel = "linear",
##     ranges = list(cost = c(0.001, 0.01, 1, 5, 10, 100)))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  118
##
##  ( 58 60 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```r
#prediction of SV classifier, table and error
pred.svmlinear <- predict(svmlinear,newdata = test_data)
table(test_data$origin2,pred.svmlinear)
```

```
##    pred.svmlinear
##      0  1
##   0 26  4
##   1  6 42
```

```r
mean(pred.svmlinear!=test_data$origin2)
```

```
## [1] 0.1282051
```

As can be seen above the cross-validation gives a better error than just the classifier model

f) Now repeat (e), this time using Support Vector Machines (SVMs) with radial and polynomial basis kernels, with different values of gamma and degree and cost. Comment on your results.

```r
set.seed(439)
#SVM polynomial model
svmpol =  svm(origin2 ~ ., data = train_data, kernel = "polynomial", ranges = list(cost = c(0.1, 1, 5,
summary(svmpol)
```

```
##
## Call:
## svm(formula = origin2 ~ ., data = train_data, kernel = "polynomial",
```

```
##     ranges = list(cost = c(0.1, 1, 5, 10), degree = c(2, 3, 4)))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  157
##
##  ( 78 79 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```r
#predicting new data on the SVM model polynomial model
pred.svmpol <- predict(svmpol,newdata = test_data)
#confusion matrix
table(test_data$origin2,pred.svmpol)
```

```
##    pred.svmpol
##      0  1
##   0 17 13
##   1  4 44
```

```r
#computing test error obtained
mean(pred.svmpol!=test_data$origin2)
```

```
## [1] 0.2179487
```

```r
set.seed(833)
svmpolt =  tune(svm,origin2 ~ ., data = train_data, kernel = "polynomial", ranges = list(cost = c(0.1, 
summary(svmpolt)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##    10      3
##
## - best performance: 0.1431452
##
## - Detailed performance results:
##    cost degree     error dispersion
## 1   0.1      2 0.3562500 0.09172365
## 2   1.0      2 0.3180444 0.08002005
## 3   5.0      2 0.2735887 0.05204582
## 4  10.0      2 0.2705645 0.04426871
```

```
## 5   0.1      3 0.2670363 0.07297231
## 6   1.0      3 0.1811492 0.04855261
## 7   5.0      3 0.1556452 0.04460918
## 8  10.0      3 0.1431452 0.04785918
## 9   0.1      4 0.3468750 0.09002549
## 10  1.0      4 0.2990927 0.06622275
## 11  5.0      4 0.2709677 0.07136609
## 12 10.0      4 0.2806452 0.07468481
```

The lowest error was achieved by using a cost of 10 and degree of 3, which was 0.1431. The highest error was .3563 which was when cost was set to 0.1 and a degree 2. And the lowest CV error is lower than that of the regular model which is expected.

```r
set.seed(443)
svmrad =  svm(origin2 ~ ., data = train_data, kernel = "radial", ranges = list(cost = c(0.1, 1, 5, 10),
summary(svmrad)
```

```
##
## Call:
## svm(formula = origin2 ~ ., data = train_data, kernel = "radial",
##     ranges = list(cost = c(0.1, 1, 5, 10), gamma = c(0.01, 0.1, 1,
##         5, 10, 100)))
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  146
##
##  ( 76 70 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```r
#prediction of SVM radial classifier, table and error
pred.svmrad <- predict(svmrad,newdata = test_data)
table(test_data$origin2,pred.svmrad)
```

```
##    pred.svmrad
##      0  1
##   0 26  4
##   1  7 41
```

```r
mean(pred.svmrad!=test_data$origin2)
```

```
## [1] 0.1410256
```

```r
set.seed(838)
svmradt =  tune(svm,origin2 ~ ., data = train_data, kernel = "radial", ranges = list(cost = c(0.1, 1, 5
summary(svmradt)
```

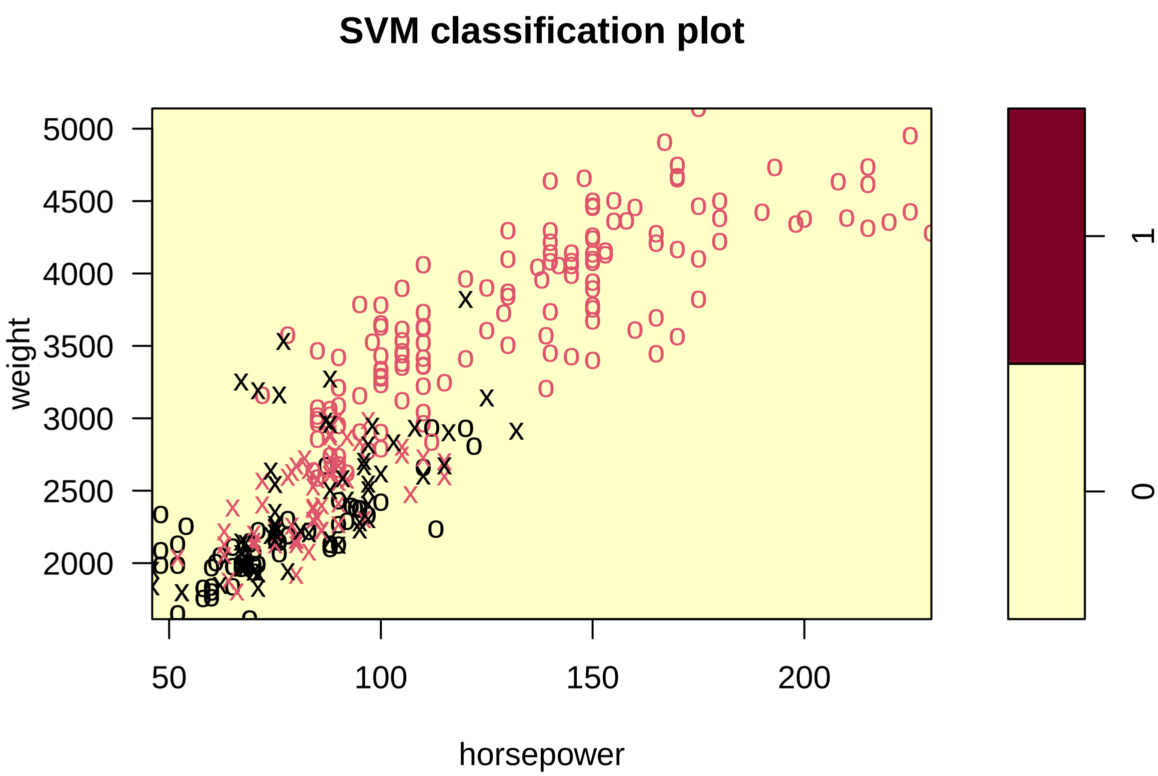
```
##
```

```
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.1
##
## - best performance: 0.1212702
##
## - Detailed performance results:
##    cost gamma     error dispersion
## 1   0.1 1e-02 0.3537298 0.09632565
## 2   1.0 1e-02 0.2168347 0.06485729
## 3   5.0 1e-02 0.1976815 0.05904702
## 4  10.0 1e-02 0.1657258 0.06740260
## 5   0.1 1e-01 0.2072581 0.05609084
## 6   1.0 1e-01 0.1722782 0.06719834
## 7   5.0 1e-01 0.1370968 0.06112452
## 8  10.0 1e-01 0.1212702 0.04811812
## 9   0.1 1e+00 0.3120968 0.07070870
## 10  1.0 1e+00 0.1627016 0.08281304
## 11  5.0 1e+00 0.1625000 0.04428299
## 12 10.0 1e+00 0.1719758 0.05074526
## 13  0.1 5e+00 0.3726815 0.05587271
## 14  1.0 5e+00 0.2487903 0.05128221
## 15  5.0 5e+00 0.2453629 0.05894702
## 16 10.0 5e+00 0.2485887 0.05455400
## 17  0.1 1e+01 0.3726815 0.05587271
## 18  1.0 1e+01 0.3537298 0.06525993
## 19  5.0 1e+01 0.3253024 0.07770900
## 20 10.0 1e+01 0.3253024 0.07770900
## 21  0.1 1e+02 0.3726815 0.05587271
## 22  1.0 1e+02 0.3726815 0.05587271
## 23  5.0 1e+02 0.3663306 0.05241763
## 24 10.0 1e+02 0.3663306 0.05241763
```

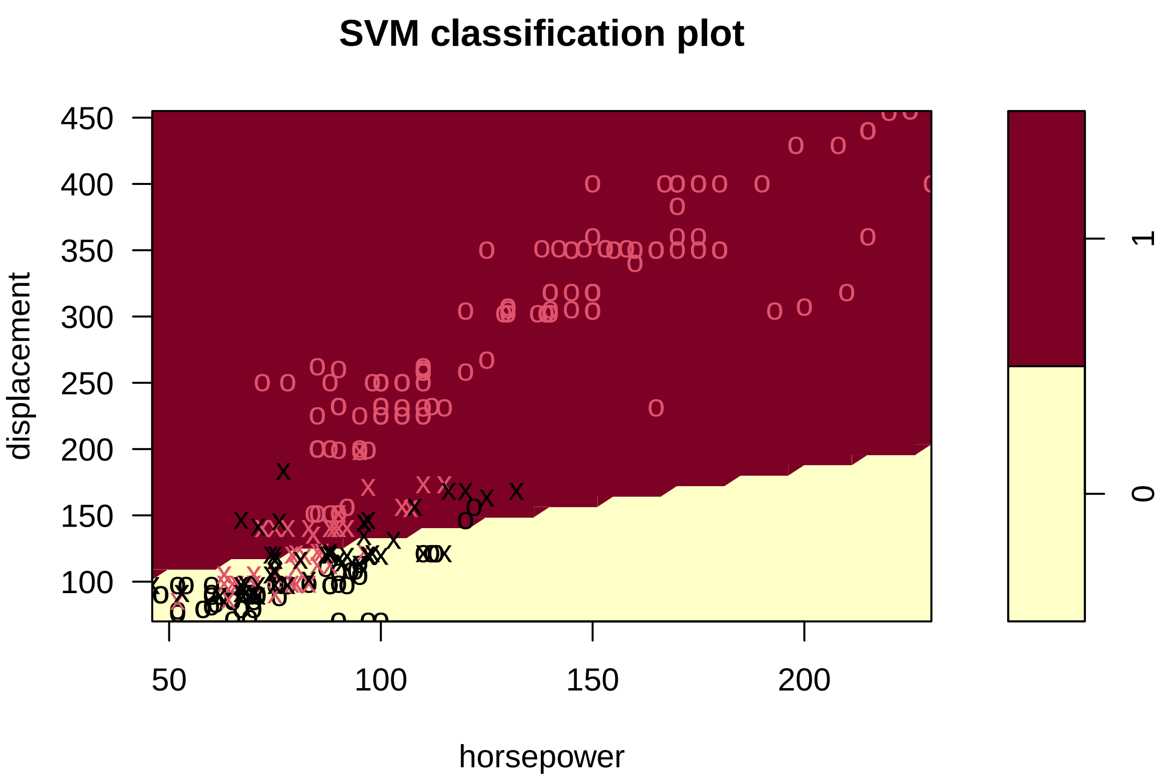
The lowest error was achieved by using a cost of 10 and gamma of 1e-01, which was 0.1212. The highest error was 0.3727 which was when cost was set to 0.1 and a gamma of 1e+01. And the lowest CV error is lower than that of the regular model which is expected.

(g) Make some plots to back up your assertions in (e) and (f).

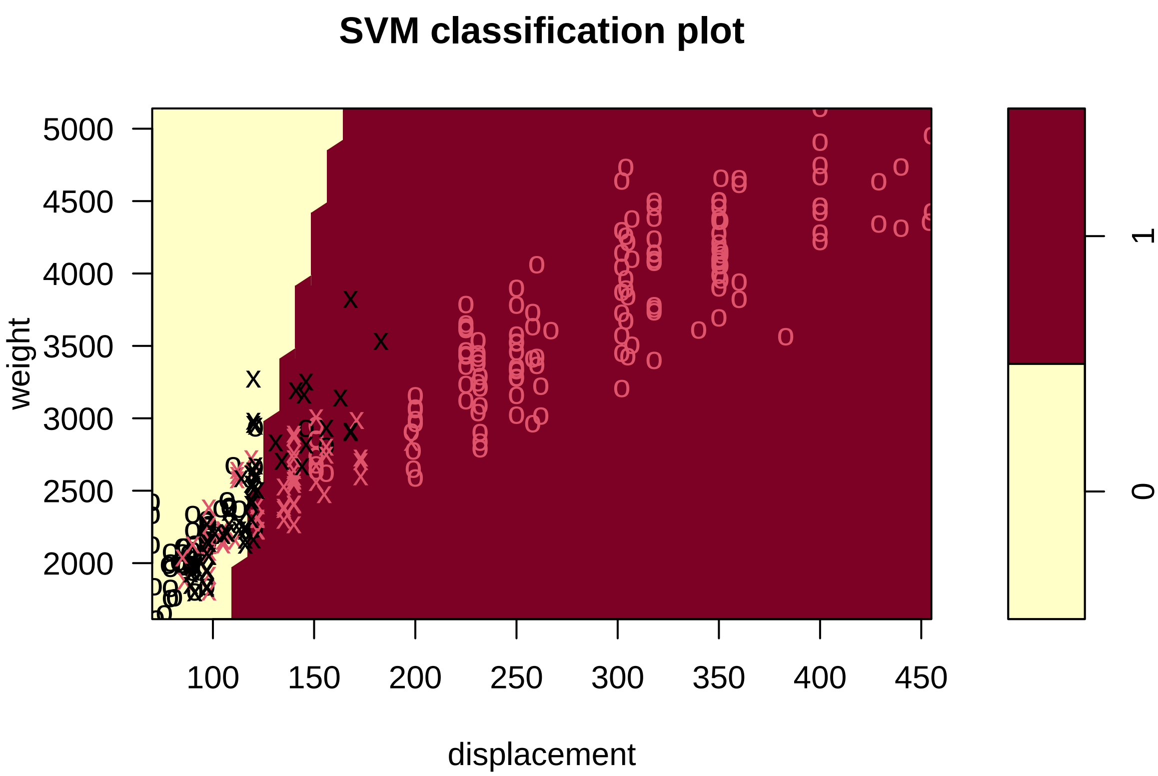
```r
library(kernlab)

plot(svmlinear,train_data,weight~horsepower)
```



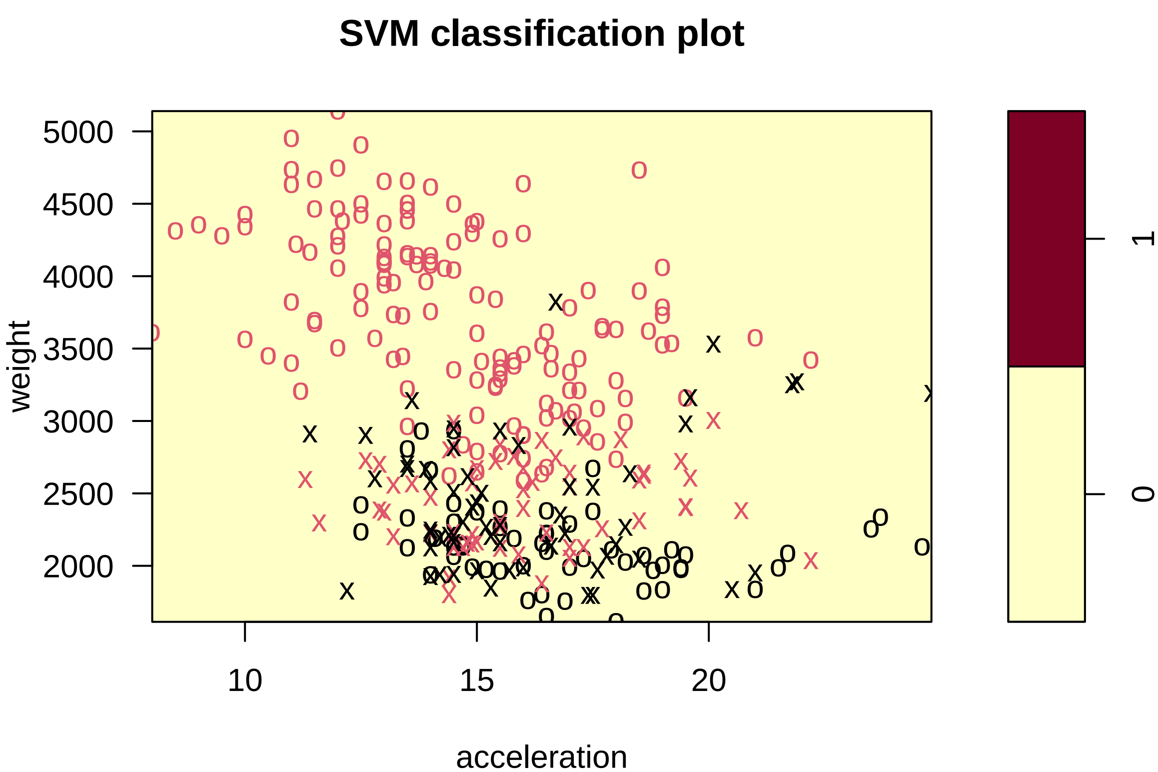

```r
plot(svmlinear,train_data,displacement~horsepower)
```



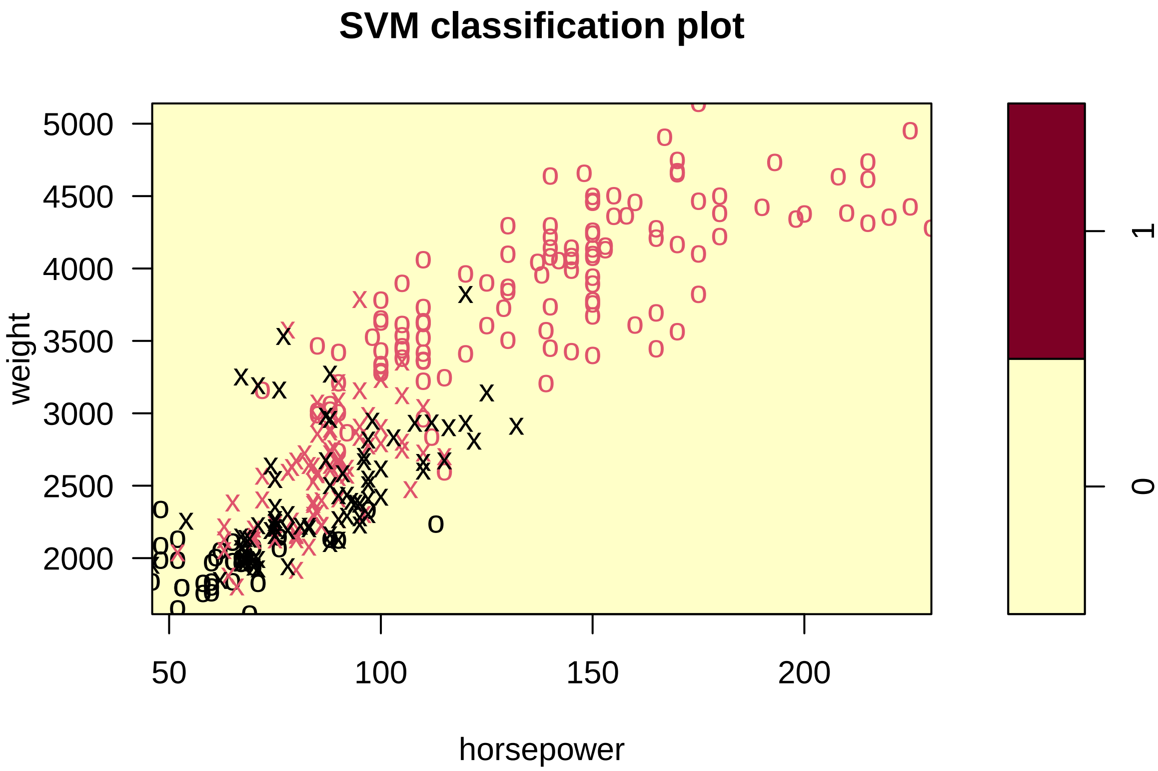

```r
plot(svmlinear,train_data,weight~displacement)
```



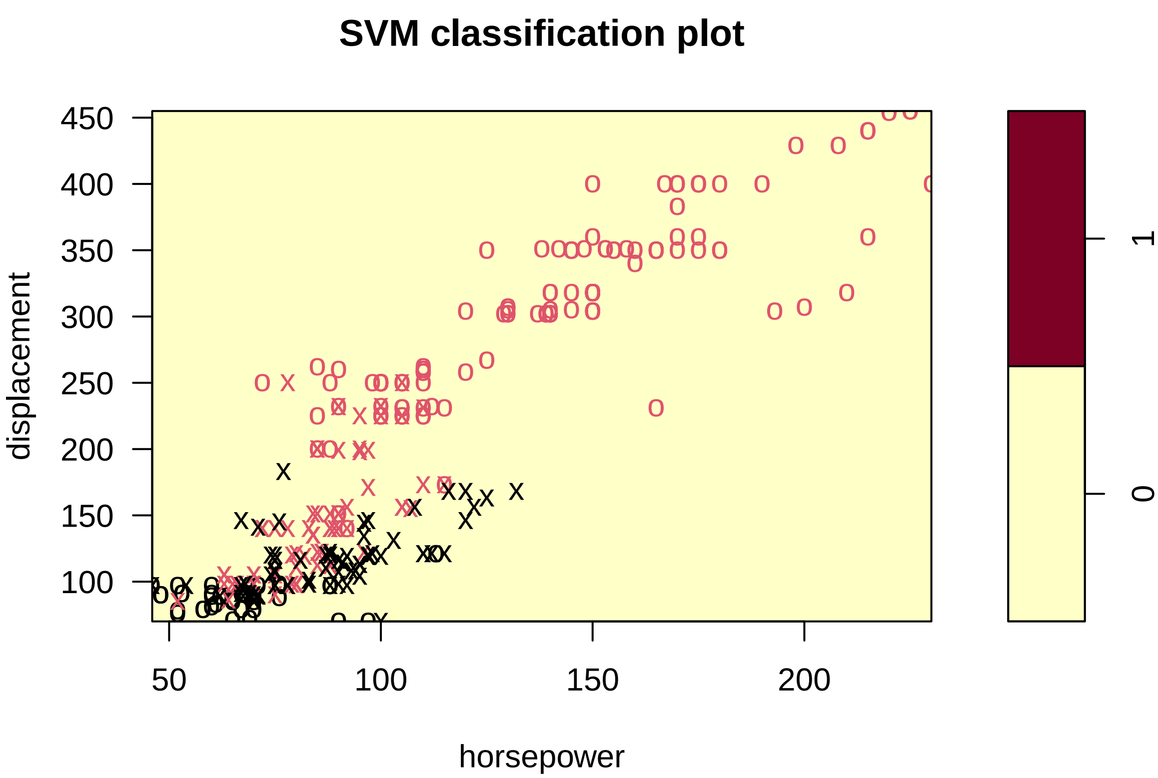

```
plot(svmlinear,train_data,weight~acceleration)
```



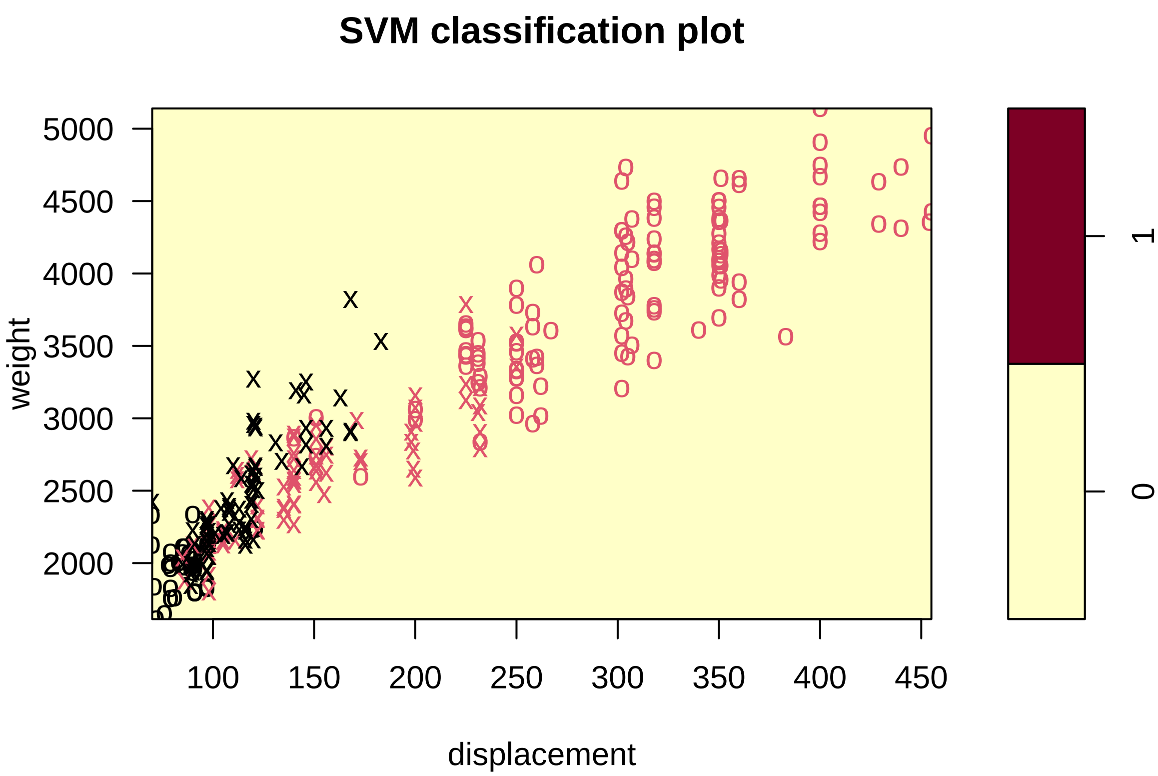

```
plot(svmpol,train_data,weight~horsepower)
```



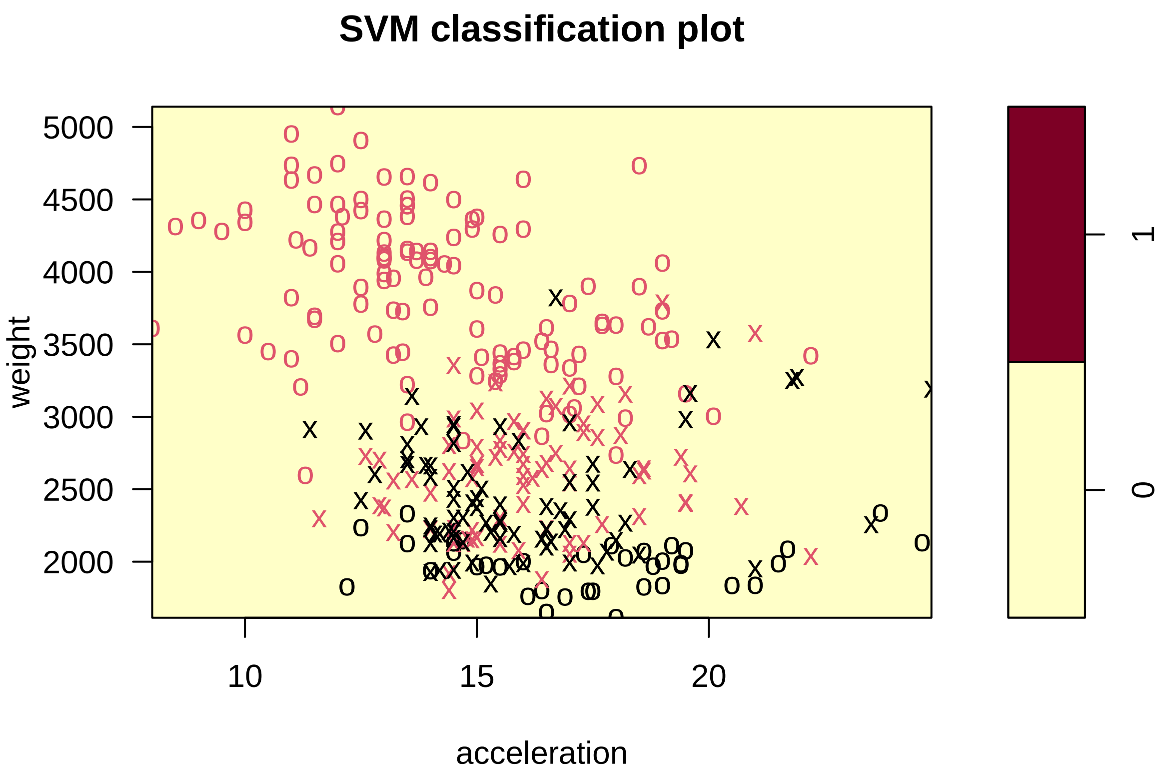

```
plot(svmpol,train_data,displacement~horsepower)
```



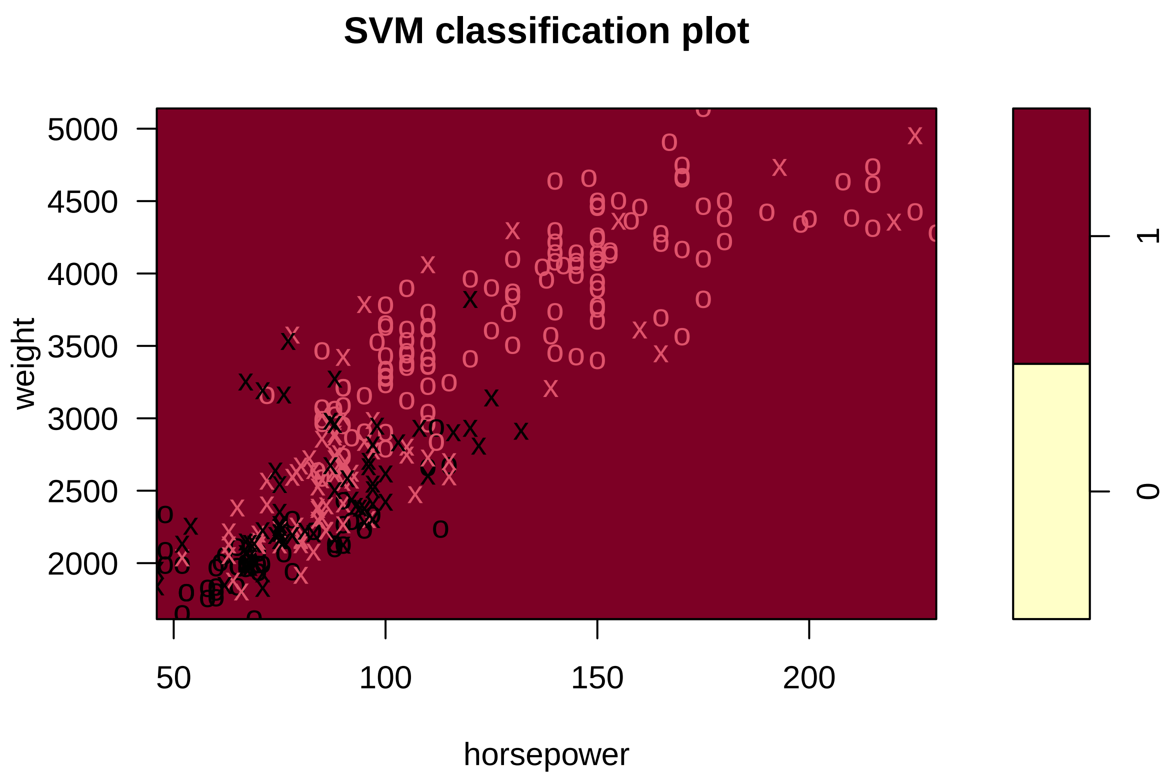

```
plot(svmpol,train_data,weight~displacement)
```



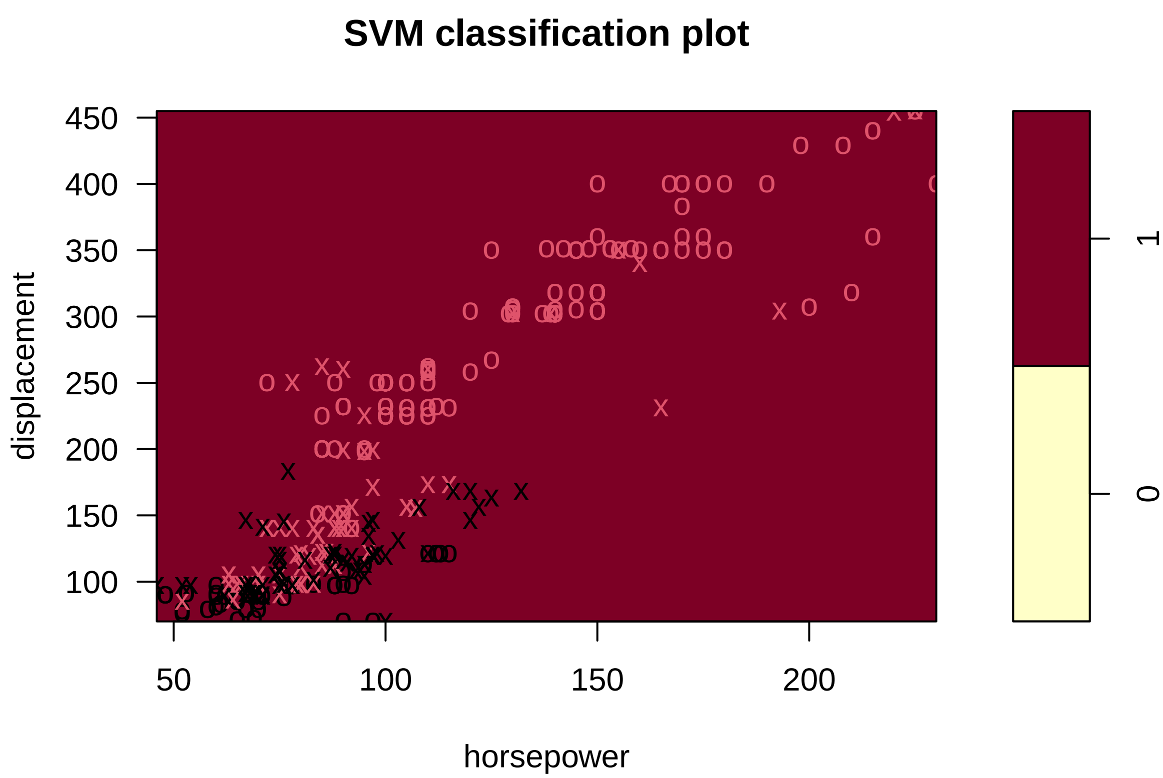

```r
plot(svmpol,train_data,weight~acceleration)
```



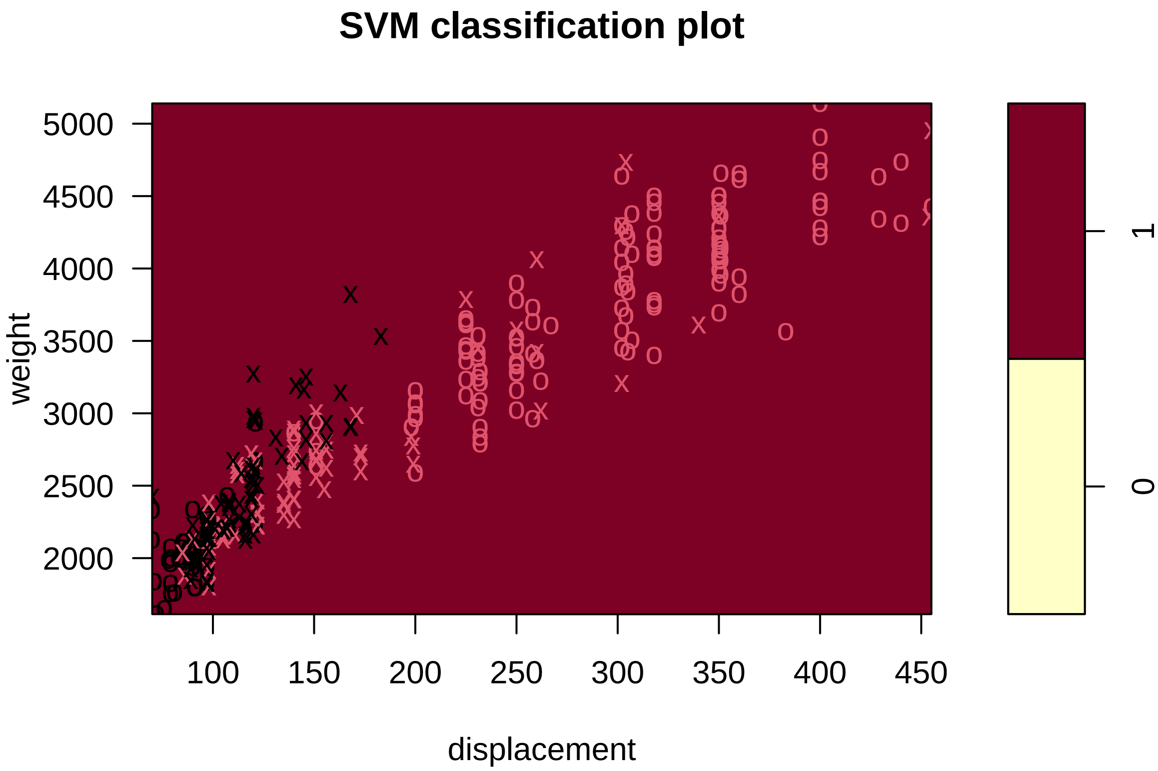

```r
plot(svmrad,train_data,weight~horsepower)
```



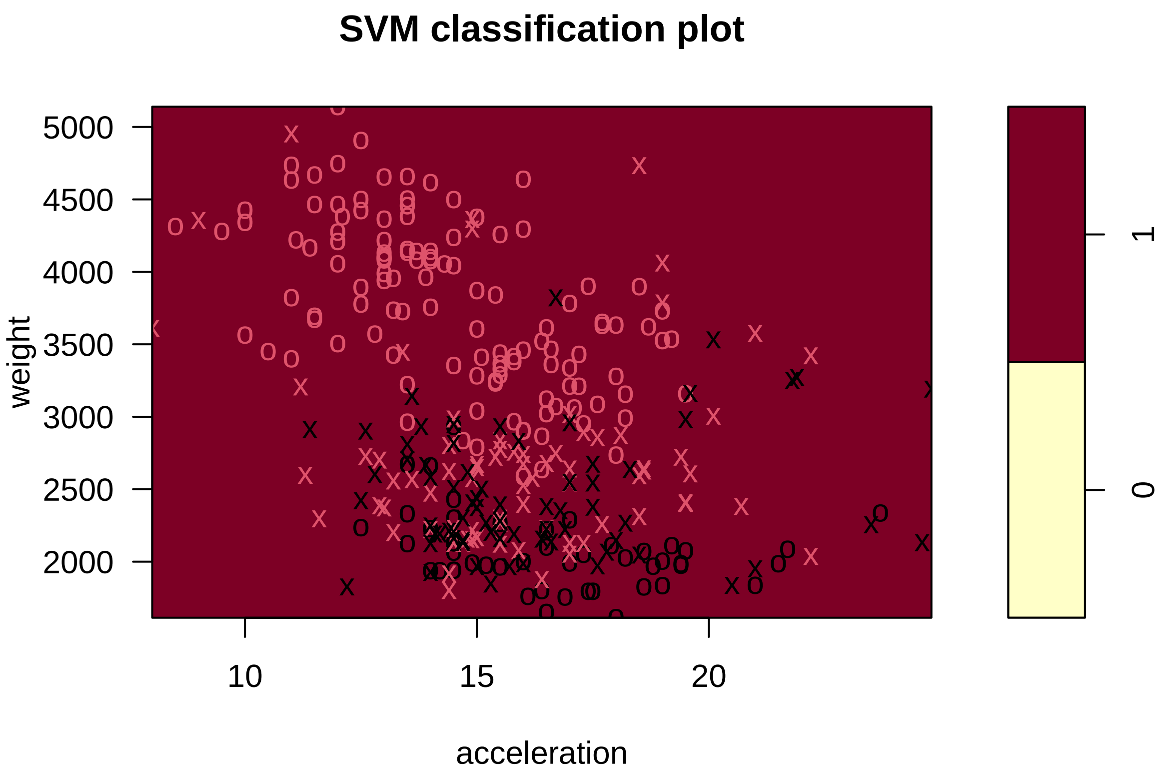

```
plot(svmrad,train_data,displacement~horsepower)
```




```
plot(svmrad,train_data,weight~displacement)
```




```r
plot(svmrad,train_data,weight~acceleration)
```

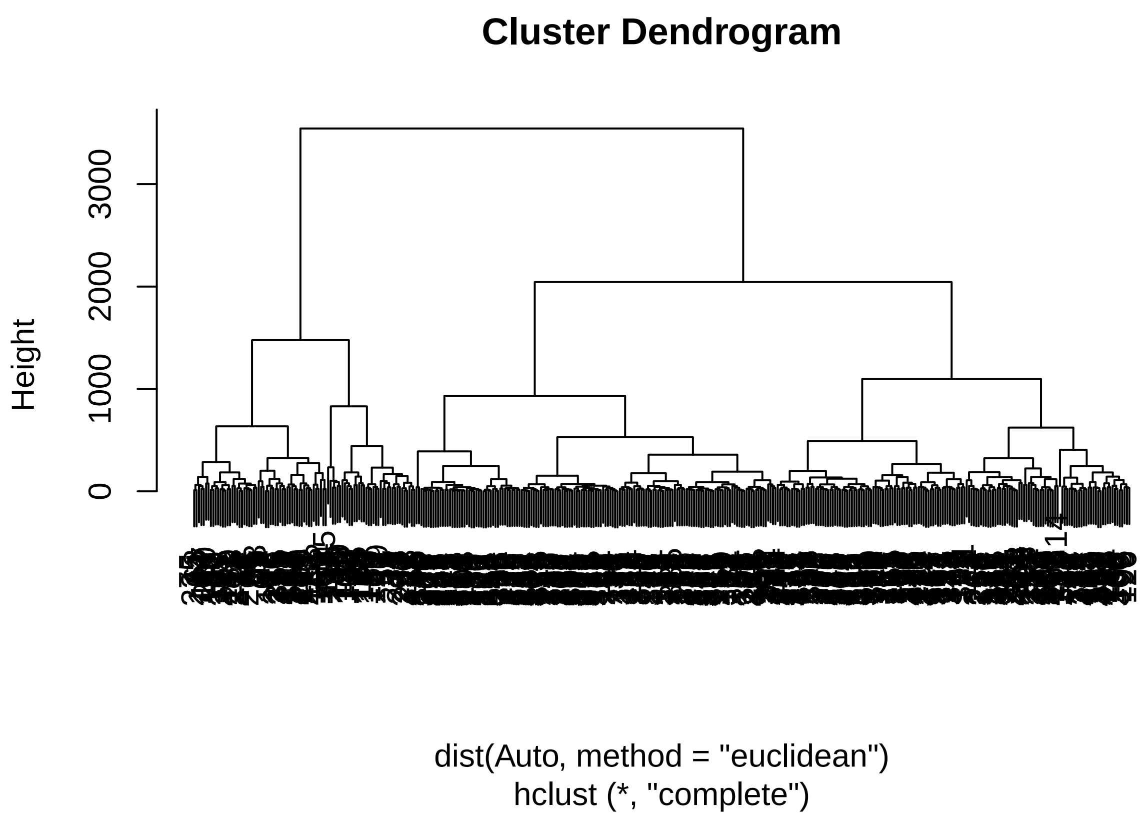


(h) Using Auto data (without origin and origin2 variables) and hierarchical clustering with complete linkage and Euclidean distance, cluster cars. Visualize the dendrogram and cut at a height that results in three (3) distinct clusters. Create a confusion matrix to compare the resulting clustering solution to origin variable.

```r
Auto$origin <- NULL
Auto$name <- NULL
head(Auto)
```

```
##   mpg cylinders displacement horsepower weight acceleration year
## 1  18         8          307        130   3504         12.0   70
## 2  15         8          350        165   3693         11.5   70
## 3  18         8          318        150   3436         11.0   70
## 4  16         8          304        150   3433         12.0   70
## 5  17         8          302        140   3449         10.5   70
## 6  15         8          429        198   4341         10.0   70
```

```r
hc.complete <- hclust(dist(Auto, method = "euclidean"), method = "complete")
plot(hc.complete)
```



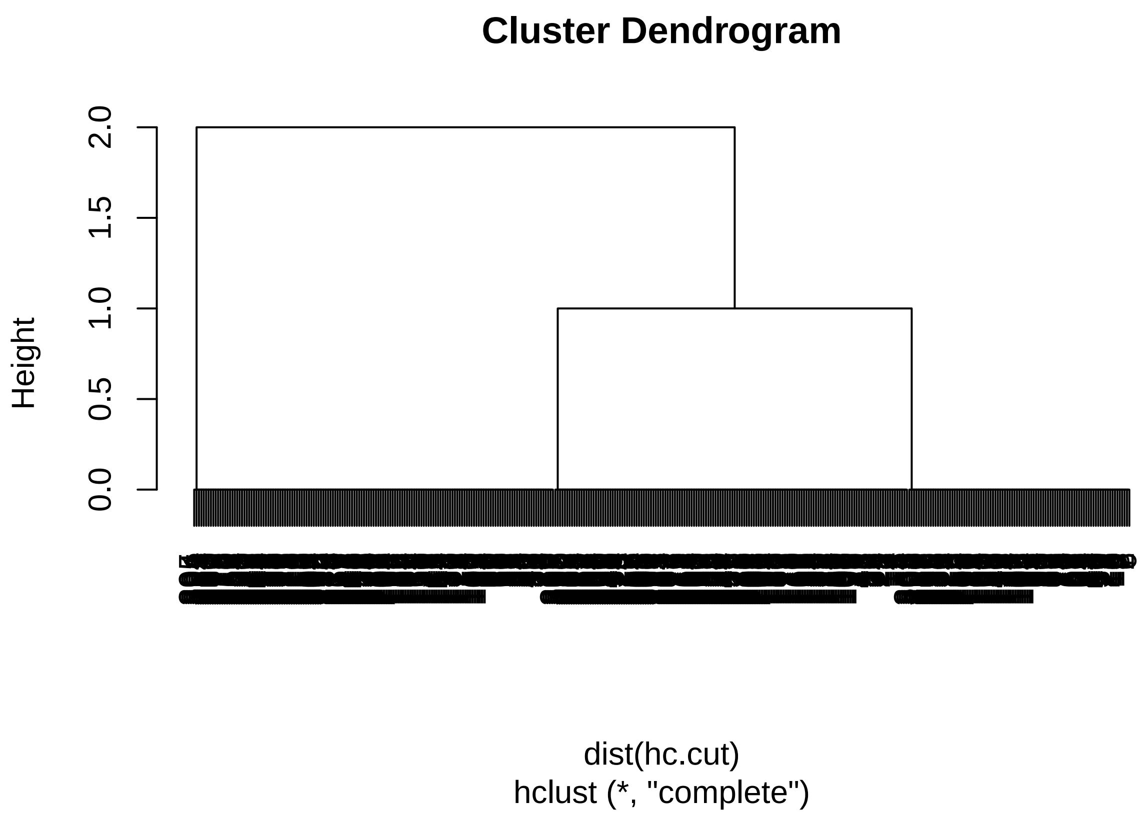
Now cut tree in three clusters

```r
hc.cut <- cutree(hc.complete,3)
table(hc.cut,Auto2$origin)
```

```
##
## hc.cut   0   1
##      1  36 112
##      2   1  92
##      3 110  41
```

In this table the small number shows misclassification and in this case there is only one misclassification

```r
hc.cut <- hclust(dist(hc.cut), method = "complete")
plot(hc.cut)
```



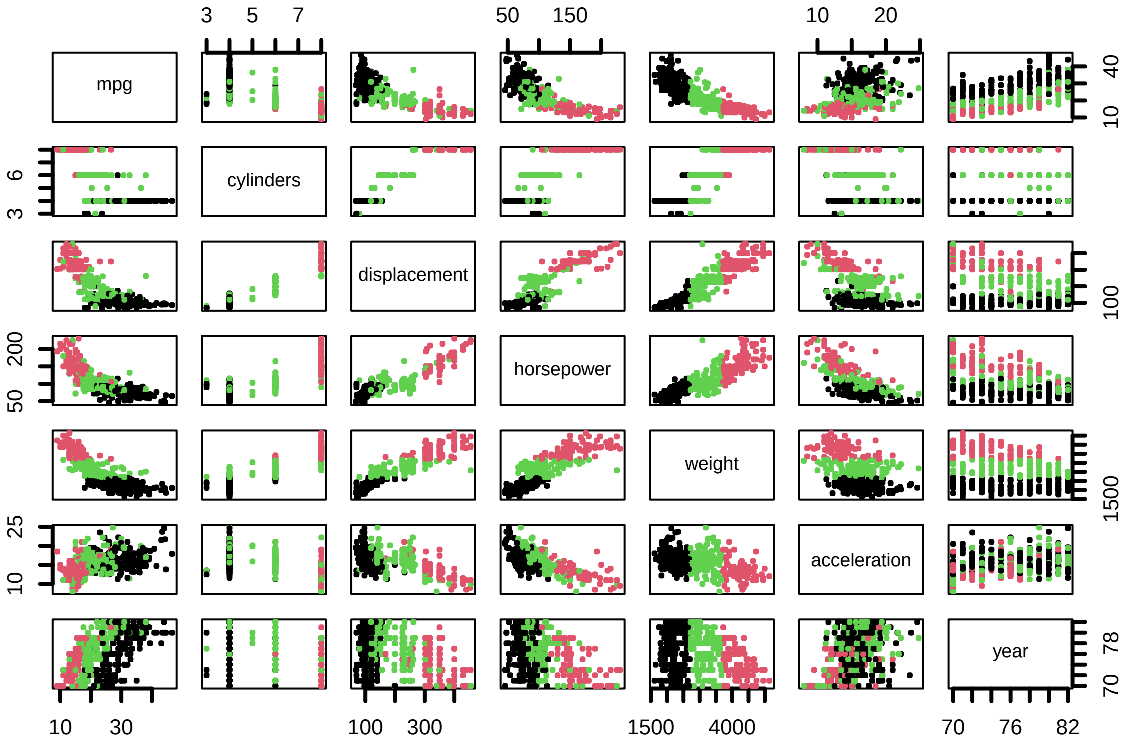
(i) Repeat (h) for K-Means clustering with K=3.

```
kcao <- kmeans(Auto, 3, nstart = 15)
kcao
```

```
## K-means clustering with 3 clusters of sizes 180, 90, 122
##
## Cluster means:
##        mpg cylinders displacement horsepower   weight acceleration     year
## 1 29.65167  4.038889     107.2083   77.16667 2222.828     16.33444 76.71111
## 2 14.63556  7.866667     344.1444  157.81111 4236.322     13.46333 74.01111
## 3 20.78934  5.819672     212.6148  105.40164 3162.582     15.90410 76.35246
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##   3   2   3   3   3   2   2   2   2   2   3   3   2   3   1   3   3   1   1   1
##  21  22  23  24  25  26  27  28  29  30  31  32  34  35  36  37  38  39  40  41
##   1   1   1   1   1   2   2   2   2   1   1   1   1   3   3   3   3   2   2   2
##  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61
##   2   2   2   2   3   1   3   3   1   1   1   1   1   1   1   1   1   1   1   1
##  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81
##   1   2   2   2   2   3   2   2   2   2   1   2   2   2   2   3   1   3   1   1
##  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101
##   1   1   1   1   2   3   2   2   2   2   2   2   2   2   2   2   3   3   3   3
## 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121
##   3   1   2   2   2   2   3   1   1   1   1   1   1   1   2   2   1   1   1   3
## 122 123 124 125 126 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142
##   3   1   3   3   3   3   3   1   1   1   1   2   3   3   2   2   2   2   2   1
## 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
```

```
##   1   1   1   1   1   1   1   1   1   1   3   3   3   3   2   2   2   2   2   2
## 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182
##   2   2   3   3   3   1   1   3   1   3   1   1   3   1   3   1   3   3   1   1
## 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202
##   1   1   1   1   1   2   2   2   2   3   3   3   3   1   1   1   1   3   3   3
## 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222
##   3   1   1   1   1   3   2   3   3   2   2   2   2   2   1   1   1   1   1   2
## 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242
##   2   2   2   3   3   3   3   2   2   2   2   1   3   1   3   1   1   1   1   3
## 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262
##   1   3   1   1   1   1   1   3   2   3   3   3   3   3   3   3   3   3   3   3
## 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282
##   3   3   3   2   1   1   1   1   1   3   3   1   3   3   3   3   1   1   3   3
## 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302
##   3   3   3   2   2   2   2   2   2   3   2   1   1   1   1   3   2   3   3   1
## 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322
##   1   1   1   1   1   3   1   1   1   1   1   1   3   3   3   1   3   1   1   1
## 323 324 325 326 327 328 329 330 332 333 334 335 336 338 339 340 341 342 343 344
##   1   3   1   1   1   3   3   1   1   1   3   1   1   1   1   1   1   3   1   1
## 345 346 347 348 349 350 351 352 353 354 356 357 358 359 360 361 362 363 364 365
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   3   3   3   3   3   2
## 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385
##   3   3   1   1   1   1   1   3   3   1   1   1   1   1   1   1   1   1   1   1
## 386 387 388 389 390 391 392 393 394 395 396 397
##   3   3   1   3   1   1   3   3   1   1   1   3
##
## Within cluster sum of squares by cluster:
## [1] 12164071 11029122 10655835
##  (between_SS / total_SS =  88.2 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```r
plot(Auto, col = kcao$cluster, cex = 0.2, pch=0.1, lwd=2)
```



(j) Using Auto data (without origin, and origin2 variables) perform PCA. Print the summary and interpret the results. Specifically, comment on the possible number of components and factor loadings. How can you characterize the first 2 principal components?

```r
dimnames(Auto)
```

```
## [[1]]
##   [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"
##  [13] "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"  "23"  "24"
##  [25] "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "34"  "35"  "36"  "37"
##  [37] "38"  "39"  "40"  "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48"  "49"
##  [49] "50"  "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"  "61"
##  [61] "62"  "63"  "64"  "65"  "66"  "67"  "68"  "69"  "70"  "71"  "72"  "73"
##  [73] "74"  "75"  "76"  "77"  "78"  "79"  "80"  "81"  "82"  "83"  "84"  "85"
##  [85] "86"  "87"  "88"  "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96"  "97"
##  [97] "98"  "99"  "100" "101" "102" "103" "104" "105" "106" "107" "108" "109"
## [109] "110" "111" "112" "113" "114" "115" "116" "117" "118" "119" "120" "121"
## [121] "122" "123" "124" "125" "126" "128" "129" "130" "131" "132" "133" "134"
## [133] "135" "136" "137" "138" "139" "140" "141" "142" "143" "144" "145" "146"
## [145] "147" "148" "149" "150" "151" "152" "153" "154" "155" "156" "157" "158"
## [157] "159" "160" "161" "162" "163" "164" "165" "166" "167" "168" "169" "170"
## [169] "171" "172" "173" "174" "175" "176" "177" "178" "179" "180" "181" "182"
## [181] "183" "184" "185" "186" "187" "188" "189" "190" "191" "192" "193" "194"
## [193] "195" "196" "197" "198" "199" "200" "201" "202" "203" "204" "205" "206"
## [205] "207" "208" "209" "210" "211" "212" "213" "214" "215" "216" "217" "218"
## [217] "219" "220" "221" "222" "223" "224" "225" "226" "227" "228" "229" "230"
## [229] "231" "232" "233" "234" "235" "236" "237" "238" "239" "240" "241" "242"
## [241] "243" "244" "245" "246" "247" "248" "249" "250" "251" "252" "253" "254"
## [253] "255" "256" "257" "258" "259" "260" "261" "262" "263" "264" "265" "266"
## [265] "267" "268" "269" "270" "271" "272" "273" "274" "275" "276" "277" "278"
## [277] "279" "280" "281" "282" "283" "284" "285" "286" "287" "288" "289" "290"
```

```
## [289] "291" "292" "293" "294" "295" "296" "297" "298" "299" "300" "301" "302"
## [301] "303" "304" "305" "306" "307" "308" "309" "310" "311" "312" "313" "314"
## [313] "315" "316" "317" "318" "319" "320" "321" "322" "323" "324" "325" "326"
## [325] "327" "328" "329" "330" "332" "333" "334" "335" "336" "338" "339" "340"
## [337] "341" "342" "343" "344" "345" "346" "347" "348" "349" "350" "351" "352"
## [349] "353" "354" "356" "357" "358" "359" "360" "361" "362" "363" "364" "365"
## [361] "366" "367" "368" "369" "370" "371" "372" "373" "374" "375" "376" "377"
## [373] "378" "379" "380" "381" "382" "383" "384" "385" "386" "387" "388" "389"
## [385] "390" "391" "392" "393" "394" "395" "396" "397"
##
## [[2]]
## [1] "mpg"          "cylinders"    "displacement" "horsepower"   "weight"
## [6] "acceleration" "year"
```

```
apply(Auto, 2, mean)
```

```
##          mpg    cylinders displacement   horsepower       weight acceleration
##    23.445918     5.471939   194.411990   104.469388  2977.584184    15.541327
##         year
##    75.979592
```

```
apply(Auto, 2, mean)
```

```
##          mpg    cylinders displacement   horsepower       weight acceleration
##    23.445918     5.471939   194.411990   104.469388  2977.584184    15.541327
##         year
##    75.979592
```

Standardize the the data

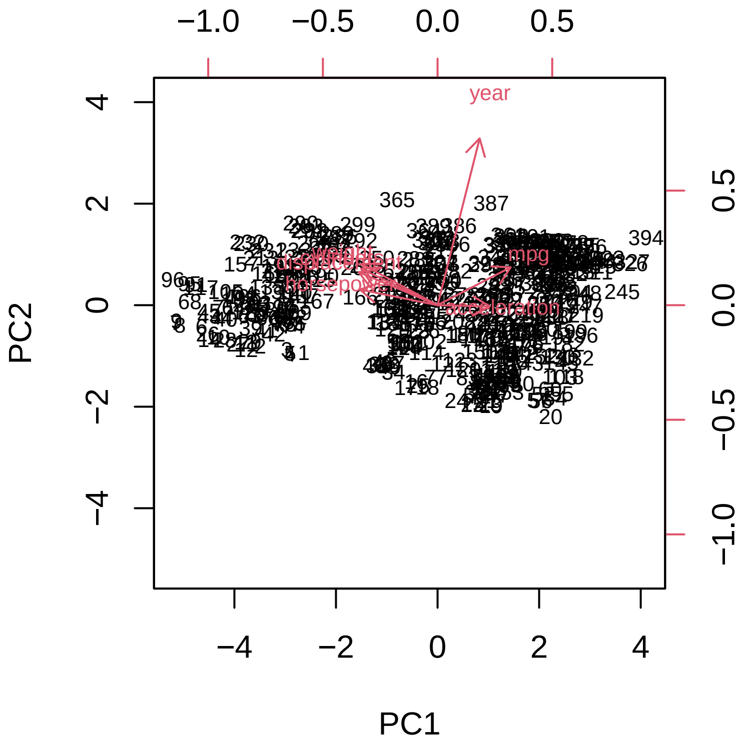
```
pca.out <- prcomp(Auto, scale=TRUE)
pca.out
```

```
## Standard deviations (1, .., p=7):
## [1] 2.2384450 0.9303716 0.8534599 0.4288532 0.3491652 0.2329317 0.1878575
##
## Rotation (n x k) = (7 x 7):
##                     PC1          PC2         PC3         PC4         PC5
## mpg           0.3981348  0.206758641  0.25721494 -0.75096624 -0.34077556
## cylinders    -0.4161242  0.198541133 -0.13915928 -0.47730649  0.49322226
## displacement -0.4292827  0.180362422 -0.10031610 -0.29784705  0.05658084
## horsepower   -0.4228129  0.085241832  0.16968441  0.04207625 -0.71128893
## weight       -0.4140457  0.224674565 -0.27610337  0.10773508 -0.26515768
## acceleration  0.2848971 -0.006971629 -0.89330772 -0.12112398 -0.23075501
## year          0.2295100  0.909674802  0.03724635  0.30243525  0.08896075
##                     PC6         PC7
## mpg          -0.2097589  0.09221162
## cylinders     0.3325483  0.43171605
## displacement -0.1429671 -0.81287676
## horsepower    0.5228025  0.06438539
## weight       -0.6965178  0.36715386
## acceleration  0.2237849 -0.05279944
## year          0.1281955 -0.05113155
```

```
names(pca.out)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

```
?biplot
biplot(pca.out, scale = 0, cex= 0.7)
```



The values of the variables are the components while the loadings are the names of the variables and the arrows show their direction. Displacement, year, and accelaration are negative while the rest of the variables are positive.