ISLR_CH9

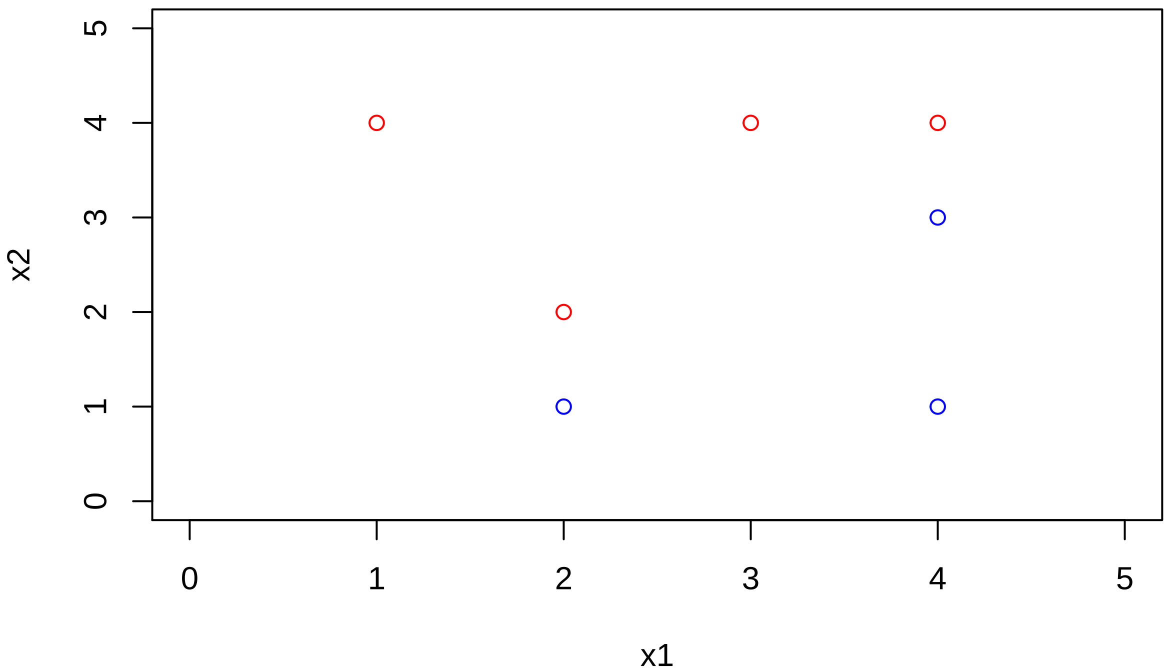
Sky Liu

3/21/2019

9.7.3

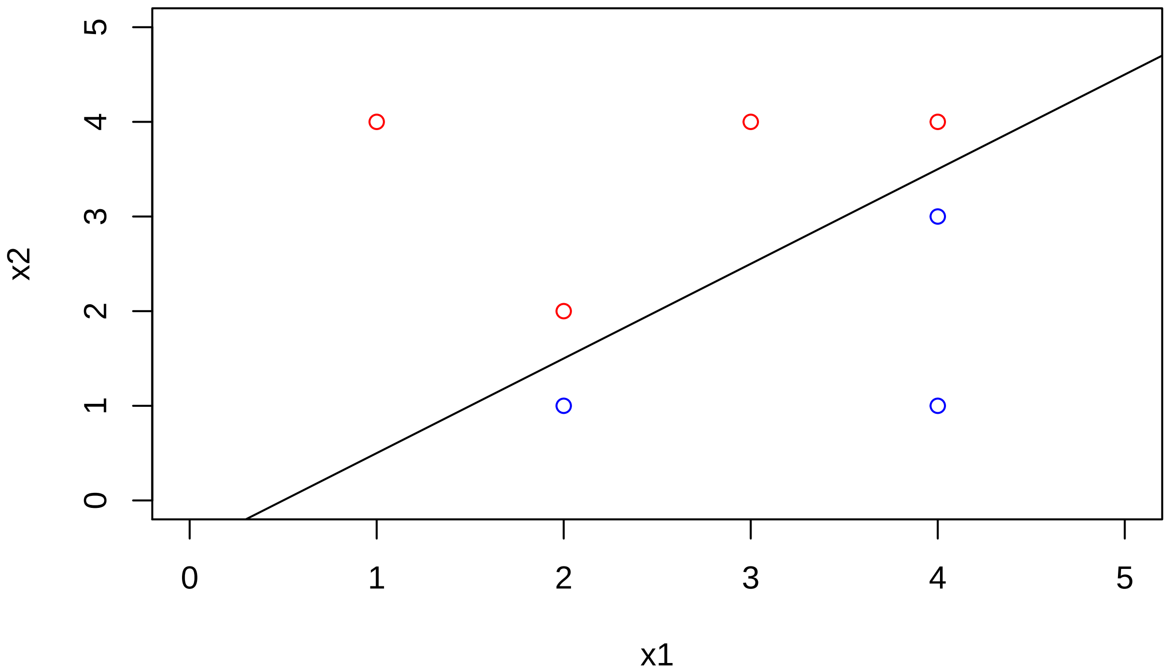
part a

```
x1 = c(3, 2, 4, 1, 2, 4, 4)
x2 = c(4, 2, 4, 4, 1, 3, 1)
colors = c("red", "red", "red", "red", "blue", "blue", "blue")
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
```



part b

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
```

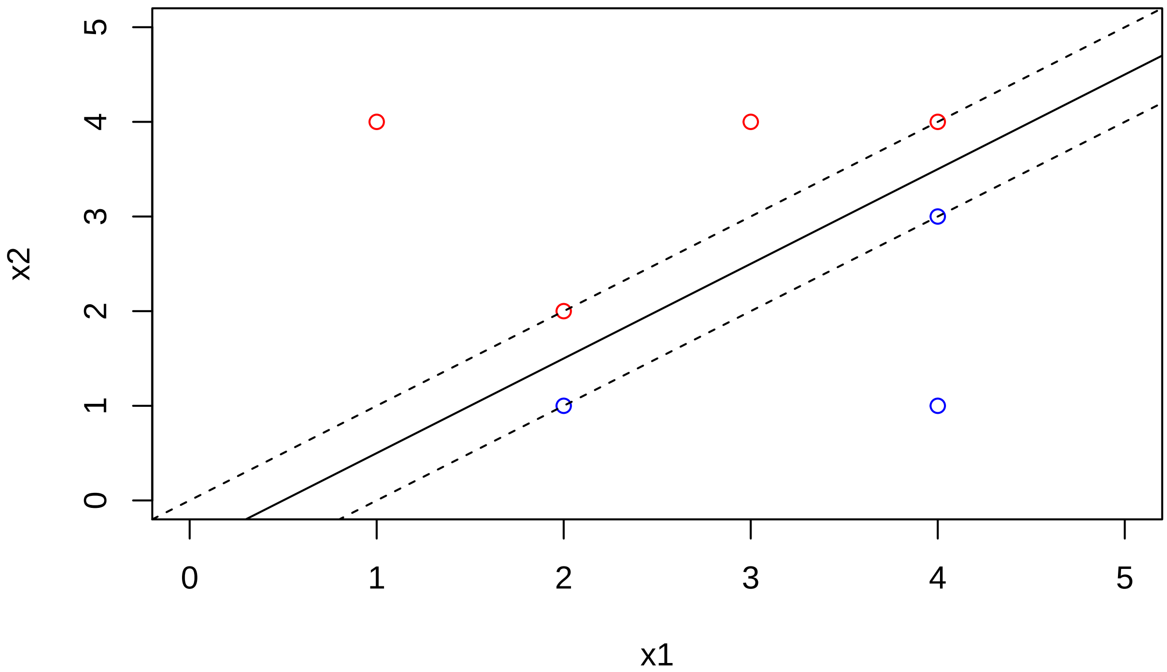


part c

$$-0.5 + X_1 - X_2 = 0.$$

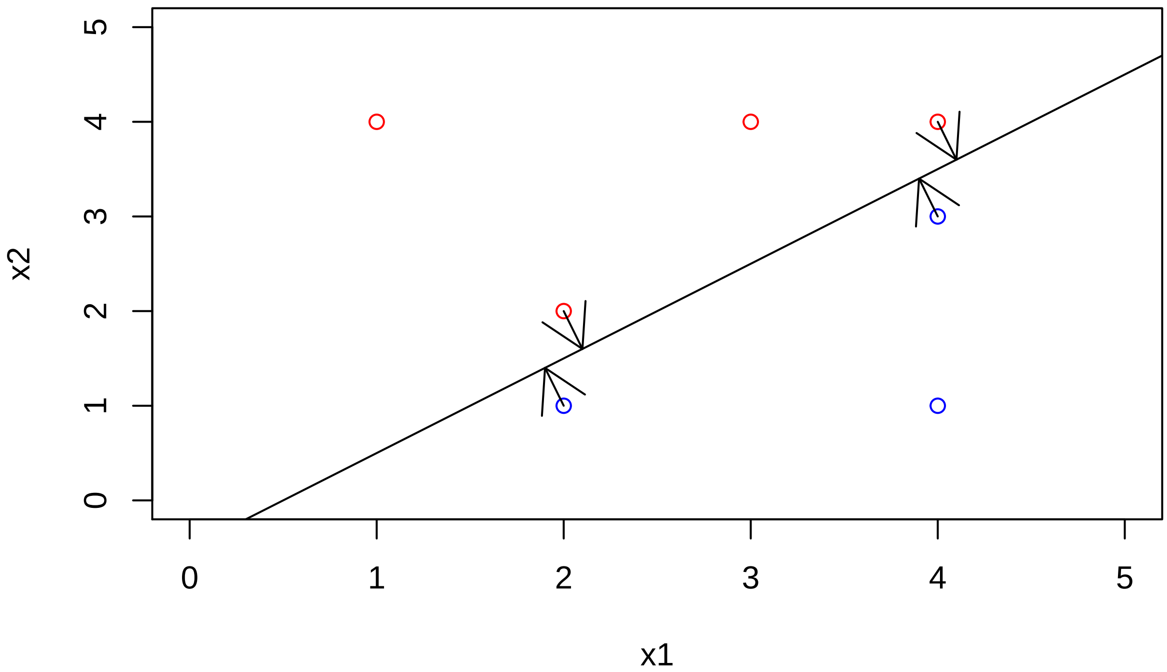
part d

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
abline(-1, 1, lty = 2)
abline(0, 1, lty = 2)
```



part e

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
arrows(2, 1, 1.9, 1.4)
arrows(2, 2, 2.1, 1.6)
arrows(4, 4, 4.1, 3.6)
arrows(4, 3, 3.9, 3.4)
```

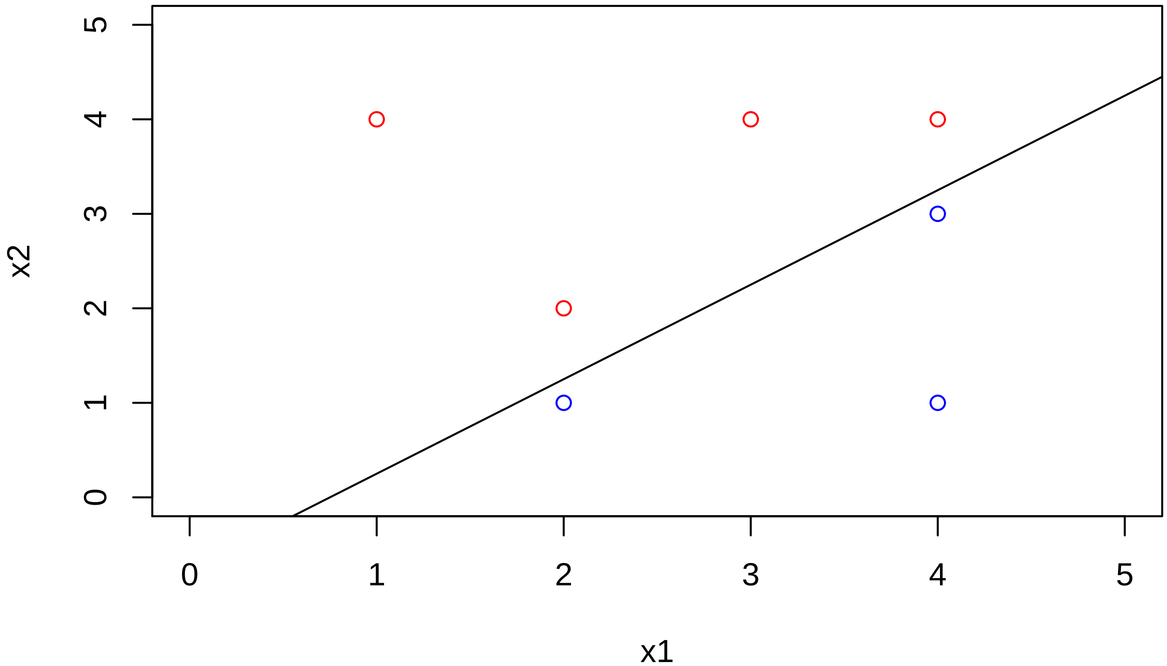


part f

because it's outside of the margin.

part g

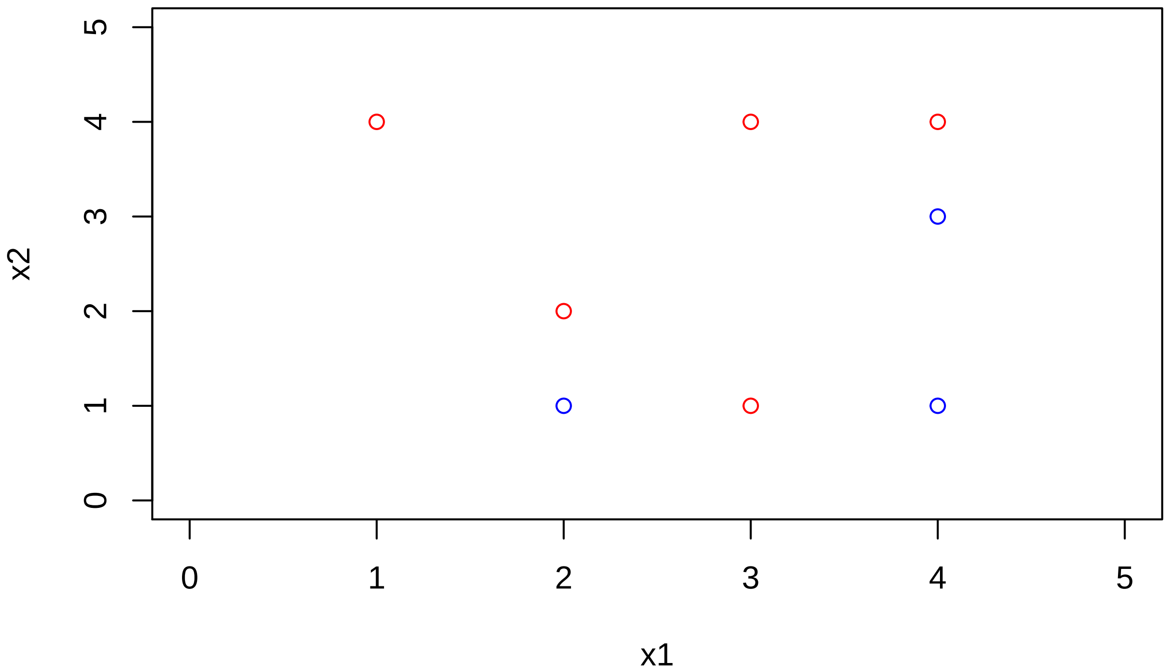
```r
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.75, 1)
```



$$-0.75 + X_1 - X_2 = 0.$$

h

```r
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
points(c(3),c(1), col = c("red"))
```



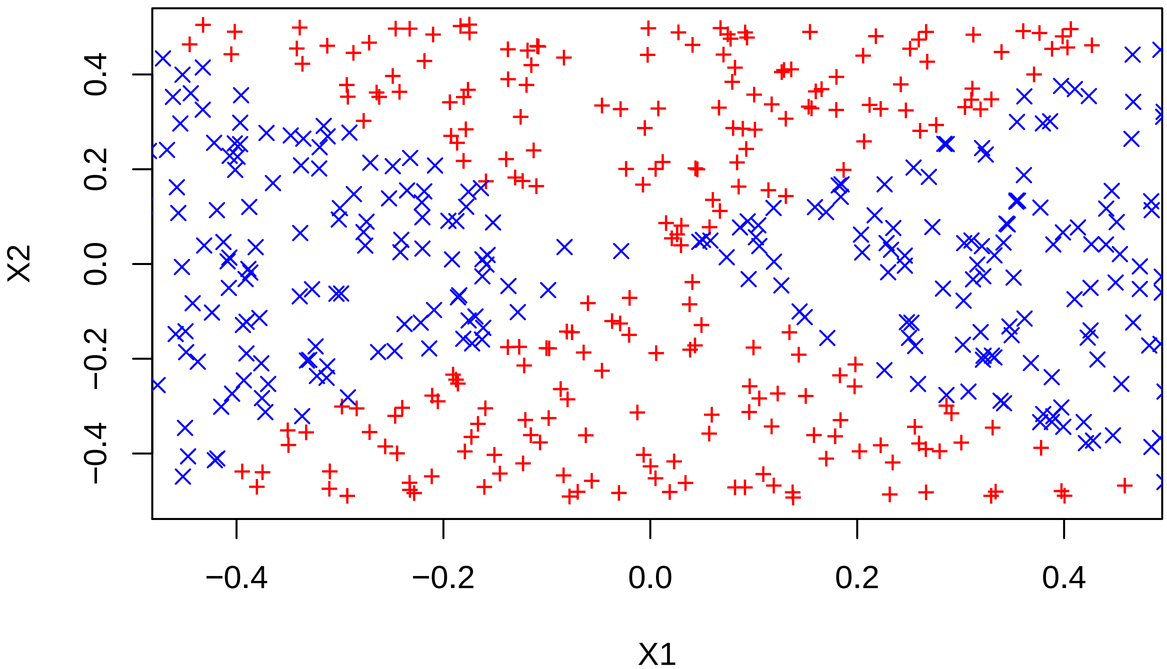
9.7.5

part a

```
set.seed(3)
x1 = runif(500) - 0.5
x2 = runif(500) - 0.5
y = 1 * (x1^2 - x2^2 > 0)
```

part b

```
plot(x1[y == 0], x2[y == 0], col = "red", xlab = "X1", ylab = "X2", pch = "+")
points(x1[y == 1], x2[y == 1], col = "blue", pch = 4)
```



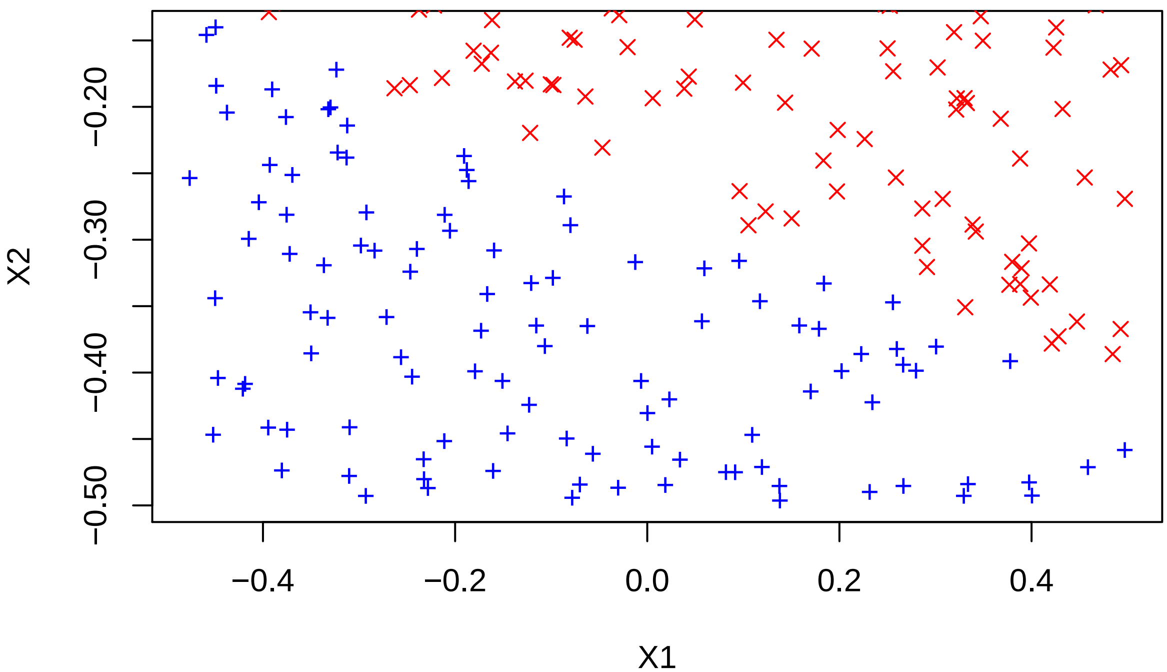
part c

```r
lm_fit = glm(y ~ x1 + x2, family = binomial)
summary(lm_fit)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = binomial)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.257  -1.194   1.115   1.152   1.201
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.06632    0.08960   0.740    0.459
## x1          -0.06040    0.31237  -0.193    0.847
## x2          -0.20247    0.30630  -0.661    0.509
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.64  on 499  degrees of freedom
## Residual deviance: 692.16  on 497  degrees of freedom
## AIC: 698.16
##
## Number of Fisher Scoring iterations: 3
```

part d

```
data = data.frame(x1 = x1, x2 = x2, y = y)
lm_prob = predict(lm_fit, data, type = "response")
# probability threshold = 0.53
lm_pred = ifelse(lm_prob > 0.53, 1, 0)
data_pos = data[lm_pred == 1, ]
data_neg = data[lm_pred == 0, ]
plot(data_pos$x1, data_pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(data_neg$x1, data_neg$x2, col = "red", pch = 4)
```



part e

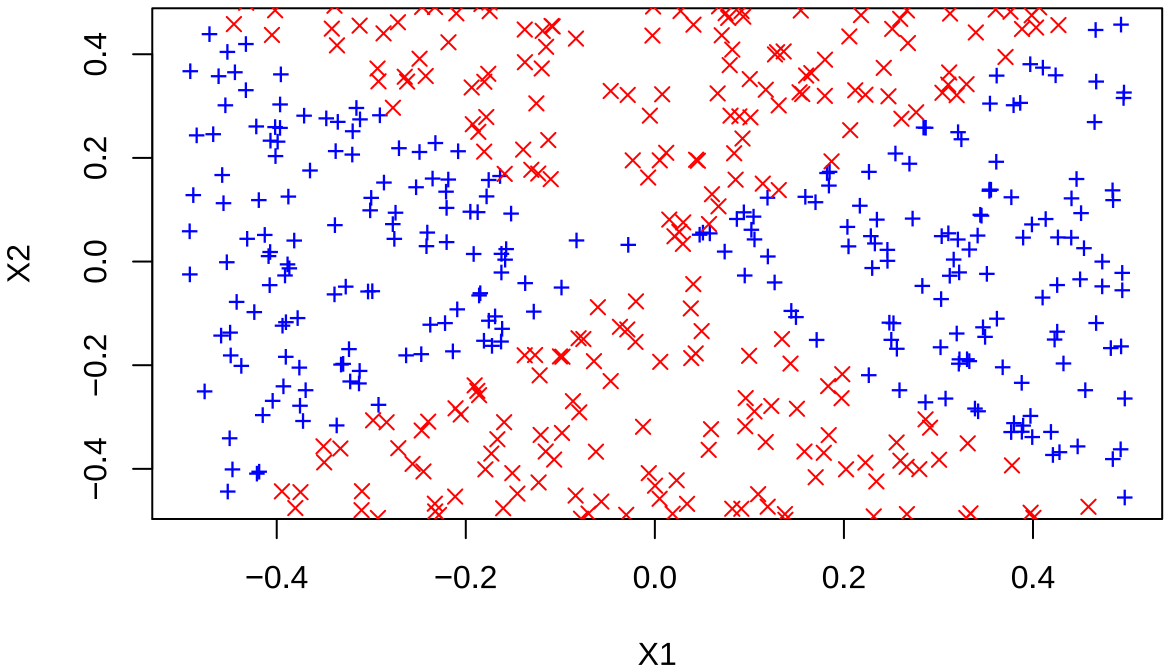
```
lm_fit = glm(y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), data = data, family = binomial)
```

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

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

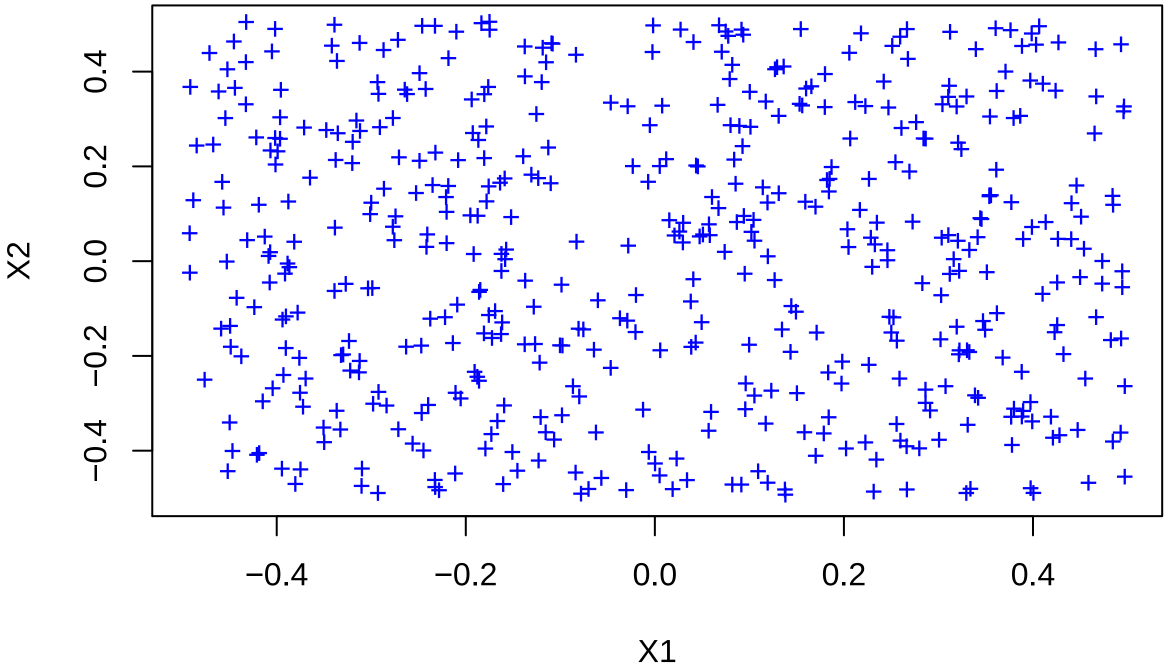
part f

```
lm_prob = predict(lm_fit, data, type = "response")
# probability threshold = 0.5
lm_pred = ifelse(lm_prob > 0.5, 1, 0)
data_pos = data[lm_pred == 1, ]
data_neg = data[lm_pred == 0, ]
plot(data_pos$x1, data_pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(data_neg$x1, data_neg$x2, col = "red", pch = 4)
```



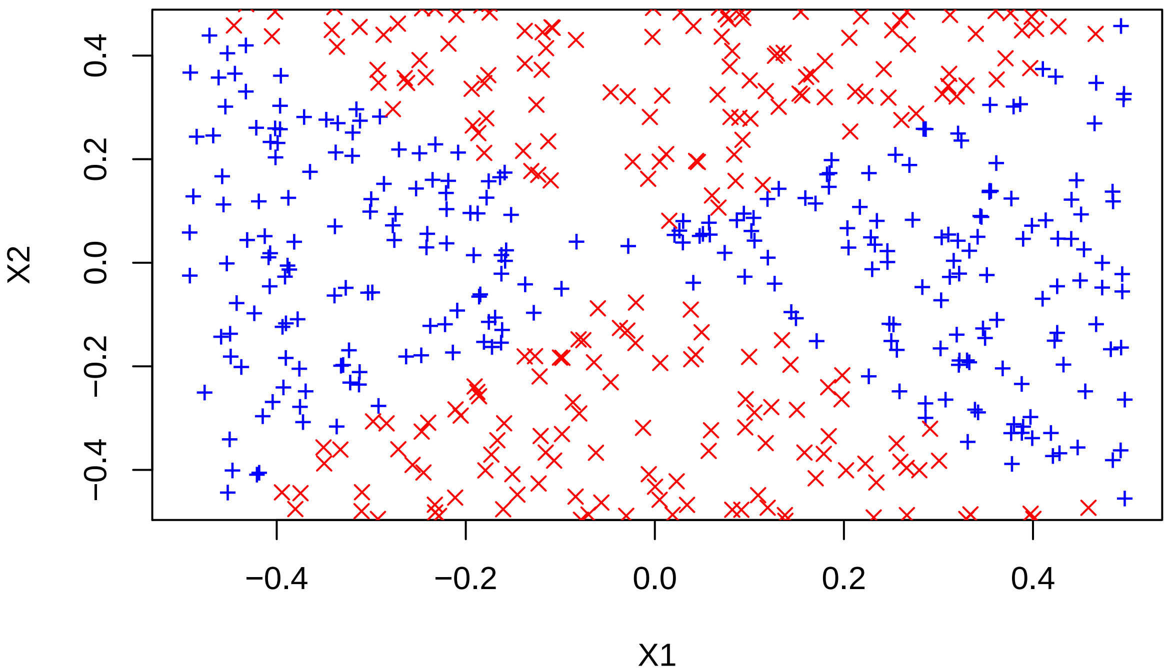
part g

```
svm_fit = svm(as.factor(y) ~ x1 + x2, data, kernel = "linear", cost = 0.1)
svm_pred = predict(svm_fit, data)
data_pos = data[svm_pred == 1, ]
data_neg = data[svm_pred == 0, ]
plot(data_pos$x1, data_pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(data_neg$x1, data_neg$x2, col = "red", pch = 4)
```



part h

```
svm_fit = svm(as.factor(y) ~ x1 + x2, data, gamma = 1)
svm_pred = predict(svm_fit, data)
data_pos = data[svm_pred == 1, ]
data_neg = data[svm_pred == 0, ]
plot(data_pos$x1, data_pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(data_neg$x1, data_neg$x2, col = "red", pch = 4)
```



part i

Logistic regression with non-interactions and SVMs with linear kernels fail to find the decision boundary. Logistic regression with interactions find a decision boundary that is very close to the real one. But finding the correct interactions might become a challenge. However, using radial basis kernels, and only change the parameter gamma, saves lots of effort and simple CV could accomplish that. Thus, SVMs with non-linear kernel is very useful to find a non-linear decision boundary.

9.7.7

part a

```
gas.med = median(Auto$mpg)
new.var = ifelse(Auto$mpg > gas.med, 1, 0)
Auto$mpglevel = as.factor(new.var)
```

part b

linear kernel

```r
set.seed(4)
tune_out = tune(svm, mpglevel ~ ., data = Auto, kernel = "linear", ranges = list(cost = c(0.01,
    0.1, 1, 5, 10, 100)))
summary(tune_out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.01275641
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-02 0.07378205 0.03437509
## 2 1e-01 0.05346154 0.03026685
## 3 1e+00 0.01275641 0.01344780
## 4 5e+00 0.02044872 0.01619554
## 5 1e+01 0.02038462 0.01074682
## 6 1e+02 0.03320513 0.01737168
```

When cost = 1, the test MSE is the smallest.

part c

polynomial kernal

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

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##    10      2
##
## - best performance: 0.5635897
##
## - Detailed performance results:
##    cost degree     error dispersion
## 1   0.1      2 0.5712179 0.04971286
## 2   1.0      2 0.5712179 0.04971286
## 3   5.0      2 0.5712179 0.04971286
## 4  10.0      2 0.5635897 0.05854295
## 5   0.1      3 0.5712179 0.04971286
```

```
## 6   1.0      3 0.5712179 0.04971286
## 7   5.0      3 0.5712179 0.04971286
## 8  10.0      3 0.5712179 0.04971286
## 9   0.1      4 0.5712179 0.04971286
## 10  1.0      4 0.5712179 0.04971286
## 11  5.0      4 0.5712179 0.04971286
## 12 10.0      4 0.5712179 0.04971286
```

When cost = 10, degree = 2, the error is the smallest.

radial kernel

```r
set.seed(6)
tune_out = tune(svm, mpglevel ~ ., data = Auto, kernel = "radial", ranges = list(cost = c(0.1,
    1, 5, 10), gamma = c(0.01, 0.1, 1, 5, 10, 100)))
summary(tune_out)
```

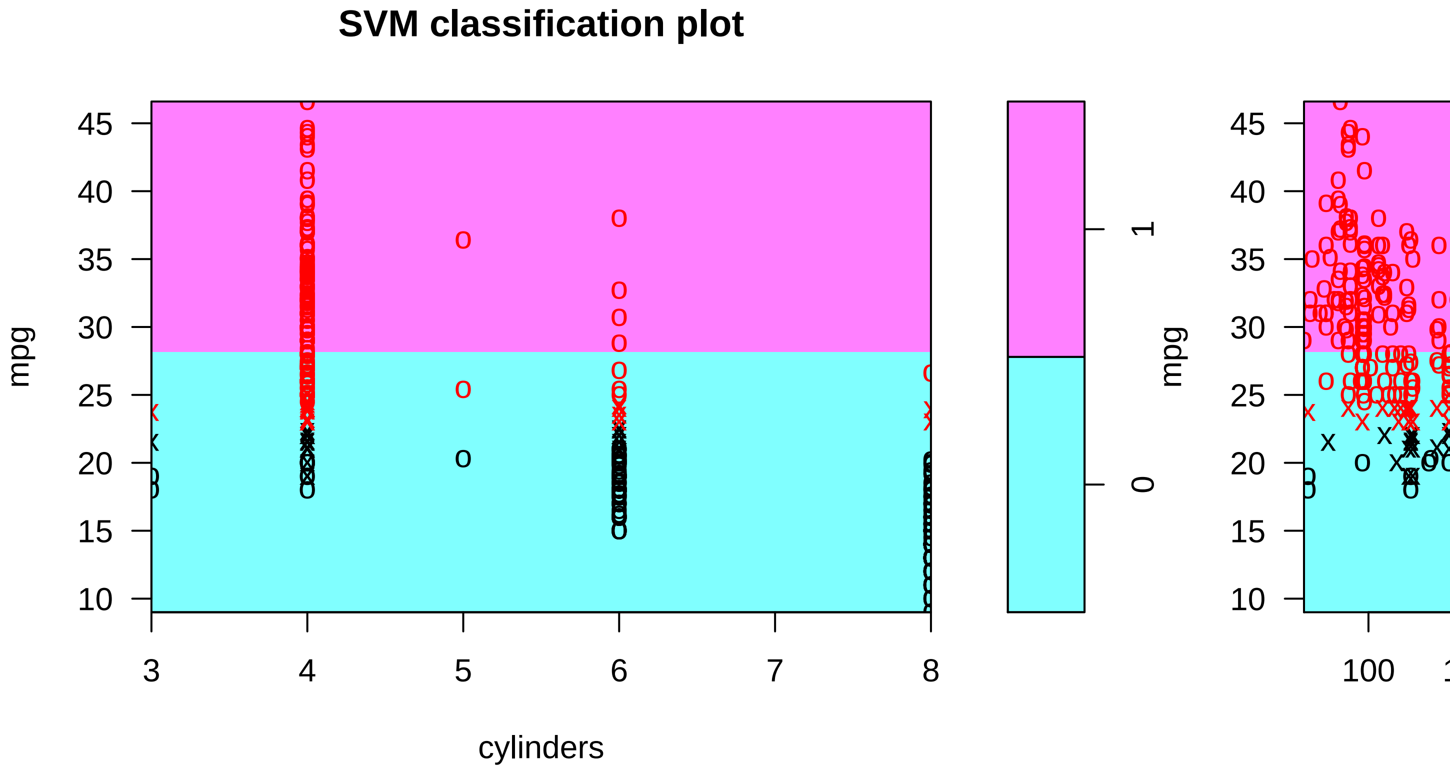
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.1
##
## - best performance: 0.02051282
##
## - Detailed performance results:
##    cost gamma      error dispersion
## 1   0.1 1e-02 0.09179487 0.04023204
## 2   1.0 1e-02 0.07141026 0.03778996
## 3   5.0 1e-02 0.04852564 0.03074258
## 4  10.0 1e-02 0.02301282 0.02549182
## 5   0.1 1e-01 0.07903846 0.03893119
## 6   1.0 1e-01 0.05371795 0.03716942
## 7   5.0 1e-01 0.02820513 0.03299190
## 8  10.0 1e-01 0.02051282 0.02911006
## 9   0.1 1e+00 0.52128205 0.14693828
## 10  1.0 1e+00 0.05878205 0.03646034
## 11  5.0 1e+00 0.05871795 0.03205442
## 12 10.0 1e+00 0.05871795 0.03205442
## 13  0.1 5e+00 0.55628205 0.05494470
## 14  1.0 5e+00 0.48987179 0.06983539
## 15  5.0 5e+00 0.48987179 0.06983539
## 16 10.0 5e+00 0.48987179 0.06983539
## 17  0.1 1e+01 0.55878205 0.05128210
## 18  1.0 1e+01 0.52557692 0.06532066
## 19  5.0 1e+01 0.50775641 0.06213538
## 20 10.0 1e+01 0.50775641 0.06213538
## 21  0.1 1e+02 0.55878205 0.05128210
## 22  1.0 1e+02 0.55878205 0.05128210
## 23  5.0 1e+02 0.55878205 0.05128210
```

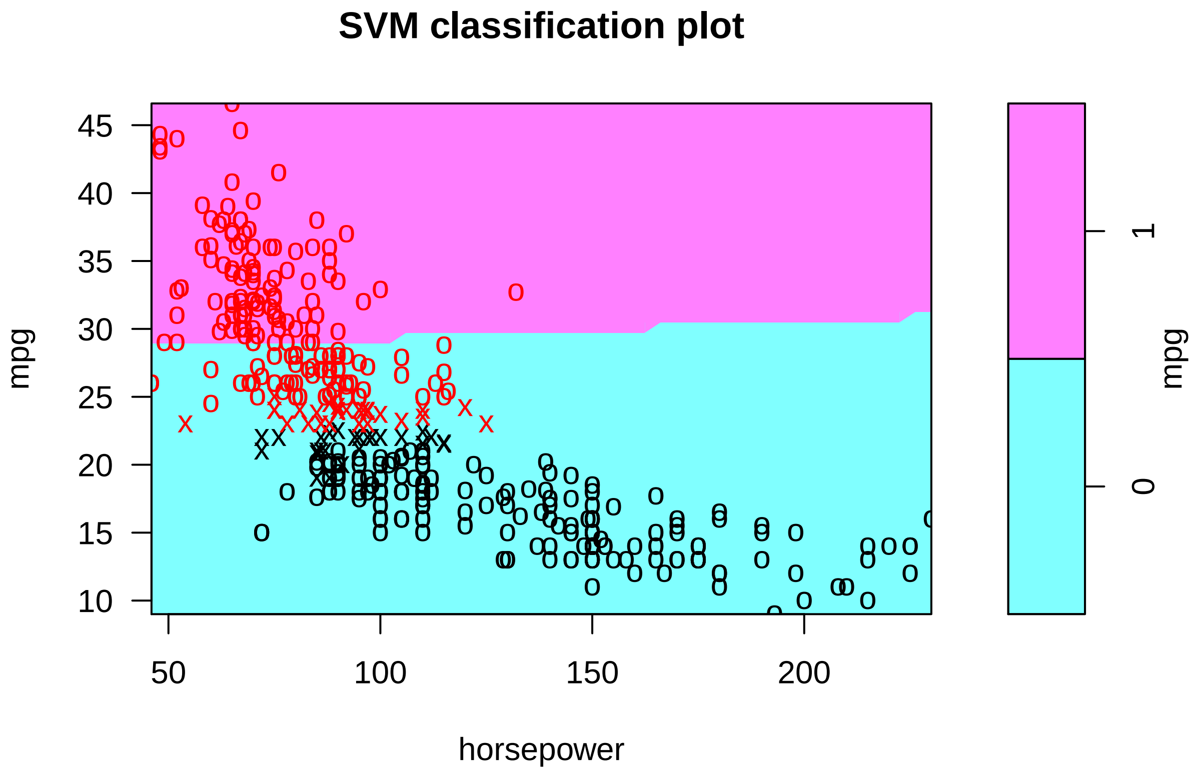
```
## 24 10.0 1e+02 0.55878205 0.05128210
```

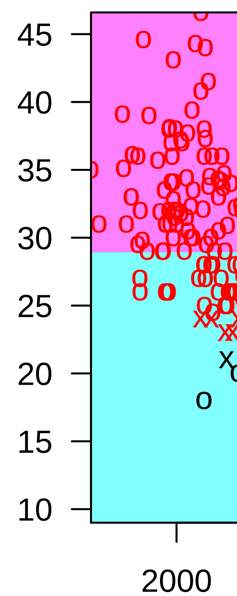
When cost = 10, gamma = 0.1, the error is the smallest.

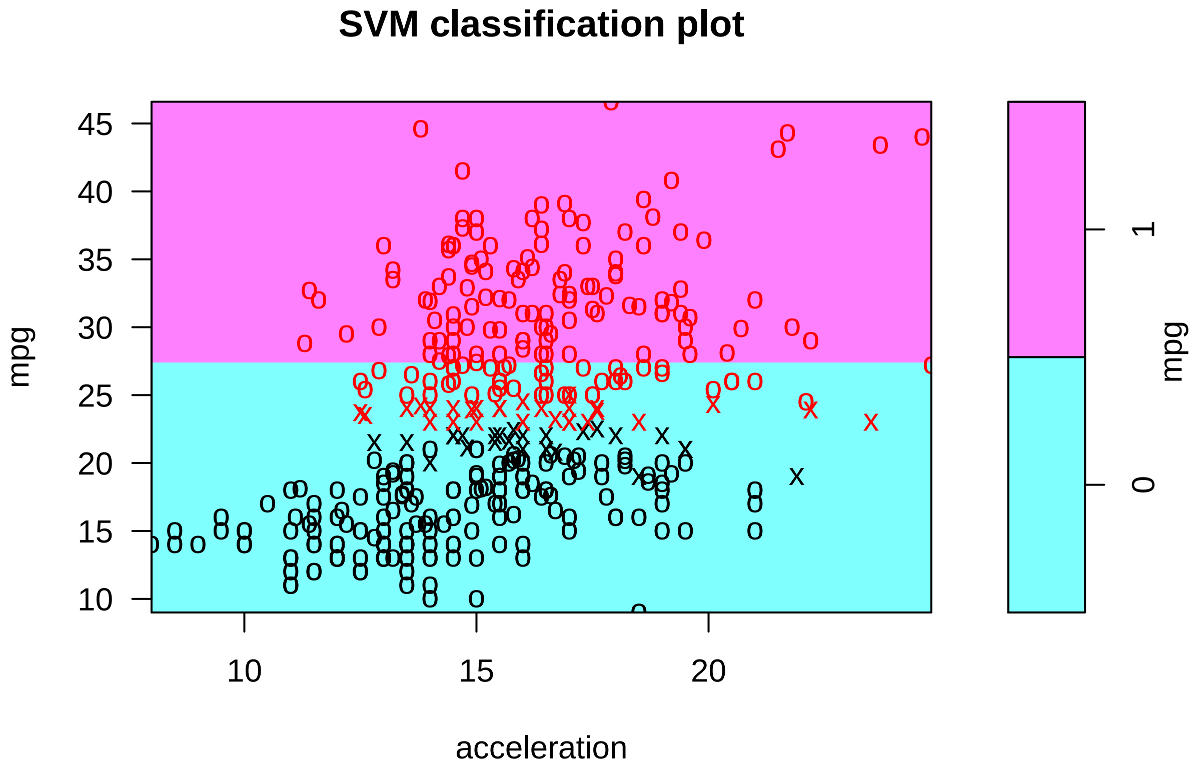
part d

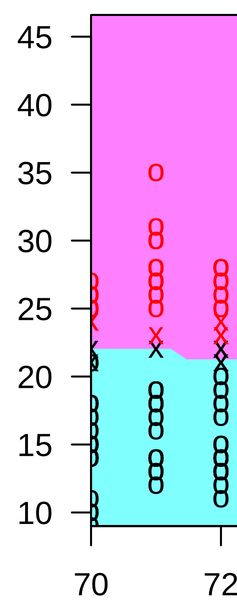
```r
svm_linear = svm(mpglevel ~ ., data = Auto, kernel = "linear", cost = 1)
svm_poly = svm(mpglevel ~ ., data = Auto, kernel = "polynomial", cost = 10,
    degree = 2)
svm_radial = svm(mpglevel ~ ., data = Auto, kernel = "radial", cost = 10, gamma = 0.01)
plotpairs = function(fit) {
    for (name in names(Auto)[!(names(Auto) %in% c("mpg", "mpglevel", "name"))]) {
        plot(fit, Auto, as.formula(paste("mpg~", name, sep = "")))
    }
}
plotpairs(svm_linear)
```

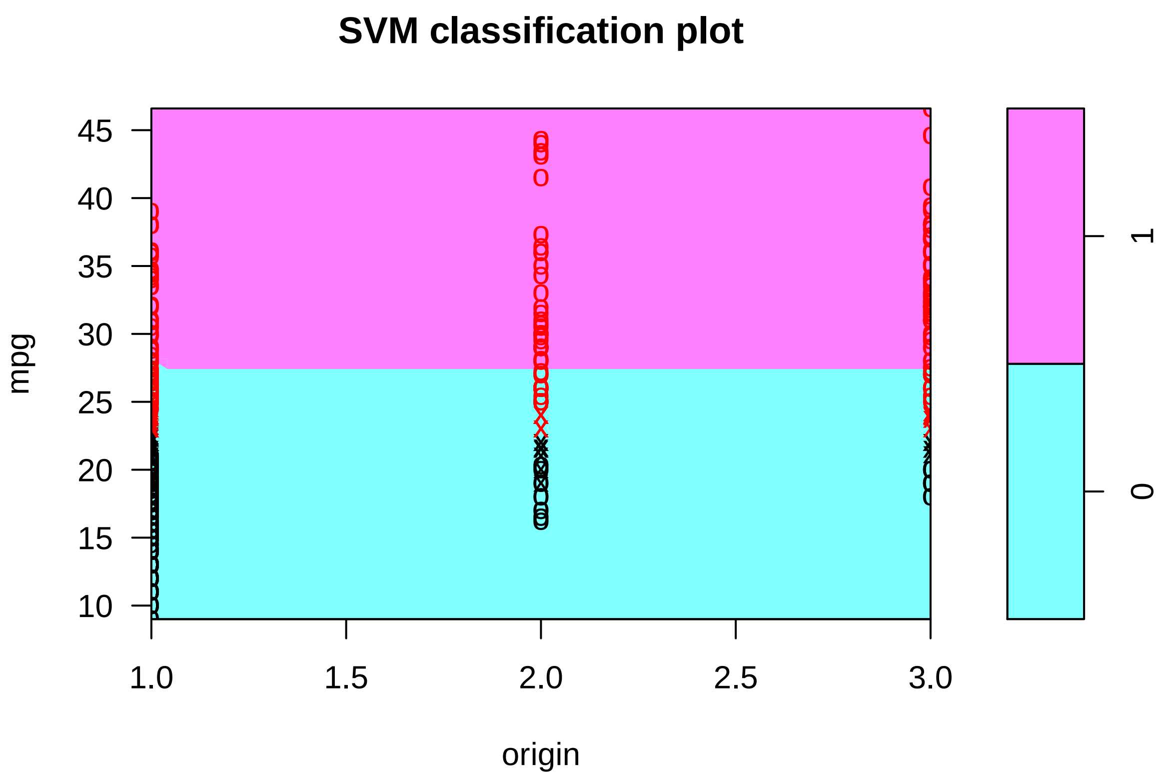



SVM classification plot
mpg
horsepower
45
40
35
30
25
20
15
10
50
100
150
200
1
0

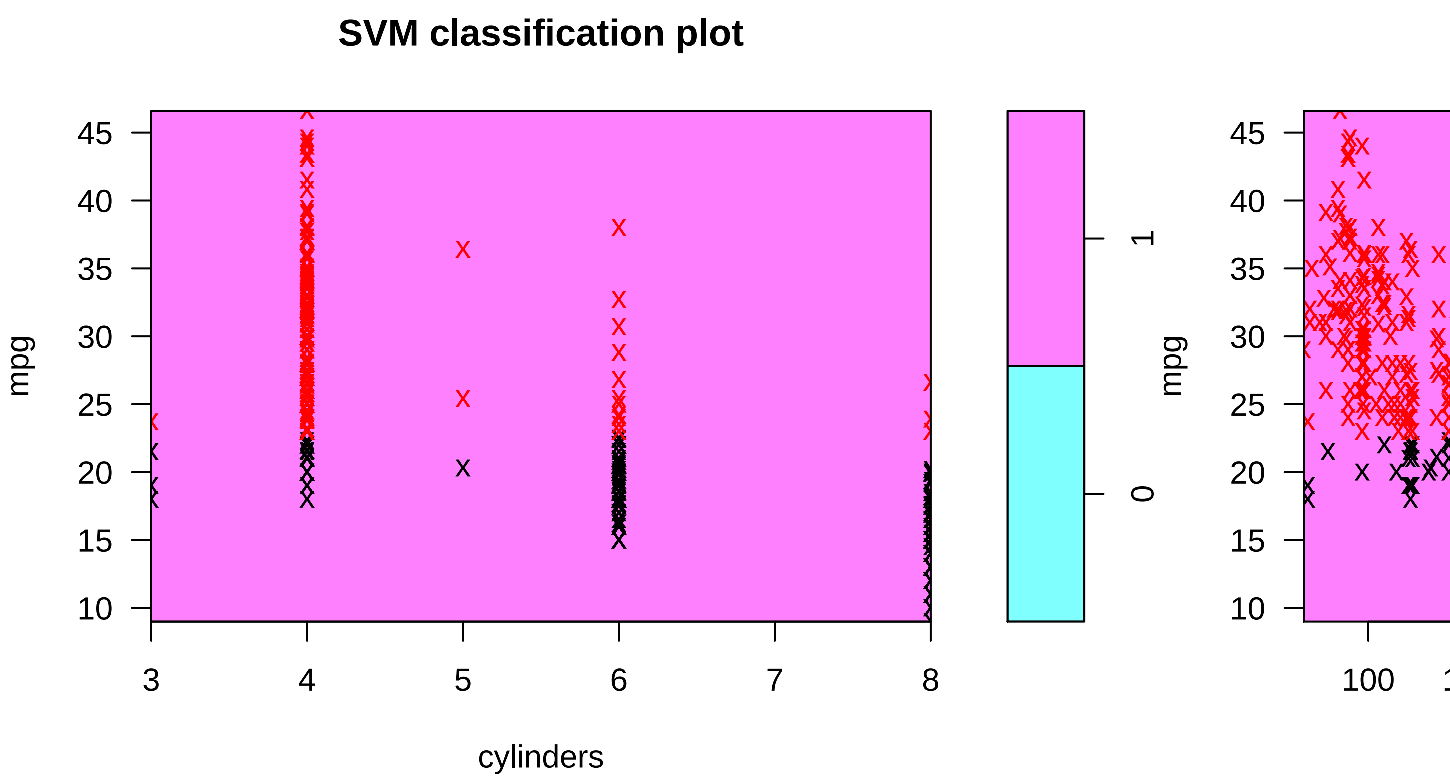

45
40
35
30
25
20
15
10
2000

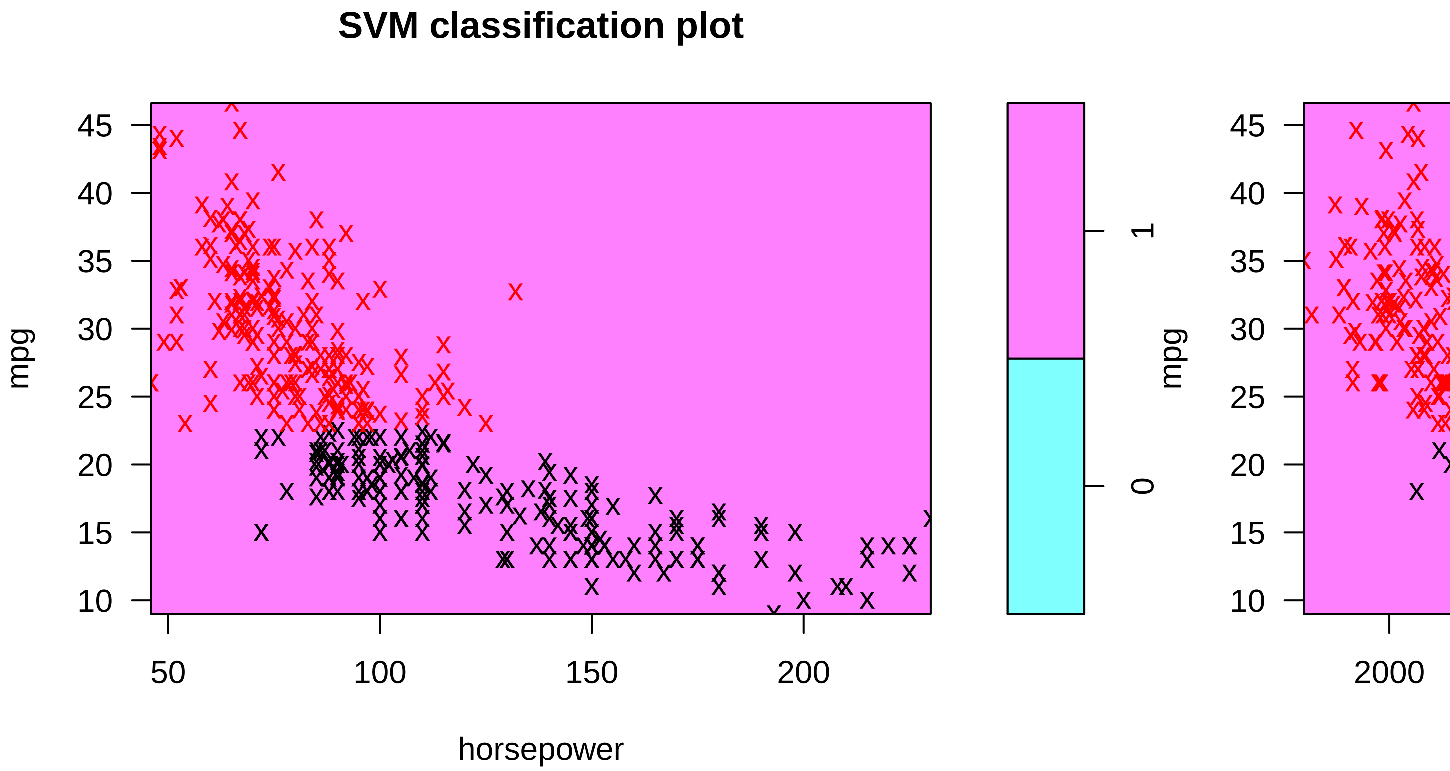

SVM classification plot
mpg
acceleration
45
40
35
30
25
20
15
10
10
15
20
1
0

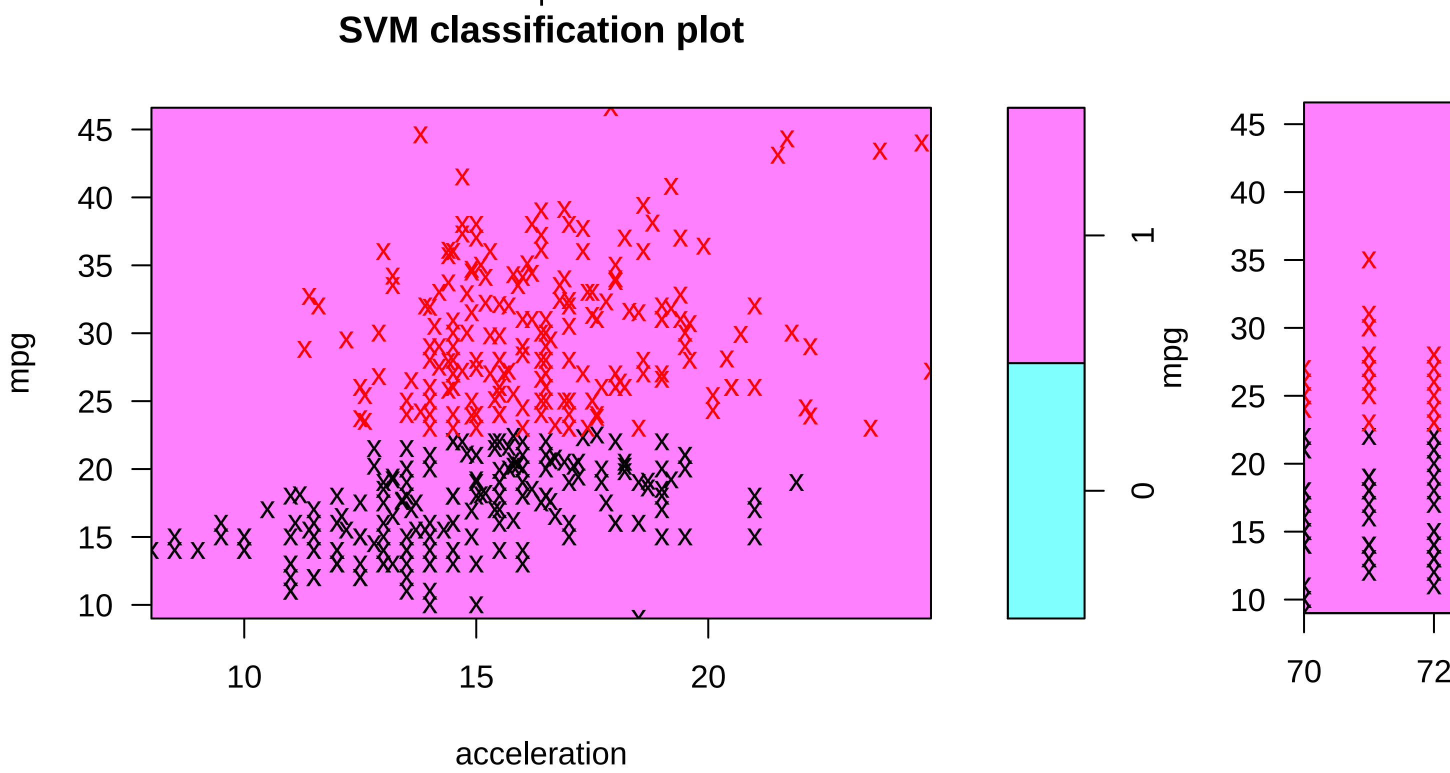

45
40
35
30
25
20
15
10
70
72

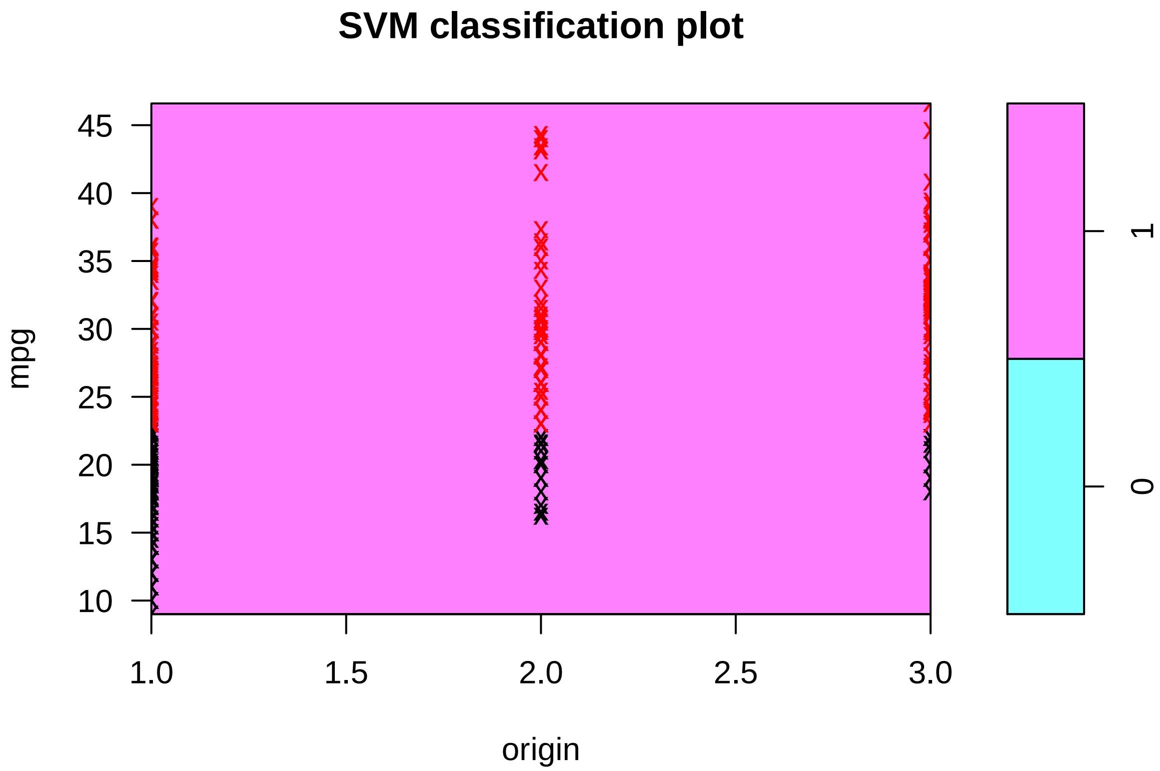



```
plotpairs(svm_poly)
```

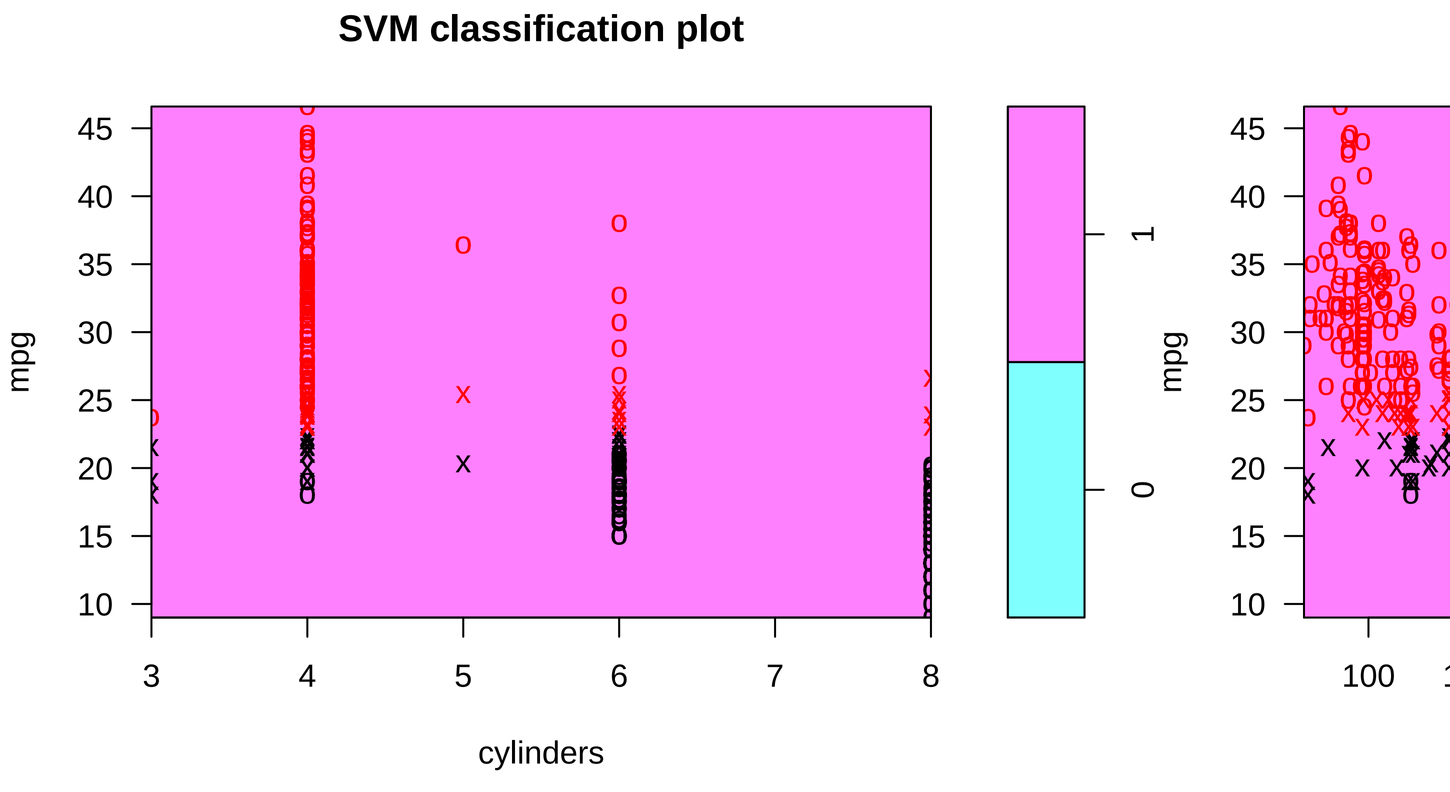


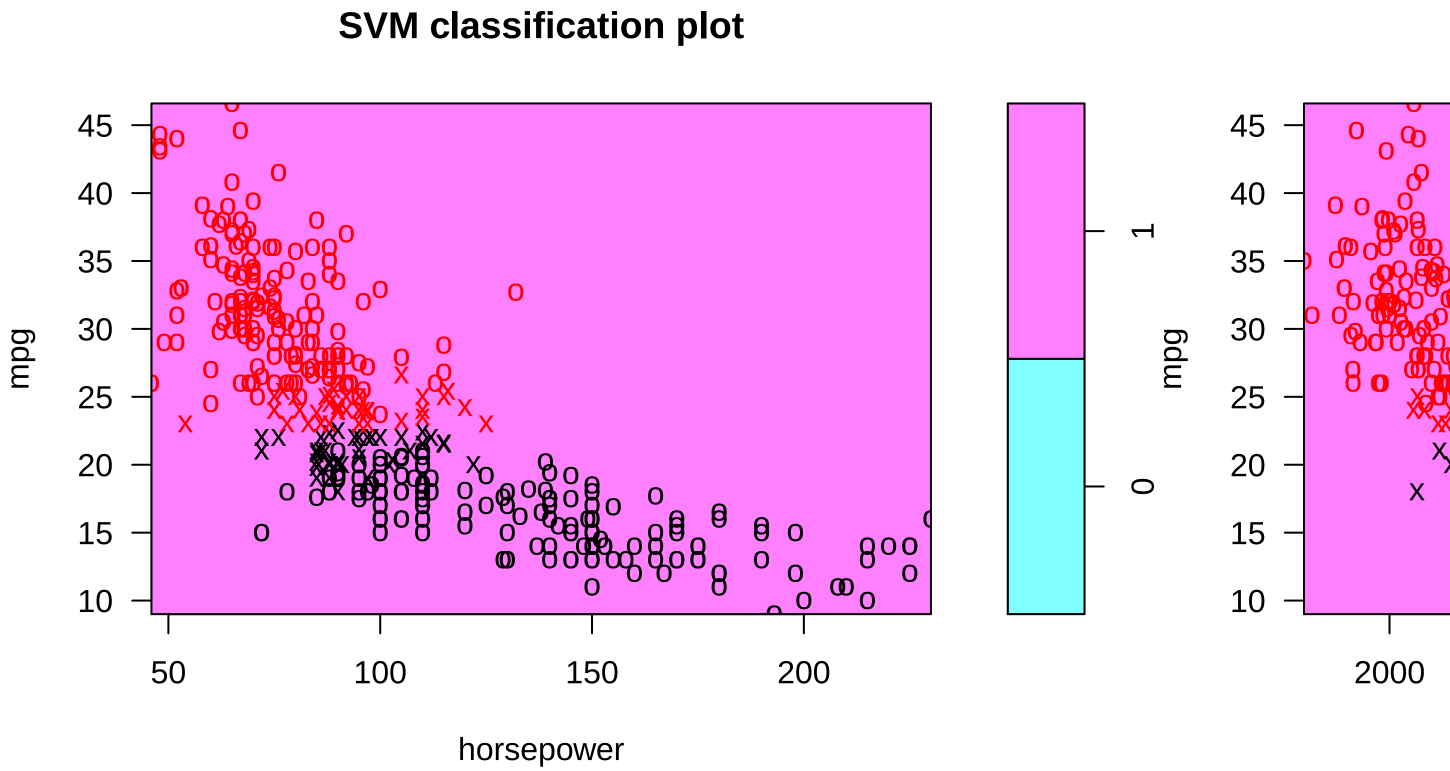

SVM classification plot
mpg
45
40
35
30
25
20
15
10
50
100
150
200
horsepower
1
0
mpg
2000

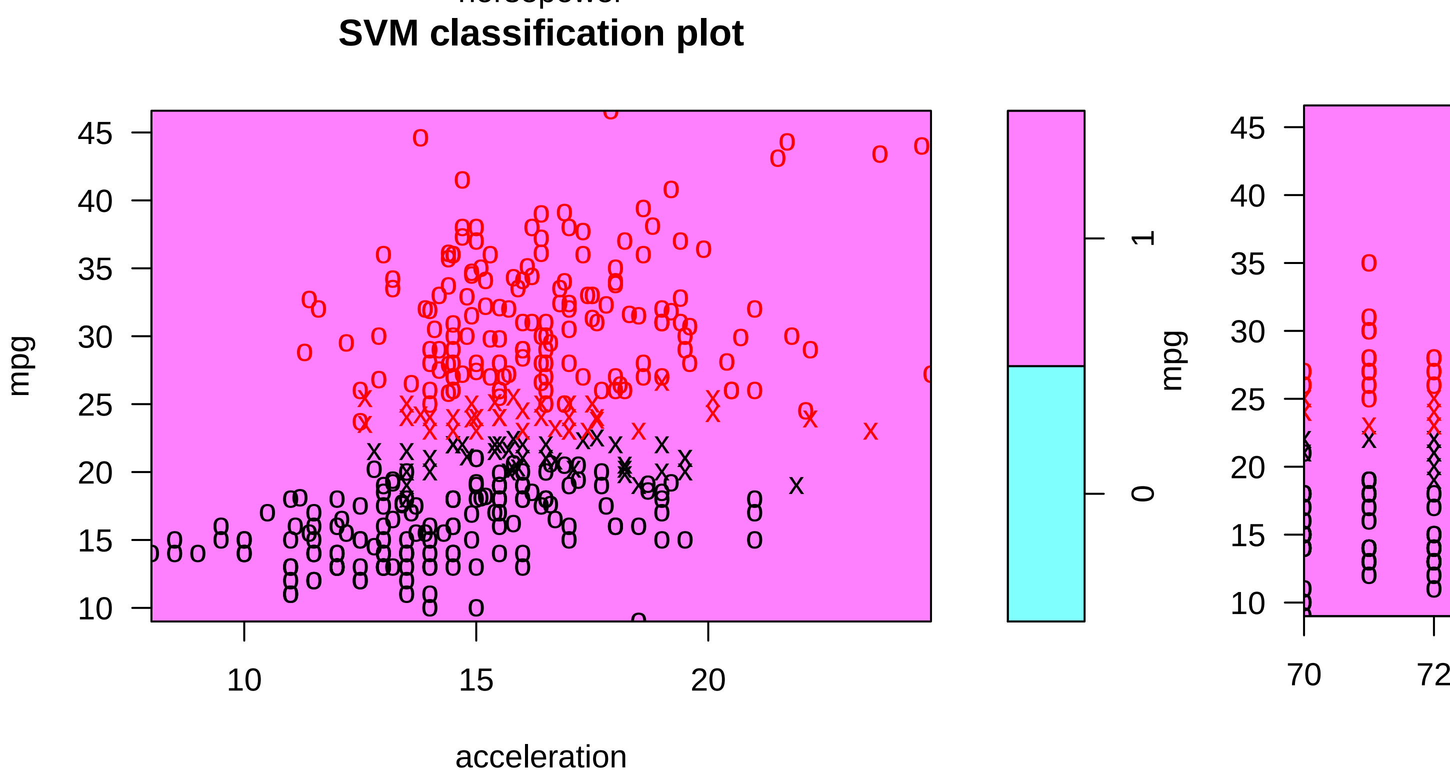

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0
mpg
70
72

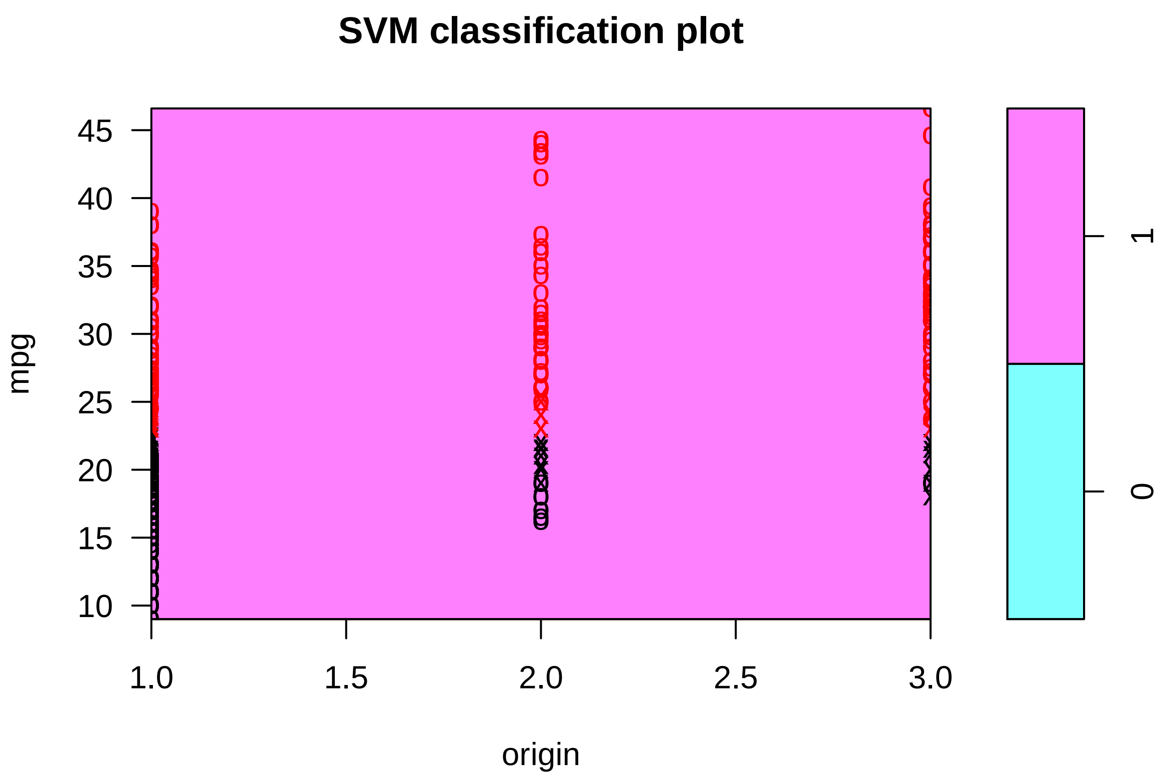



```r
plotpairs(svm_radial)
```




SVM classification plot
mpg
horsepower
45
40
35
30
25
20
15
10
50
100
150
200
2000
1
0


SVM classification plot
mpg
acceleration
45
40
35
30
25
20
15
10
10
15
20
70
72
1
0



9.7.8

part a

```r
set.seed(7)
train = sample(dim(OJ)[1], 800)
OJ_train = OJ[train, ]
OJ_test = OJ[-train, ]
```

part b

```r
svm_linear = svm(Purchase ~ ., kernel = "linear", data = OJ_train, cost = 0.01)
summary(svm_linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ_train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
```

```
## Number of Support Vectors:  443
##
##  ( 223 220 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Support vector classifier creates 443 support vectors from 800 training points. In the rest of points, 223 belong to level CH and 220 belong to level MM.

part c

```r
train_pred = predict(svm_linear, OJ_train)
table(OJ_train$Purchase, train_pred)
```

```
##     train_pred
##       CH  MM
##   CH 426  59
##   MM  83 232
```

```r
(59+83)/(426+232+59+83)
```

```
## [1] 0.1775
```

The training data error rate is 17.75%

```r
test_pred = predict(svm_linear, OJ_test)
table(OJ_test$Purchase, test_pred)
```

```
##     test_pred
##       CH  MM
##   CH 151  17
##   MM  26  76
```

```r
(17+26)/(151+17+26+76)
```

```
## [1] 0.1592593
```

The test data error rate is 16%

part d

```r
set.seed(8)
tune_out = tune(svm, Purchase ~ ., data = OJ_train, kernel = "linear", ranges = list(cost = 10^seq(-2,1
summary(tune_out)
```

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

```
##  3.162278
##
## - best performance: 0.17375
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.19250 0.03782269
## 2   0.01778279 0.18875 0.03508422
## 3   0.03162278 0.18750 0.03004626
## 4   0.05623413 0.18000 0.03238227
## 5   0.10000000 0.17750 0.03322900
## 6   0.17782794 0.17875 0.03120831
## 7   0.31622777 0.17750 0.03525699
## 8   0.56234133 0.17875 0.03634805
## 9   1.00000000 0.17500 0.03679900
## 10  1.77827941 0.17750 0.04073969
## 11  3.16227766 0.17375 0.04466309
## 12  5.62341325 0.17625 0.04101575
## 13 10.00000000 0.17625 0.03928617
```

The optimal cost is 3.1623

part e

```r
svm_linear = svm(Purchase ~ ., kernel = "linear", data = OJ_train, cost = tune_out$best.parameters$cost)
train_pred = predict(svm_linear, OJ_train)
table(OJ_train$Purchase, train_pred)
```

```
##     train_pred
##       CH  MM
##   CH 425  60
##   MM  71 244
```

```r
(60+71)/(60+71+425+244)
```

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

The training error rate after tuning is 16.375%

```r
test_pred = predict(svm_linear, OJ_test)
table(OJ_test$Purchase, test_pred)
```

```
##     test_pred
##       CH  MM
##   CH 152  16
##   MM  26  76
```

```r
(16+26)/(16+26+152+76)
```

```
## [1] 0.1555556
```

The test error rate after tuing is 15.56%

part f

```
set.seed(9)
svm_radial = svm(Purchase ~ ., data = OJ_train, kernel = "radial")
summary(svm_radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ_train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.05555556
##
## Number of Support Vectors:  368
##
##  ( 183 185 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Support vector classifier with radial kernel creates 368 support vectors from 800 training points. In the rest of points, 183 belong to level CH and 185 belong to level MM.

```
train_pred = predict(svm_radial, OJ_train)
table(OJ_train$Purchase, train_pred)
```

```
##     train_pred
##       CH  MM
##   CH 438  47
##   MM  82 233
```

```
(47+82)/(438+233+47+82)
```

```
## [1] 0.16125
```

The training error rate is 16.125%

```
test_pred = predict(svm_radial, OJ_test)
table(OJ_test$Purchase, test_pred)
```

```
##     test_pred
##       CH  MM
##   CH 153  15
##   MM  30  72
```

```
(15+30)/(153+72+15+30)
```

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

The test error rate is 16.67%

```r
set.seed(10)
tune_out = tune(svm, Purchase ~ ., data = OJ_train, kernel = "radial", ranges = list(cost = 10^seq(-2, 
#summary(tune_out)
svm_radial = svm(Purchase ~ ., data = OJ_train, kernel = "radial", cost = tune_out$best.parameters$cost
train_pred = predict(svm_radial, OJ_train)
table(OJ_train$Purchase, train_pred)
```

```
##     train_pred
##       CH  MM
##   CH 444  41
##   MM  74 241
```

```r
(41+74)/(41+74+444+241)
```

```
## [1] 0.14375
```

The training error rate after tuning is 14.375%

```r
test_pred = predict(svm_radial, OJ_test)
table(OJ_test$Purchase, test_pred)
```

```
##     test_pred
##       CH  MM
##   CH 152  16
##   MM  30  72
```

```r
(16+30)/(152+72+16+30)
```

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

The test rate after tuning is 17%

part g

```r
set.seed(11)
svm_poly = svm(Purchase ~ ., data = OJ_train, kernel = "poly", degree = 2)
summary(svm_poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ_train, kernel = "poly",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  441
##
##  ( 223 218 )
##
```

```
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Support vector classifier with radial kernel creates 441 support vectors from 800 training points. In the rest of points, 183 belong to level CH and 223 belong to level 218

```r
train_pred = predict(svm_poly, OJ_train)
table(OJ_train$Purchase, train_pred)
```

```
##     train_pred
##       CH  MM
##   CH 449  36
##   MM 111 204
```

```r
(36+111)/(36+111+449+204)
```

```
## [1] 0.18375
```

The training error rate is 18.375%

```r
test_pred = predict(svm_poly, OJ_test)
table(OJ_test$Purchase, test_pred)
```

```
##     test_pred
##       CH  MM
##   CH 158  10
##   MM  41  61
```

```r
(10+41)/(10+41+158+61)
```

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

The test error rate is 18.89%

```r
set.seed(12)
tune_out = tune(svm, Purchase ~ ., data = OJ_train, kernel = "poly", degree = 2, ranges = list(cost = 1(
#summary(tune_out)
svm_poly = svm(Purchase ~ ., data = OJ_train, kernel = "poly", cost = tune_out$best.parameters$cost)
train_pred = predict(svm_poly, OJ_train)
table(OJ_train$Purchase, train_pred)
```

```
##     train_pred
##       CH  MM
##   CH 446  39
##   MM  80 235
```

```r
(39+80)/(39+80+446+235)
```

```
## [1] 0.14875
```

The training error rate after tuning is 14.875%

```r
test_pred = predict(svm_poly, OJ_test)
table(OJ_test$Purchase, test_pred)
```

```
##     test_pred
##       CH  MM
##   CH 156  12
```

```
##    MM  32  70
```

```r
(12+32)/(12+32+156+70)
```

```
## [1] 0.162963
```

The test rate after tuning is 16.3%

part h

With linear kernel, the training error rate after tuning is 16.375% and the test error rate after tuing is 15.56%

With radial kernel, the training error rate after tuning is 14.375% and the test error rate after tuing is 17%

With polynomial kernel, the training error rate after tuning is 14.875% and the test error rate after tuing is 16.3%

Suprisingly, on this data, linear kernel has smaller test error rate.