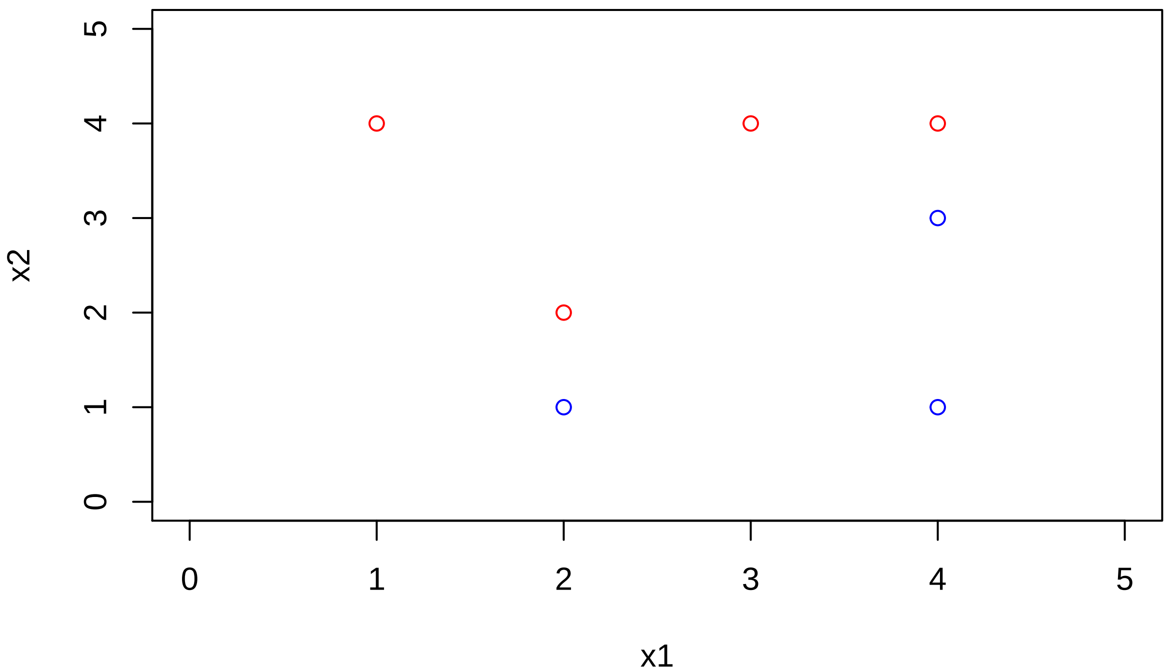
07 SVM Homework

Xinci Chen

3/5/2020

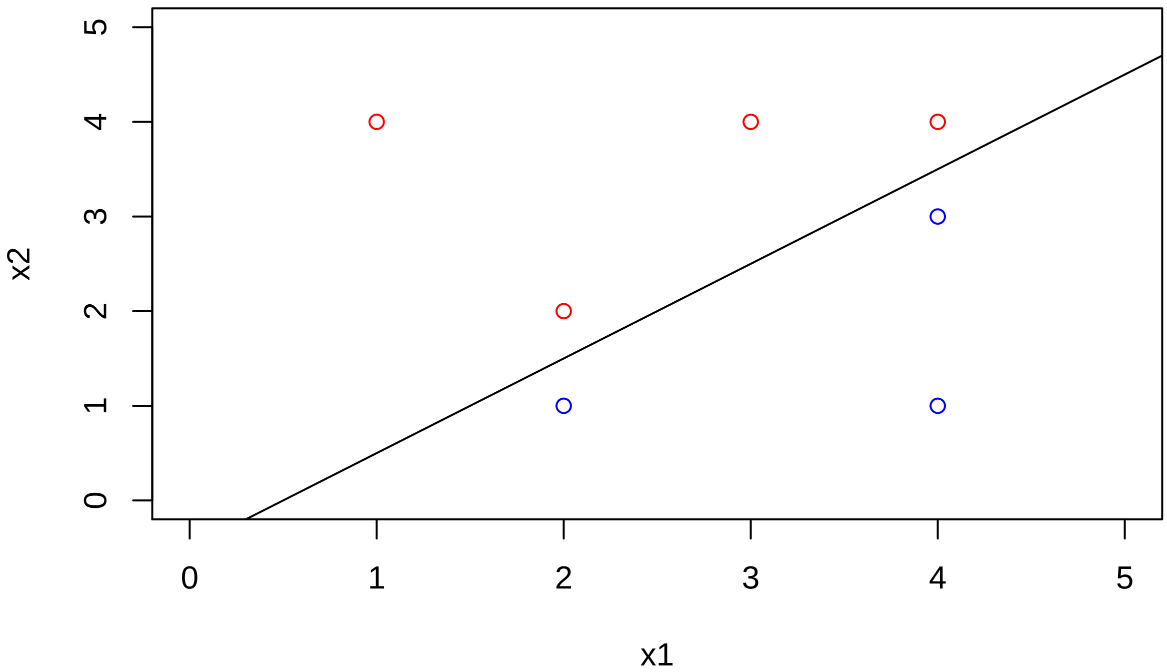
3 (a)

```
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))
```



(b)

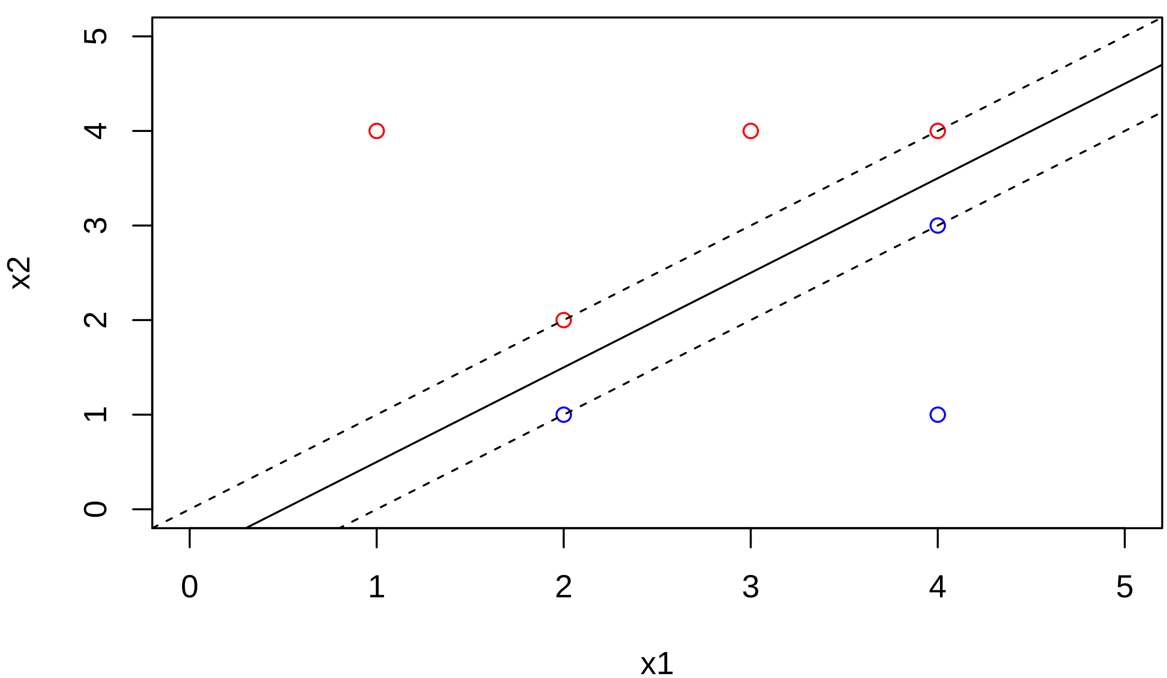
```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
```



(c)

(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)
```

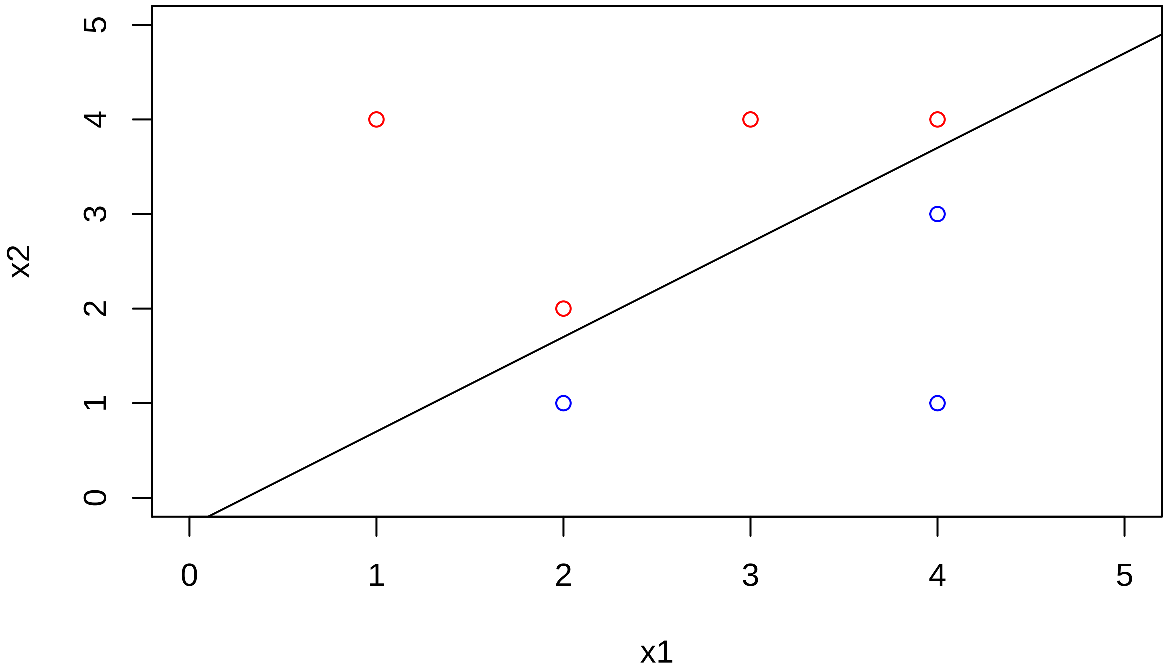


(e)

(f)

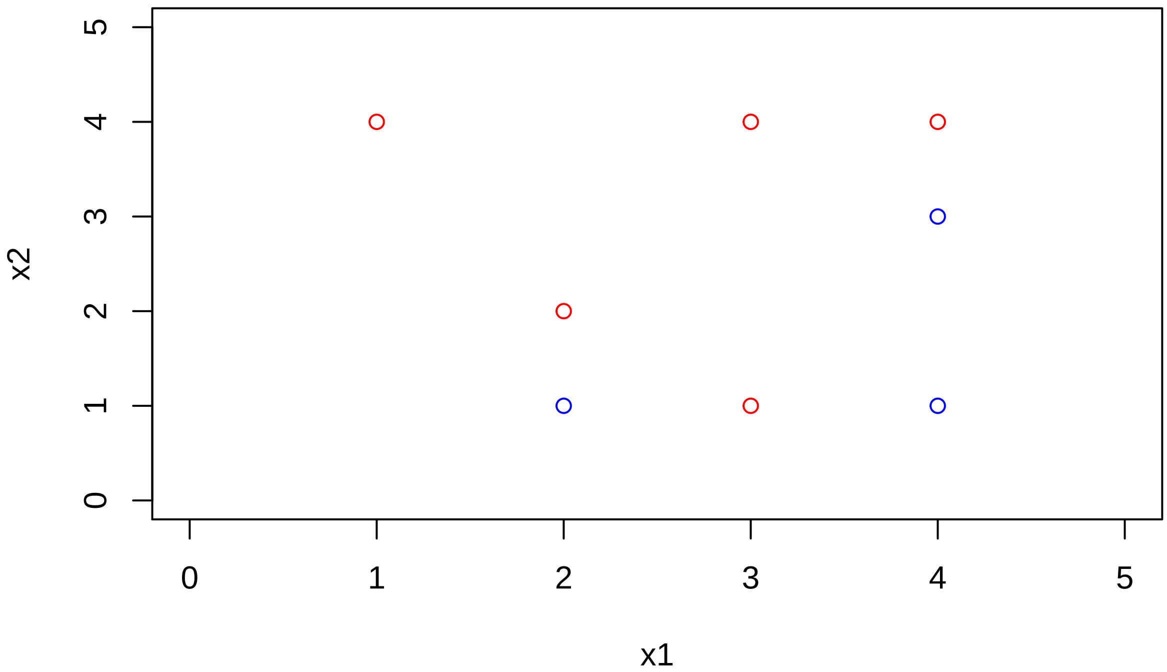
(g)

```r
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.3, 1)
```



(h)

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

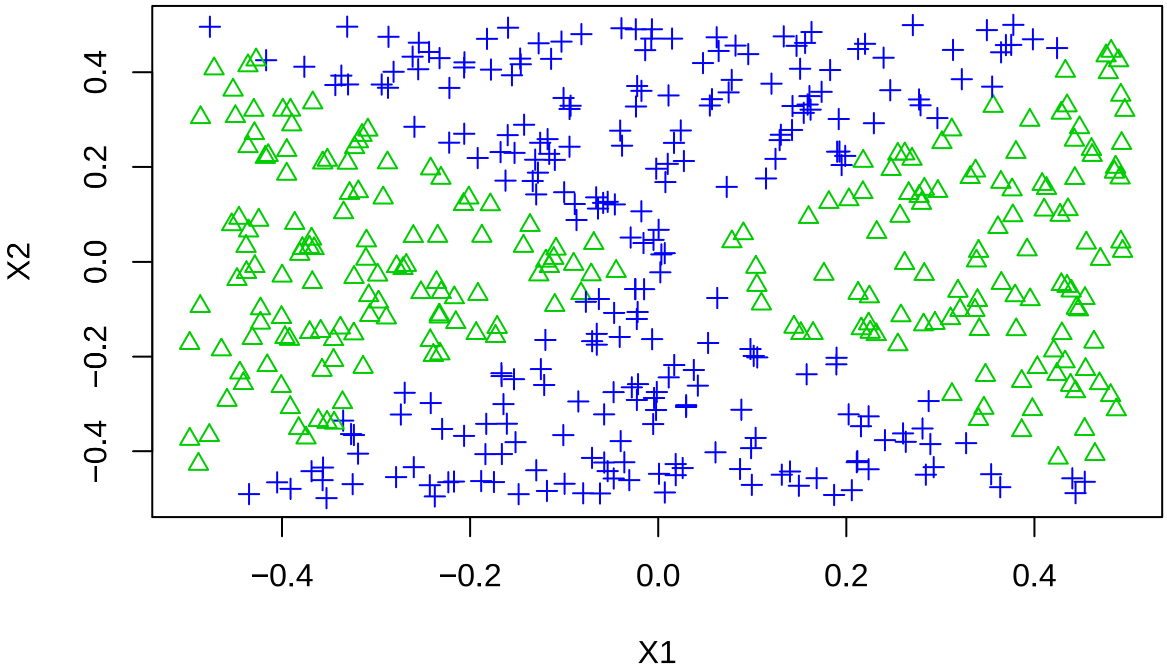


5(a)

```r
set.seed(1)
x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- as.integer(x1 ^ 2 - x2 ^ 2 > 0)
```

(b)

```r
plot(x1, x2, xlab = "X1", ylab = "X2", col = (4 - y), pch = (3 - y))
```



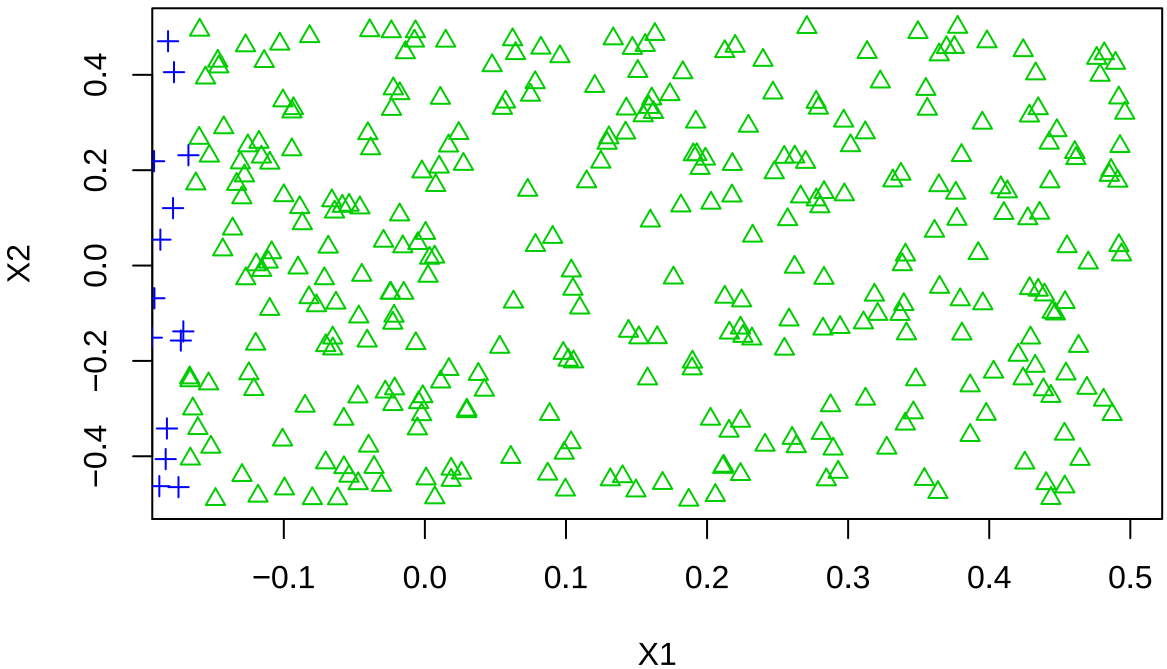
(c)

```r
logit.fit <- glm(y ~ x1 + x2, family = "binomial")
summary(logit.fit)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = "binomial")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.179  -1.139  -1.112   1.206   1.257
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.087260   0.089579  -0.974    0.330
## x1           0.196199   0.316864   0.619    0.536
## x2          -0.002854   0.305712  -0.009    0.993
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.18  on 499  degrees of freedom
## Residual deviance: 691.79  on 497  degrees of freedom
## AIC: 697.79
##
## Number of Fisher Scoring iterations: 3
```

(d)

```r
data <- data.frame(x1 = x1, x2 = x2, y = y)
probs <- predict(logit.fit, data, type = "response")
preds <- rep(0, 500)
preds[probs > 0.47] <- 1
plot(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1), xlab = "X1", ylab = "X
points(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0))
```



(e)

```
logitnl.fit <- glm(y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family = "binomial")
```

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

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

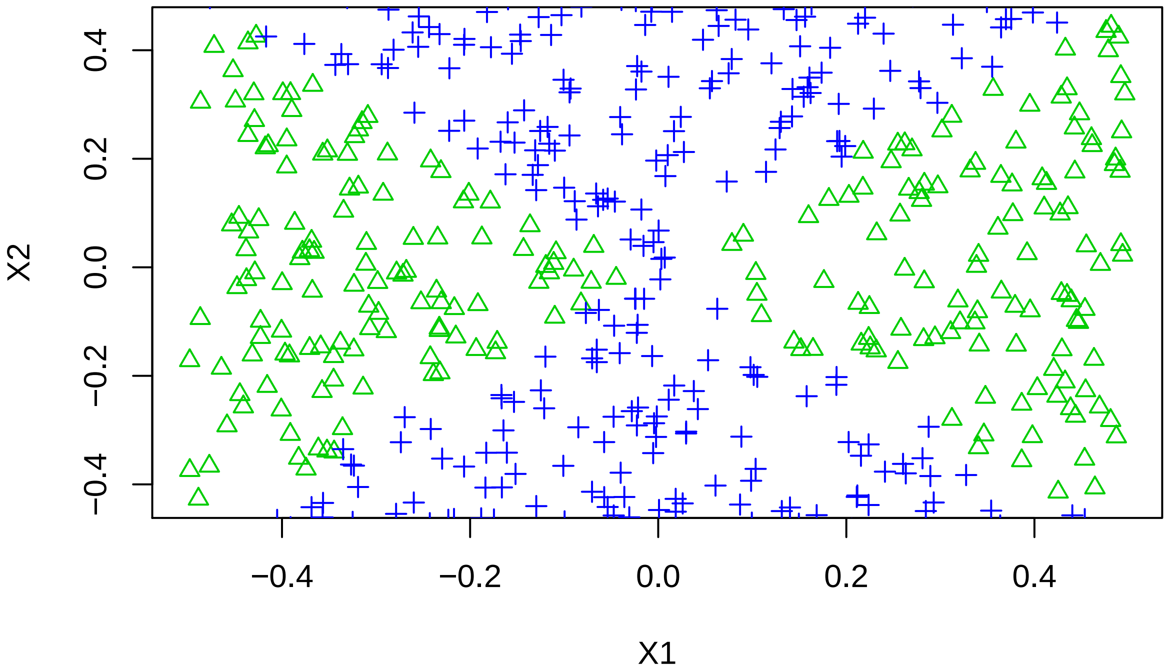
```
summary(logitnl.fit)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family = "binomial")
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -8.240e-04  -2.000e-08  -2.000e-08   2.000e-08   1.163e-03
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -102.2     4302.0  -0.024    0.981
## poly(x1, 2)1    2715.3   141109.5   0.019    0.985
## poly(x1, 2)2   27218.5   842987.2   0.032    0.974
## poly(x2, 2)1    -279.7    97160.4  -0.003    0.998
## poly(x2, 2)2 -28693.0   875451.3  -0.033    0.974
## I(x1 * x2)      -206.4    41802.8  -0.005    0.996
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9218e+02  on 499  degrees of freedom
## Residual deviance: 3.5810e-06  on 494  degrees of freedom
## AIC: 12
##
```

```
## Number of Fisher Scoring iterations: 25
```

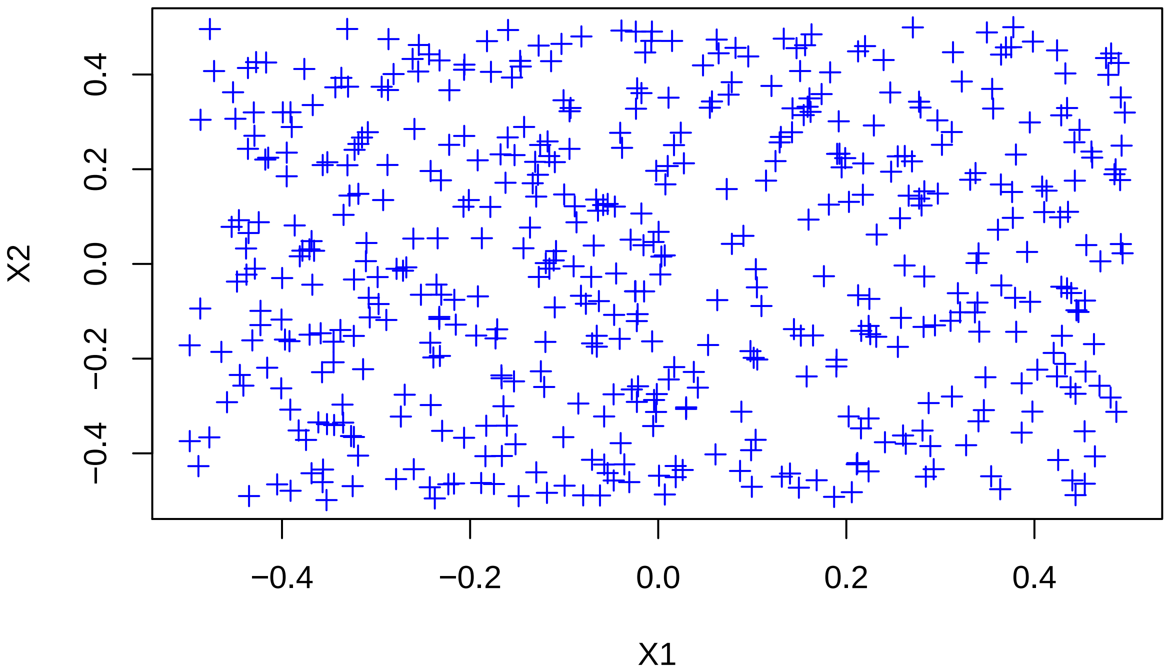
(f)

```r
probs <- predict(logitnl.fit, data, type = "response")
preds <- rep(0, 500)
preds[probs > 0.47] <- 1
plot(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1), xlab = "X1", ylab = "X2
points(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0))
```



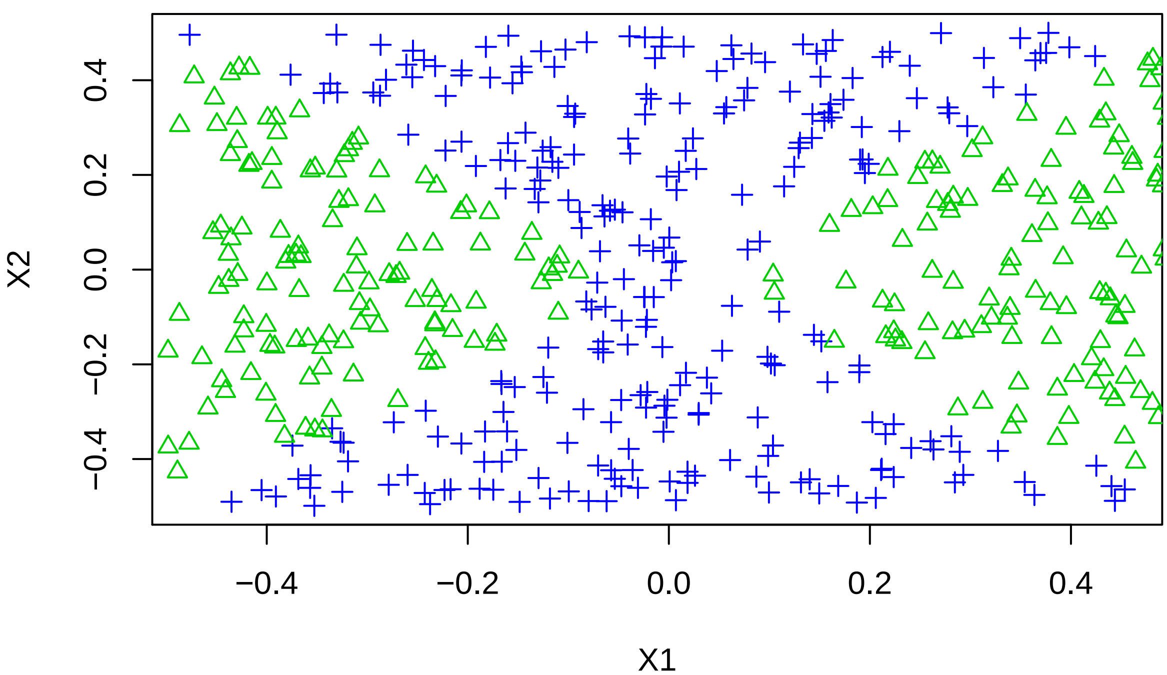
(g)

```r
data$y <- as.factor(data$y)
svm.fit <- svm(y ~ x1 + x2, data, kernel = "linear", cost = 0.01)
preds <- predict(svm.fit, data)
plot(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0), xlab = "X1", ylab = "X2
points(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1))
```



(h)

```
data$y <- as.factor(data$y)
svmnl.fit <- svm(y ~ x1 + x2, data, kernel = "radial", gamma = 1)
preds <- predict(svmnl.fit, data)
plot(data[preds == 0, ]$x1, data[preds == 0, ]$x2, col = (4 - 0), pch = (3 - 0), xlab = "X1", ylab = "X
points(data[preds == 1, ]$x1, data[preds == 1, ]$x2, col = (4 - 1), pch = (3 - 1))
```



(i)

7 (a)

```
var <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
Auto$mpglevel <- as.factor(var)
```

(b)

```
set.seed(1)
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "linear", ranges = list(cost = c(0.01, 0.1, 1
summary(tune.out)
```

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

(c)

```
set.seed(1)
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "polynomial", ranges = list(cost = c(0.01, 0.
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      2
##
## - best performance: 0.3013462
##
## - Detailed performance results:
##     cost degree     error dispersion
## 1  1e-02      2 0.5511538 0.04366593
```

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

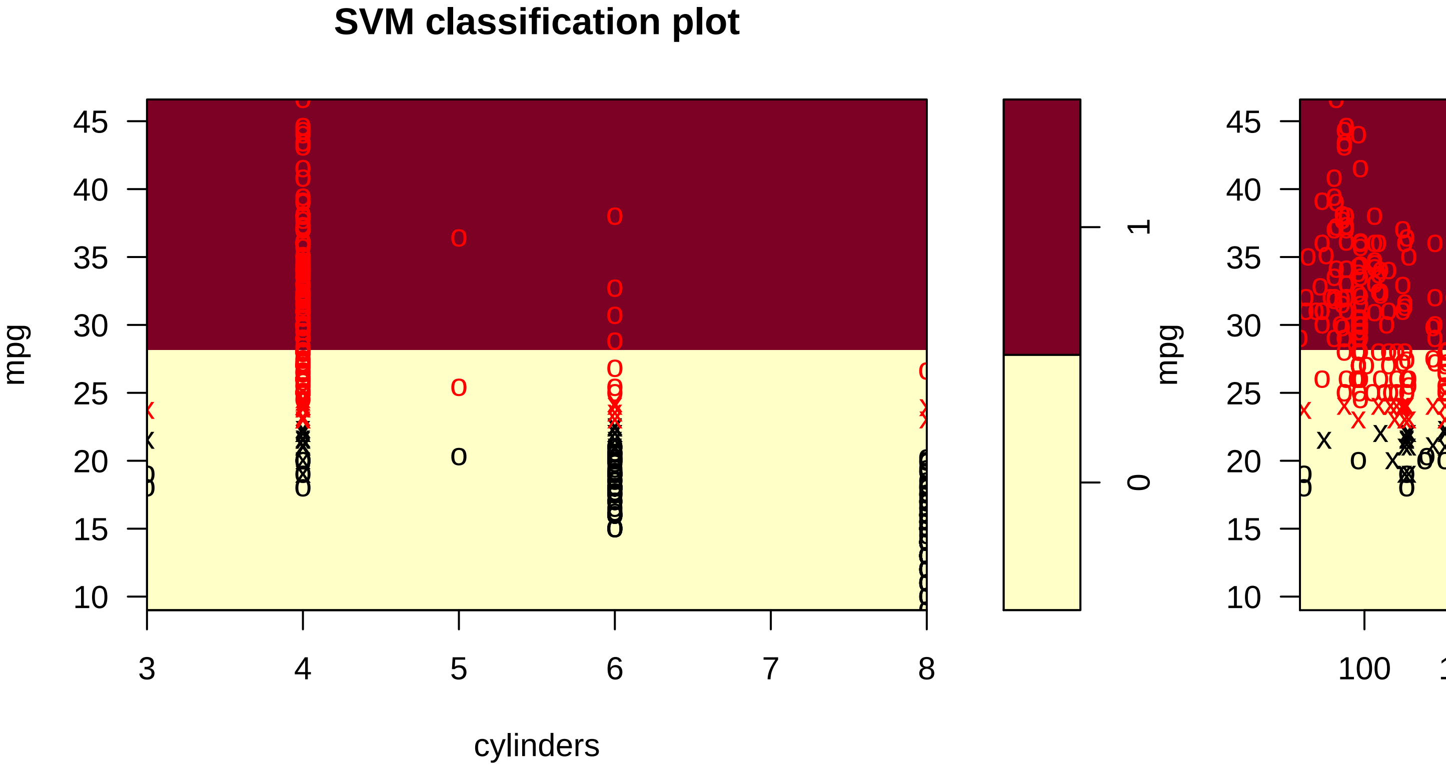
```r
set.seed(1)
tune.out <- tune(svm, mpglevel ~ ., data = Auto, kernel = "radial", ranges = list(cost = c(0.01, 0.1, 1
summary(tune.out)
```

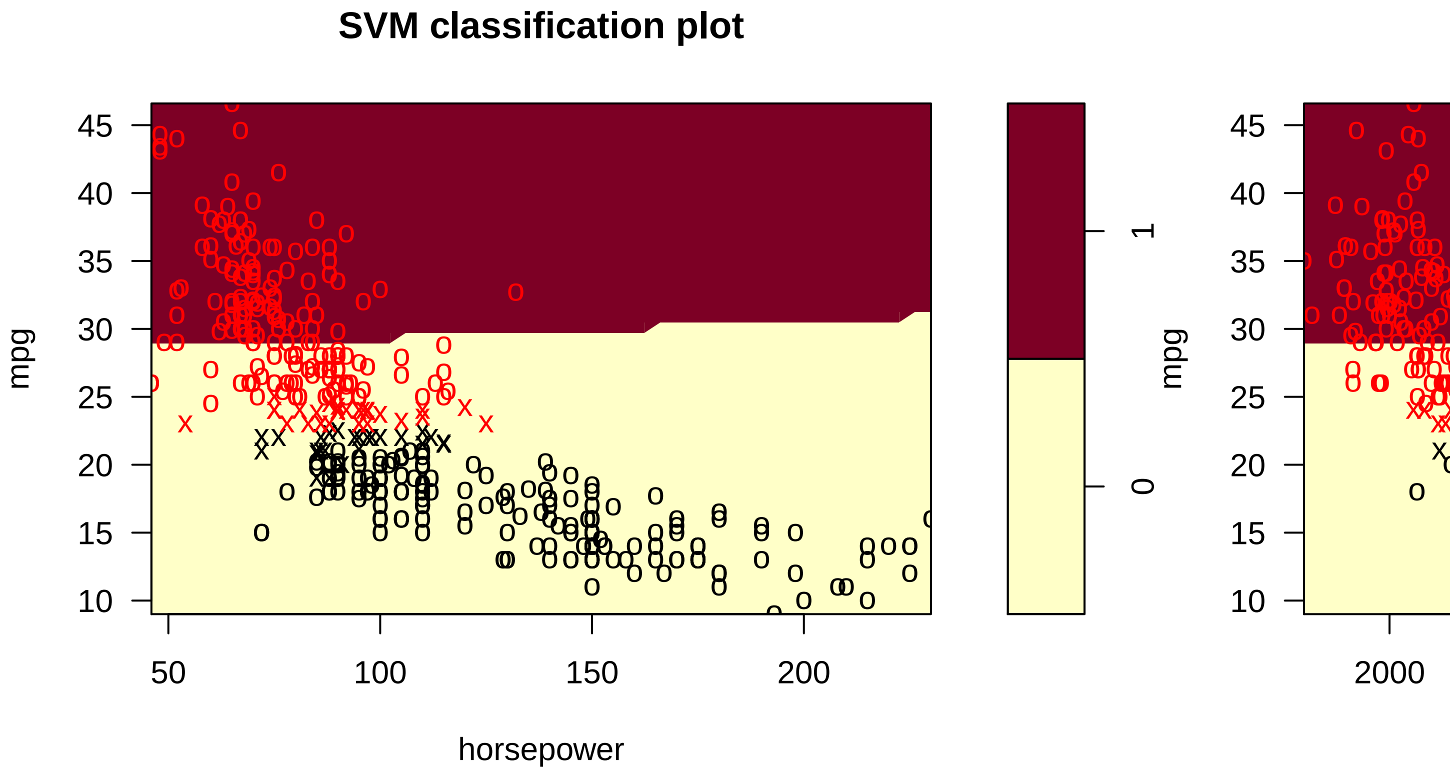
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100  0.01
##
## - best performance: 0.01282051
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-02 1e-02 0.55115385 0.04366593
## 2  1e-01 1e-02 0.08929487 0.04382379
## 3  1e+00 1e-02 0.07403846 0.03522110
## 4  5e+00 1e-02 0.04852564 0.03303346
## 5  1e+01 1e-02 0.02557692 0.02093679
## 6  1e+02 1e-02 0.01282051 0.01813094
## 7  1e-02 1e-01 0.21711538 0.09865227
## 8  1e-01 1e-01 0.07903846 0.03874545
## 9  1e+00 1e-01 0.05371795 0.03525162
## 10 5e+00 1e-01 0.02820513 0.03299190
## 11 1e+01 1e-01 0.03076923 0.03375798
## 12 1e+02 1e-01 0.03583333 0.02759051
## 13 1e-02 1e+00 0.55115385 0.04366593
## 14 1e-01 1e+00 0.55115385 0.04366593
## 15 1e+00 1e+00 0.06384615 0.04375618
## 16 5e+00 1e+00 0.05884615 0.04020934
## 17 1e+01 1e+00 0.05884615 0.04020934
## 18 1e+02 1e+00 0.05884615 0.04020934
## 19 1e-02 5e+00 0.55115385 0.04366593
## 20 1e-01 5e+00 0.55115385 0.04366593
```

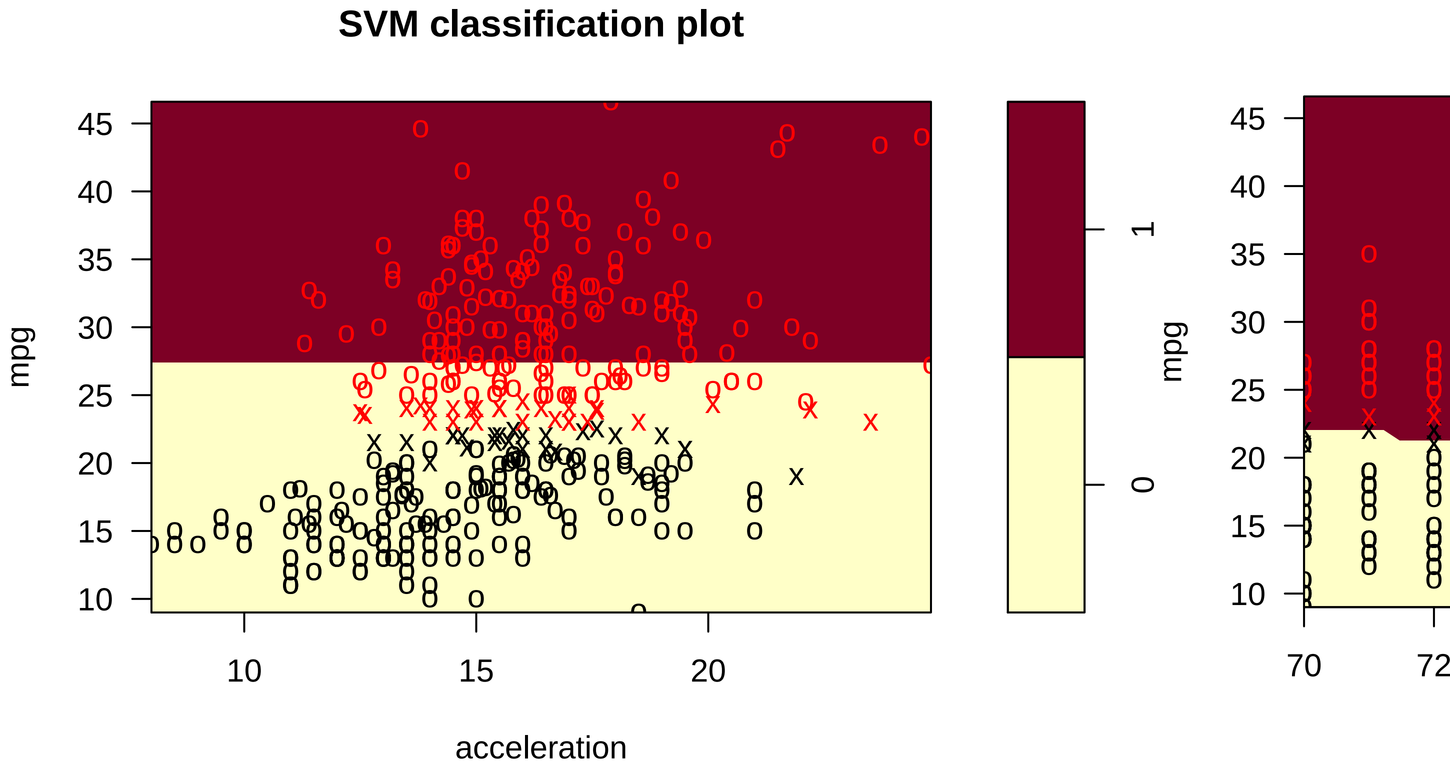
```
## 21 1e+00 5e+00 0.49493590 0.04724924
## 22 5e+00 5e+00 0.48217949 0.05470903
## 23 1e+01 5e+00 0.48217949 0.05470903
## 24 1e+02 5e+00 0.48217949 0.05470903
## 25 1e-02 1e+01 0.55115385 0.04366593
## 26 1e-01 1e+01 0.55115385 0.04366593
## 27 1e+00 1e+01 0.51794872 0.05063697
## 28 5e+00 1e+01 0.51794872 0.04917316
## 29 1e+01 1e+01 0.51794872 0.04917316
## 30 1e+02 1e+01 0.51794872 0.04917316
## 31 1e-02 1e+02 0.55115385 0.04366593
## 32 1e-01 1e+02 0.55115385 0.04366593
## 33 1e+00 1e+02 0.55115385 0.04366593
## 34 5e+00 1e+02 0.55115385 0.04366593
## 35 1e+01 1e+02 0.55115385 0.04366593
## 36 1e+02 1e+02 0.55115385 0.04366593
```

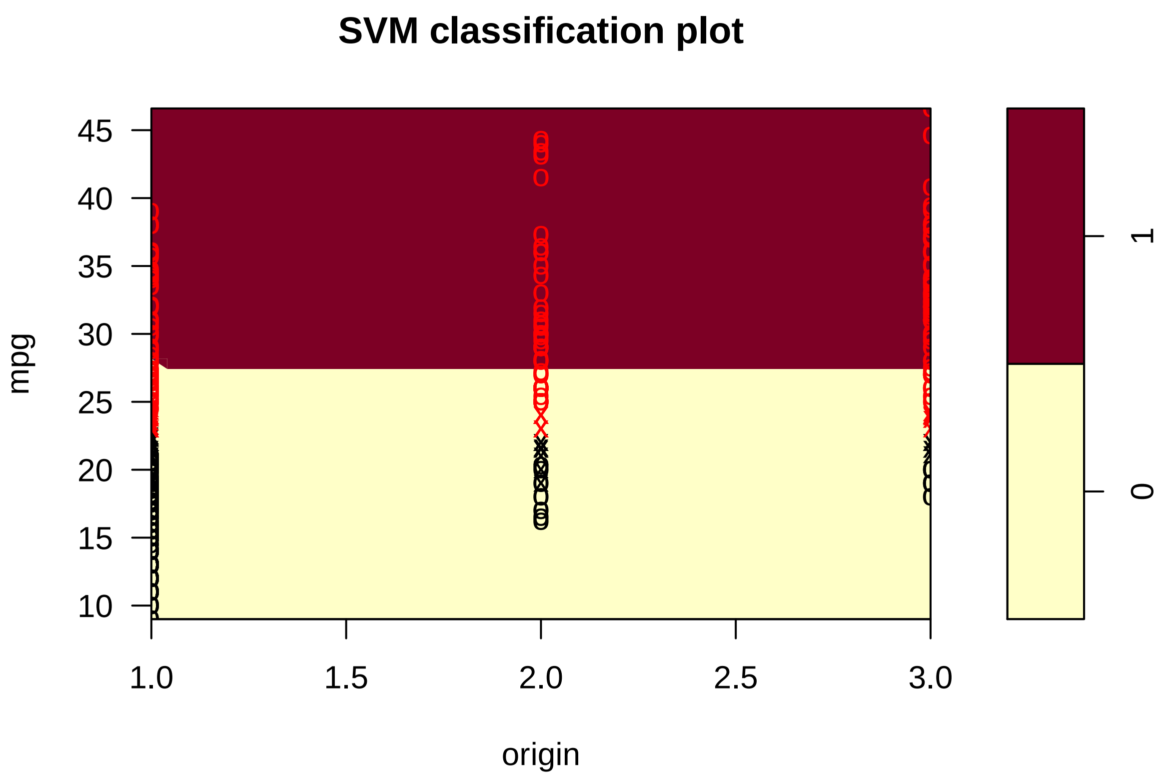
(d)

```
svm.linear <- svm(mpglevel ~ ., data = Auto, kernel = "linear", cost = 1)
svm.poly <- svm(mpglevel ~ ., data = Auto, kernel = "polynomial", cost = 100, degree = 2)
svm.radial <- svm(mpglevel ~ ., data = Auto, kernel = "radial", cost = 100, 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)
```

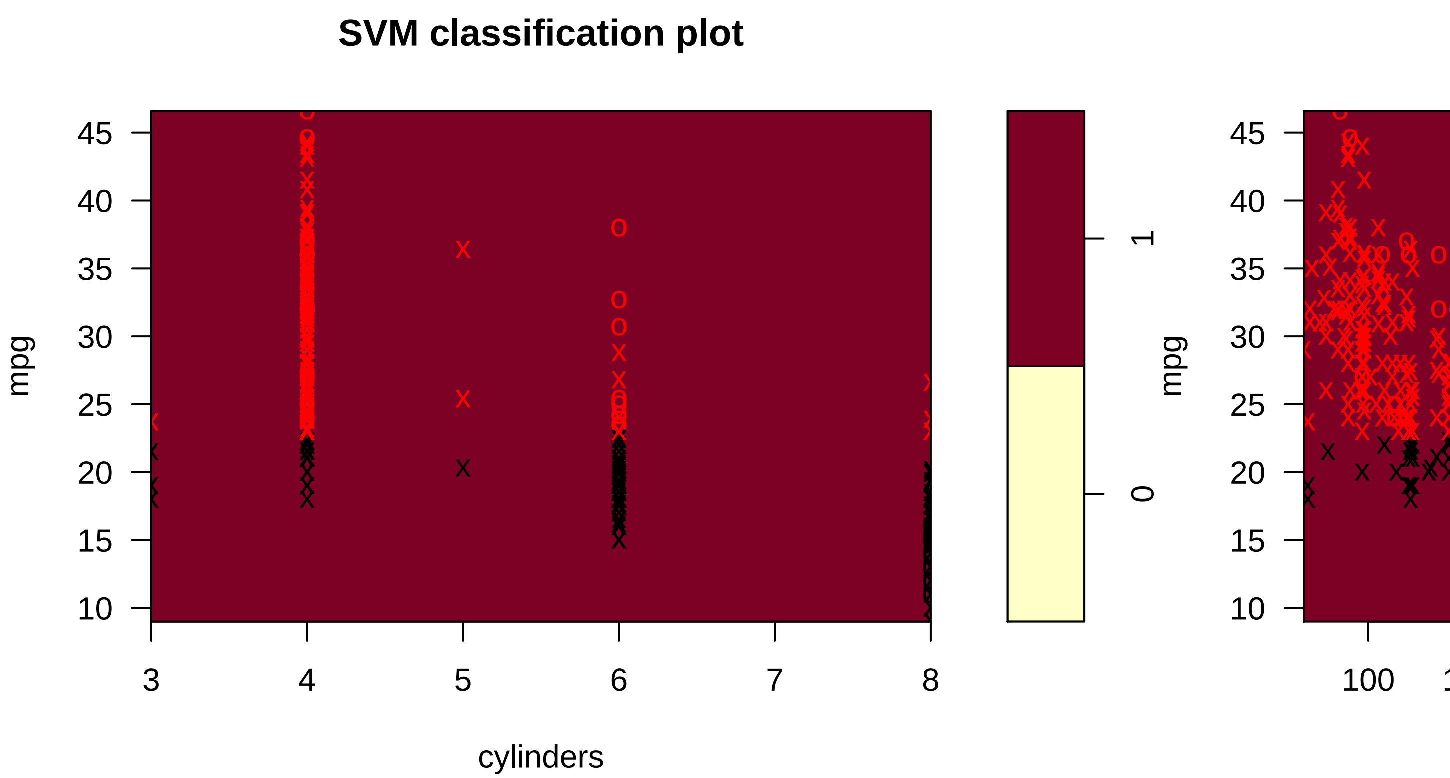


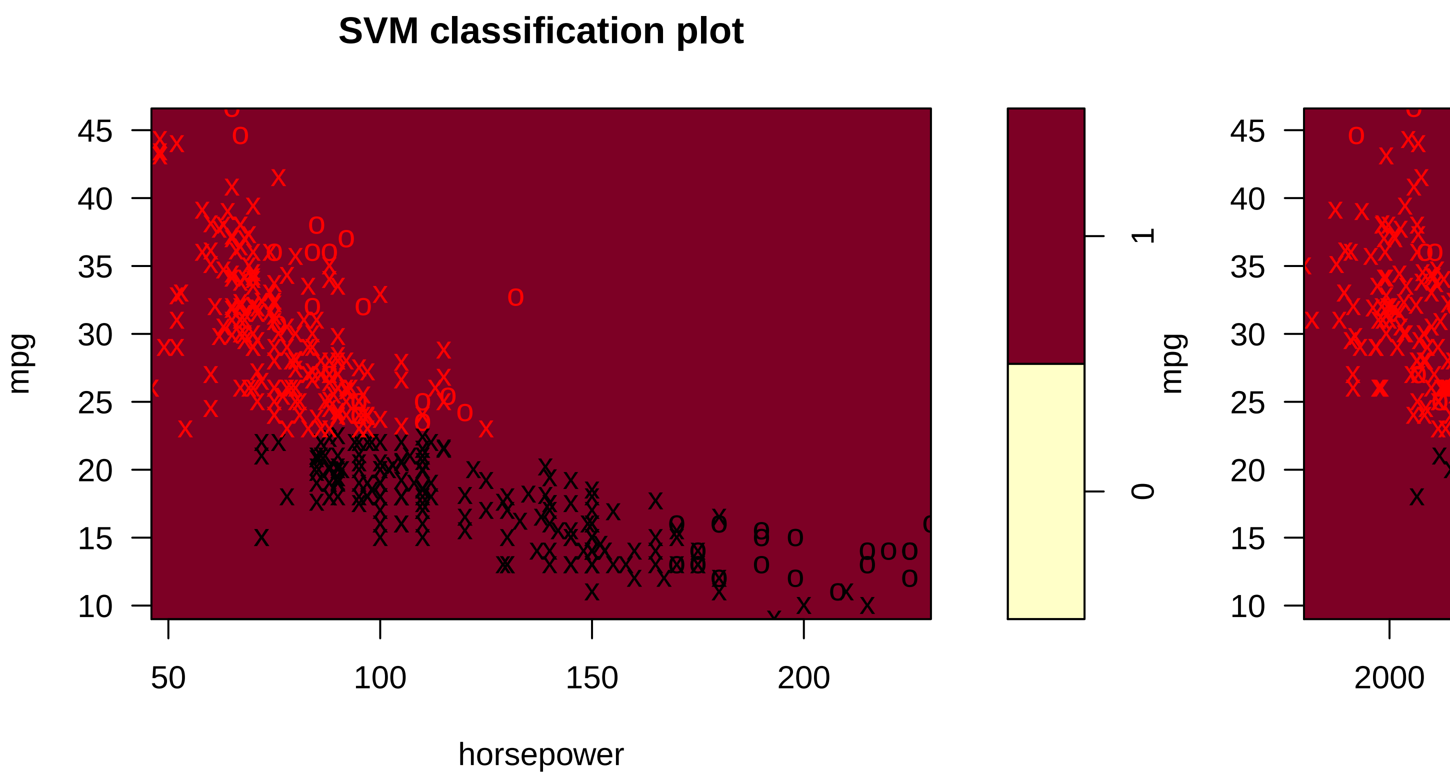

SVM classification plot
mpg
45
40
35
30
25
20
15
10
50
100
150
200
horsepower
1
0
mpg
2000

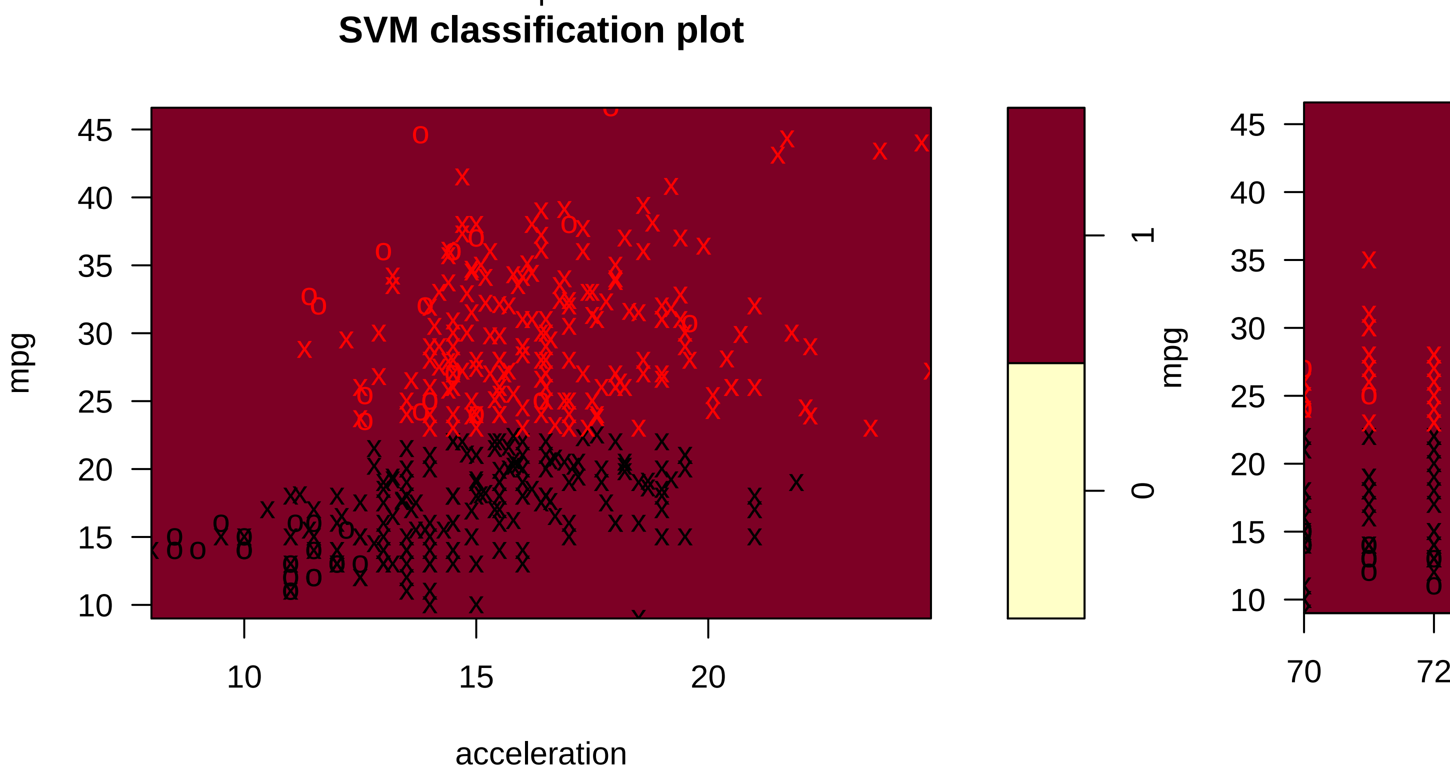

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0
mpg
70
72

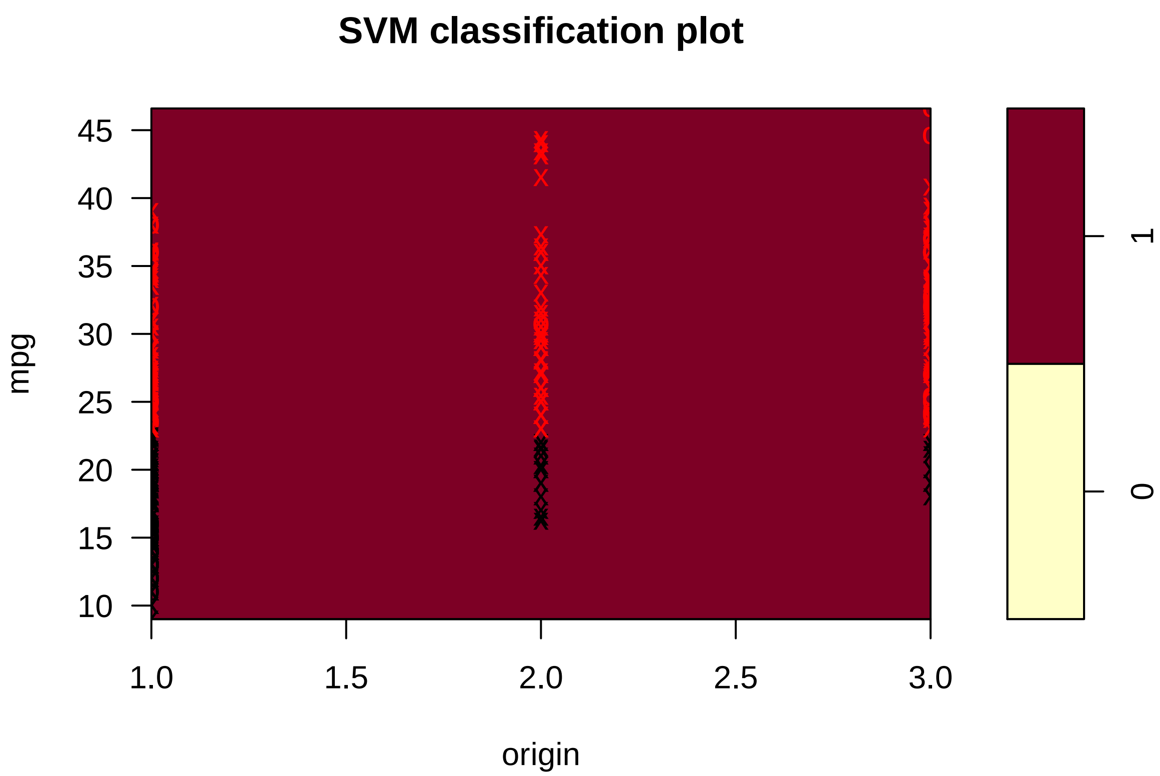



```r
plotpairs(svm.poly)
```

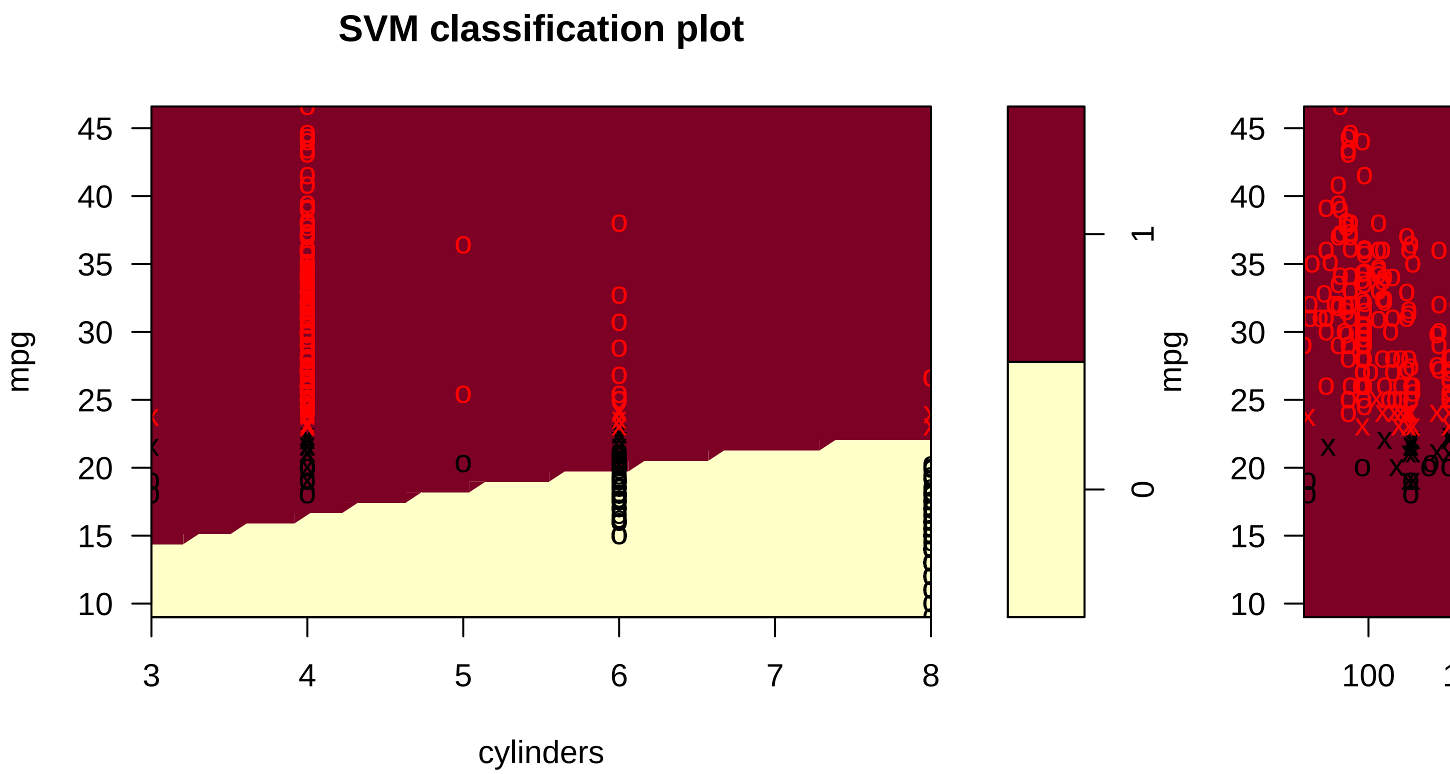


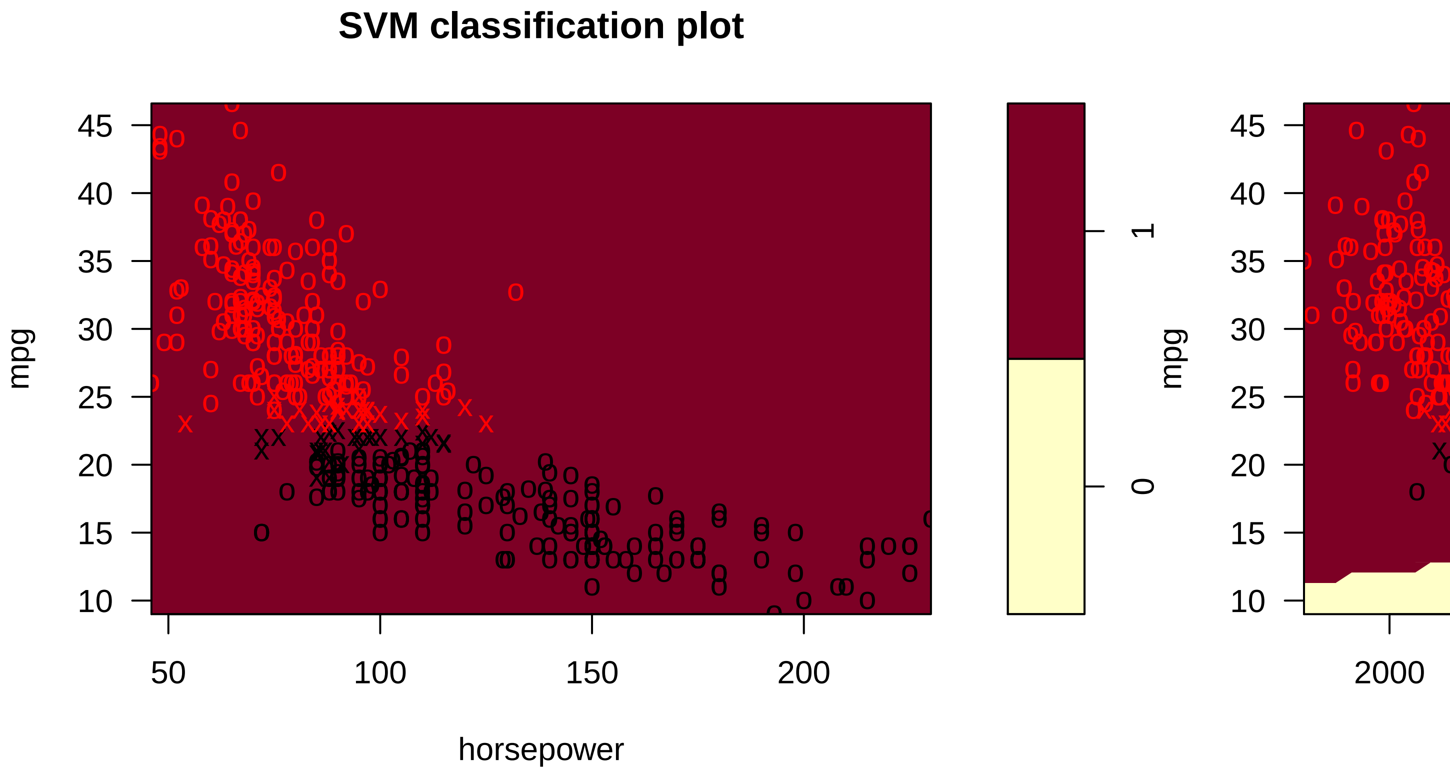

SVM classification plot
mpg
45
40
35
30
25
20
15
10
50
100
150
200
horsepower
1
0
mpg
45
40
35
30
25
20
15
10
2000

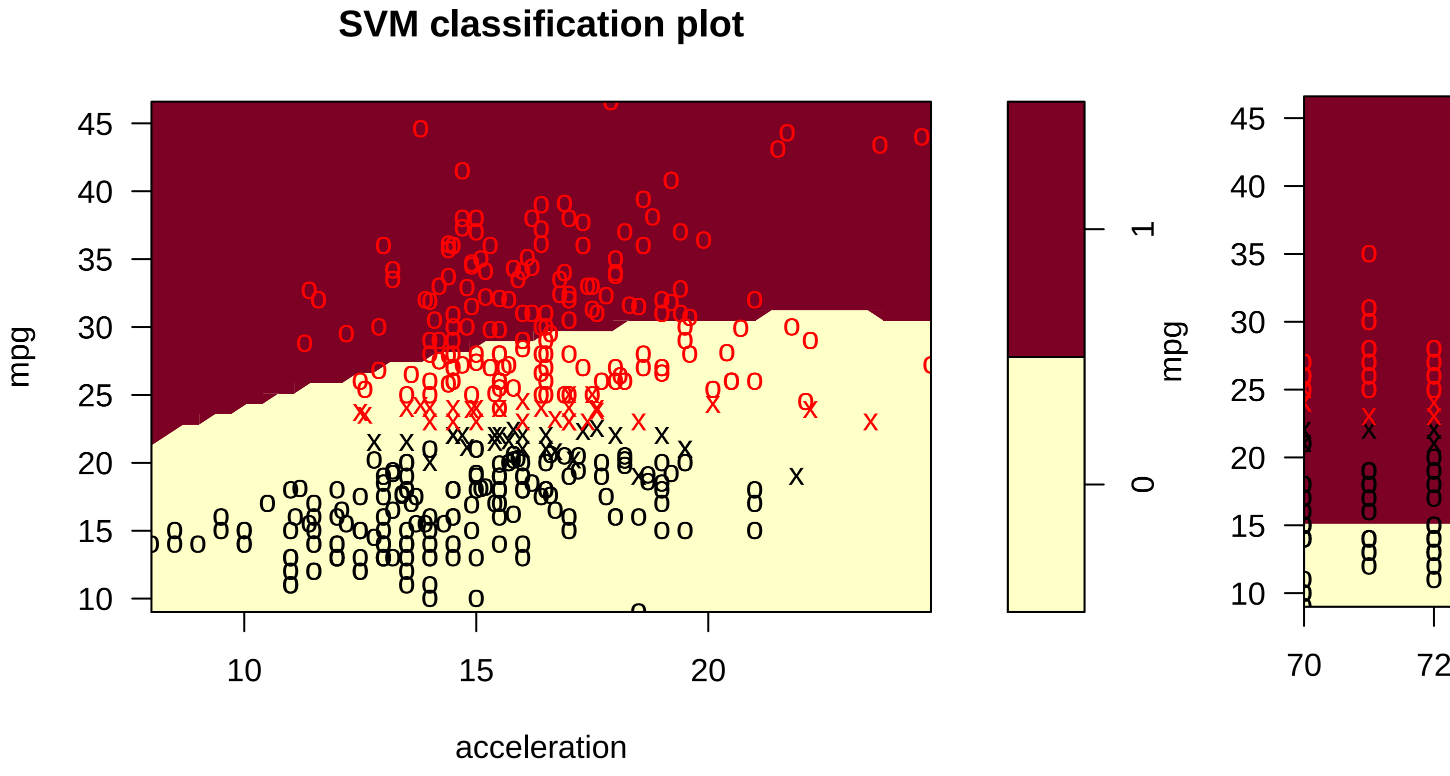

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0
mpg
45
40
35
30
25
20
15
10
70
72

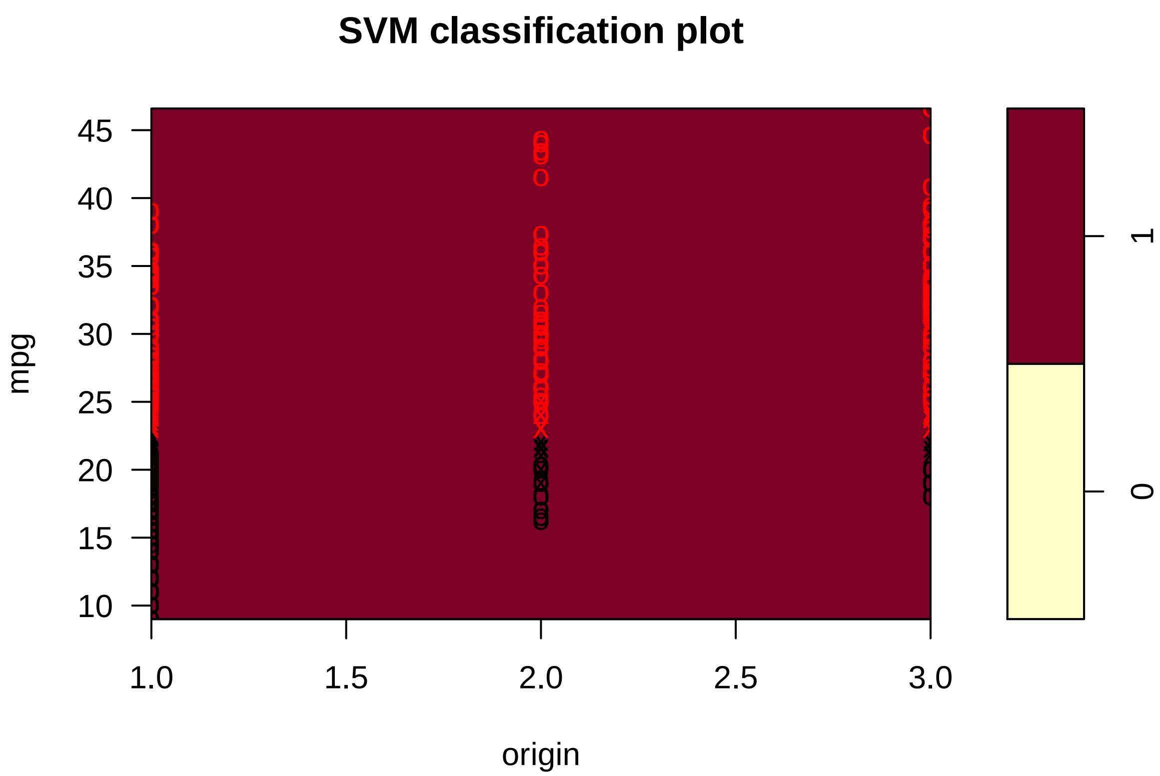



```r
plotpairs(svm.radial)
```




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


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



8 (a)

```
set.seed(1)
train <- sample(nrow(OJ), 800)
OJ.train <- OJ[train, ]
OJ.test <- OJ[-train, ]
```

(b)

```
svm.linear <- svm(Purchase ~ ., data = OJ.train, kernel = "linear", 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
##
## Number of Support Vectors:  435
##
##  ( 219 216 )
##
##
```

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

(c)

```r
train.pred <- predict(svm.linear, OJ.train)
table(OJ.train$Purchase, train.pred)
```

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

```r
(78 + 55) / (439 + 228 + 78 + 55)
```

```
## [1] 0.16625
```

```r
test.pred <- predict(svm.linear, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 153  15
##   MM  33  69
```

```r
(31 + 18) / (141 + 80 + 31 + 18)
```

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

(d)

```r
set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", ranges = list(cost = 10^seq(-2,
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  1.778279
##
## - best performance: 0.1675
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.17625 0.04059026
## 2   0.01778279 0.17625 0.04348132
## 3   0.03162278 0.17125 0.04604120
## 4   0.05623413 0.17000 0.04005205
## 5   0.10000000 0.17125 0.04168749
## 6   0.17782794 0.17000 0.04090979
```

```
## 7   0.31622777 0.17125 0.04411554
## 8   0.56234133 0.17125 0.04084609
## 9   1.00000000 0.17000 0.04090979
## 10  1.77827941 0.16750 0.03782269
## 11  3.16227766 0.16750 0.03782269
## 12  5.62341325 0.16750 0.03545341
## 13 10.00000000 0.17000 0.03736085
```

(e)

```
svm.linear <- svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = tune.out$best.parameter$cost)
train.pred <- predict(svm.linear, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 423  62
##   MM  69 246
```

```
(71 + 56) / (438 + 235 + 71 + 56)
```

```
## [1] 0.15875
```

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

```
##     test.pred
##       CH  MM
##   CH 156  12
##   MM  29  73
```

```
(32 + 19) / (140 + 79 + 32 + 19)
```

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

(f)

```
svm.radial <- svm(Purchase ~ ., kernel = "radial", data = OJ.train)
summary(svm.radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  373
##
##  ( 188 185 )
##
##
## Number of Classes:  2
```

```
##
## Levels:
##  CH MM
```

```r
train.pred <- predict(svm.radial, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 441  44
##   MM  77 238
```

```r
(77 + 39) / (455 + 229 + 77 + 39)
```

```
## [1] 0.145
```

```r
test.pred <- predict(svm.radial, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 151  17
##   MM  33  69
```

```r
(28 + 18) / (141 + 83 + 28 + 18)
```

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

```r
set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", ranges = list(cost = 10^seq(-2,
    1, by = 0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.1725
##
## - Detailed performance results:
##          cost   error dispersion
## 1   0.01000000 0.39375 0.03240906
## 2   0.01778279 0.39375 0.03240906
## 3   0.03162278 0.34750 0.05552777
## 4   0.05623413 0.19250 0.03016160
## 5   0.10000000 0.19500 0.03782269
## 6   0.17782794 0.18000 0.04048319
## 7   0.31622777 0.17250 0.03809710
## 8   0.56234133 0.17500 0.04124790
## 9   1.00000000 0.17250 0.03162278
## 10  1.77827941 0.17750 0.03717451
## 11  3.16227766 0.18375 0.03438447
## 12  5.62341325 0.18500 0.03717451
```

```
## 13 10.00000000 0.18750 0.03173239
```

```
svm.radial <- svm(Purchase ~ ., kernel = "radial", data = OJ.train, cost = tune.out$best.parameter$cost)
summary(svm.radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial",
##     cost = tune.out$best.parameter$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  373
##
##  ( 188 185 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.radial, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 441  44
##   MM  77 238
```

```
(77 + 39) / (455 + 229 + 77 + 39)
```

```
## [1] 0.145
```

```
test.pred <- predict(svm.radial, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 151  17
##   MM  33  69
```

```
(28 + 18) / (141 + 83 + 28 + 18)
```

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

(g)

```
svm.poly <- svm(Purchase ~ ., kernel = "polynomial", data = OJ.train, degree = 2)
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial",
```

```
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  447
##
##  ( 225 222 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
train.pred <- predict(svm.poly, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 449  36
##   MM 110 205
```

```r
(105 + 33) / (461 + 201 + 105 + 33)
```

```
## [1] 0.1725
```

```r
test.pred <- predict(svm.poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 153  15
##   MM  45  57
```

```r
(41 + 10) / (149 + 70 + 41 + 10)
```

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

```r
set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "polynomial", degree = 2, ranges = list(c
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  3.162278
##
## - best performance: 0.18
```

```
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39000 0.03670453
## 2   0.01778279 0.37000 0.03395258
## 3   0.03162278 0.36375 0.03197764
## 4   0.05623413 0.34500 0.03291403
## 5   0.10000000 0.32125 0.03866254
## 6   0.17782794 0.24750 0.03322900
## 7   0.31622777 0.20250 0.04073969
## 8   0.56234133 0.20250 0.03670453
## 9   1.00000000 0.19625 0.03910900
## 10  1.77827941 0.19125 0.03586723
## 11  3.16227766 0.18000 0.04005205
## 12  5.62341325 0.18000 0.04133199
## 13 10.00000000 0.18125 0.03830162
```

```
svm.poly <- svm(Purchase ~ ., kernel = "polynomial", degree = 2, data = OJ.train, cost = tune.out$best.p
summary(svm.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial",
##     degree = 2, cost = tune.out$best.parameter$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  3.162278
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  385
##
##  ( 197 188 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred <- predict(svm.poly, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 451  34
##   MM  90 225
```

```
(72 + 44) / (450 + 234 + 72 + 44)
```

```
## [1] 0.145
```

```
test.pred <- predict(svm.poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 154  14
##   MM  41  61
```

```r
(31 + 19) / (140 + 80 + 31 + 19)
```

```
## [1] 0.1851852
```

(h)

The radial basis kernel seems to be producing minimum misclassification error on both train and test data.