SVM-hw

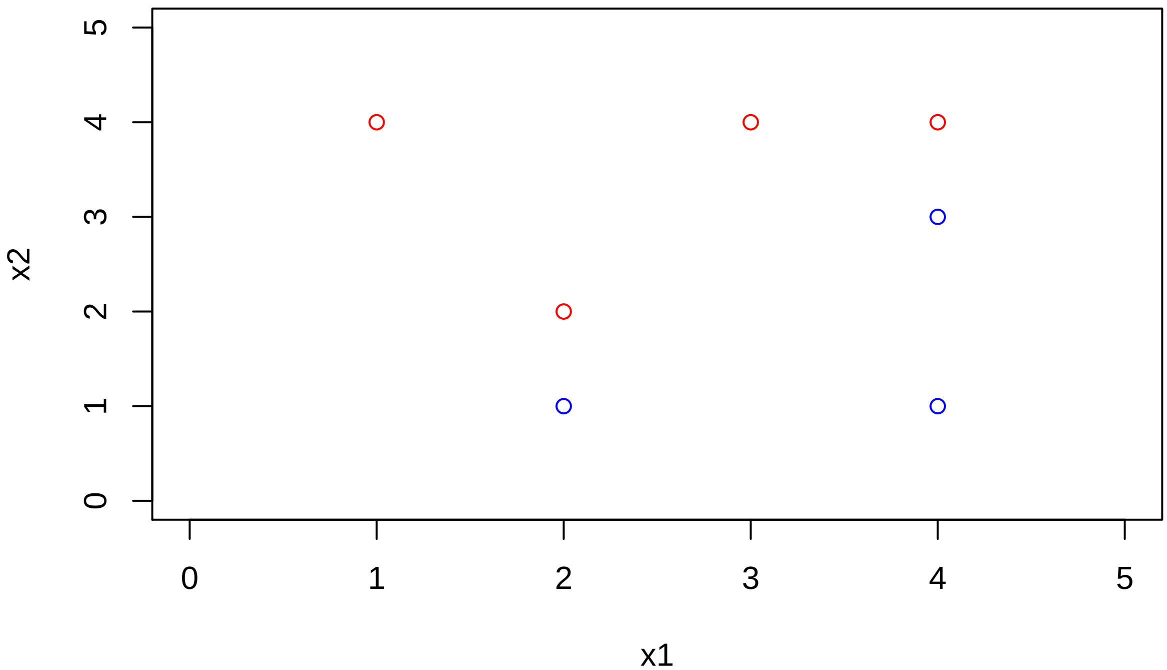
Ziyi Bai

2021/3/11

9.3

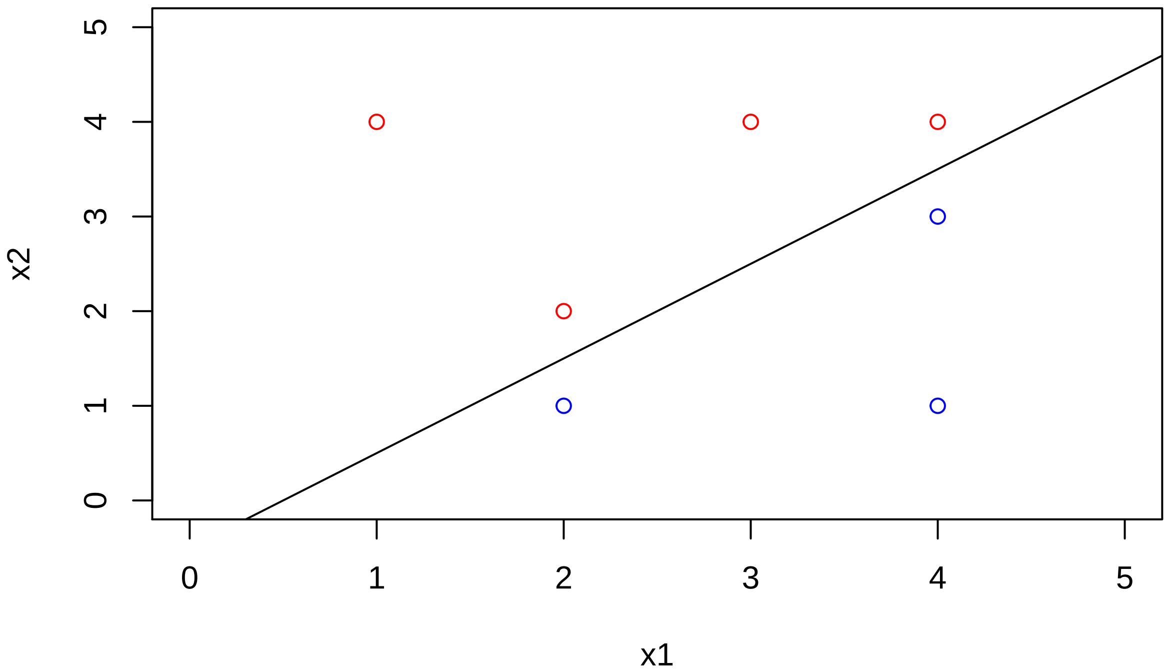
##(a)

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



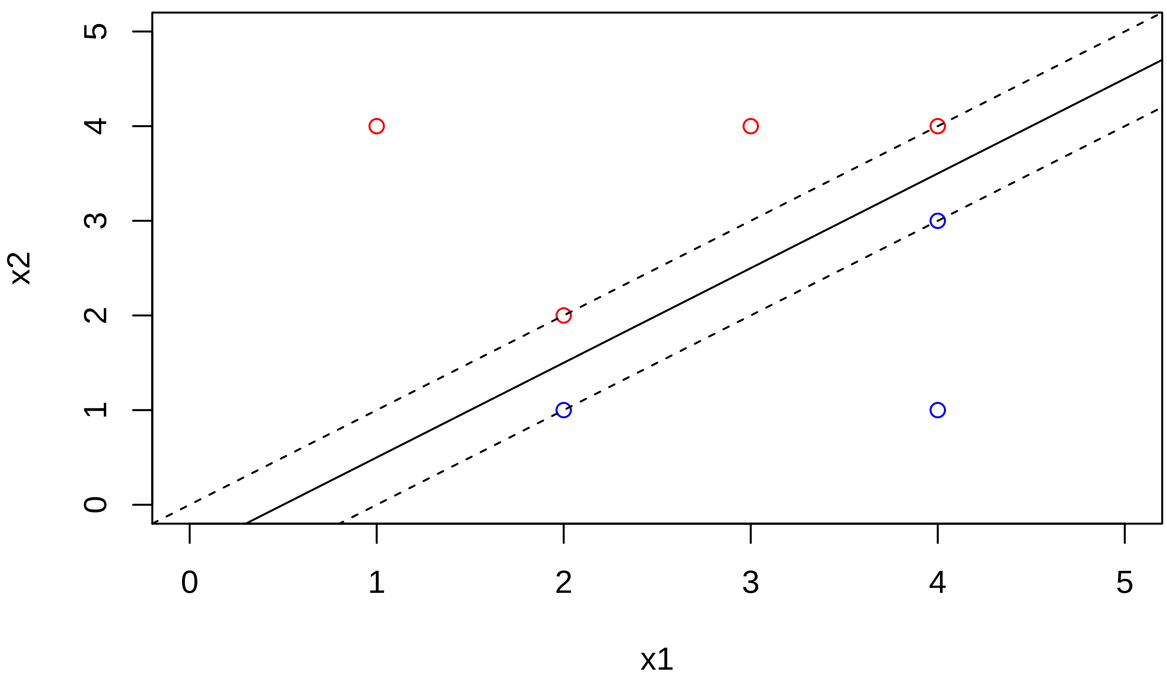
##(b)

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



##(d)

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



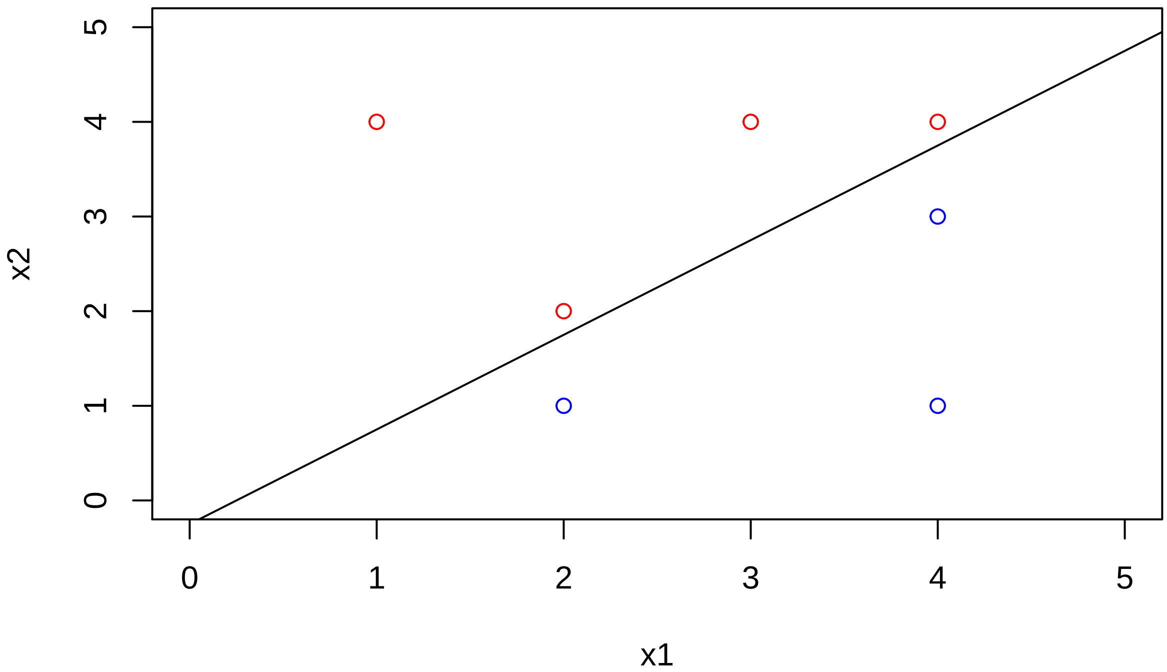
##(e)

The support vectors for the maximal margin classifier are the points (2,1), (2,2), (4,3) and (4,4)

##(f) The 7th observations is not a support vector, so move it won't affect the outcome.

##(g)

```
plot(x1,x2,col=cols,xlim = c(0,5),ylim = c(0,5))
abline(-0.25,1)
```



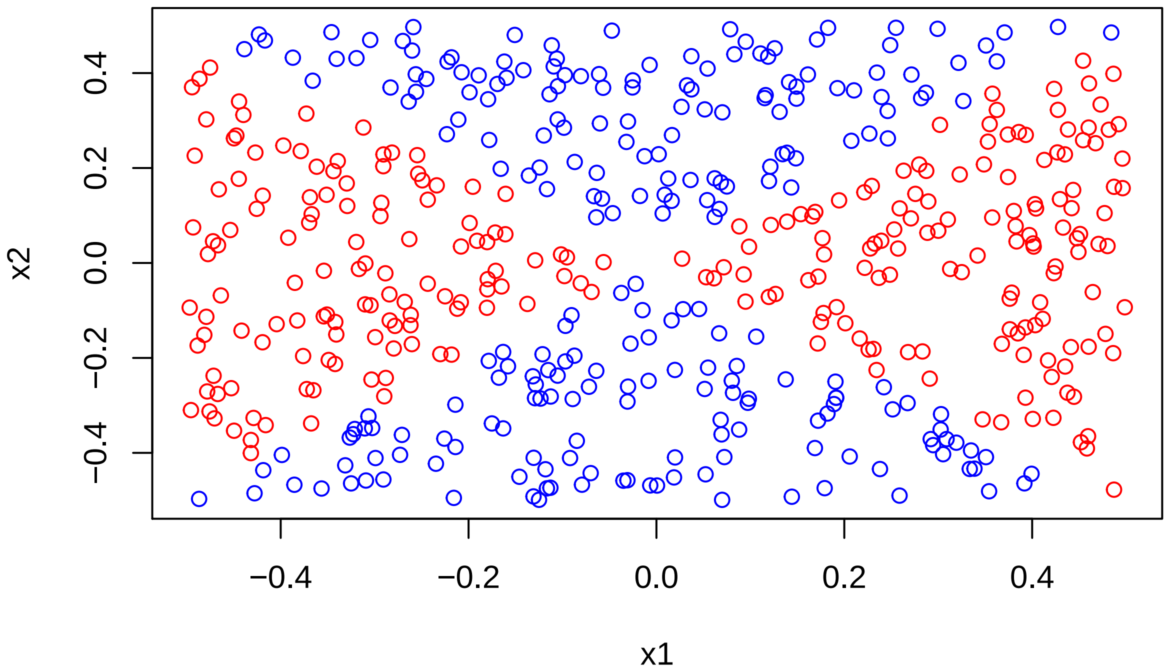
9.5

##(a)

```
set.seed(9)
x1=runif(500)-0.5
x2=runif(500)-0.5
y=1*(x1^2-x2^2 > 0)
```

##(b)

```
plot(x1,x2,col=ifelse(y,"red","blue"))
```



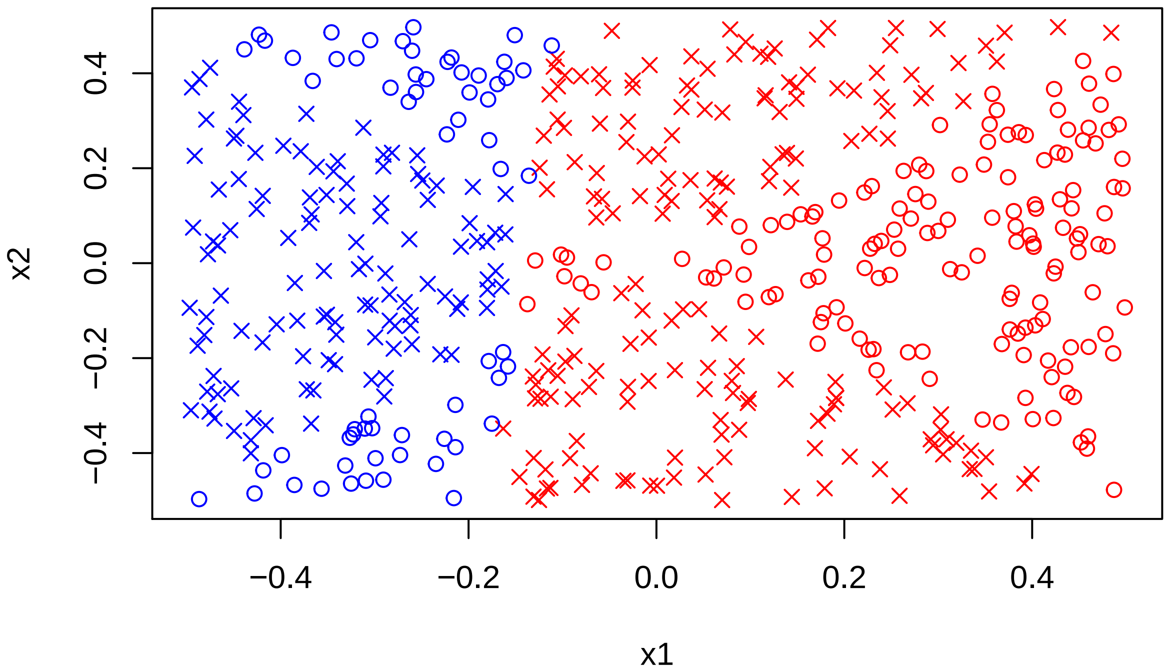
##(c)

```r
df1 <- data.frame(x1,x2,y)
fit_glm <- glm(y~x1+x2, data=df1,family = binomial)
fit_glm
```

```
##
## Call:  glm(formula = y ~ x1 + x2, family = binomial, data = df1)
##
## Coefficients:
## (Intercept)           x1           x2
##     0.05514      0.38587     -0.02653
##
## Degrees of Freedom: 499 Total (i.e. Null);  497 Residual
## Null Deviance:       692.8
## Residual Deviance: 691.2     AIC: 697.2
```

##(d)

```r
pred_fit <- predict(fit_glm,data.frame(x1,x2))
plot(x1,x2,col=ifelse(pred_fit>0,"red","blue"),pch=ifelse(as.integer(pred_fit>0)==y,1,4))
```



##(e)

```r
fit_glm1 <- glm(y~poly(x1,2)+poly(x2,2),data=df1,family = binomial)
```

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

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

```r
summary(fit_glm1)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2) + poly(x2, 2), family = binomial,
##     data = df1)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
```

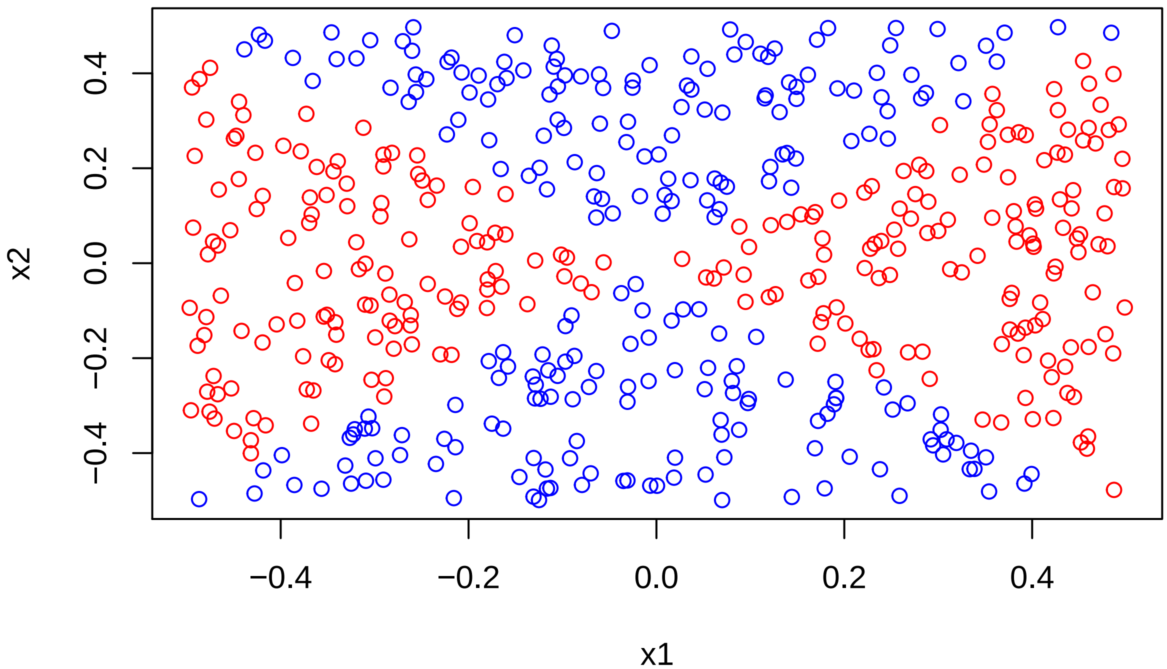
```
## -1.079e-03  -2.000e-08   2.000e-08   2.000e-08   9.076e-04
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)       43.78    3063.63   0.014    0.989
## poly(x1, 2)1    1360.39  102905.10   0.013    0.989
## poly(x1, 2)2   21374.91  785951.63   0.027    0.978
## poly(x2, 2)1    -119.10   88918.85  -0.001    0.999
## poly(x2, 2)2  -21333.50  788724.67  -0.027    0.978
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9276e+02  on 499  degrees of freedom
## Residual deviance: 2.4730e-06  on 495  degrees of freedom
## AIC: 10
##
## Number of Fisher Scoring iterations: 25
```

```r
fit_glm2 <- glm(y~x1+x2+x1*x2,data=df1,family=binomial)
summary(fit_glm2)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2 + x1 * x2, family = binomial, data = df1)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.342  -1.199   1.050   1.144   1.291
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.05206    0.08983   0.580    0.562
## x1           0.38234    0.31036   1.232    0.218
## x2          -0.01969    0.31760  -0.062    0.951
## x1:x2        0.64537    1.12041   0.576    0.565
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.76  on 499  degrees of freedom
## Residual deviance: 690.87  on 496  degrees of freedom
## AIC: 698.87
##
## Number of Fisher Scoring iterations: 3
```

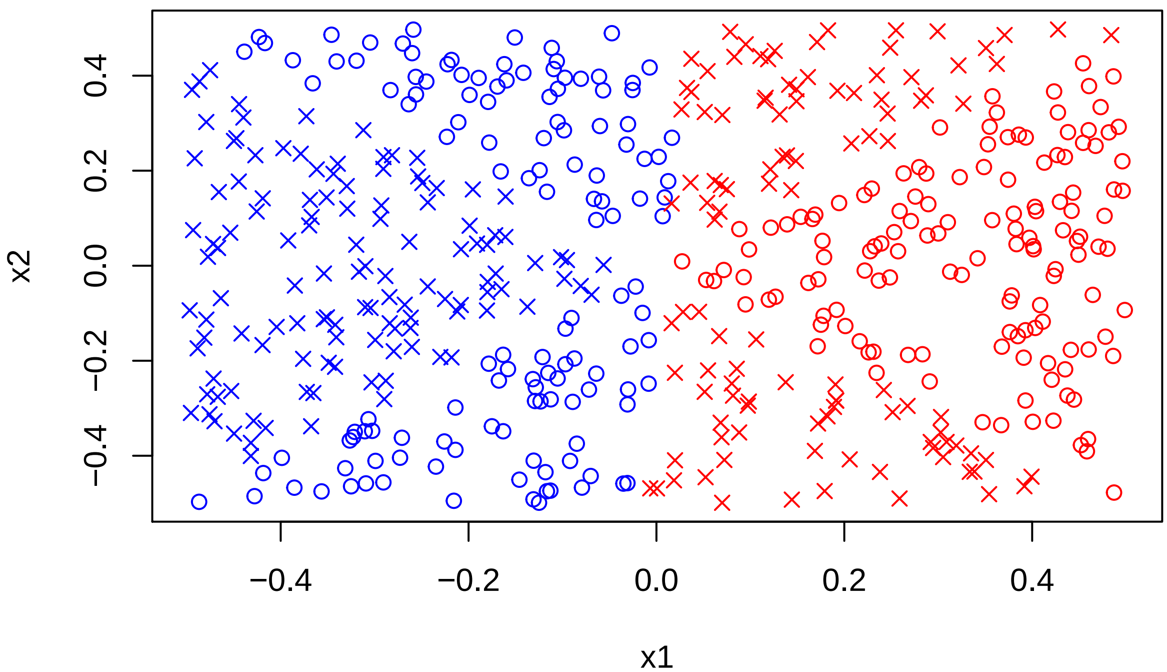
##(f)

```r
pred_fit1 <- predict(fit_glm1, df1)
plot(x1, x2, col = ifelse(pred_fit1 > 0, "red", "blue"), pch = ifelse(as.integer(pred_fit1 > 0) == y, 1
```



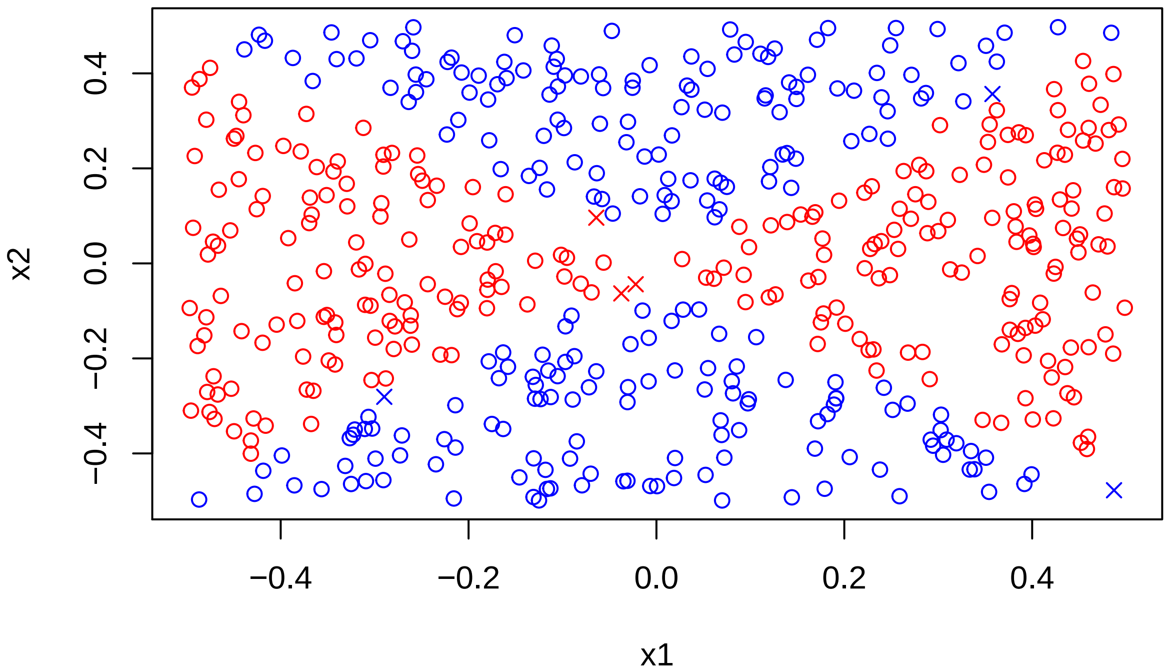
##(g)

```
df1$y <- as.factor(df1$y)
fit_svc <- svm(y~x1+x2,data=df1,kernel="linear")
pred_svc <- predict(fit_svc,df1,type="response")
plot(x1,x2,col=ifelse(pred_svc!=0,"red","blue"),pch=ifelse(pred_svc==y,1,4))
```



##(h)

```
fit_svm <- svm(y ~ x1 + x2, data = df1, kernel = "polynomial", degree = 2)
pred_svm <- predict(fit_svm, df1, type = "response")
plot(x1, x2, col = ifelse(pred_svm != 0, "red", "blue"), pch = ifelse(pred_svm == y, 1,4))
```



##(i)

Using polynomial gives us better result.

9.7

##(a)

```
data("Auto")
Auto$Y <- ifelse(Auto$mpg > median(Auto$mpg),1,0)
Auto$Y <- as.factor(Auto$Y)
```

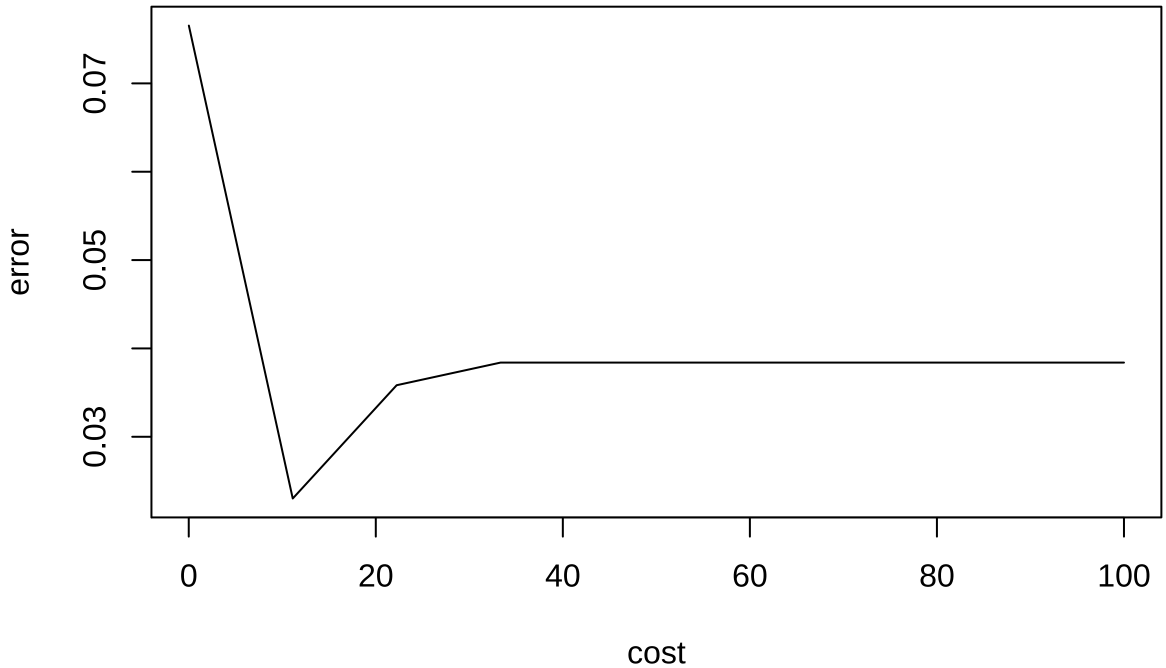
##(b)

```
set.seed(9)
cost <- data.frame(cost=seq(0.01,100,length.out = 10))
svm_tune <- tune(svm,Y~.,data=Auto,kernel="linear",ranges = cost)
summary(svm_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##    cost
##   11.12
##
## - best performance: 0.02301282
##
## - Detailed performance results:
##       cost      error dispersion
## 1     0.01 0.07653846 0.05100638
## 2    11.12 0.02301282 0.01891104
## 3    22.23 0.03583333 0.03245677
## 4    33.34 0.03839744 0.03872235
## 5    44.45 0.03839744 0.03872235
```

```
## 6   55.56 0.03839744 0.03872235
## 7   66.67 0.03839744 0.03872235
## 8   77.78 0.03839744 0.03872235
## 9   88.89 0.03839744 0.03872235
## 10 100.00 0.03839744 0.03872235
```

```r
plot(svm_tune$performances[,c(1,2)],type="l")
```



cost=11.12 has the best performance.

##(c)

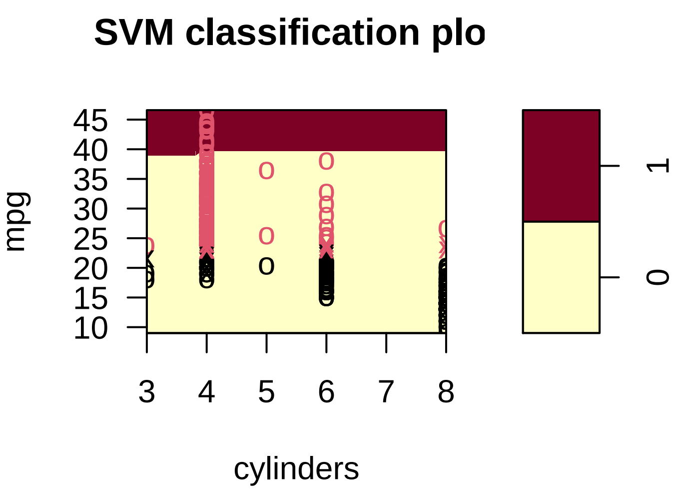
```r
para <- data.frame(cost=seq(0.01,100,length.out = 5),degree=seq(1,100,length.out = 5))
svm_poly <- tune(svm,Y~.,data=Auto,kernel="polynomial",ranges = para)
summary(svm_poly)
```

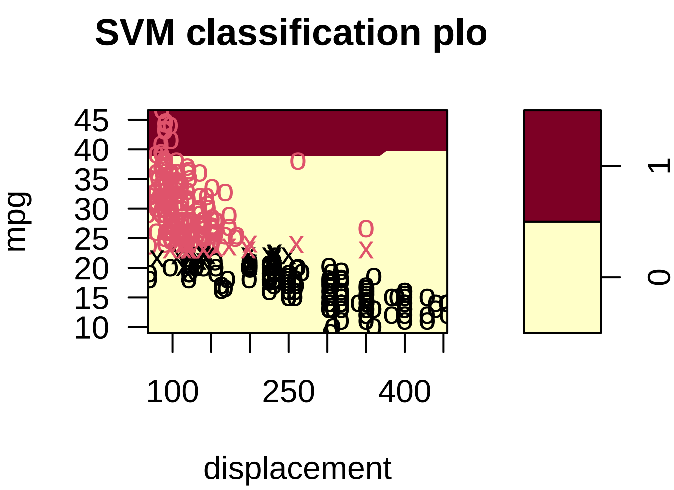
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##     cost degree
##  75.0025      1
##
## - best performance: 0.02814103
##
## - Detailed performance results:
##        cost degree      error dispersion
## 1    0.0100   1.00 0.54852564 0.01899199
## 2   25.0075   1.00 0.05102564 0.03419231
## 3   50.0050   1.00 0.03326923 0.02434857
## 4   75.0025   1.00 0.02814103 0.01893035
## 5  100.0000   1.00 0.02814103 0.01893035
## 6    0.0100  25.75 0.54852564 0.01899199
```

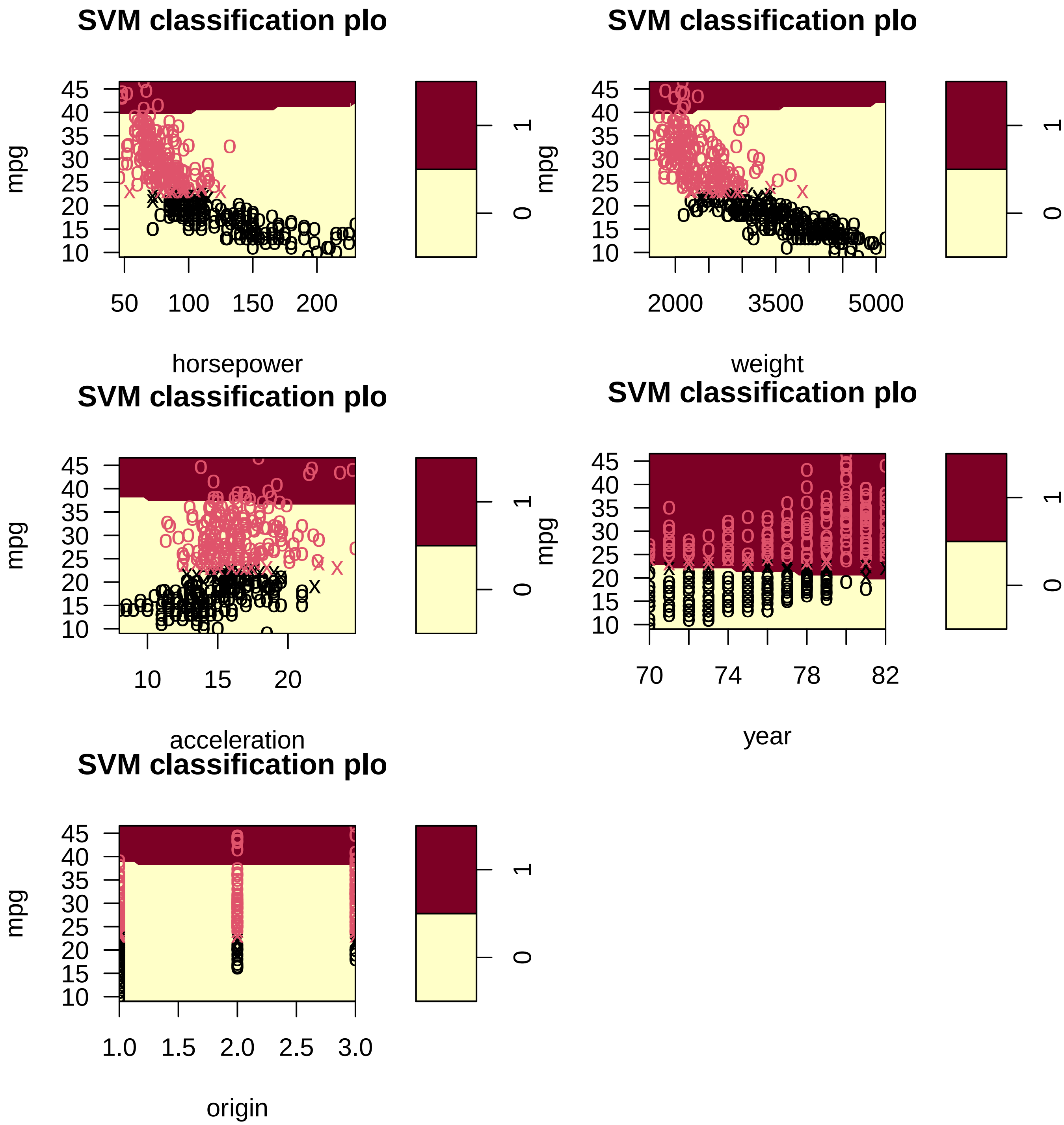
```
## 7    25.0075   25.75 0.54852564 0.01899199
## 8    50.0050   25.75 0.54852564 0.01899199
## 9    75.0025   25.75 0.54852564 0.01899199
## 10  100.0000   25.75 0.54852564 0.01899199
## 11    0.0100   50.50 0.54852564 0.01899199
## 12   25.0075   50.50 0.54852564 0.01899199
## 13   50.0050   50.50 0.54852564 0.01899199
## 14   75.0025   50.50 0.54852564 0.01899199
## 15  100.0000   50.50 0.54852564 0.01899199
## 16    0.0100   75.25 0.54852564 0.01899199
## 17   25.0075   75.25 0.54852564 0.01899199
## 18   50.0050   75.25 0.54852564 0.01899199
## 19   75.0025   75.25 0.54852564 0.01899199
## 20  100.0000   75.25 0.54852564 0.01899199
## 21    0.0100  100.00 0.54852564 0.01899199
## 22   25.0075  100.00 0.54852564 0.01899199
## 23   50.0050  100.00 0.54852564 0.01899199
## 24   75.0025  100.00 0.54852564 0.01899199
## 25  100.0000  100.00 0.54852564 0.01899199
```

##(d)

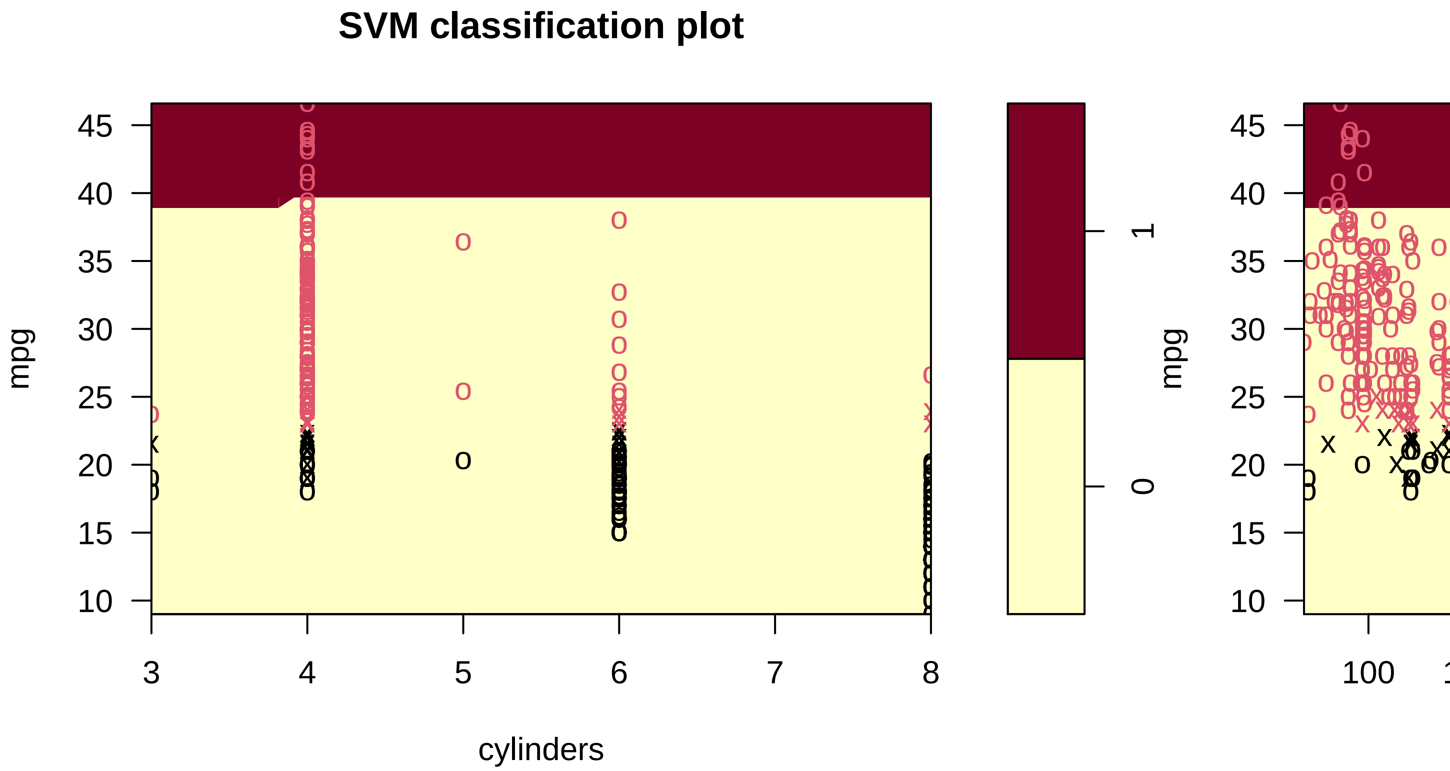
```
linear <- svm(Y ~ ., data = Auto, kernel = "linear", cost = 11.12)
polynomial <- svm(Y ~ ., data = Auto, kernel = "polynomial", cost = 100, degree = 1)
radial <- svm(Y ~ ., data = Auto, kernel = "radial", cost = 25.0075, gamma = 0.1)
pair_plot <- function(a){
  for (name in names(Auto)[!(names(Auto) %in% c("mpg", "Y", "name"))])
    plot(a, Auto, as.formula(paste("mpg~", name, sep = "")))
}
pair_plot(linear)
```

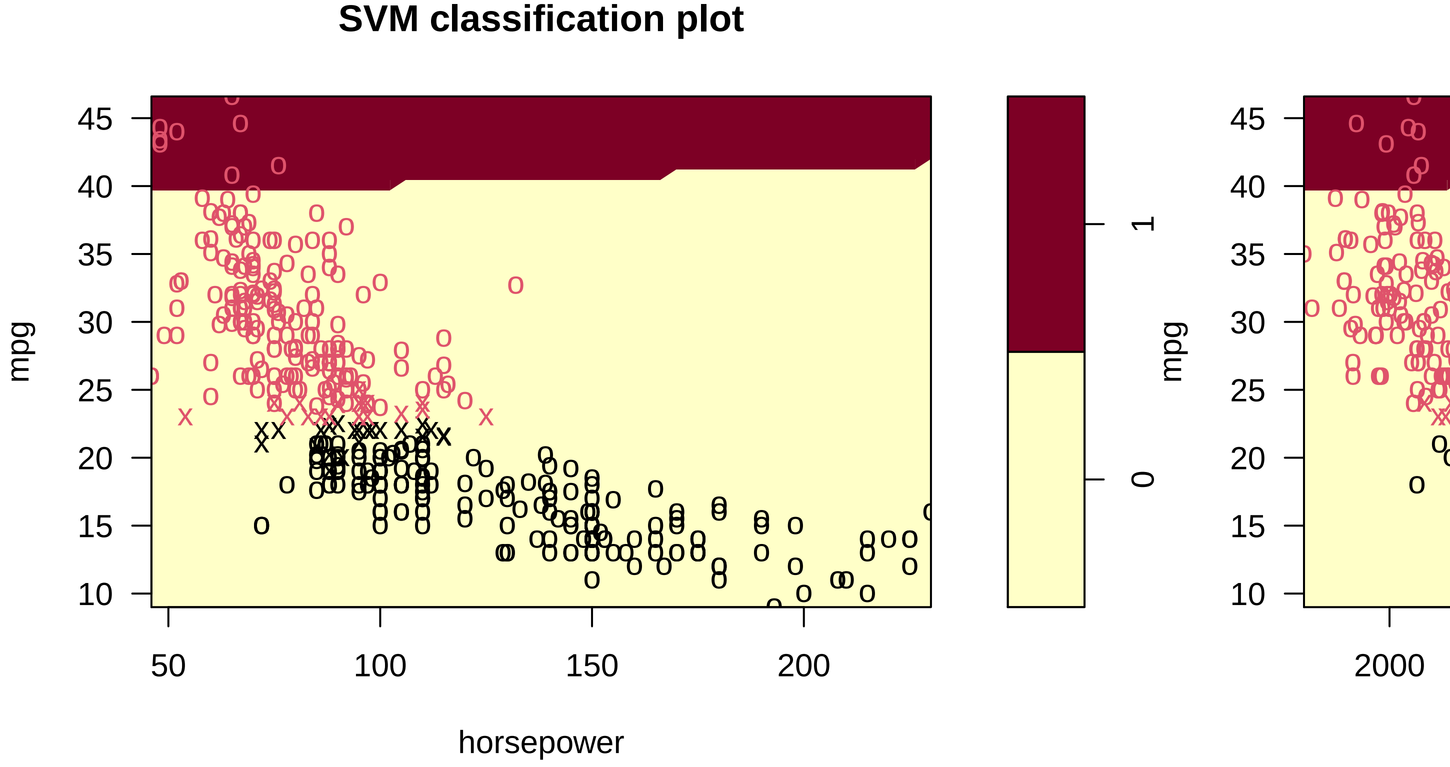


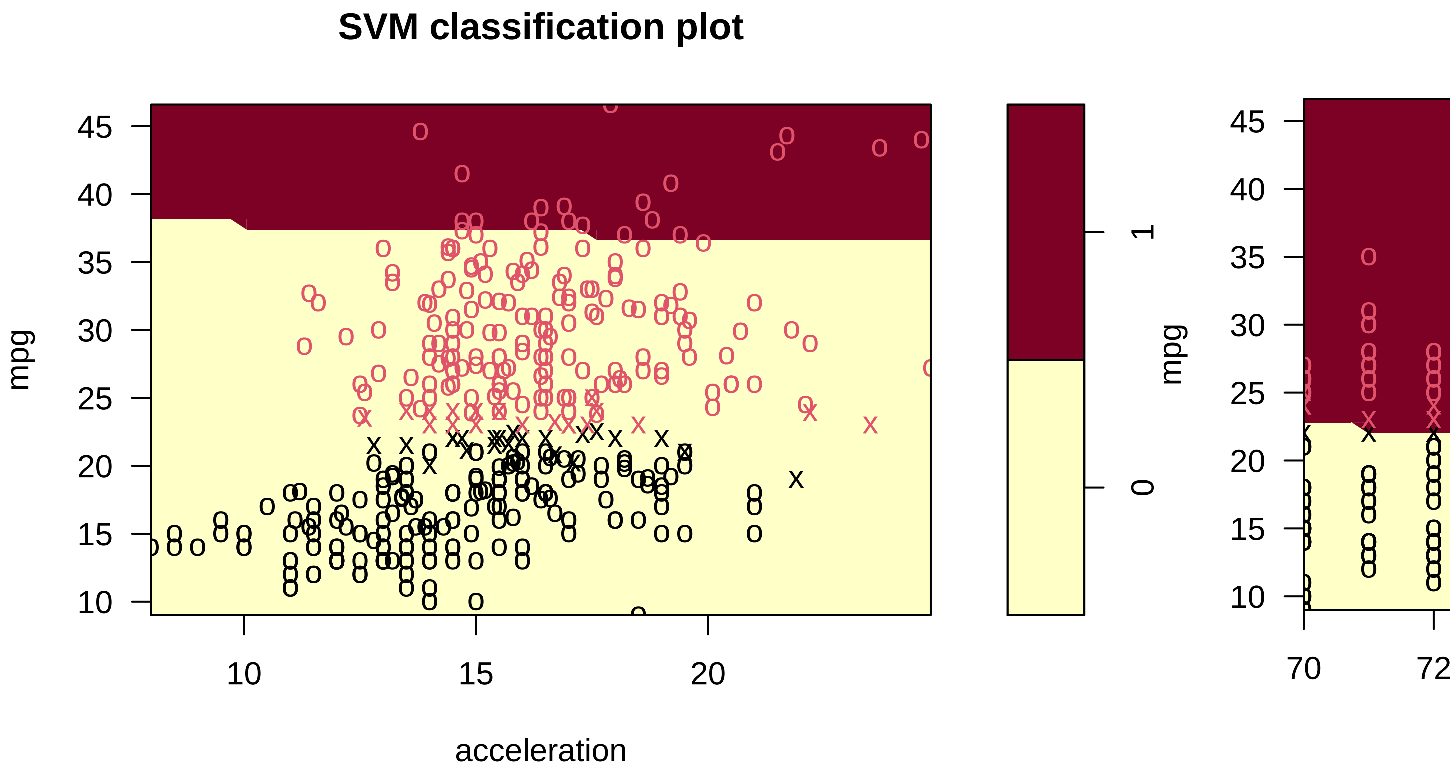


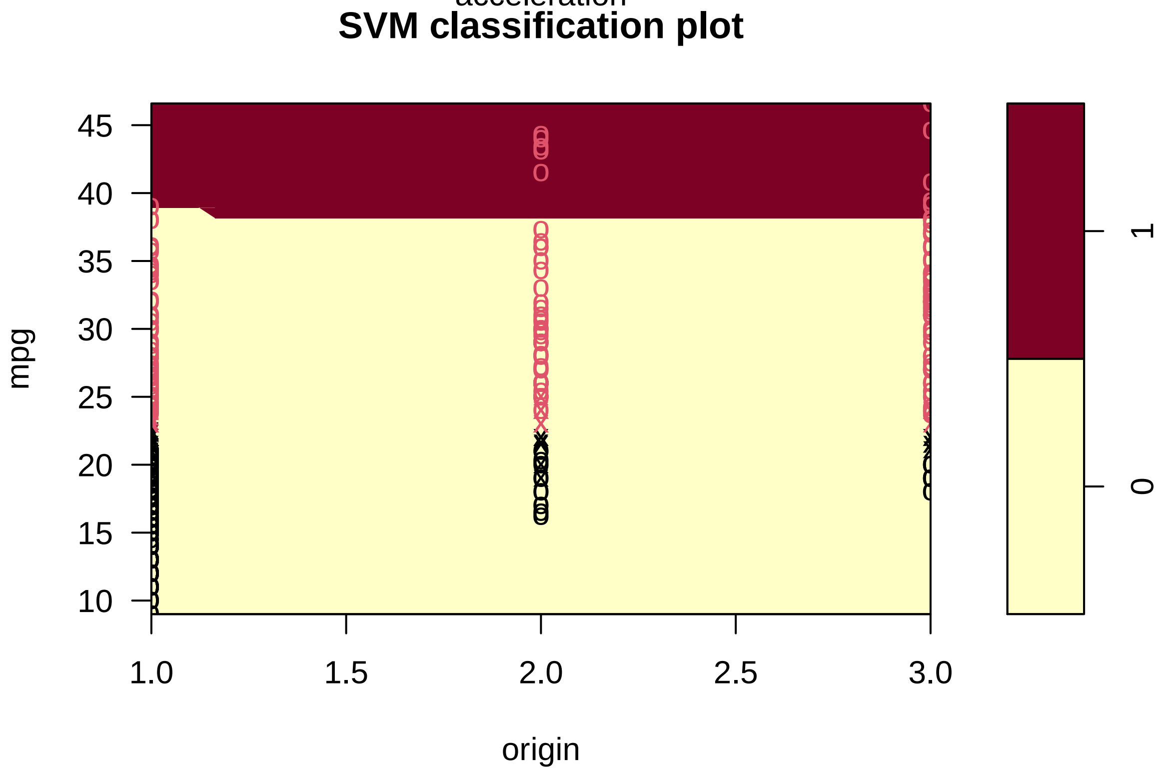



```r
pair_plot(linear)
```

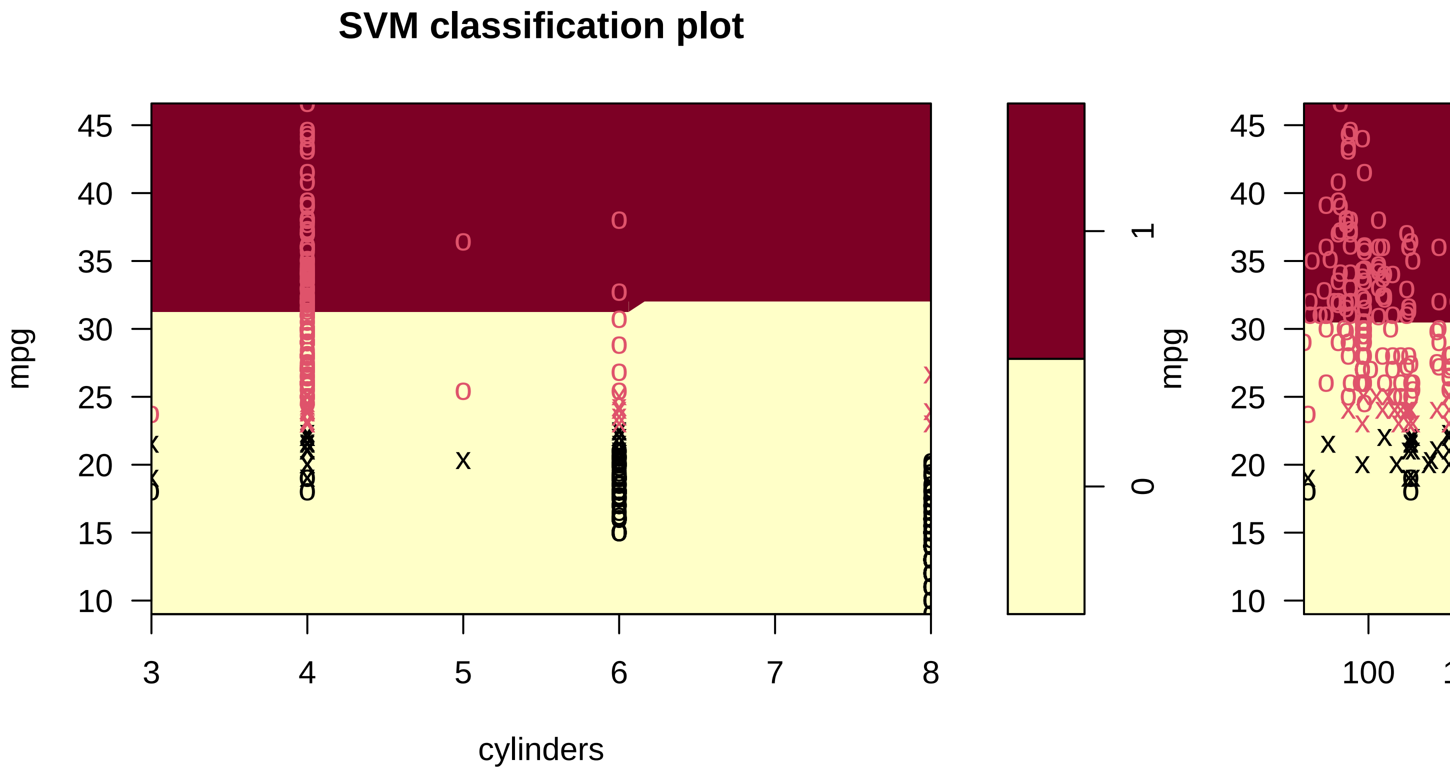

SVM classification plot
mpg
45
40
35
30
25
20
15
10
3
4
5
6
7
8
cylinders
1
0
mpg
100

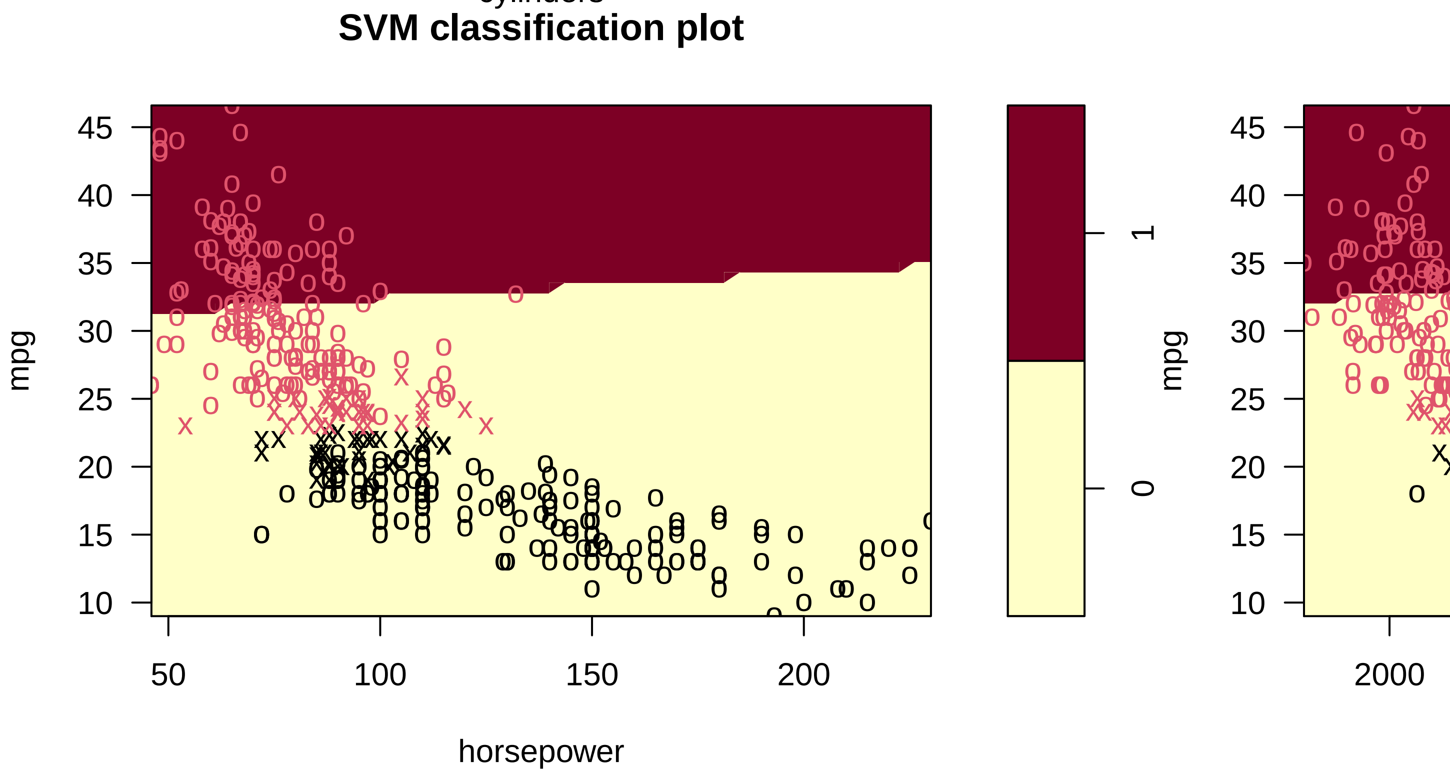

SVM classification plot
mpg
45
40
35
30
25
20
15
10
50
100
150
200
horsepower
1
0
mpg
2000

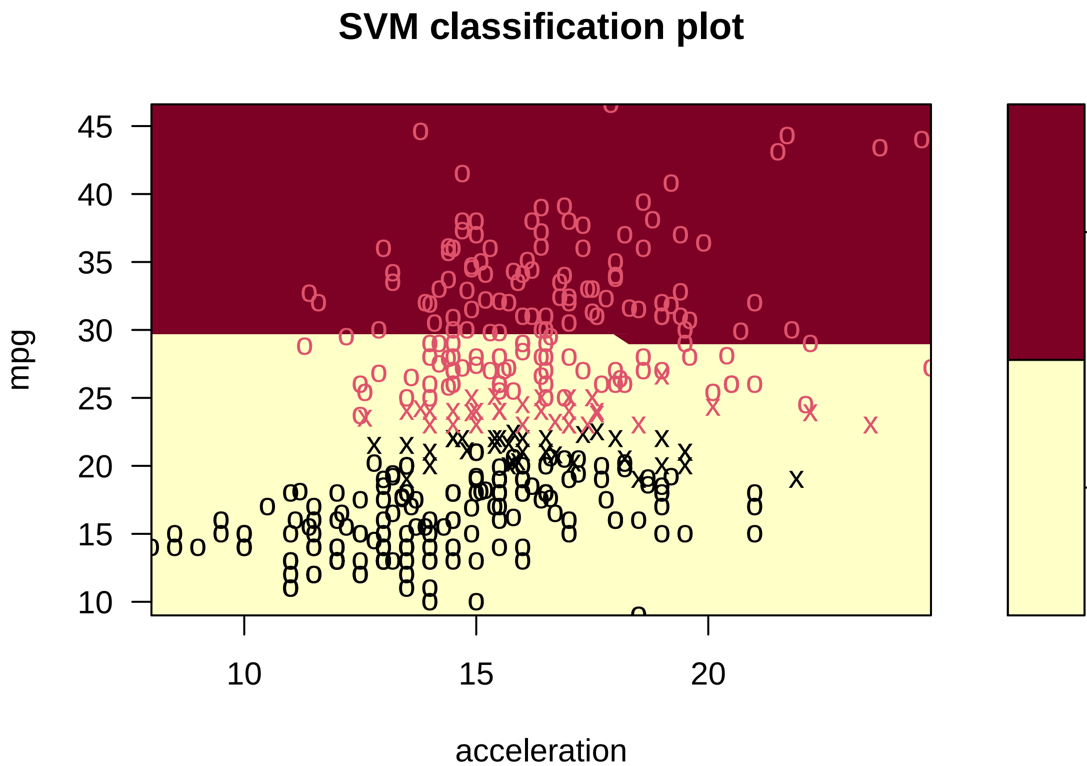


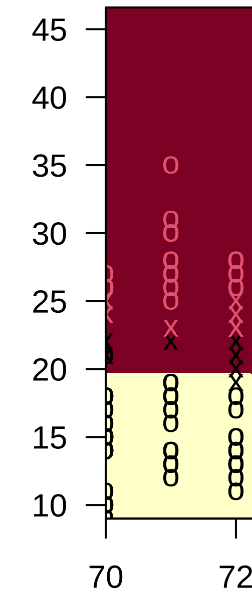


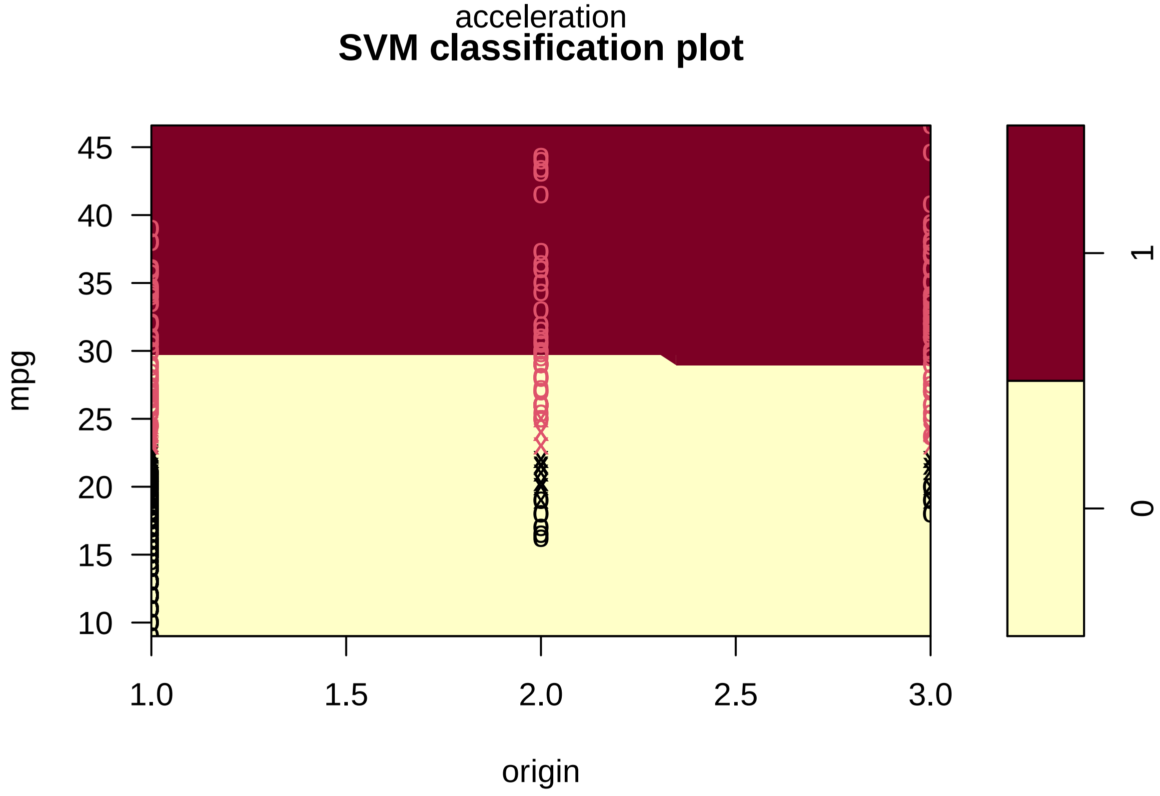

```
pair_plot(polynomial)
```


SVM classification plot
mpg
45
40
35
30
25
20
15
10
3
4
5
6
7
8
cylinders
1
0
100


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







9.8

##(a)

```r
data("OJ")
set.seed(9)
train_oj <- sample(nrow(OJ),800)
oj_train <- OJ[train_oj,]
oj_test <- OJ[-train_oj,]
```

##(b)

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

```
##
## 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:  426
##
##  ( 213 213 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

##(c)

```
#Training error rate
pred_train <- predict(oj_svc, oj_train)
table(pred_train, oj_train$Purchase)
```

```
##
## pred_train  CH  MM
##         CH 455  77
##         MM  50 218
```

```
#Test error rate
pred_test <- predict(oj_svc, oj_test)
table(pred_test, oj_test$Purchase)
```

```
##
## pred_test  CH  MM
##        CH 131  39
##        MM  17  83
```

```
(tr_error<- (70+61)/(428+70+61+241))
```

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

```
(te_error <- (33+21)/(143+33+21+73))
```

```
## [1] 0.2
```

##(d)

```
oj_tune <- tune(svm, Purchase~.,data=oj_train,kernel="linear",ranges = data.frame(cost=seq(0.01,10,leng
summary(oj_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  3.34
##
## - best performance: 0.1575
##
## - Detailed performance results:
##        cost   error dispersion
## 1   0.01000 0.16625 0.03175973
## 2   0.42625 0.16125 0.03557562
## 3   0.84250 0.16125 0.03408018
## 4   1.25875 0.16125 0.03197764
## 5   1.67500 0.16125 0.03408018
## 6   2.09125 0.16000 0.03670453
## 7   2.50750 0.15875 0.03586723
## 8   2.92375 0.15875 0.03729108
## 9   3.34000 0.15750 0.03593976
## 10  3.75625 0.15750 0.03593976
## 11  4.17250 0.15750 0.03593976
## 12  4.58875 0.15750 0.03593976
## 13  5.00500 0.15875 0.03586723
## 14  5.42125 0.15875 0.03586723
## 15  5.83750 0.15875 0.03586723
## 16  6.25375 0.15875 0.03586723
## 17  6.67000 0.15875 0.03586723
## 18  7.08625 0.15875 0.03586723
## 19  7.50250 0.15875 0.03586723
## 20  7.91875 0.15875 0.03586723
## 21  8.33500 0.15875 0.03586723
## 22  8.75125 0.15875 0.03586723
## 23  9.16750 0.15875 0.03586723
## 24  9.58375 0.15875 0.03586723
## 25 10.00000 0.15875 0.03586723
```

The optimal cost is 3.75625.

##(e)

```r
#Training error rate
oj_svm <- svm(Purchase ~ ., data = oj_train, kernel = "linear", cost = oj_tune$best.parameters$cost)
svm_train <- predict(oj_svm, oj_train)
table(svm_train, oj_train$Purchase)
```

```
##
## svm_train  CH  MM
##        CH 456  75
##        MM  49 220
```

```r
(tr_err <- (58+64)/(425+58+64+253))
```

```
## [1] 0.1525
```

```r
#Test error rate
svm_test <- predict(oj_svm, oj_test)
table(svm_test, oj_test$Purchase)
```

```
##
## svm_test  CH  MM
##       CH 131  35
##       MM  17  87
```

```r
(te_err <- (27+22)/(142+27+22+79))
```

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

##(f)

```r
## radial kernel
oj_radial <- svm(Purchase~., data = oj_train, kernel="radial")
summary(oj_radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj_train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  365
##
##  ( 187 178 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
## training error rate
radial_train <- predict(oj_radial,oj_train)
table(radial_train,oj_train$Purchase)
```

```
##
## radial_train  CH  MM
##           CH 466  73
##           MM  39 222
```

```r
## test error rate
radial_test <- predict(oj_radial,oj_test)
table(radial_test,oj_test$Purchase)
```

```
##
## radial_test  CH  MM
##          CH 132  37
##          MM  16  85
```

```r
#The training error rate is 14.5% and the test error rate is 18.89%.

## optimal cost
radial_tune <- tune(svm,Purchase~.,data=oj_train,kernel="radial",ranges = data.frame(cost=seq(0.01,10,le
summary(radial_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##     cost
##  1.25875
##
## - best performance: 0.155
##
## - Detailed performance results:
##        cost   error dispersion
## 1   0.01000 0.36875 0.04218428
## 2   0.42625 0.16125 0.02461509
## 3   0.84250 0.15500 0.02648375
## 4   1.25875 0.15500 0.03016160
## 5   1.67500 0.15500 0.03238227
## 6   2.09125 0.15625 0.03294039
## 7   2.50750 0.15750 0.03593976
## 8   2.92375 0.16125 0.03557562
## 9   3.34000 0.16000 0.03574602
## 10  3.75625 0.16125 0.03408018
## 11  4.17250 0.16250 0.03535534
## 12  4.58875 0.16625 0.03488573
## 13  5.00500 0.16750 0.03343734
## 14  5.42125 0.16625 0.03682259
## 15  5.83750 0.16625 0.03682259
## 16  6.25375 0.16625 0.03488573
## 17  6.67000 0.16625 0.03488573
## 18  7.08625 0.16625 0.03488573
## 19  7.50250 0.16750 0.03593976
## 20  7.91875 0.16875 0.03448530
## 21  8.33500 0.16875 0.03448530
## 22  8.75125 0.16875 0.03448530
## 23  9.16750 0.17000 0.03343734
## 24  9.58375 0.17000 0.03343734
## 25 10.00000 0.17125 0.03283481
```

```r
## training error rate
radial_svm <- svm(Purchase ~ ., data = oj_train, kernel = "radial", cost = radial_tune$best.parameters$c
svm_rad <- predict(radial_svm, oj_train)
table(svm_rad, oj_train$Purchase)
```

```
##
## svm_rad  CH  MM
##      CH 469  73
##      MM  36 222
```

```r
#Test error rate
svm_rad_test <- predict(radial_svm, oj_test)
table(svm_rad_test, oj_test$Purchase)
```

```
##
## svm_rad_test  CH  MM
##           CH 132  36
##           MM  16  86
```

```r
#The training error rate is 14.5% and the test error rate is 18.89%.
```

##(g)

```r
## polynomial kernel
oj_poly <- svm(Purchase~.,data=oj_train,kernel="polynomial",degree=2)
summary(oj_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:  431
##
##  ( 219 212 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

##(h) linear kernel with cost of 3.75625 has the best result.