ANLY512_HW9

Hongyang Zheng

2019/4/15

```
library(ISLR)
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.5.2
```

```
library(mlbench)
library("pROC")
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(randomForest)
```

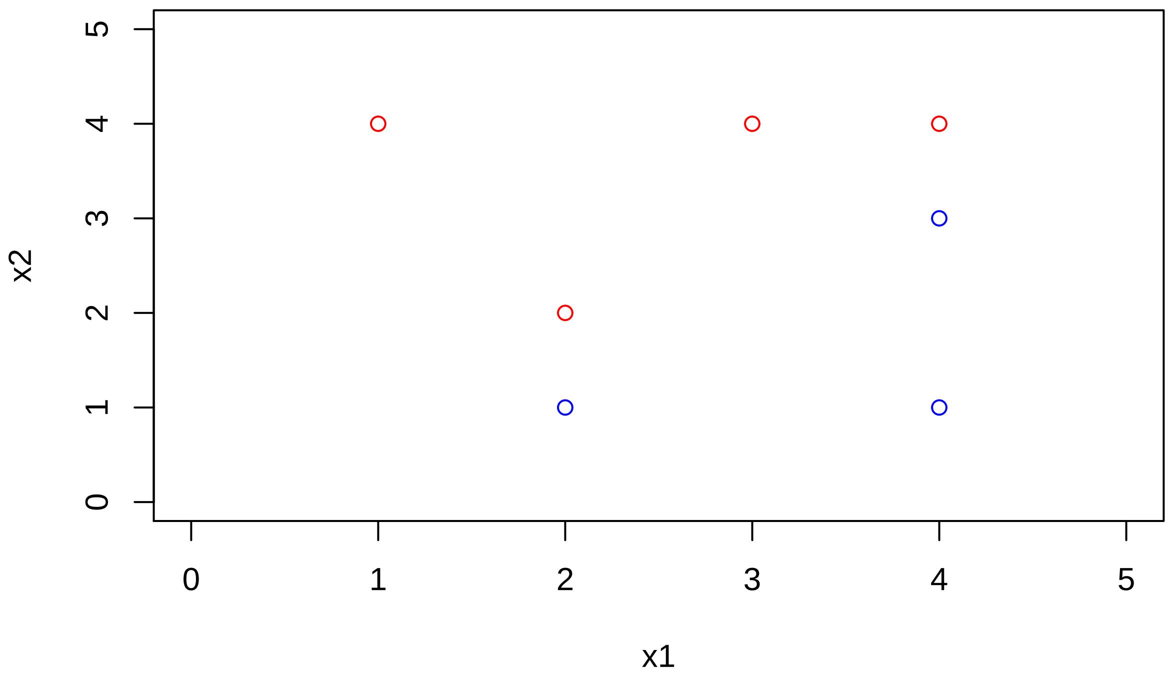
```
## randomForest 4.6-14
```

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

Problem #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)

The optimal separating hyperplane must between the points (2,2), (4,4), (2,1), (4,3).

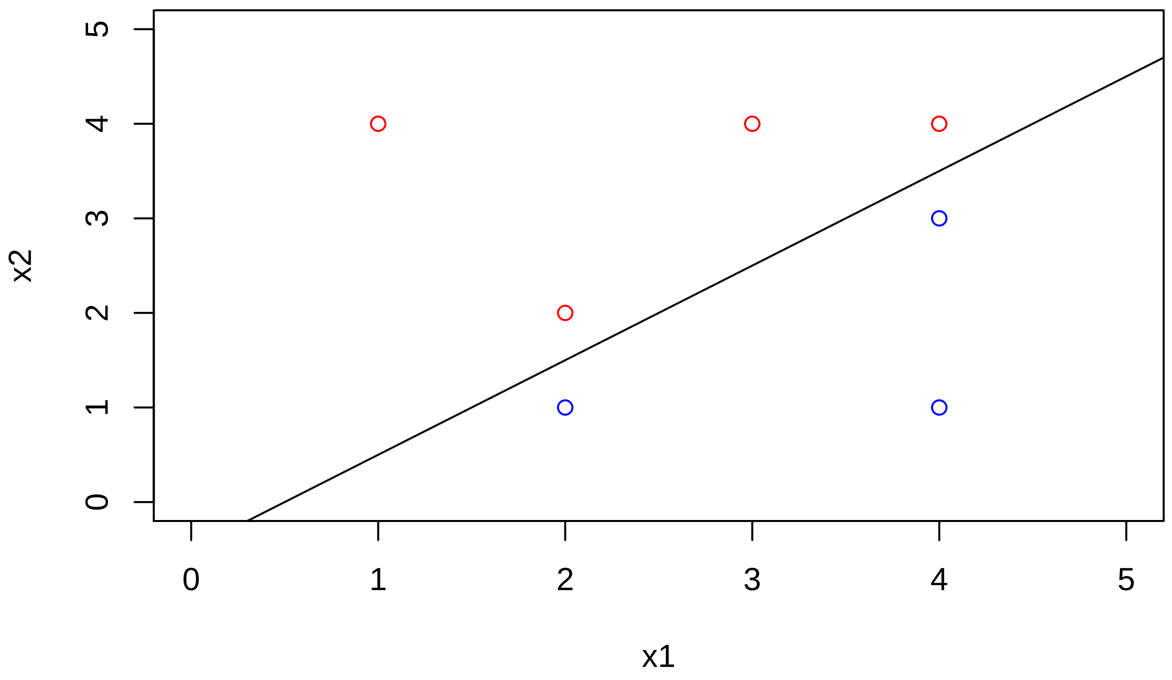
The average of the two sets points is (2,1.5), (4,3.5).

slope=(3.5-1.5)/(4-2)=1

intercept=1.5-2=-0.5

The hyperplane is 0.5-X1+X2=0

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

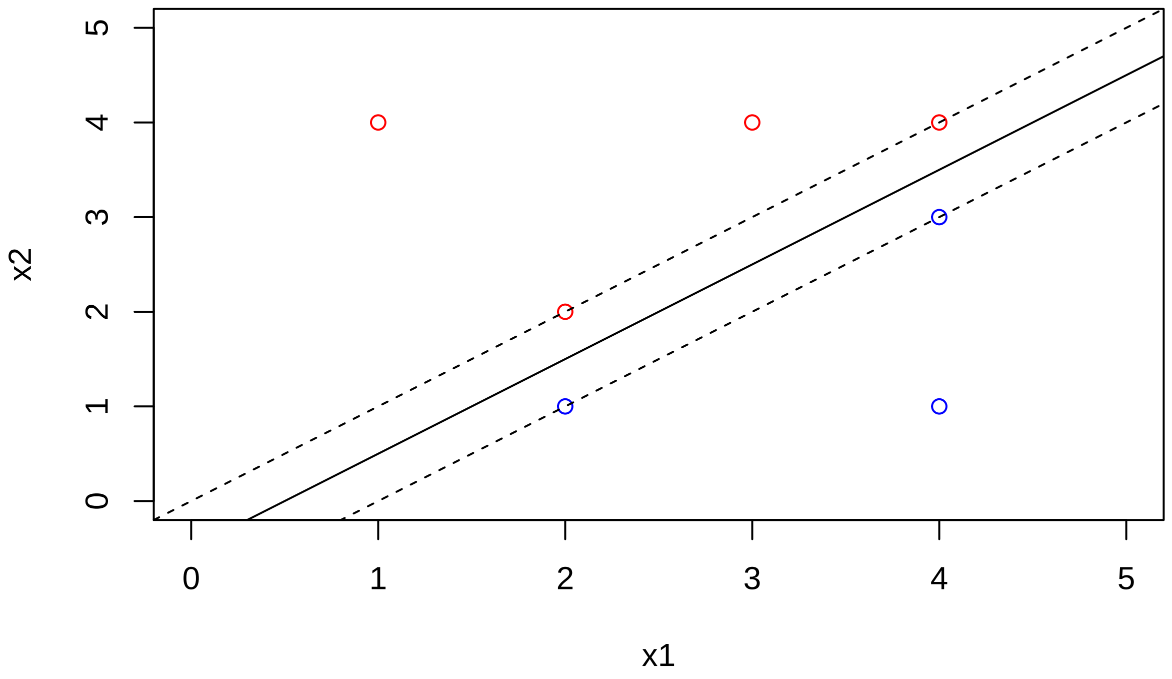


c)

Classify to red if 0.5-X1+X2>0, and classify to blue otherwise.

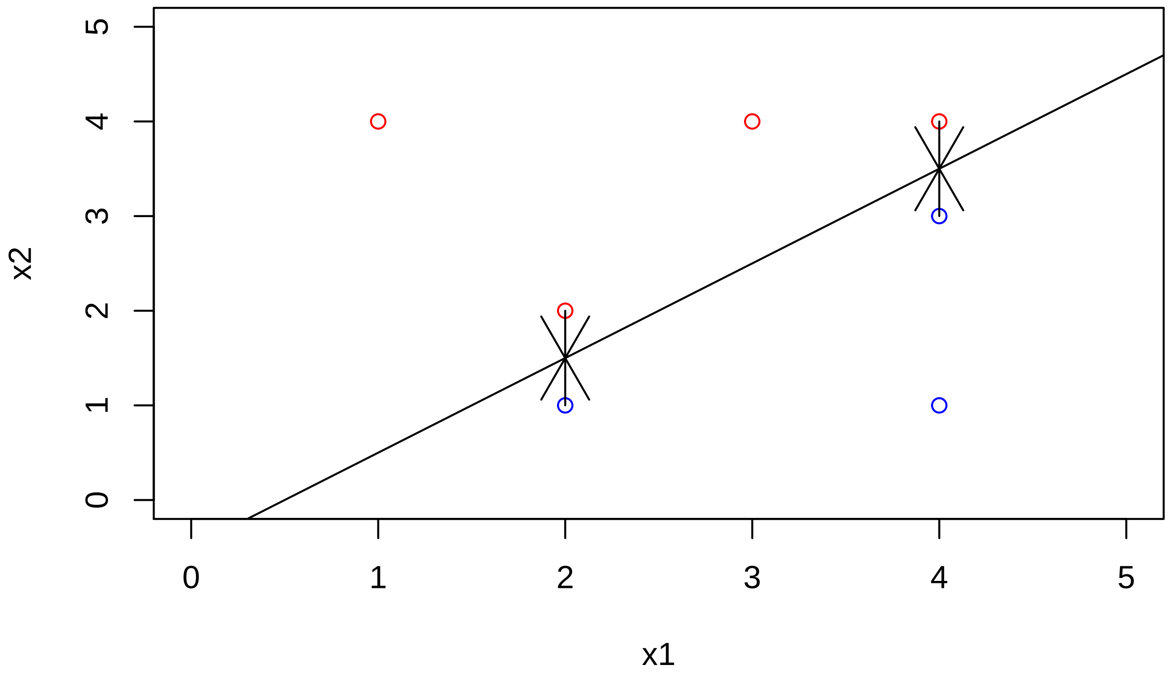
d)

```r
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)

```r
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(-0.5, 1)
arrows(2, 1, 2, 1.5)
arrows(2, 2, 2, 1.5)
arrows(4, 4, 4, 3.5)
arrows(4, 3, 4, 3.5)
```

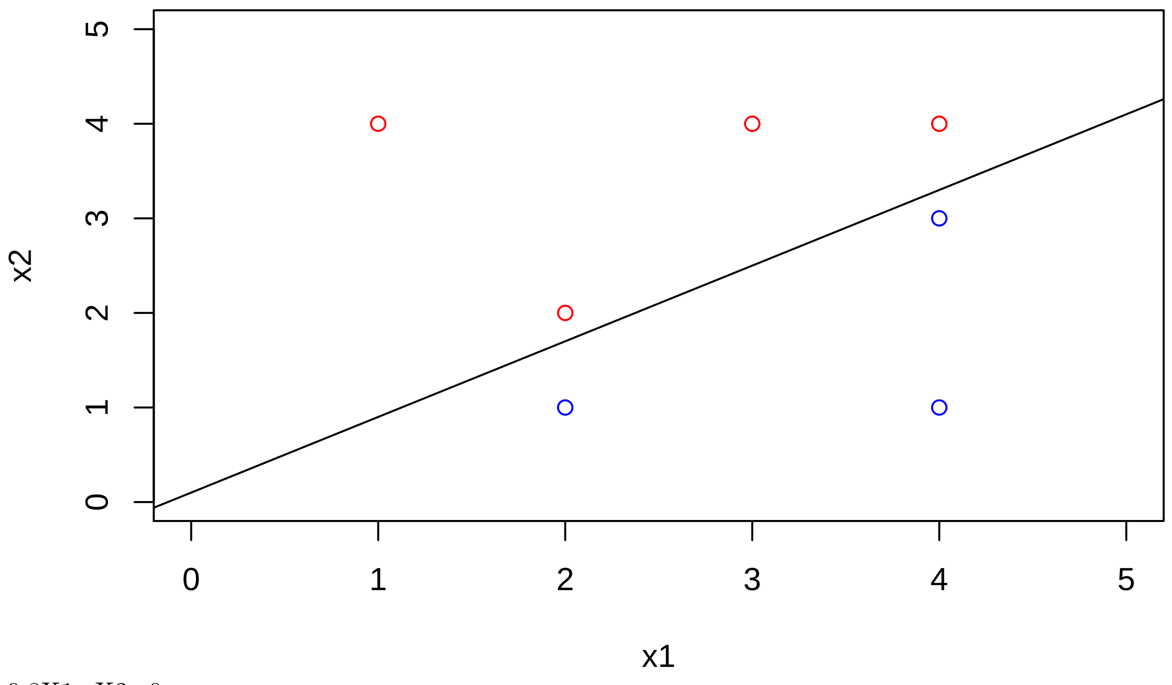


f)

A slight movement of observation #7 (not a support vector) (4,1) blue would not have an effect on the maximal margin hyperplane since its movement would be outside of the margin.

g)

```r
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
abline(0.1, 0.8)
```

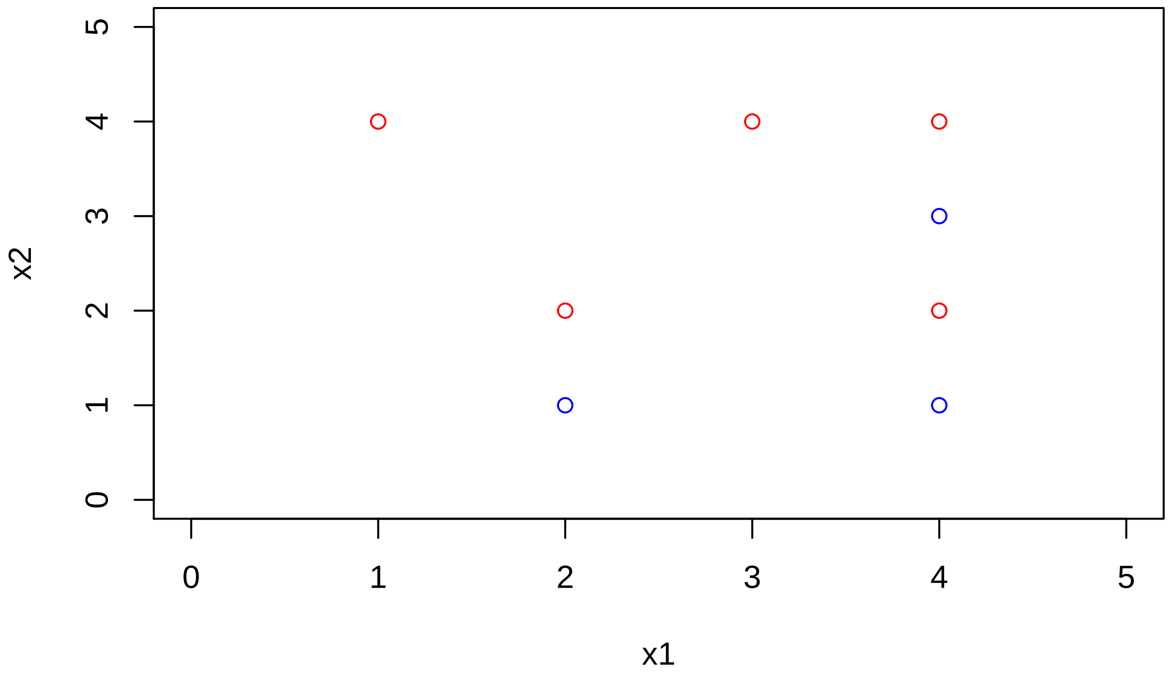


-0.1-

0.8X1+X2>0

h)

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



Problem #7

a)

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

b)

```r
set.seed(1234)
# Build svm with linear kernel with different cost
fit.linear = tune(svm, mpglevel ~ ., data = Auto, kernel = "linear",
                  ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 20, 50, 100)))
summary(fit.linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.01532051
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-02 0.07384615 0.04710732
## 2 1e-01 0.05602564 0.03751957
## 3 1e+00 0.01532051 0.01788871
## 4 5e+00 0.01532051 0.01788871
## 5 1e+01 0.02044872 0.01619554
## 6 2e+01 0.02814103 0.01893035
## 7 5e+01 0.03320513 0.01737168
## 8 1e+02 0.03320513 0.01737168
```

The lowest cross-validation error is gained when cost = 1 or 5, and the optimal cost is 1. With the cost increasing, the cross-validation error first decreases and then increases.

c)

```r
# Build svm with poly kernel with different cost and degree
fit.poly = tune(svm, mpglevel ~ ., data = Auto, kernel = "polynomial",
                ranges = list(cost = c(0.1, 1, 5, 10), degree = c(2, 3, 4, 5)))
summary(fit.poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##    10      2
##
## - best performance: 0.4844231
```

```
##
## - Detailed performance results:
##    cost degree     error dispersion
## 1   0.1      2 0.5458974 0.04287924
## 2   1.0      2 0.5458974 0.04287924
## 3   5.0      2 0.5458974 0.04287924
## 4  10.0      2 0.4844231 0.12292995
## 5   0.1      3 0.5383974 0.05626799
## 6   1.0      3 0.5383974 0.05626799
## 7   5.0      3 0.5383974 0.05626799
## 8  10.0      3 0.5383974 0.05626799
## 9   0.1      4 0.5458974 0.04287924
## 10  1.0      4 0.5458974 0.04287924
## 11  5.0      4 0.5458974 0.04287924
## 12 10.0      4 0.5458974 0.04287924
## 13  0.1      5 0.5458974 0.04287924
## 14  1.0      5 0.5458974 0.04287924
## 15  5.0      5 0.5458974 0.04287924
## 16 10.0      5 0.5458974 0.04287924
```

The lowest cross-validation error is gained when cost = 10 and degree = 2. With degree = 2 increasing cost leads to the cross-validation error decrease; while when degree= 3, 4, 5, increasing cost does not have an obvious impact on decreasing the error.

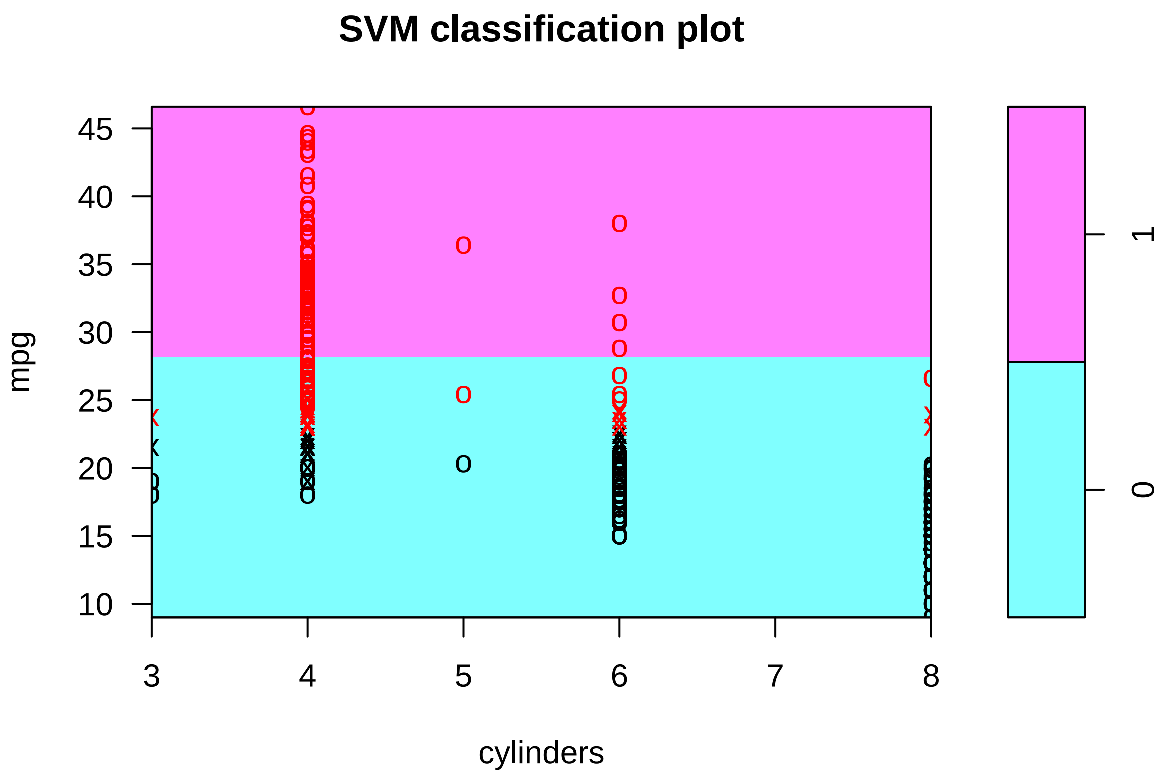
```
# Build svm with radial kernel with different cost and gamma
fit.radial = tune(svm, mpglevel ~ ., data = Auto, kernel = "radial",
                  ranges = list(cost = c(0.1, 1, 5, 10), gamma = c(0.01, 0.1, 1, 5, 10)))
summary(fit.radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.1
##
## - best performance: 0.02038462
##
## - Detailed performance results:
##    cost gamma      error dispersion
## 1   0.1  0.01 0.08423077 0.03636273
## 2   1.0  0.01 0.07147436 0.03571286
## 3   5.0  0.01 0.04858974 0.03724672
## 4  10.0  0.01 0.02044872 0.02354784
## 5   0.1  0.10 0.07660256 0.03187594
## 6   1.0  0.10 0.05628205 0.03597976
## 7   5.0  0.10 0.02551282 0.02402645
## 8  10.0  0.10 0.02038462 0.02001214
## 9   0.1  1.00 0.57641026 0.05372884
## 10  1.0  1.00 0.06647436 0.02498091
## 11  5.0  1.00 0.06134615 0.03871008
## 12 10.0  1.00 0.06134615 0.03871008
## 13  0.1  5.00 0.57641026 0.05372884
## 14  1.0  5.00 0.51275641 0.06179792
## 15  5.0  5.00 0.50762821 0.06388457
## 16 10.0  5.00 0.50762821 0.06388457
## 17  0.1 10.00 0.57641026 0.05372884
## 18  1.0 10.00 0.54076923 0.06527363
## 19  5.0 10.00 0.53820513 0.06805695
## 20 10.0 10.00 0.53820513 0.06805695
```

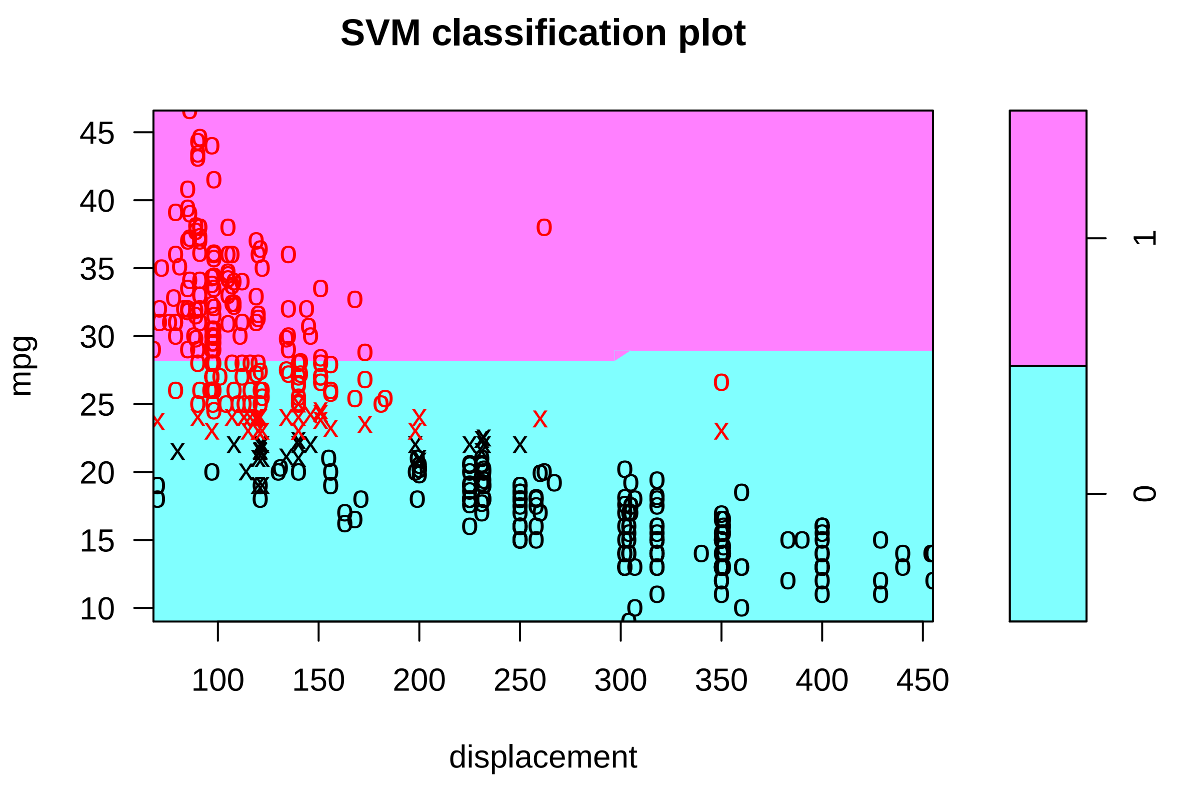
The lowest cross-validation error is gained when cost $= 10$ and gamma $= 0.1$. With gamma fixed, increasing cost leads to the cross-validation error decrease.

d)

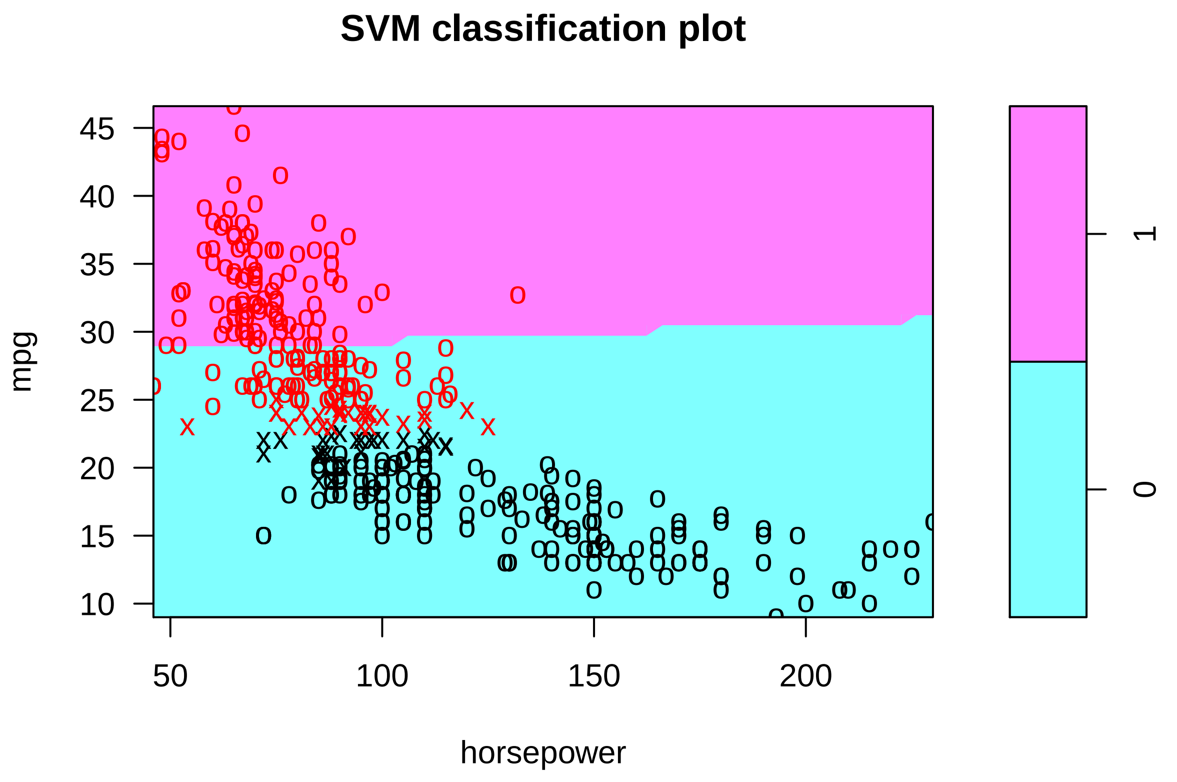
```
# Use the best parameters to build the models
svm.linear = svm(mpglevel ~ ., data = Auto, kernel = "linear", cost = 1)
svm.poly = svm(mpglevel ~ ., data = Auto, kernel = "polynomial", cost = 10, degree = 2)
svm.radial = svm(mpglevel ~ ., data = Auto, kernel = "radial", cost = 10, gamma = 0.1)

# Make plot for linear kernel
# random select 4 variables
plot(svm.linear, Auto, mpg~cylinders)
```



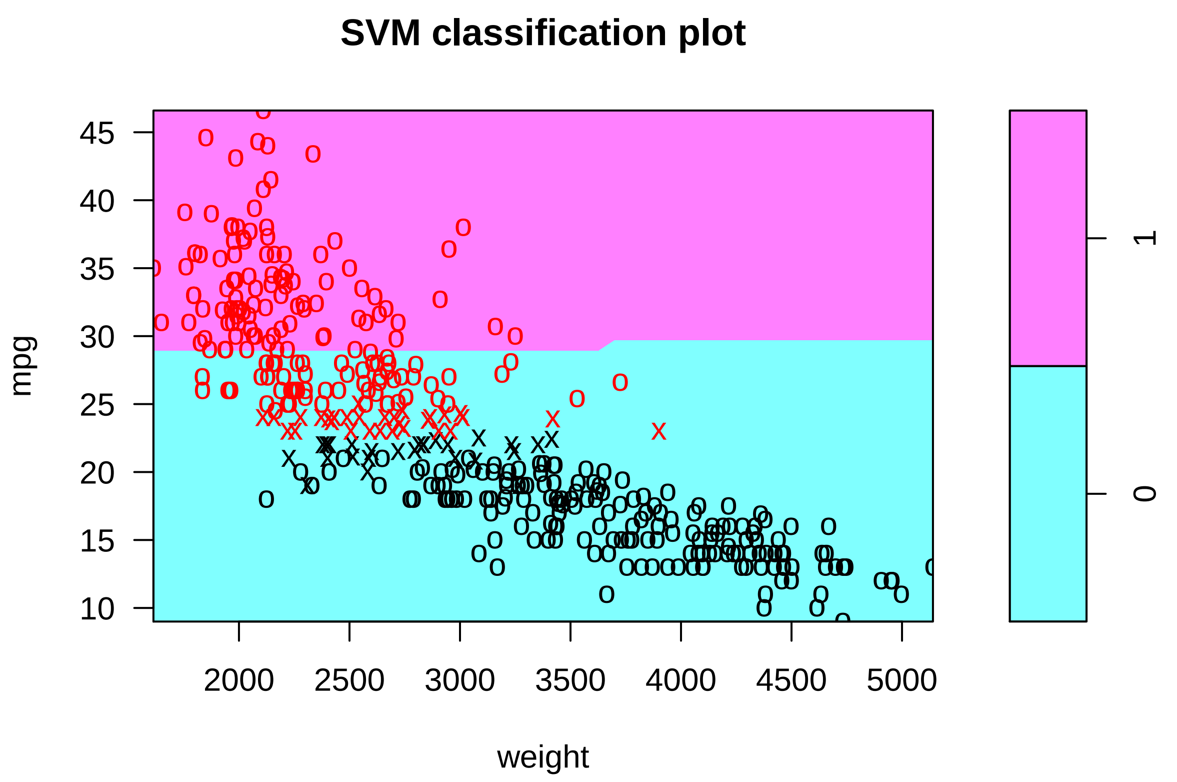
```
plot(svm.linear, Auto, mpg~displacement)
```



```r
plot(svm.linear, Auto, mpg~horsepower)
```

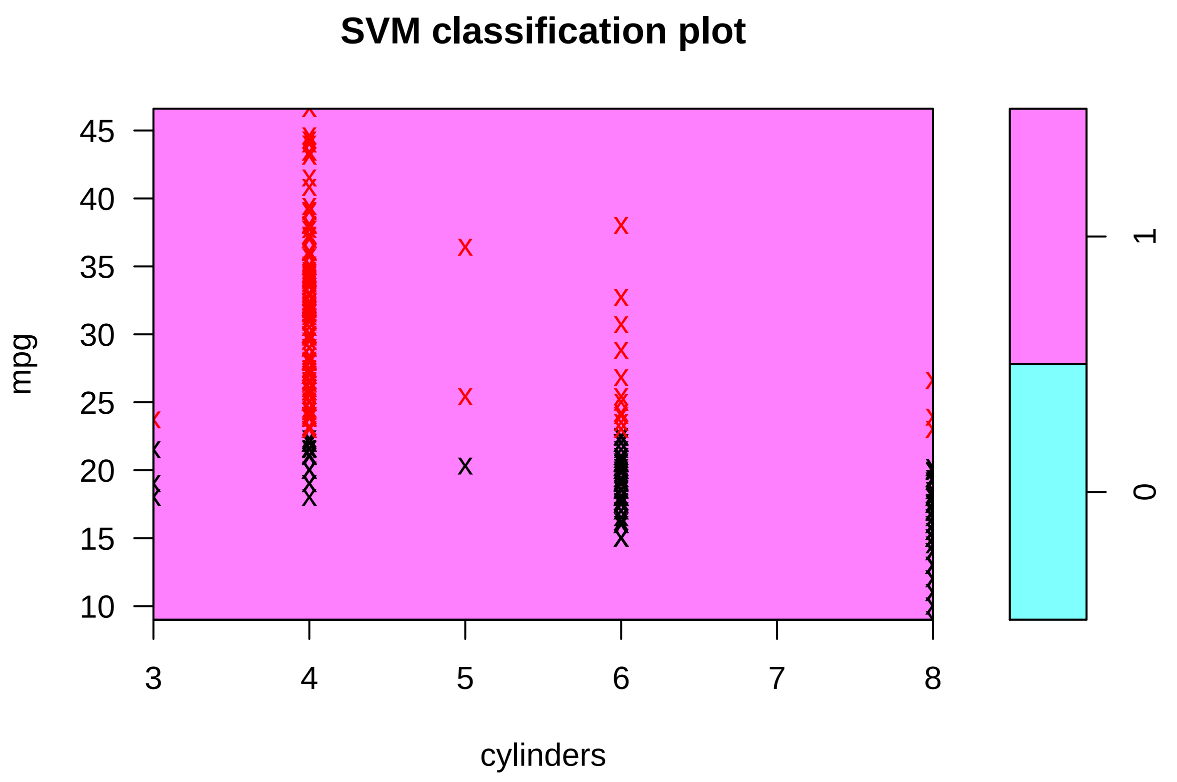


```r
plot(svm.linear, Auto, mpg~weight)
```

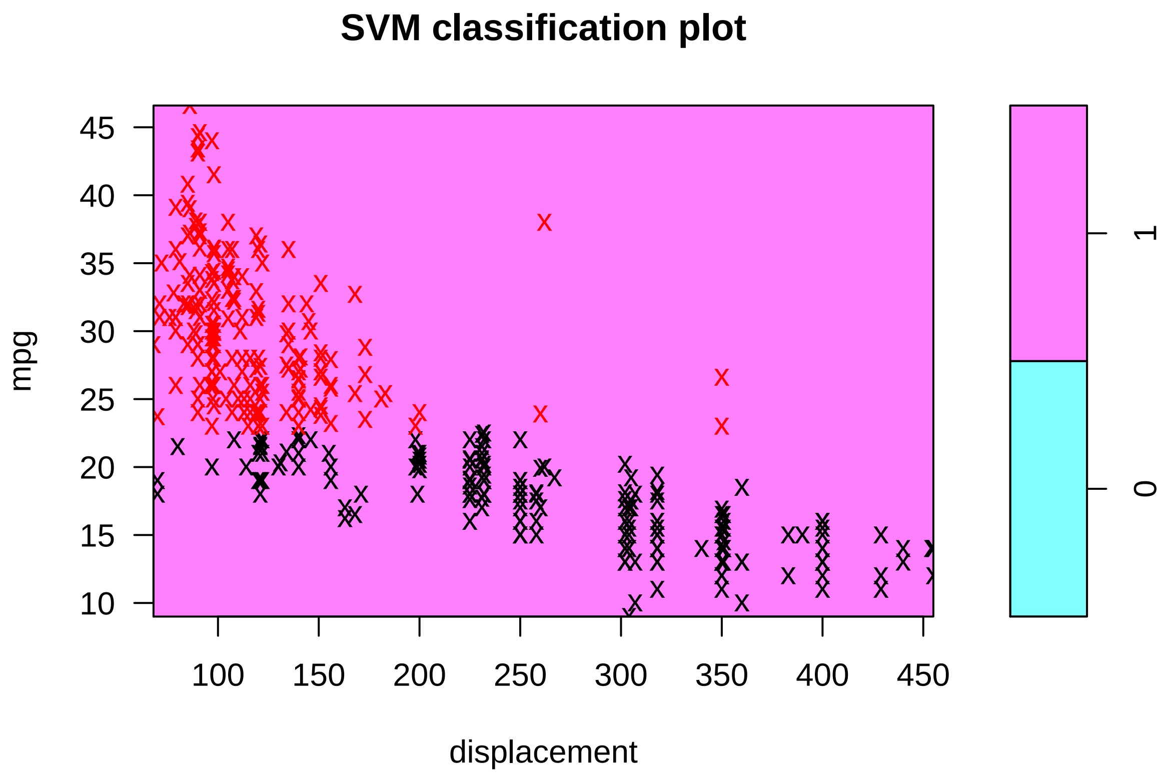


We can see that this model overall does a good job for separating two classes (two colors).

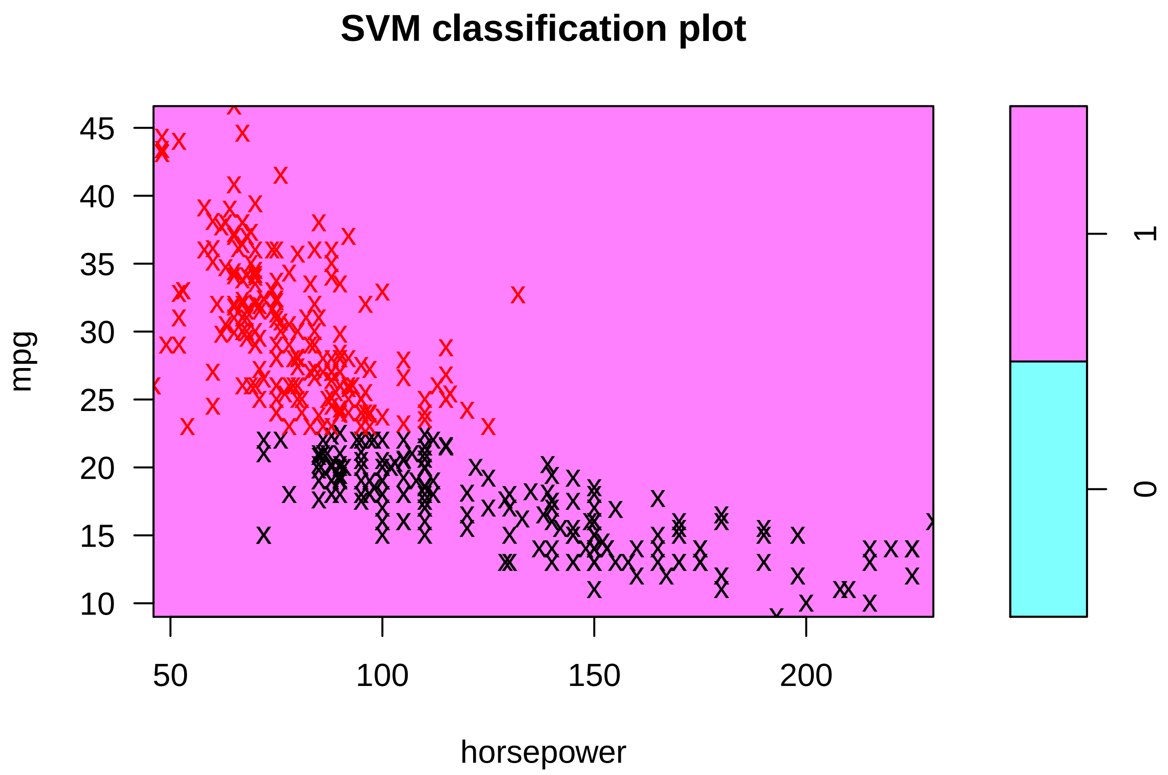
```r
# Make plot for poly kernel
# random select 4 variables
plot(svm.poly, Auto, mpg~cylinders)
```



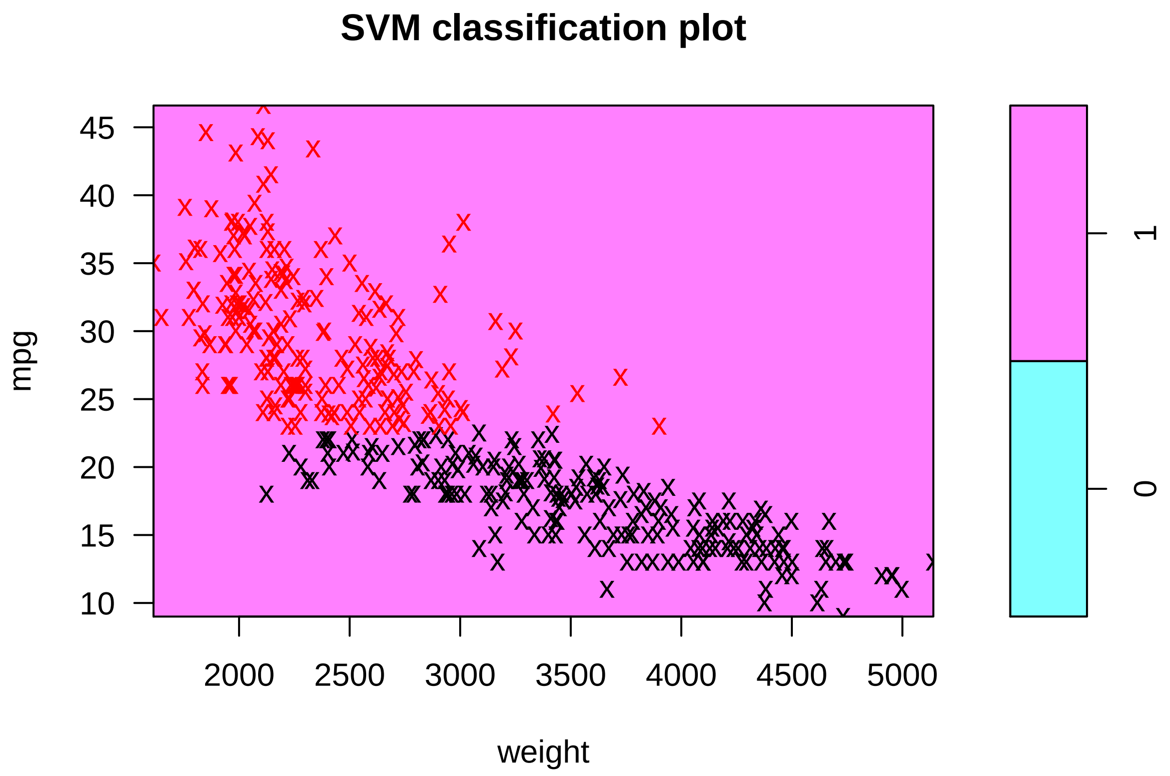
```r
plot(svm.poly, Auto, mpg~displacement)
```



```r
plot(svm.poly, Auto, mpg~horsepower)
```

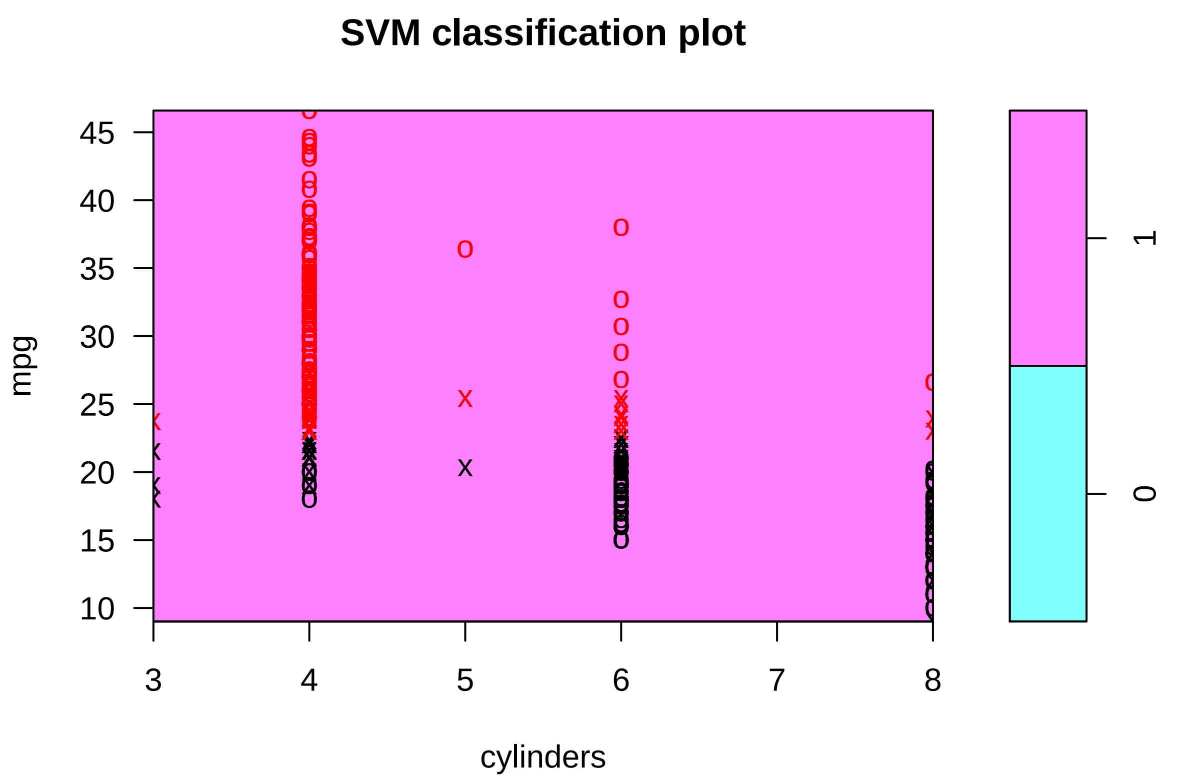


```r
plot(svm.poly, Auto, mpg~weight)
```

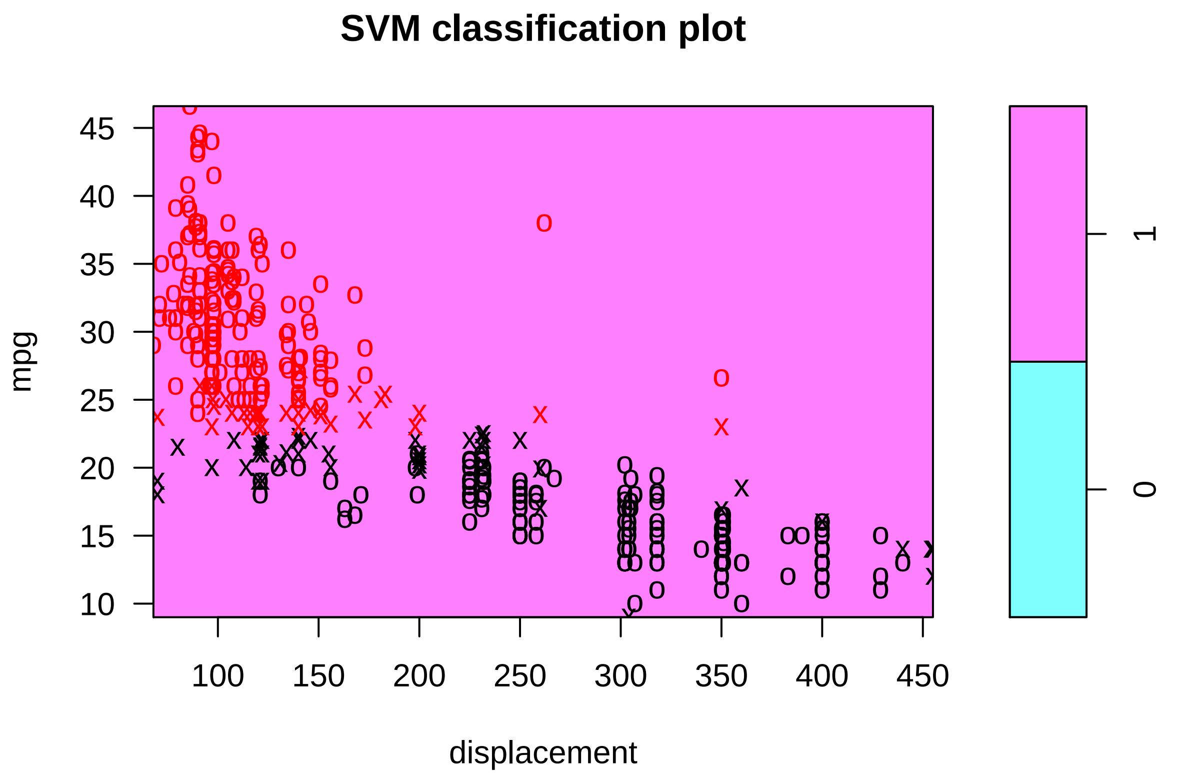


This model with poly kernel does not perform good compared to linear kernel. The error for poly kernel is also much higher than the error for linear kernel.

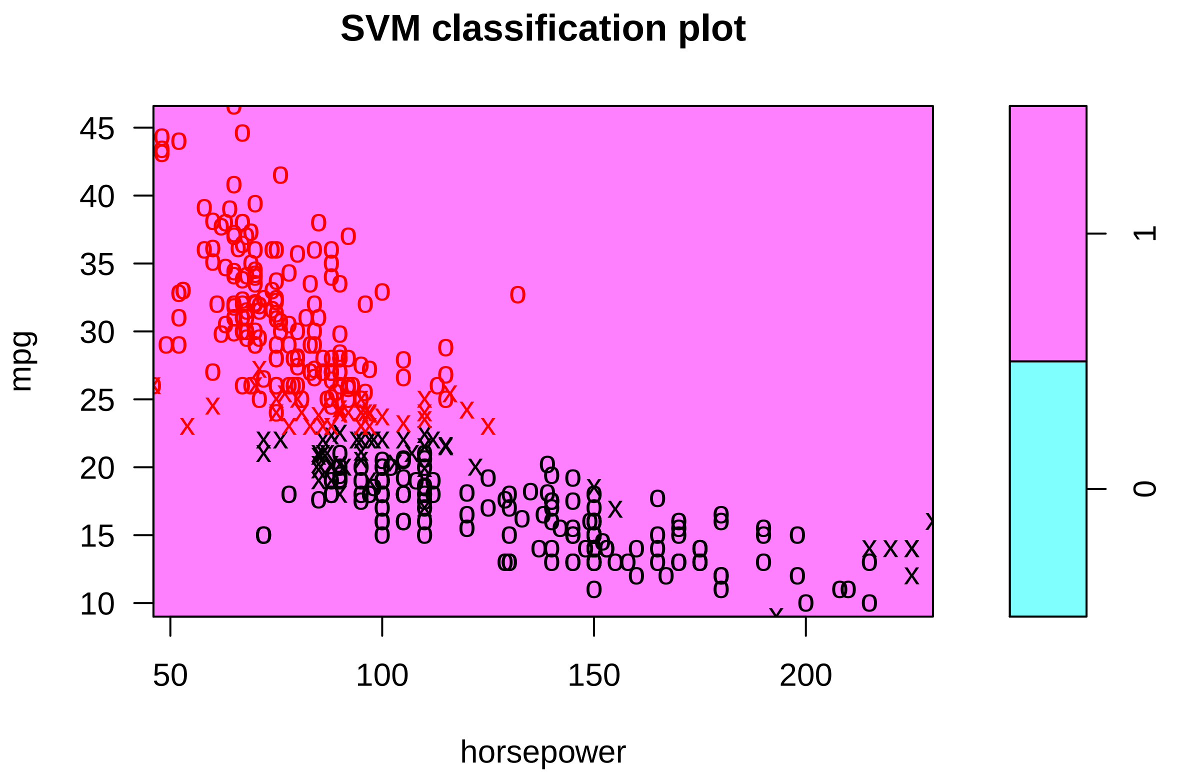
```r
# Make plot for radial kernel
# random select 4 variables
plot(svm.radial, Auto, mpg~cylinders)
```



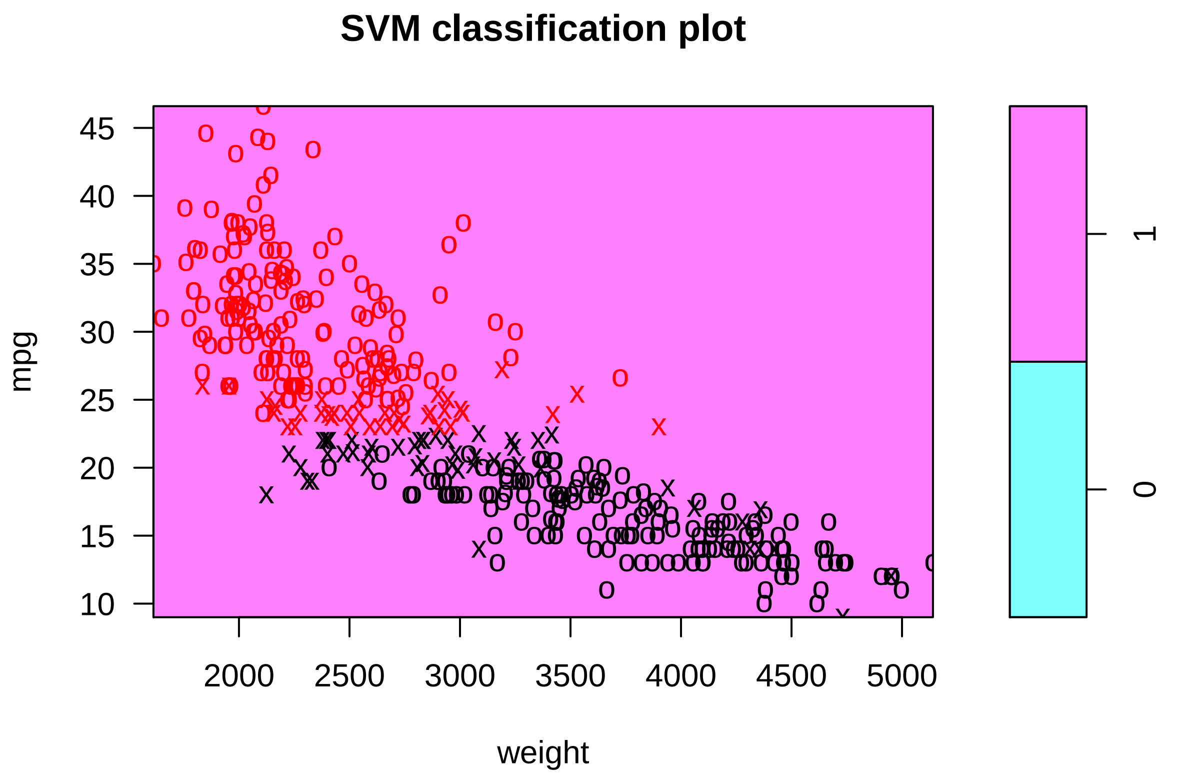
```r
plot(svm.radial, Auto, mpg~displacement)
```



```r
plot(svm.radial, Auto, mpg~horsepower)
```



```
plot(svm.radial, Auto, mpg~weight)
```



This model with radial kernel does not perform good compared to linear kernel, but is similar to poly kernal. The error for radial kernel is also much higher than the error for linear kernel.

Problem #8

a)

```r
set.seed(2345)

# Split the data into train and test
train = sample(dim(OJ)[1], 800)
OJ.train = OJ[train, ]
OJ.test = OJ[-train, ]
```

b)

```r
fit.cost.0.01=svm(Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.01)
summary(fit.cost.0.01)
```

```
##
## 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:  449
##
##  ( 223 226 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

There are 449 support vectors out of 800 observations, and 223 vectors belong to CH, and 226 vectors belong to MM.

c)

```r
# Training error
train.pred.1 = predict(fit.cost.0.01, OJ.train)
table(OJ.train$Purchase, train.pred.1)
```

```
##     train.pred.1
##       CH  MM
##   CH 430  57
##   MM  85 228
```

```r
train.error.1 = (85+57)/800
```

The training error is 0.1775.

```r
# Test error
test.pred.1 = predict(fit.cost.0.01, OJ.test)
table(OJ.test$Purchase, test.pred.1)
```

```
##     test.pred.1
##       CH  MM
##   CH 151  15
##   MM  19  85
```

```r
test.error.1 = (19+15)/270
```

The test error is 0.1259259.

d)

```r
tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear",
                ranges = list(cost = 10^seq(-2, 1, by = 0.1)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  7.943282
##
## - best performance: 0.175
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.18500 0.05027701
## 2   0.01258925 0.18375 0.04825065
## 3   0.01584893 0.18500 0.04518481
## 4   0.01995262 0.18250 0.04721405
## 5   0.02511886 0.18500 0.04743416
## 6   0.03162278 0.18625 0.04619178
## 7   0.03981072 0.18750 0.04750731
## 8   0.05011872 0.18125 0.04903584
## 9   0.06309573 0.18250 0.04758034
## 10  0.07943282 0.18250 0.04684490
## 11  0.10000000 0.18125 0.04796194
## 12  0.12589254 0.18500 0.04594683
## 13  0.15848932 0.18375 0.04489571
## 14  0.19952623 0.18375 0.04489571
## 15  0.25118864 0.18125 0.04379958
## 16  0.31622777 0.18250 0.04216370
## 17  0.39810717 0.18000 0.04377975
## 18  0.50118723 0.18125 0.04177070
## 19  0.63095734 0.18125 0.04177070
## 20  0.79432823 0.18125 0.04177070
## 21  1.00000000 0.18125 0.04177070
## 22  1.25892541 0.18000 0.04338138
## 23  1.58489319 0.18125 0.04218428
## 24  1.99526231 0.18000 0.04005205
## 25  2.51188643 0.17875 0.03998698
## 26  3.16227766 0.18125 0.03875224
## 27  3.98107171 0.17875 0.03998698
## 28  5.01187234 0.17750 0.04116363
## 29  6.30957344 0.17625 0.04308019
## 30  7.94328235 0.17500 0.04289846
## 31 10.00000000 0.17625 0.04226652
```

The optimal cost is 7.943282

e)

```r
# Build a new model using the optimal cost
fit.new.cost = svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = 7.943282)

# Training error
train.pred.2 = predict(fit.new.cost, OJ.train)
table(OJ.train$Purchase, train.pred.2)
```

```
##     train.pred.2
##       CH  MM
##   CH 424  63
##   MM  81 232
```

```r
train.error.2 = (81+63)/800

# Test error
test.pred.2 = predict(fit.new.cost, OJ.test)
table(OJ.test$Purchase, test.pred.2)
```

```
##     test.pred.2
##       CH  MM
##   CH 149  17
##   MM  16  88
```

```r
test.error.2 = (16+17)/270
```

With the new cost, the training error increases from 0.1775 to 0.18, while the test error decreases slightly from 0.1259 to 0.12222.

f)

```r
# Using a radial kernel with cost = 0.01 and gamma = default
fit.radial=svm(Purchase ~ ., data = OJ.train, kernel = "radial", cost = 0.01)
summary(fit.radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  630
##
##  ( 313 317 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# Training error
train.radial = predict(fit.radial, OJ.train)
table(OJ.train$Purchase, train.radial)
```

```
##     train.radial
##       CH  MM
##   CH 487   0
##   MM 313   0
```

```
train.error.r = (313)/800

# Test error
test.radial = predict(fit.radial, OJ.test)
table(OJ.test$Purchase, test.radial)
```

```
##     test.radial
##       CH  MM
##   CH 166   0
##   MM 104   0
```

```
test.error.r = (104)/270
```

In this situation, the error for both training data and test data are high. All observations are classified as CH.

```
# Find the optimal cost
tune.out.r = tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial",
                 ranges = list(cost = 10^seq(-2, 1, by = 0.1)))
summary(tune.out.r)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.185
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39125 0.05337563
## 2   0.01258925 0.39125 0.05337563
## 3   0.01584893 0.39125 0.05337563
## 4   0.01995262 0.39125 0.05337563
## 5   0.02511886 0.39125 0.05337563
## 6   0.03162278 0.37750 0.05797509
## 7   0.03981072 0.29750 0.05230785
## 8   0.05011872 0.23750 0.03726780
## 9   0.06309573 0.21250 0.03486083
## 10  0.07943282 0.20375 0.03682259
## 11  0.10000000 0.19750 0.04241004
## 12  0.12589254 0.19500 0.04417453
## 13  0.15848932 0.19250 0.04456581
## 14  0.19952623 0.18750 0.04487637
## 15  0.25118864 0.18625 0.04387878
## 16  0.31622777 0.18875 0.04226652
## 17  0.39810717 0.18750 0.04289846
## 18  0.50118723 0.18750 0.04894725
## 19  0.63095734 0.18750 0.04750731
## 20  0.79432823 0.18625 0.04693746
## 21  1.00000000 0.18500 0.04779877
## 22  1.25892541 0.18625 0.04466309
## 23  1.58489319 0.18625 0.04267529
## 24  1.99526231 0.19000 0.04158325
## 25  2.51188643 0.18875 0.04656611
## 26  3.16227766 0.19000 0.04594683
## 27  3.98107171 0.18875 0.04693746
## 28  5.01187234 0.18875 0.04656611
## 29  6.30957344 0.18875 0.04543387
## 30  7.94328235 0.19250 0.04972145
## 31 10.00000000 0.19250 0.05277047
```

```r
# Build a new model using the optimal cost
fit.new.cost.2 = svm(Purchase ~ ., kernel = "radial", data = OJ.train,
                     cost = tune.out.r$best.parameters$cost)

# Training error
train.radial.2 = predict(fit.new.cost.2, OJ.train)
table(OJ.train$Purchase, train.radial.2)
```

```
##     train.radial.2
##       CH  MM
##   CH 444  43
##   MM  82 231
```

```r
train.error.r2 = (82+43)/800

# Test error
test.radial.2 = predict(fit.new.cost.2, OJ.test)
table(OJ.test$Purchase, test.radial.2)
```

```
##     test.radial.2
##       CH  MM
##   CH 153  13
##   MM  22  82
```

```r
test.error.r2 = (22+13)/270
```

The optimal cost is 1, and the train error is 0.15625, which is lower than using the linear kernel. The test error is 0.1296296, which is a little higher than using the linear kernel.

g)

```r
# Using a polynomial kernel with cost = 0.01, degree = 2
fit.poly=svm(Purchase ~ ., data = OJ.train, kernel = "poly", cost = 0.01, degree=2)
summary(fit.poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "poly",
##     cost = 0.01, degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  630
##
##  ( 313 317 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
# Training error
train.poly = predict(fit.poly, OJ.train)
table(OJ.train$Purchase, train.poly)
```

```
##     train.poly
##       CH  MM
##   CH 484   3
##   MM 296  17
```

```r
train.error.p = (296+3)/800

# Test error
test.poly = predict(fit.poly, OJ.test)
table(OJ.test$Purchase, test.poly)
```

```
##     test.poly
##       CH  MM
##   CH 162   4
##   MM  95   9
```

```r
test.error.p = (95+4)/270
```

In this situation, the error for training and test is also very high.

```r
tune.out.p = tune(svm, Purchase ~ ., data = OJ.train, kernel = "poly",
                ranges = list(cost = 10^seq(-2, 1, by = 0.1)), degree=2)
summary(tune.out.p)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  7.943282
##
## - best performance: 0.19125
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39000 0.05737305
## 2   0.01258925 0.37125 0.05172376
## 3   0.01584893 0.37125 0.05172376
## 4   0.01995262 0.36625 0.04860913
## 5   0.02511886 0.36500 0.04816061
## 6   0.03162278 0.36375 0.05015601
## 7   0.03981072 0.35375 0.04896498
## 8   0.05011872 0.34250 0.04533824
## 9   0.06309573 0.33250 0.04216370
## 10  0.07943282 0.33375 0.03682259
## 11  0.10000000 0.32375 0.03304563
## 12  0.12589254 0.32125 0.03283481
## 13  0.15848932 0.30000 0.04082483
## 14  0.19952623 0.27125 0.03821086
## 15  0.25118864 0.24750 0.02751262
## 16  0.31622777 0.22875 0.02949223
## 17  0.39810717 0.22250 0.02751262
## 18  0.50118723 0.21625 0.03007514
## 19  0.63095734 0.21250 0.03061862
## 20  0.79432823 0.20750 0.03343734
## 21  1.00000000 0.20750 0.03184162
## 22  1.25892541 0.20375 0.03064696
## 23  1.58489319 0.19250 0.02776389
## 24  1.99526231 0.19750 0.02934469
## 25  2.51188643 0.19750 0.03050501
## 26  3.16227766 0.20500 0.02713137
## 27  3.98107171 0.20250 0.02751262
## 28  5.01187234 0.19375 0.02960973
## 29  6.30957344 0.19250 0.03343734
## 30  7.94328235 0.19125 0.03682259
## 31 10.00000000 0.19375 0.04177070
```

```r
# Build a new model using the optimal cost
fit.new.cost.3 = svm(Purchase ~ ., kernel = "poly", data = OJ.train,
                     cost = tune.out.p$best.parameters$cost)

# Training error
train.poly.2 = predict(fit.new.cost.3, OJ.train)
table(OJ.train$Purchase, train.poly.2)
```

```
##     train.poly.2
##       CH  MM
##   CH 453  34
##   MM  84 229
```

```
train.error.p2 = (85+35)/800

# Test error
test.poly.2 = predict(fit.new.cost.3, OJ.test)
table(OJ.test$Purchase, test.poly.2)
```

```
##     test.poly.2
##       CH  MM
##   CH 152  14
##   MM  26  78
```

```
test.error.p2 = (26+15)/270
```

The optimal cost is 7.943282, and the train error is 0.15, which is lower than using the linear kernel and radial kernel. The test error is 0.1518519, which is much higher than using the linear kernel.

h)

Overall, polynomial basis kernel seems to be producing minimum misclassification error on train data, while linear basis kernel seems to be producing minimum misclassification error on test data. Therefore, I think the linear basis kernel still gives the best result, since it has the minimum test error.

Extra 63

a)

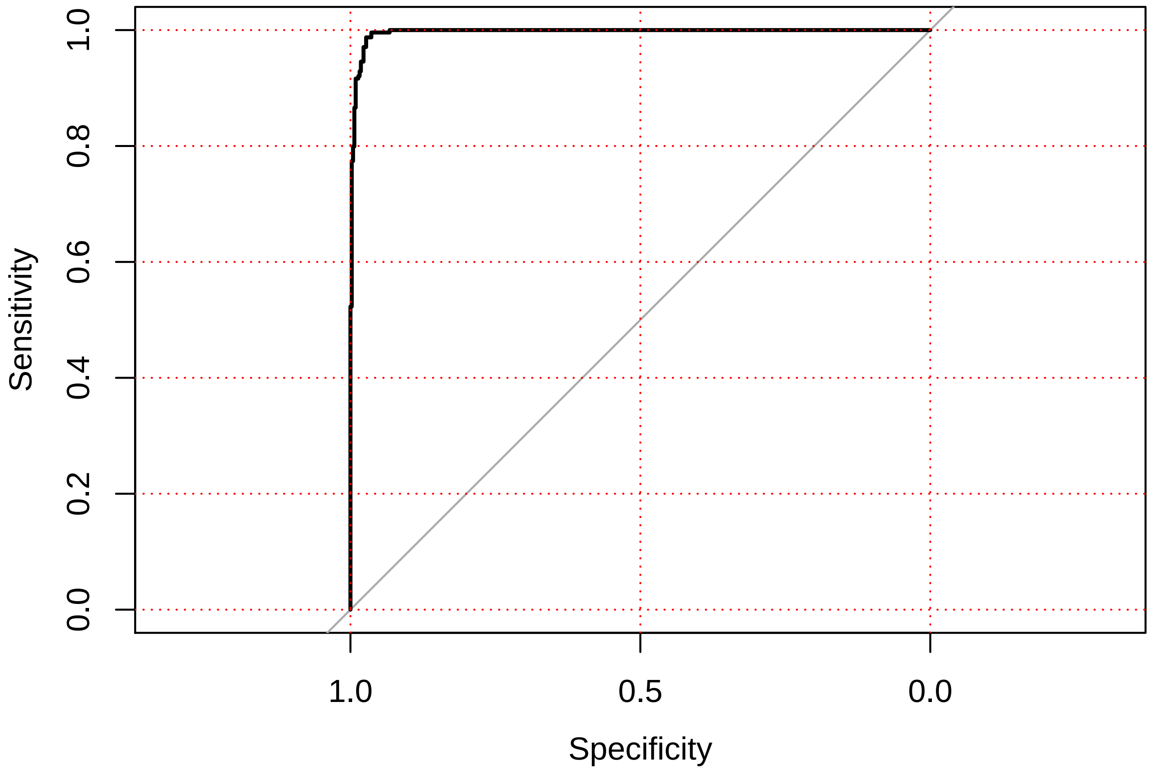
```
set.seed(1234)
data(BreastCancer)

# Drop NA rows
sum(is.na(BreastCancer))
```

```
## [1] 16
```

```
new.bc=na.omit(BreastCancer)

# Convert response variable into a factor variable of 1s and 0s
new.bc$Class <- ifelse(new.bc$Class == "malignant", 1, 0)
new.bc$Class <- factor(new.bc$Class, levels = c(0, 1))

# Remove id column
new.bc <- new.bc[,-1]

# Convert predictors to numeric
```

```r
for(i in 1:9) {
 new.bc[, i] <- as.numeric(as.character(new.bc[, i]))
}

# Fit a logistic model
fit.log=glm(Class~., data=new.bc, family='binomial')
```

```r
# Plot the ROC curve
pred.log <- predict(fit.log, data=new.bc, type = "response")
plot(roc(new.bc$Class, pred.log))
grid(col = 2)
```



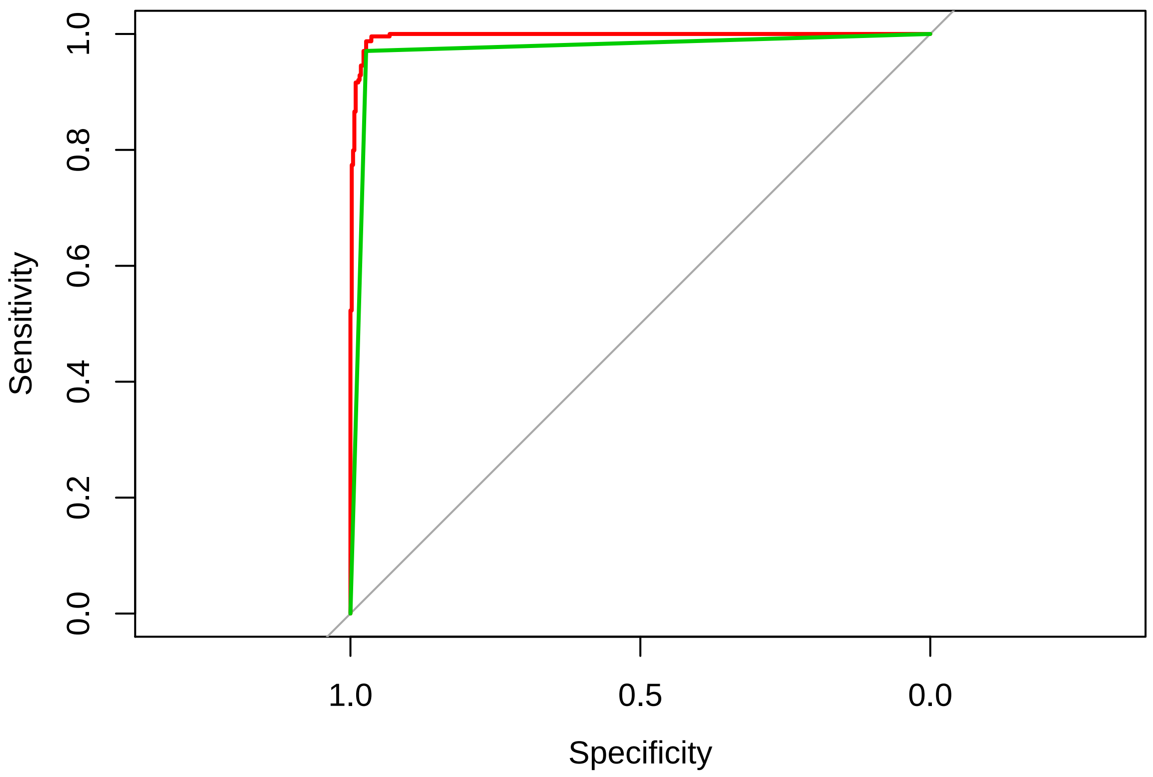
```r
# AUC score
auc(roc(new.bc$Class, pred.log))
```

```
## Area under the curve: 0.9963
```

The auc score for this logistic model is 0.9963.

b)

```r
# Use a support vector classifier (linear kernels) to classify
fit.linear=svm(Class~., kernel='linear', data=new.bc)
pred.linear=as.numeric(predict(fit.linear, data=new.bc))-1

# Plot ROC
plot(roc(new.bc$Class, pred.log), col=2)
lines(roc(new.bc$Class, pred.linear), col=3)
```



```r
# Calculate auc score
auc(roc(new.bc$Class, pred.linear))
```

```
## Area under the curve: 0.9718
```

The auc score for this svm model with linear kernel is 0.9718.

c)

```r
# Use SVMs with polynomial kernels degree = 2, 3, 4, 5 with cost fixed.
fit.poly2=svm(Class~., kernel='poly', data=new.bc, cost=10, degree=2)
fit.poly3=svm(Class~., kernel='poly', data=new.bc, cost=10, degree=3)
fit.poly4=svm(Class~., kernel='poly', data=new.bc, cost=10, degree=4)
fit.poly5=svm(Class~., kernel='poly', data=new.bc, cost=10, degree=5)

pred.poly.2 <- as.numeric(predict(fit.poly2)) - 1
pred.poly.3 <- as.numeric(predict(fit.poly3)) - 1
pred.poly.4 <- as.numeric(predict(fit.poly4)) - 1
pred.poly.5 <- as.numeric(predict(fit.poly5)) - 1

# Plot ROC curve for all models
plot(roc(new.bc$Class, pred.log), col=2)
lines(roc(new.bc$Class, pred.linear), col=3)
lines(roc(new.bc$Class, pred.poly.2), col=4)
lines(roc(new.bc$Class, pred.poly.3), col=5)
lines(roc(new.bc$Class, pred.poly.4), col=6)
lines(roc(new.bc$Class, pred.poly.5), col=7)

# Calculate auc score
auc(roc(new.bc$Class, pred.poly.2))
```

```
## Area under the curve: 0.9566
```

```
auc(roc(new.bc$Class, pred.poly.3))
```

```
## Area under the curve: 0.981
```

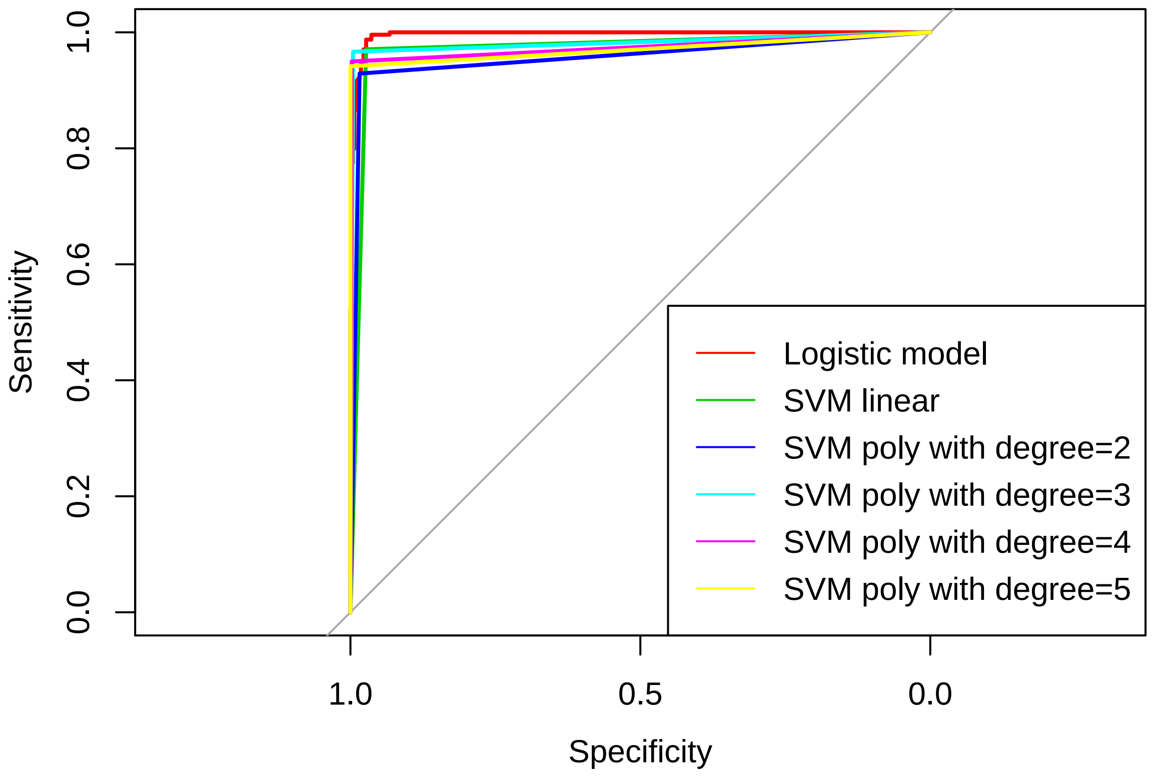
```
auc(roc(new.bc$Class, pred.poly.4))
```

```
## Area under the curve: 0.9738
```

```
auc(roc(new.bc$Class, pred.poly.5))
```

```
## Area under the curve: 0.9707
```

```
legend("bottomright", c('Logistic model', 'SVM linear', 'SVM poly with degree=2',
                     'SVM poly with degree=3', 'SVM poly with degree=4', 'SVM poly with degree=5'),
       col = c(2,3,4,5,6, 7), cex = 1, lty = 1)
```



d)

Logistic regression model is better since the auc score is near 1. With cost=10 and degree = 3, the auc score is about 0.981 for poly svm which is still not better than Logistic regression model. This appears not to depend on degree since the auc score increases first and then decreases.

Extra 66

```
set.seed(123)
MNIST=load('~/Desktop/other/Data/mnist_all.RData')
```

```r
# Generate dataframe
# Extract data for digit3 and digit8
index.train = (train$y ==3|train$y == 8)
train.df = as.data.frame(train$x[index.train,])
train.df$y=0
train.df$y[train$y[index.train]==8]=1
train.df$y = as.factor(train.df$y)

# Remove all variables with zero vaiance
var.train = apply(train.df,2,var)
train.df = train.df[,var.train>0]

# Distinguish from 3 and 8
index.test = (test$y ==3|test$y == 8)
test.df = as.data.frame(test$x[index.test,])
test.df$y=0
test.df$y[test$y[index.test]==8]=1
test.df$y = as.factor(test.df$y)

# Remove all variables with zero vaiance
test.df = test.df[,var.train>0]
```

a)

```r
# Random Forest
error_rate.rf = rep(NA,5)

for (i in 1:5){
  rf =randomForest(y~.,train.df,ntree = i*10)
  pred = predict(rf,train.df,type = 'class')
  tb = table(pred,train.df$y)
  error_rate.rf[i] = (tb[2]+tb[3])/length(train.df$y)
}
error_rate.rf
```

```
## [1] 8.345852e-04 0.000000e+00 8.345852e-05 0.000000e+00 0.000000e+00
```

The smallest error is about 0, when ntree = 20.

```r
rf.best = randomForest(y~.,train.df,ntree = 30)
pred.test = predict(rf.best,test.df,type = 'class')
tb = table(pred.test,test.df$y)
error.best.rf = (tb[2]+tb[3])/length(test.df$y)
print(error.best.rf)
```

```
## [1] 0.015625
```

The test error is about 0.015625.

b)

```r
# Write a function to calculate error
error_rate <- function(model, df)
{
  pred = predict(model, data=df)
  misclass = sum(df$y != pred)
  error_rate = misclass/length(pred)
  return(error_rate)
}

# When cost = 20
svm.20 = svm(y ~ ., kernel = "radial", data = train.df, cost = 20)
error.20 = error_rate(svm.20, train.df)

# When cost = 10
svm.10 = svm(y ~ ., kernel = "radial", data = train.df, cost = 10)
error.10 = error_rate(svm.10, train.df)

# When cost = 1
svm.1 = svm(y ~ ., kernel = "radial", data = train.df, cost = 1)
error.1 = error_rate(svm.1, train.df)

# When cost = 0.1
svm.0.1 = svm(y ~ ., kernel = "radial", data = train.df, cost = 0.1)
error.0.1 = error_rate(svm.0.1, train.df)
```

```
# Test data
pred.best.svm = predict(svm.20, newdata=test.df)
tb = table(pred.best.svm,test.df$y)
error.best.svm = (tb[2]+tb[3])/length(test.df$y)
print(error.best.svm)
```

```
## [1] 0.01310484
```

The smallest error is 0 which is gained when cost = 20, and the test error is 0.013

c)

The best train error for random forest is 0 when ntree=20, and the test error is 0.015625. So it may be overfitting.

The best train error for svm is 0 when cost = 20 and the test error is 0.013. So it may be overfitting.

Therefore, the svm test error is smaller than that for random forest, so svm is better.

Yes, the runtime is differ substantially: svm runs slower than the random forest, and when ntree or cost is greater, the runtime is longer.