HW3 Econ 187

Yiting Zhang

2022-05-26

Problem 8.8a

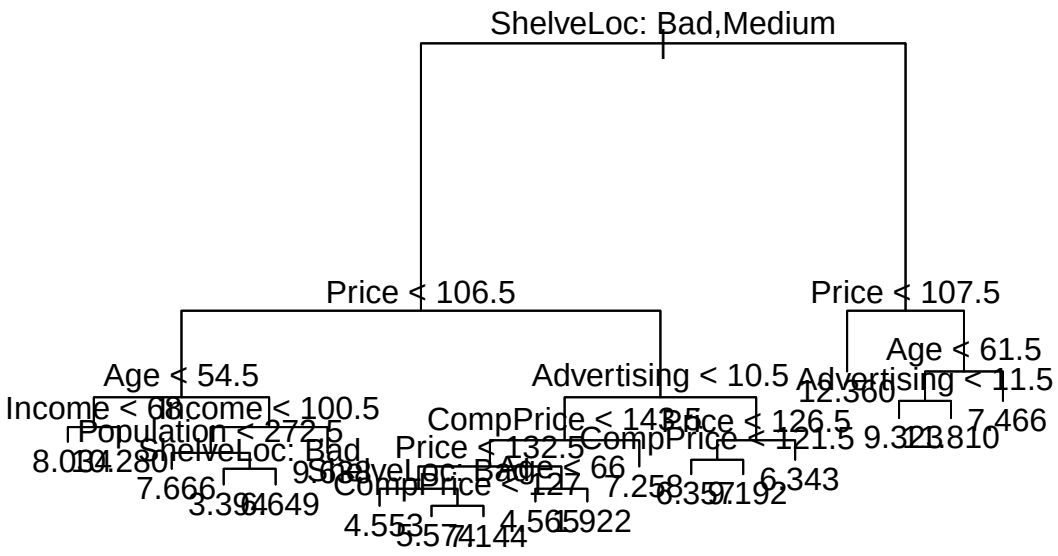
In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable. Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

(a) Split the data set into a training set and a test set.

```
data(Carseats)

set.seed(12345678)
train <- sample(1:dim(Carseats)[1], dim(Carseats )[1]*.75, rep=FALSE)
test <- -train
car.train<- Carseats[train, ]
car.test <- Carseats[test, ]
```

(b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain

```
#regression tree

reg.tree <- tree(Sales ~ ., data = car.train)
plot(reg.tree)
text(reg.tree, pretty=0)
```




```
summary(reg.tree)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = car.train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Income"      "Population"
## [6] "Advertising" "CompPrice"
## Number of terminal nodes:  19
## Residual mean deviance:  2.526 = 709.9 / 281
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.19600 -1.04700  0.06027  0.00000  1.04700  3.97600
```

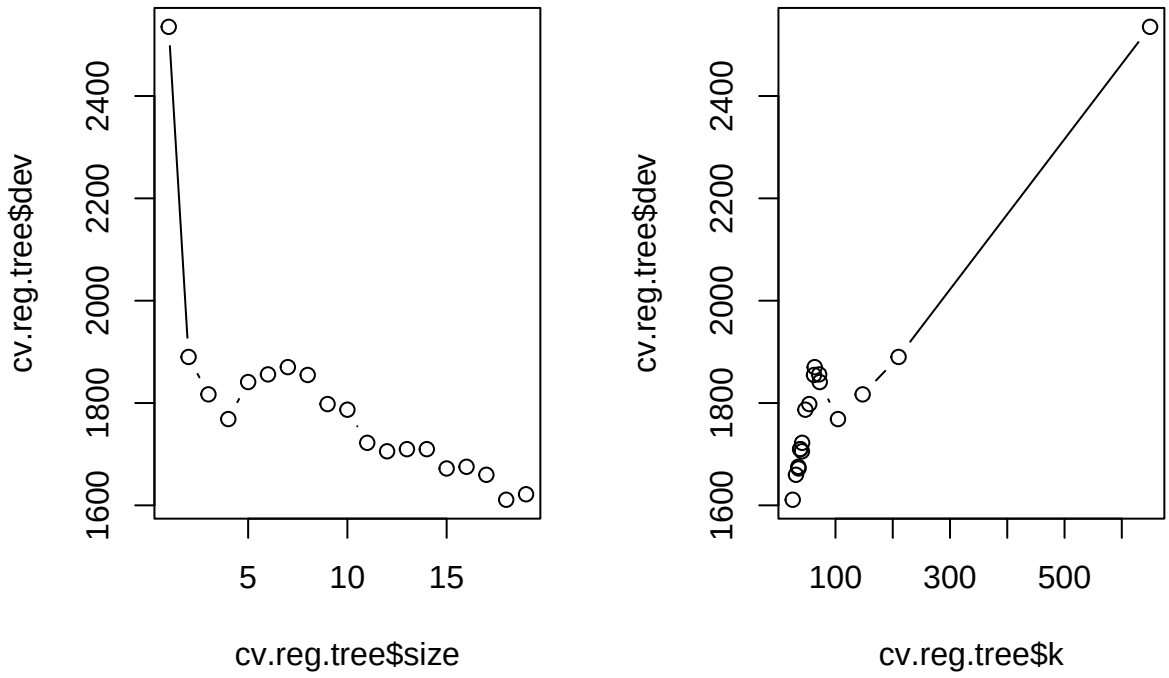
```
#testing MSE

pred.reg.tree <- predict(reg.tree, newdata = car.test)
test.mse <- mean((pred.reg.tree - car.test$Sales)^2)

cat('The testing MSE is :', test.mse)
```

```
## The testing MSE is : 4.329655
```

Using the regression tree, the most important indicator of Sales appears to be shelving location, because the first knot differentiates Good locations from Bad and Medium locations. And the testing MSE is 4.32.

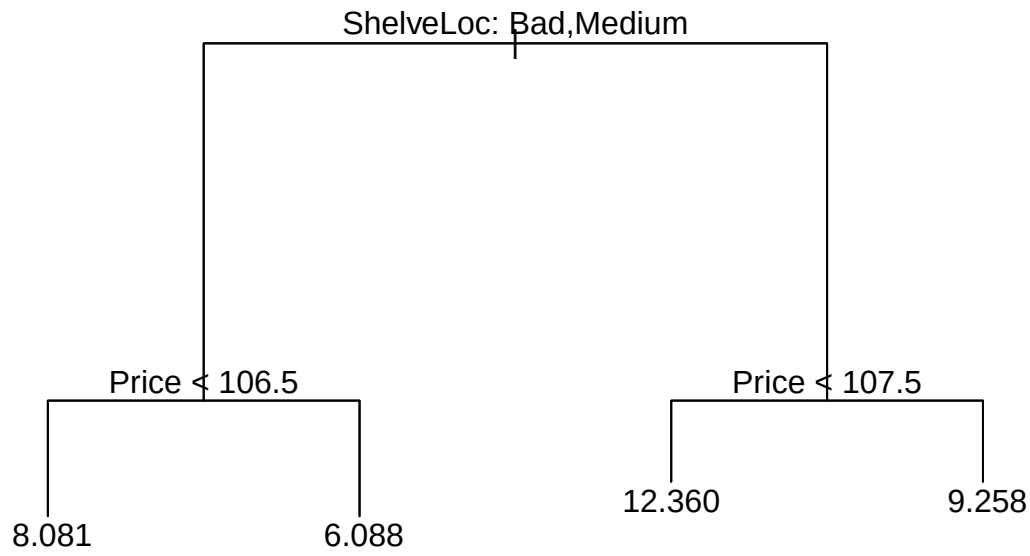
(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

```
set.seed(123)
cv.reg.tree <- cv.tree(reg.tree)

par(mfrow = c(1, 2))
plot(cv.reg.tree$size, cv.reg.tree$dev, type = "b")
plot(cv.reg.tree$k, cv.reg.tree$dev, type = "b")
```



From the plots, it looks like 4 or 12 is the best number of terminal nodes with under 1800 cross validation errors. Therefore, we should try to prune the tree.

```
#Let's try 4
prune.car <- prune.tree(reg.tree, best = 4)
plot(prune.car)
text(prune.car, pretty = 0)
```



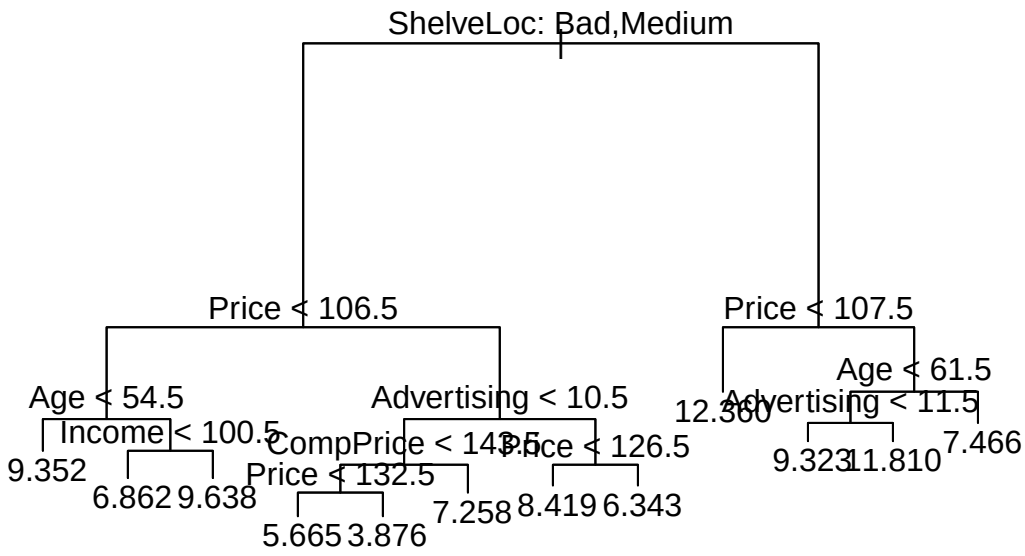

```
predict.prune <- predict(prune.car, newdata = car.test)
mean((predict.prune - car.test$Sales)^2)
```

```
## [1] 4.314584
```

```
cat('The testing MSE for 4 terminal nodes is :', mean((predict.prune - car.test$Sales)^2))
```

```
## The testing MSE for 4 terminal nodes is : 4.314584
```

```
#Let's try 12
prune.car <- prune.tree(reg.tree, best = 12)
plot(prune.car)
text(prune.car, pretty = 0)
```




```
predict.prune <- predict(prune.car, newdata = car.test)
mean((predict.prune - car.test$Sales)^2)
```

```
## [1] 4.302057
```

```
cat('The testing MSE for 12 terminal nodes is :', mean((predict.prune - car.test$Sales)^2))
```

```
## The testing MSE for 12 terminal nodes is : 4.302057
```

In this case, the MSE improves from the original tree, and we would choose 4 terminal nodes because the MSE for 12 terminal nodes did not improve that much from 4. So for the sake of interpretability, we would choose 4 terminal nodes for the prune tree.

(d) Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important.

```
set.seed(123)
car.bagging <- randomForest(Sales ~ ., data = Carseats, subset = train, ntree=500, mtry = 10,
                            importance = TRUE)
car.bagging
```

```
##
## Call:
```

```
##  randomForest(formula = Sales ~ ., data = Carseats, ntree = 500,      mtry = 10, importance = TRUE, s
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 2.756364
##                     % Var explained: 66.66
```

```
pred.bagging <- predict(car.bagging, newdata = car.test)
mean((pred.bagging - car.test$Sales)^2)
```

```
## [1] 1.54672
```

```
cat('The testing MSE is :', mean((pred.bagging - car.test$Sales)^2))
```

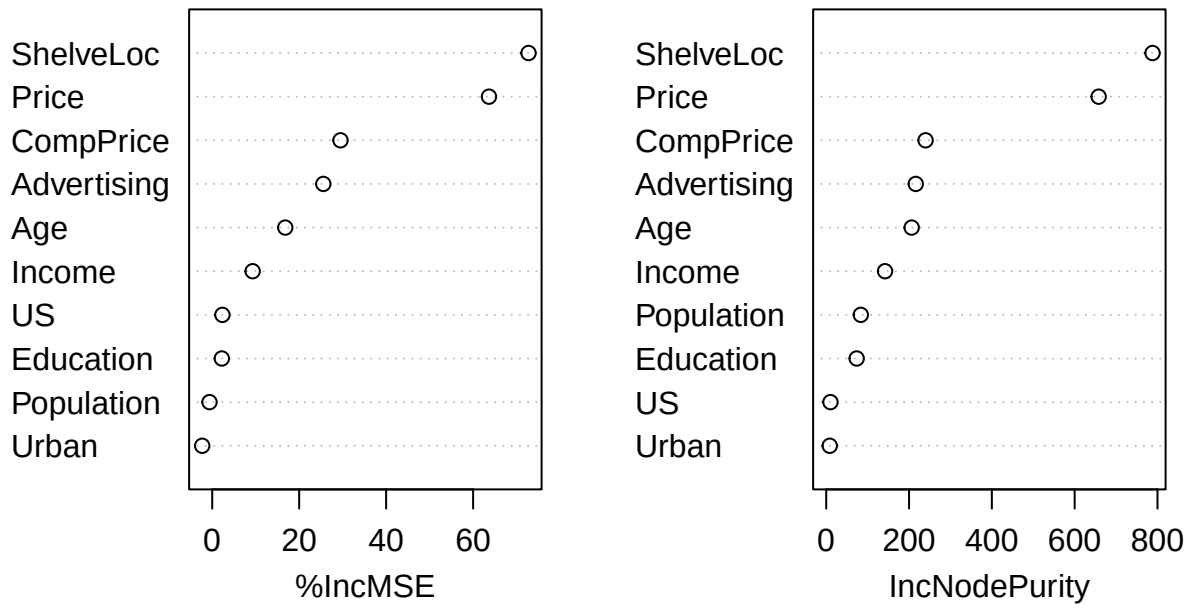
```
## The testing MSE is : 1.54672
```

The MSE is much lower than the regression tree and the prune tree. Bagging essentially reduces the variance.

```
importance(car.bagging)
```

```
##                %IncMSE IncNodePurity
## CompPrice   29.5062918    239.970363
## Income       9.2979698    142.096030
## Advertising 25.5570846    216.285592
## Population  -0.6160017     83.522245
## Price       63.6495870    657.970074
## ShelveLoc   72.7479730    788.056420
## Age         16.7988720    206.379395
## Education    2.2080871     73.653690
## Urban       -2.2998062      8.592606
## US           2.3440063     10.276359
```

```
varImpPlot(car.bagging)
```



The Importance() function shows that Shelve locations and Price are the most important variable.

(e) Use random forests to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

```
car.rf=randomForest(Sales~., data=Carseats, subset=train, importance=TRUE)
importance(car.rf)
```

```
##               %IncMSE IncNodePurity
## CompPrice   15.222395     212.49491
## Income       5.553445     191.89240
## Advertising 19.176548     244.97859
## Population  -1.054574     155.76795
## Price       40.465227     523.41097
## ShelveLoc   51.885984     592.08842
## Age         16.400972     266.08755
## Education    1.429300     104.54378
## Urban       -1.187119      20.63024
## US           3.960954      32.49367
```

```
mtry_=2:10
errors=rep(0,length(mtry_))

for(i in 1:length(mtry_)){
```

```
  m=mtry_[i]
  carseats.rf=randomForest(Sales~.,data=Carseats,
                           subset=train,mtry=mtry_[i],
                           importance=TRUE)
  pred.sales=predict(carseats.rf, car.test)
  test.mse=mean((pred.sales - car.test$Sales)^2)

  errors[i]=test.mse
}
errors
```

```
## [1] 2.769831 2.256163 1.943337 1.838787 1.742230 1.639247 1.613265 1.583473
## [9] 1.583379
```

The importance() function still indicates that Shelve locations and Price are the two most important indicators, same as Part(d). And testing MSE shows that the closer m is to 10, the smaller the MSE.

(f) Now analyze the data using BART, and report your results.

```
# gbart() function fits a Bayesian additive regression tree model to the data set.
# gbart() function is designed for quantitative outcome variables.

x <- Carseats[, 2:11]
y <- Carseats[, "Sales"]
xtrain <- x[train, ]
ytrain <- y[train]
xtest <- x[-train, ]
ytest <- y[-train]
set.seed(123)
bart.fit <- gbart(xtrain, ytrain, x.test = xtest)
```

```
## *****Calling gbart: type=1
## *****Data:
## data:n,p,np: 300, 14, 100
## y1,yn: 1.270600, 4.890600
## x1,x[n*p]: 131.000000, 1.000000
## xp1,xp[np*p]: 136.000000, 1.000000
## *****Number of Trees: 200
## *****Number of Cut Points: 70 ... 1
## *****burn,nd,thin: 100,1000,1
## *****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.287616,3,0.208994,7.5994
## *****sigma: 1.035813
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,14,0
## *****printevery: 100
##
## MCMC
## done 0 (out of 1100)
## done 100 (out of 1100)
## done 200 (out of 1100)
## done 300 (out of 1100)
## done 400 (out of 1100)
```

```
## done 500 (out of 1100)
## done 600 (out of 1100)
## done 700 (out of 1100)
## done 800 (out of 1100)
## done 900 (out of 1100)
## done 1000 (out of 1100)
## time: 5s
## trcnt,tecnt: 1000,1000
```

```
#Test MSE

yhat.bart <- bart.fit$yhat.test.mean
mean((ytest - yhat.bart)^2)
```

```
## [1] 1.137684
```

```
cat('The testing MSE is :', mean((ytest - yhat.bart)^2))
```

```
## The testing MSE is : 1.137684
```

The testing MSE is lower than both regression tree and bagging method.

```
#check how many times each variable appeared in the collection od trees

order_ <- order(bart.fit$varcount.mean, decreasing = TRUE)
bart.fit$varcount.mean[order_]
```

```
##       Price   CompPrice      Income  ShelveLoc2  ShelveLoc1         Age
##      25.606      19.961      17.754      17.333      16.757      16.684
##      Urban2         US1 Advertising         US2  ShelveLoc3      Urban1
##      16.657      16.019      15.521      15.267      15.167      15.113
##  Population   Education
##      15.089      14.869
```

It appears that Price shows up the most.

Problem 8.10a

We now use boosting to predict Salary in the Hitters data set.

(a) Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

```
rm(list = ls())
data(Hitters)
```

```
#check how many unknown there are in the dataset
sum(is.na(Hitters))
```

```
## [1] 59
```

```
#remove unknowns
hitters <- na.omit(Hitters)
#check again
sum(is.na(hitters))
```

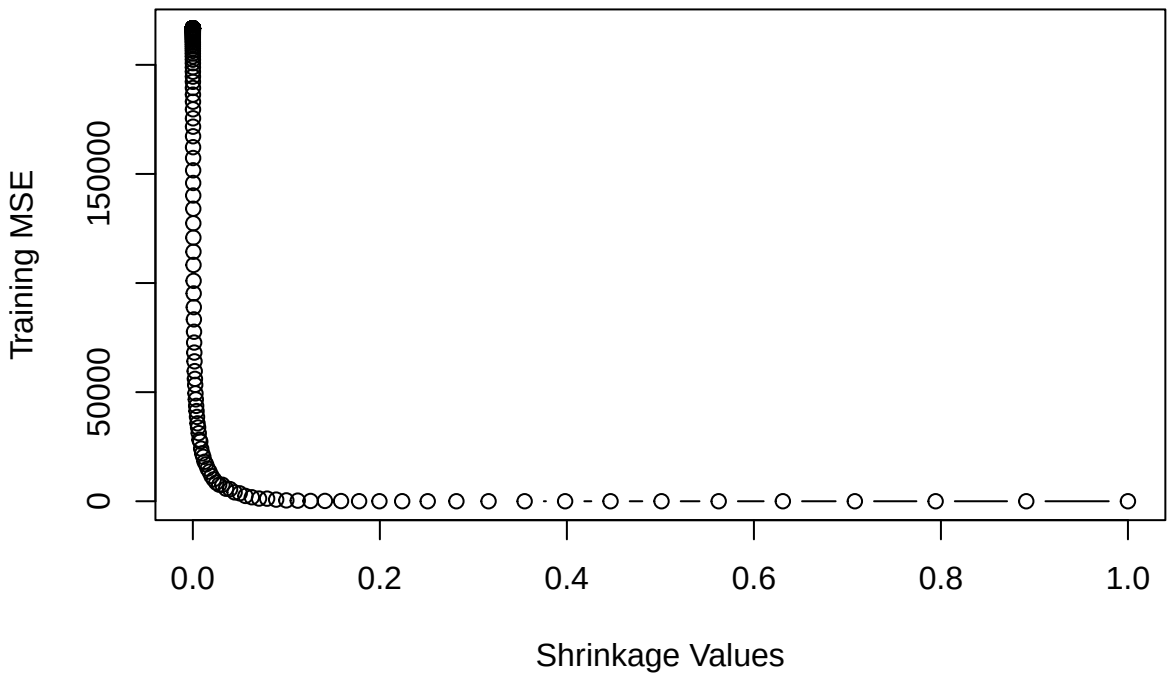
```
## [1] 0
```

(b) Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

```
set.seed(12345678)
train <- sample(1:dim(hitters)[1], dim(hitters )[1]*.75, rep=FALSE)
test <- -train
hit.train<- hitters[train, ]
hit.test <- hitters[test, ]
```

(c) Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.

```
set.seed(123)
p <- seq(from=-10, to=0, by=0.05)
shrinkage=10^p
errors=rep(0,length(shrinkage))
for (i in 1:length(shrinkage)){
  s=shrinkage[i]
  hit.boost=gbm(Salary~., data=hit.train,
                distribution="gaussian",
                n.trees=1000,
                shrinkage = s,
                interaction.depth=5)
  pred.boost=predict(hit.boost,newdata=hit.train, n.trees=1000)
  errors[i]=mean((pred.boost-hit.train$Salary)^2)
}
plot(shrinkage,errors, ylab = "Training MSE", xlab = "Shrinkage Values",
     type = "b")
```




```
#Find the minimum MSE
min(errors)
```

```
## [1] 1.548924e-11
```

```
shrinkage[which.min(errors)]
```

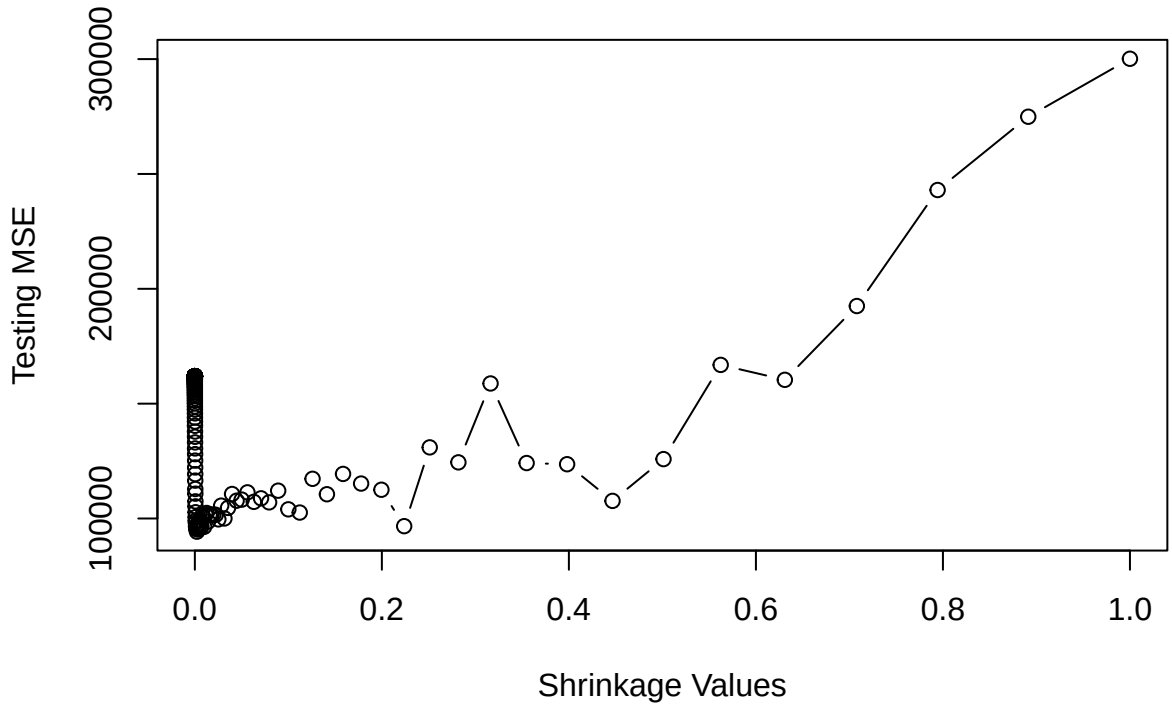
```
## [1] 0.8912509
```

The result from boosting shows that the minimum MSE is 1.548924e-11 where the shrinkage is 0.89.

(d) Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

```
set.seed(123)

for (i in 1:length(shrinkage)){
  s=shrinkage[i]
  hit.boost=gbm(Salary~., data=hit.train,
                distribution="gaussian",
                n.trees=1000,
                shrinkage = s,
                interaction.depth=5)
  pred.boost=predict(hit.boost,newdata=hit.test, n.trees=1000)
```

```
  errors[i]=mean((pred.boost-hit.test$Salary)^2)
}
plot(shrinkage,errors,ylab = "Testing MSE", xlab = "Shrinkage Values",
     type = "b")
```




```
#Find the minimum MSE
min(errors)
```

```
## [1] 94273.23
```

```
shrinkage[which.min(errors)]
```

```
## [1] 0.002238721
```

The minimum MSE is 94273.23 where the shrinkage is 0.002238721.

(e) Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

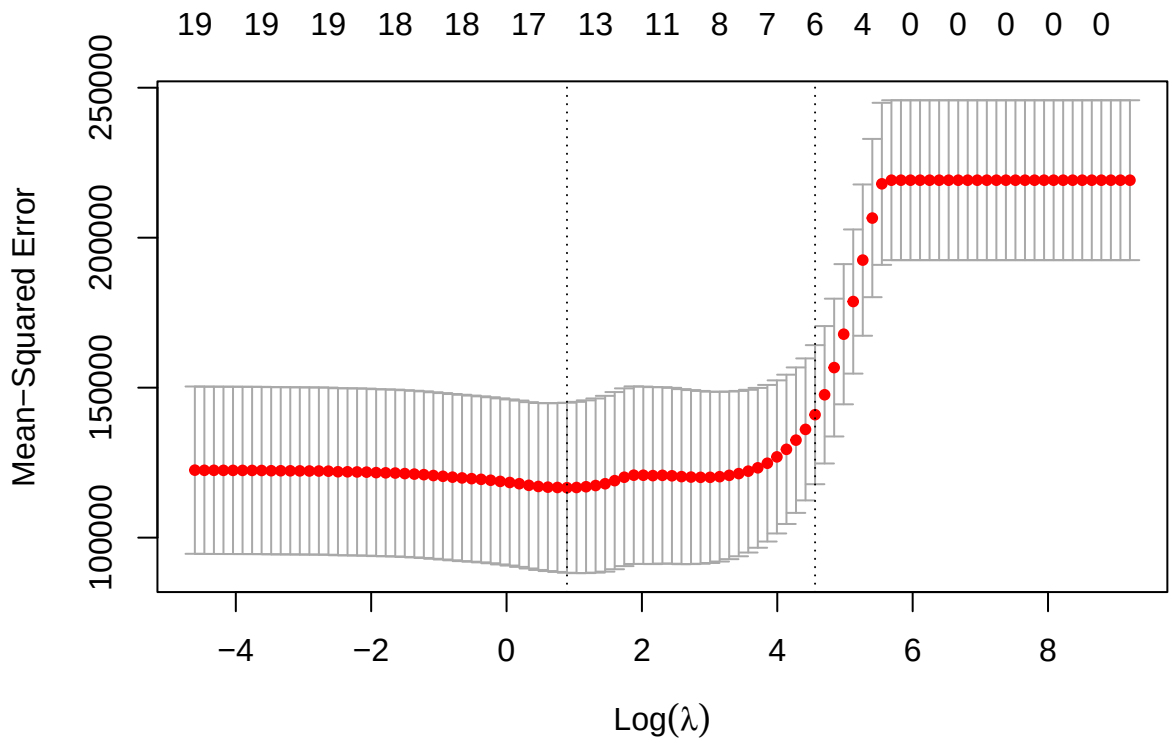
Compare Boosting with LASSO and linear regression:

```
#LASSO

library(vip)
```

```
##
## Attaching package: 'vip'
```

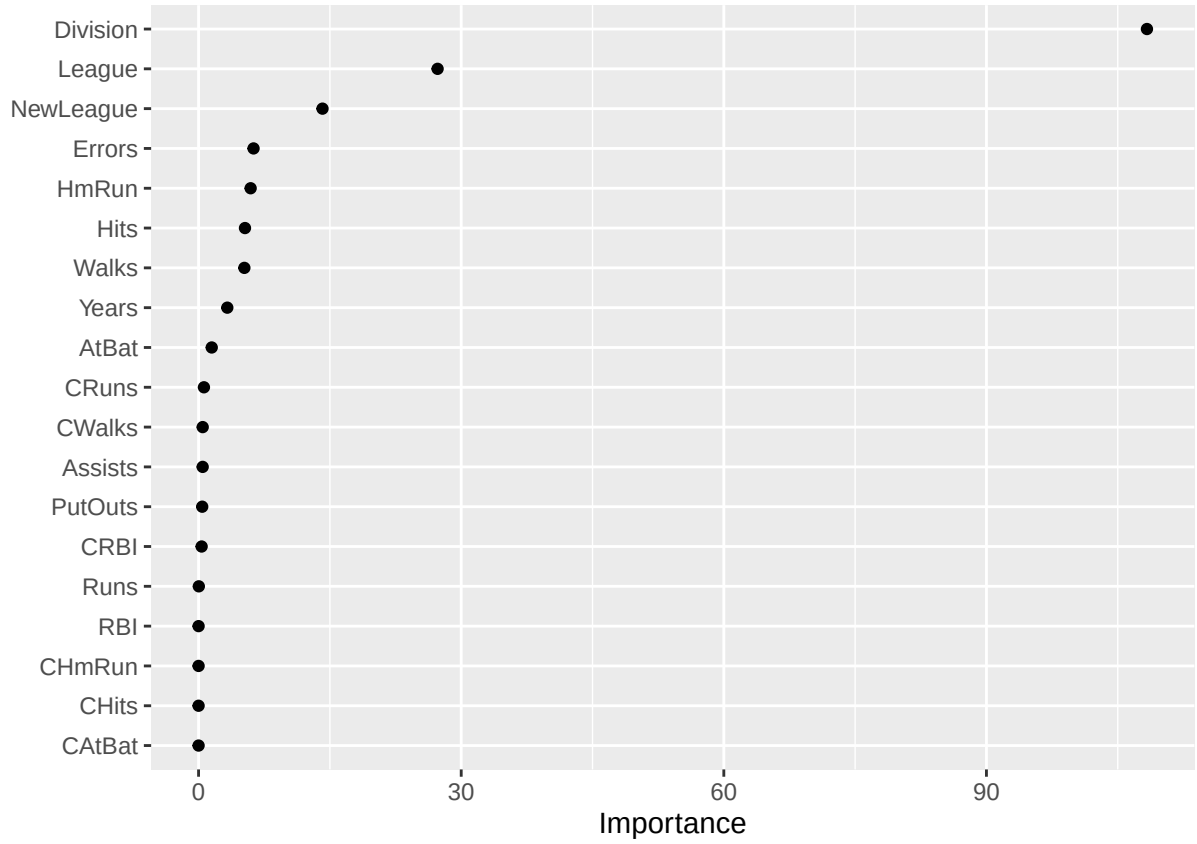
```
## The following object is masked from 'package:utils':
##
##     vi
```

```
lambda_try <- 10^seq(-2, 4, length.out = 99)
cv_lasso = cv.glmnet(x = data.matrix(hit.train[,-which(names(hit.train) %in% c("Salary"))]),
y=hit.train$Salary, alpha = 1, lambda=lambda_try,standardize = TRUE, nfolds = 10)

#choose best lambda
# Plot cross-validation results
plot(cv_lasso)
```




```
# Best cross-validated lambda
lambda_cv <- cv_lasso$lambda.min

# Fit final model
model_lasso <- glmnet(x = data.matrix(hit.train[,-which(names(hit.train) %in% c("Salary"))]),
```

```
y=hit.train$Salary, alpha = 1, lambda = lambda_cv, standardize = TRUE)

vip(model_lasso, num_features = 30, geom = "point")
```




```
train_control <- trainControl(method = "repeatedcv",
                              number = 5,
                              repeats = 5,
                              search = "random",
                              verboseIter = FALSE)

lasso_model     <- train(Salary ~ .,
                         data = hit.train,
                         metrics = 'RMSE',
                         method = "glmnet",
                         tuneGrid = expand.grid(alpha = 1,
                                                lambda = 1),
                         tuneLength = 25,
                         trControl = train_control)

# Predict using the testing data
lasso_pred = predict(lasso_model, newdata = hit.test)

# Evaluate performance
postResample(pred = na.omit(lasso_pred), obs = hit.test[,'Salary'])
```

```
##        RMSE    Rsquared         MAE
## 334.8420340   0.3863951 243.2410056
```

```
lasso_MSE <- mean((lasso_pred - hit.test[, "Salary"])^2); lasso_MSE
```

```
## [1] 112119.2
```

Lasso Model yields a MSE of 112119.2.

```
#Linear Regression Model

lm.fit <- lm(Salary ~ ., data=hit.train)
pred.lm <- predict(lm.fit, hit.test)
lm.mse <- mean((pred.lm - hit.test$Salary)^2)

lm.mse
```

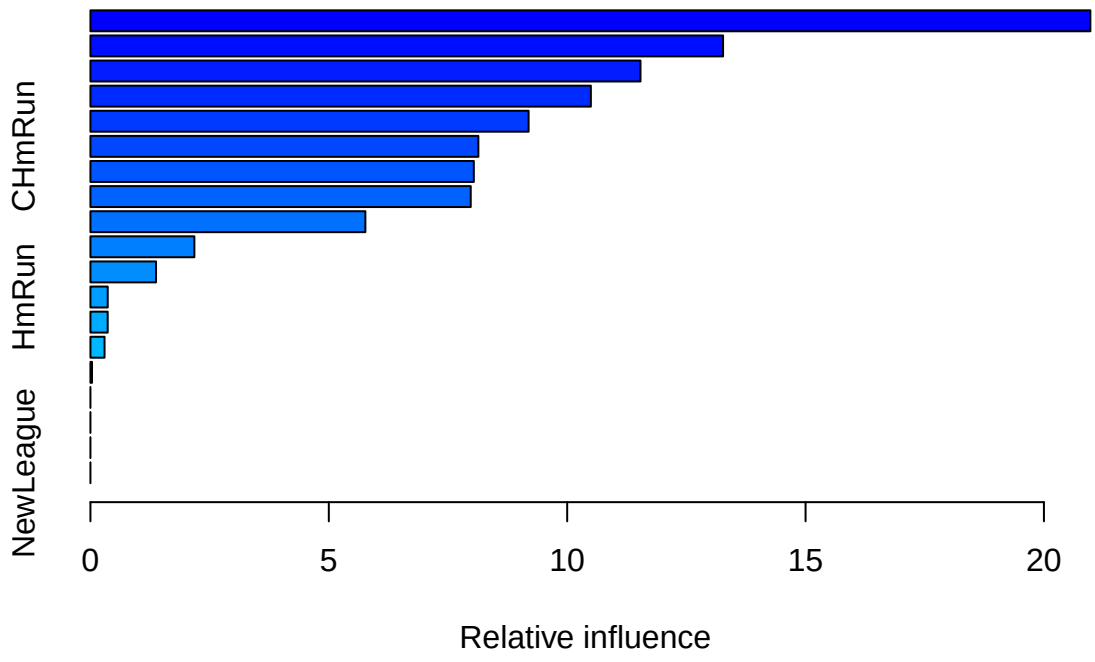
```
## [1] 112011
```

The Linear regression Model yields a MSE of 112011.

Compare to LASSO model and Linear Regression model, Boosting contributes to the lowest MSE.

(f) Which variables appear to be the most important predictors in the boosted model?

```
best.boost <- gbm(Salary ~ ., data=hit.train, distribution = 'gaussian', n.trees=1000, shrinkage = shri
summary(best.boost)
```



```
##                 var     rel.inf
## Walks         Walks 20.97383277
## CRBI           CRBI 13.27004593
## CHits         CHits 11.53785635
## CRuns         CRuns 10.49653863
## RBI             RBI  9.19073271
## CHmRun       CHmRun  8.13696673
## CAtBat       CAtBat  8.04176405
## Hits           Hits  7.97679651
## PutOuts     PutOuts  5.76688004
## CWalks       CWalks  2.17975355
## AtBat         AtBat  1.37519552
## HmRun         HmRun  0.36270979
## Years         Years  0.36185182
## Runs           Runs  0.29592803
## Division   Division  0.03314757
## League       League  0.00000000
## Assists     Assists  0.00000000
## Errors       Errors  0.00000000
## NewLeague NewLeague  0.00000000
```

It appears that Walks are the most important predictor.

(g) Now apply bagging to the training set. What is the test set MSE for this approach?

```
set.seed(123)
hit.bagging <- randomForest(Salary ~ ., data = hitters, subset = train, ntree=500, mtry = 10,
                        importance = TRUE)
hit.bagging
```

```
##
## Call:
##  randomForest(formula = Salary ~ ., data = hitters, ntree = 500,      mtry = 10, importance = TRUE, s
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 82084.24
##                     % Var explained: 62.13
```

```
pred.bagging <- predict(hit.bagging, newdata = hit.test)
mean((pred.bagging - hit.test$Salary)^2)
```

```
## [1] 100282.8
```

```
cat('The testing MSE is :', mean((pred.bagging - hit.test$Salary)^2))
```

```
## The testing MSE is : 100282.8
```

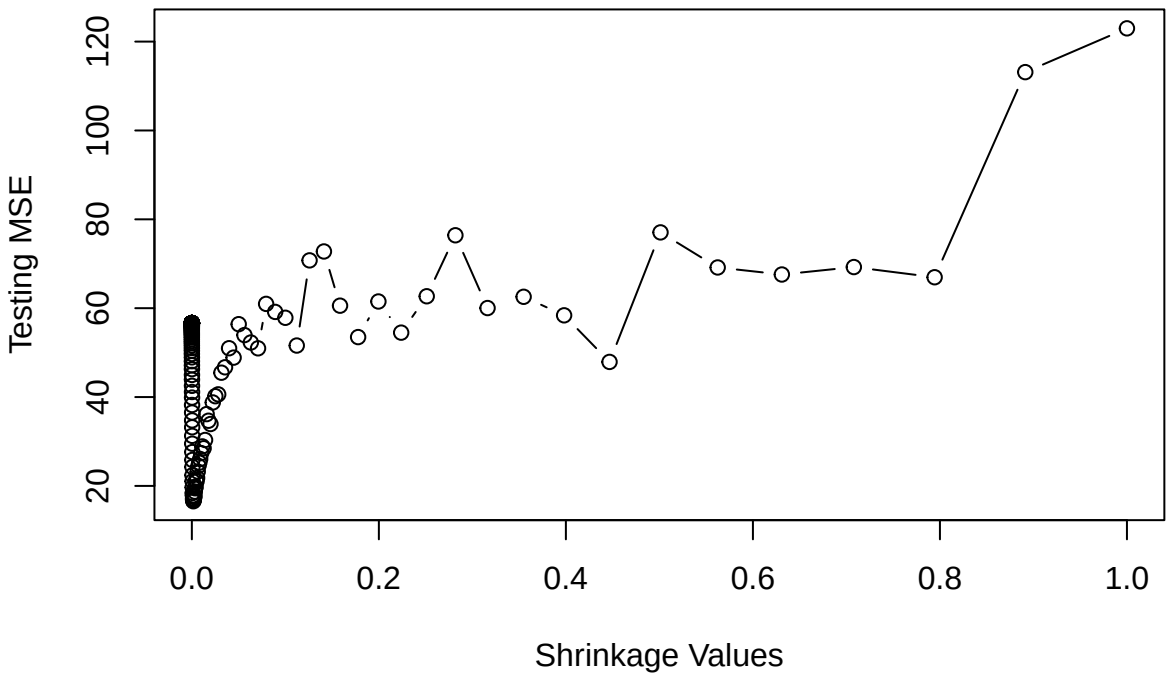
Compare with boosting method, bagging has a higher MSE. Therefore, we should choose boosting as the best method.

Problem 8.12a

Apply boosting, bagging, random forests to a data set of your choice. Be sure to fit the models on a training set and to evaluate their performance on a test set. How accurate are the results compared to simple methods like linear or logistic regression? Which of these approaches yields the best performance?

```
data("Boston")

set.seed(12345678)
train <- sample(1:dim(Boston)[1], dim(Boston )[1]*.75, rep=FALSE)
test <- -train
boston.train<- Boston[train, ]
boston.test <- Boston[test, ]
```

```
#boosting

set.seed(123)

p <- seq(from=-10, to=0, by=0.05)
shrinkage=10^p
errors=rep(0,length(shrinkage))

for (i in 1:length(shrinkage)){
  s=shrinkage[i]
  hit.boost=gbm(crim~., data=boston.train,
                distribution="gaussian",
                n.trees=1000,
                shrinkage = s,
                interaction.depth=5)
  pred.boost=predict(hit.boost,newdata=boston.test, n.trees=1000)
  errors[i]=mean((pred.boost-boston.test$crim)^2)
}
plot(shrinkage,errors,ylab = "Testing MSE", xlab = "Shrinkage Values",
     type = "b")
```




```
#Find the minimum MSE
min(errors)
```

```
## [1] 16.5519
```

```
shrinkage[which.min(errors)]
```

```
## [1] 0.001584893
```

```
cat('The testing MSE is :', min(errors))
```

```
## The testing MSE is : 16.5519
```

```
#bagging

set.seed(123)
boston.bagging <- randomForest(crim ~ ., data = Boston, subset = train, ntree=500, mtry = 10,
                               importance = TRUE)
boston.bagging
```

```
##
## Call:
##  randomForest(formula = crim ~ ., data = Boston, ntree = 500,      mtry = 10, importance = TRUE, subs
```

```
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 41.70486
##                     % Var explained: 47.61
```

```
pred.bagging <- predict(boston.bagging, newdata = boston.test)
mean((pred.bagging - boston.test$crim)^2)
```

```
## [1] 28.72875
```

```
cat('The testing MSE is :', mean((pred.bagging - boston.test$crim)^2))
```

```
## The testing MSE is : 28.72875
```

```
#random forest

boston.rf=randomForest(crim~., data=Boston, subset=train, importance=TRUE)
importance(boston.rf)
```

```
##            %IncMSE IncNodePurity
## zn       1.4631198      2.040498
## indus    6.9812258   1011.765819
## chas    -0.5574317     23.839769
## nox      4.7513237   1763.011826
## rm       5.6076573   2254.156292
## age     -1.8279019   1167.391825
## dis      3.5235684   4221.142379
## rad     11.9953852   3815.334589
## tax      9.8436087   2330.409795
## ptratio  7.5852236    339.745631
## black   -1.2160317   2161.061168
## lstat    5.2298703   1592.819412
## medv     7.0164921   6947.211725
```

```
mtry_=2:10
errors=rep(0,length(mtry_))

for(i in 1:length(mtry_)){
  m=mtry_[i]
  boston.rf=randomForest(crim~.,data=Boston,
                         subset=train,mtry=mtry_[i],
                         importance=TRUE)
  pred.crim=predict(boston.rf, boston.test)
  test.mse=mean((pred.crim - boston.test$crim)^2)

  errors[i]=test.mse
}
errors
```

```
## [1] 17.72651 21.02602 22.15983 23.95708 24.31580 25.34522 26.74185 27.41450
## [9] 28.92843
```

```
cat('The testing MSE is :', mean(errors))
```

```
## The testing MSE is : 24.17947
```

```
#Linear Regression Model

lm.fit <- lm(crim ~ ., data=boston.train)
pred.lm <- predict(lm.fit, boston.test)
lm.mse <- mean((pred.lm - boston.test$crim)^2)

cat('The testing MSE is :', lm.mse)
```

```
## The testing MSE is : 23.08197
```

```
#Logistic Regression

glm.fit=glm(crim~.,family=gaussian,data=boston.train)
glm.prob=predict(glm.fit,boston.test,type="response")

glm.mse <- mean((glm.prob - boston.test$crim)^2)

cat('The testing MSE is :', glm.mse)
```

```
## The testing MSE is : 23.08197
```

Compare the models above, we can see that boosting yields the lowest MSE and has the best performance, followed by linear regression and logistic regression model.

Problem 9.5a

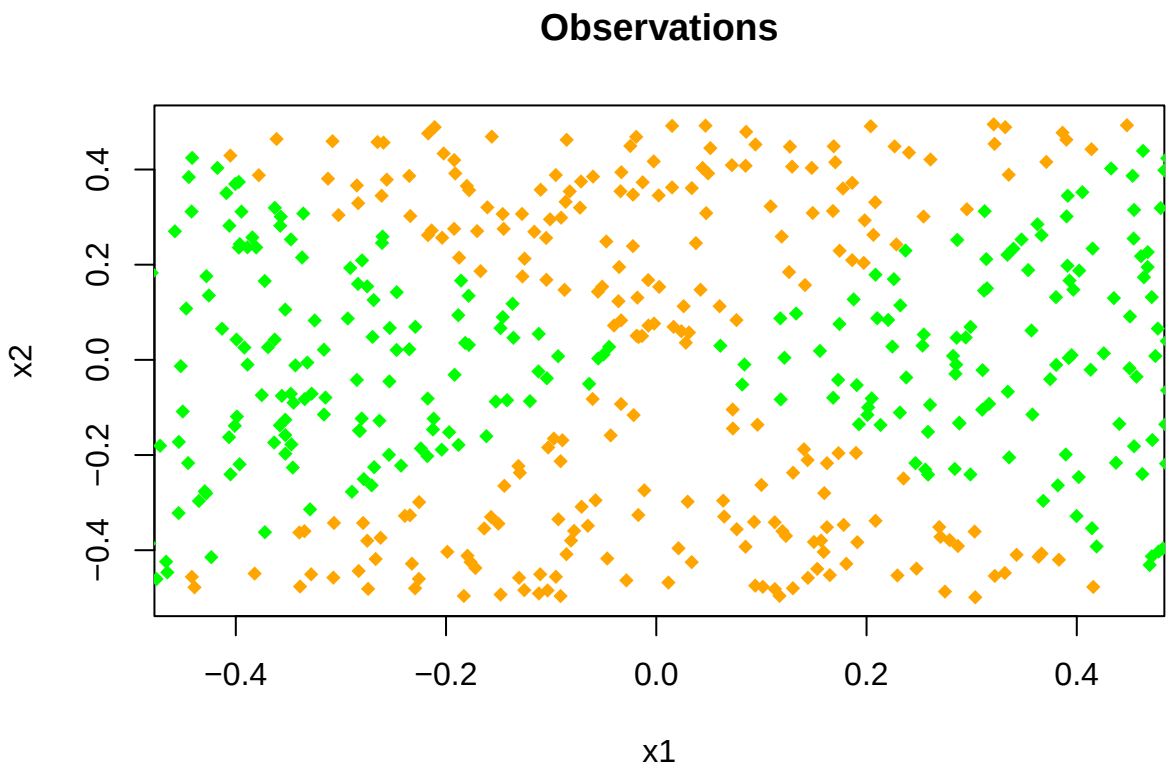
We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary.We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

(a) Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with a quadratic decision boundary between them.

```
set.seed(123)
x1 <- runif (500) - 0.5
x2 <- runif (500) - 0.5
y <- 1 * (x1^2 - x2^2 > 0)
```

(b) Plot the observations, colored according to their class labels. Your plot should display X1 on the x-axis, and X2 on the yaxis.

```
plot(x1[y == 0], x2[y == 0], col = "orange", main="Observations", xlab = "x1", ylab = "x2", pch = 18)
points(x1[y == 1], x2[y == 1], col = "green", pch = 18)
```



(c) Fit a logistic regression model to the data, using X1 and X2 as predictors.

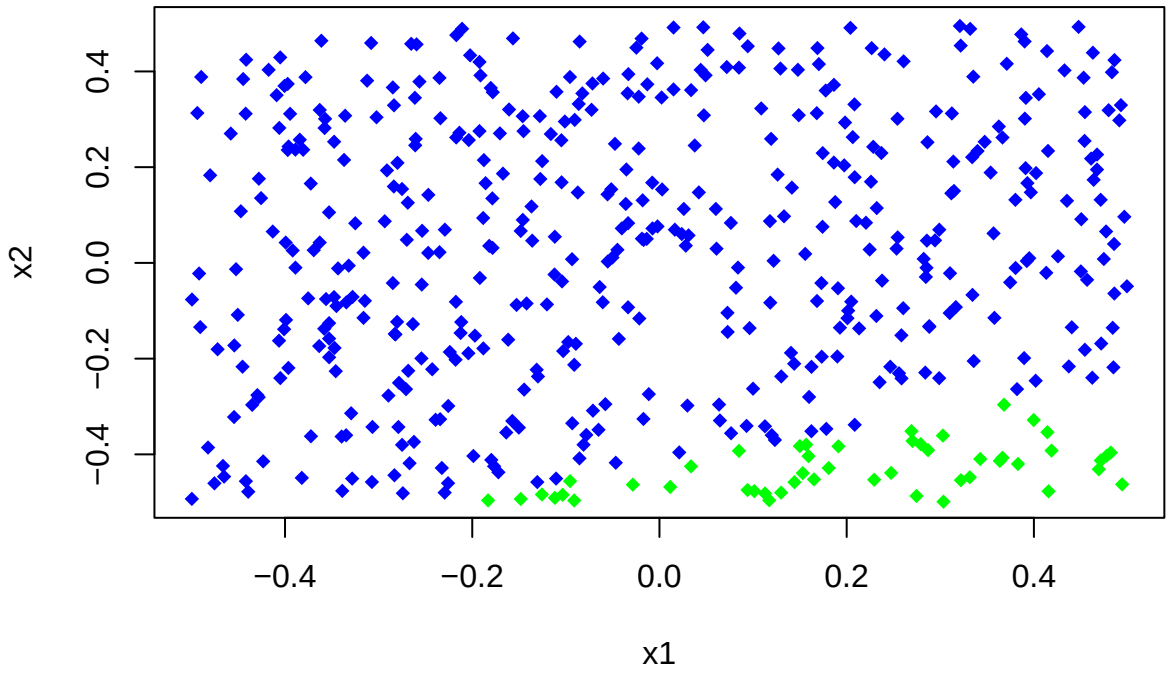
```
glm.fit=glm(y~x1+x2,family=binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = binomial)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.227  -1.200   1.133   1.157   1.188
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.04792    0.08949   0.535    0.592
## x1          -0.03999    0.31516  -0.127    0.899
## x2           0.11509    0.30829   0.373    0.709
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.86  on 499  degrees of freedom
## Residual deviance: 692.71  on 497  degrees of freedom
## AIC: 698.71
##
```

```
## Number of Fisher Scoring iterations: 3
```

(d) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be linear.

```
data = data.frame(x1 = x1, x2 = x2, y = y)
glm.prob=predict(glm.fit,data,type="response")
lm.pred = ifelse(glm.prob > 0.5, 1, 0)
data.1 = data[lm.pred == 1, ]
data.2 = data[lm.pred == 0, ]
plot(data.1$x1, data.1$x2, col = "blue", xlab = "x1", ylab = "x2", pch = 18)
points(data.2$x1, data.2$x2, col = "green", pch = 18)
```



(e) Now fit a logistic regression model to the data using non-linear functions of X1 and X2 as predictors (e.g. X21 , X1×X2, log(X2),and so forth).

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

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

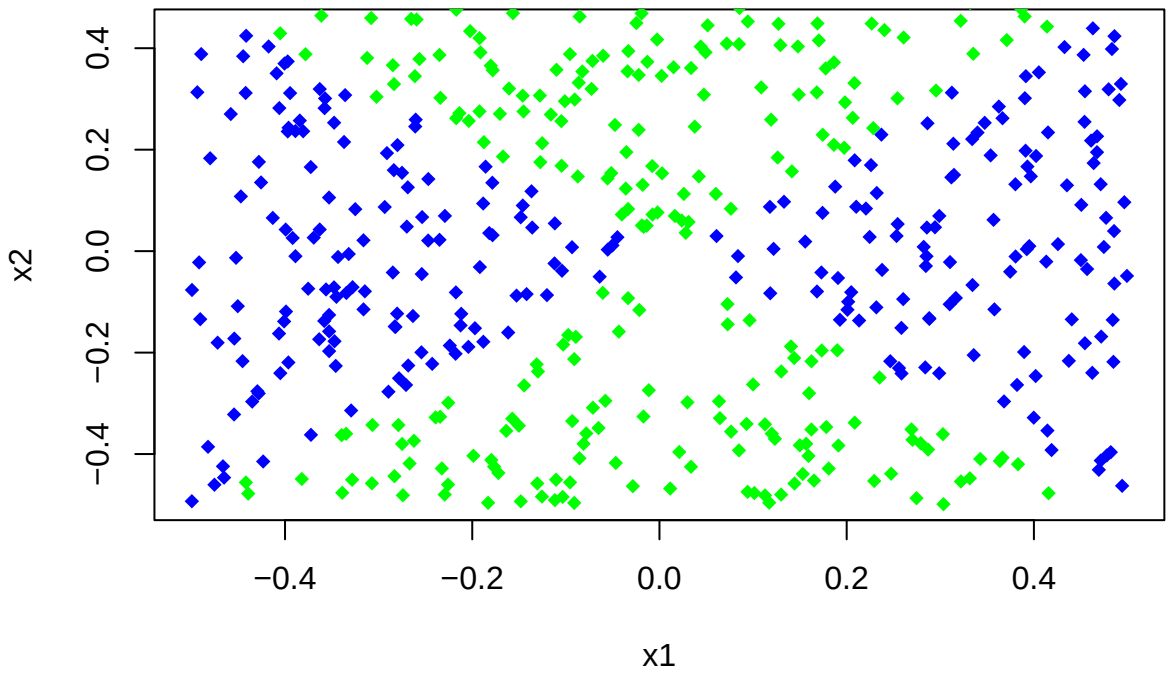
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(glm.fit.nonlinear)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family = "binomial")
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -8.625e-04  -2.000e-08   2.000e-08   2.000e-08   9.604e-04
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -54.24    3335.32  -0.016    0.987
## poly(x1, 2)1    542.33   71411.93   0.008    0.994
## poly(x1, 2)2  20838.39  778877.62   0.027    0.979
## poly(x2, 2)1   2163.06  115506.63   0.019    0.985
## poly(x2, 2)2 -21646.31  811141.33  -0.027    0.979
## I(x1 * x2)      566.19   36927.06   0.015    0.988
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9286e+02  on 499  degrees of freedom
## Residual deviance: 2.3990e-06  on 494  degrees of freedom
## AIC: 12
##
## Number of Fisher Scoring iterations: 25
```

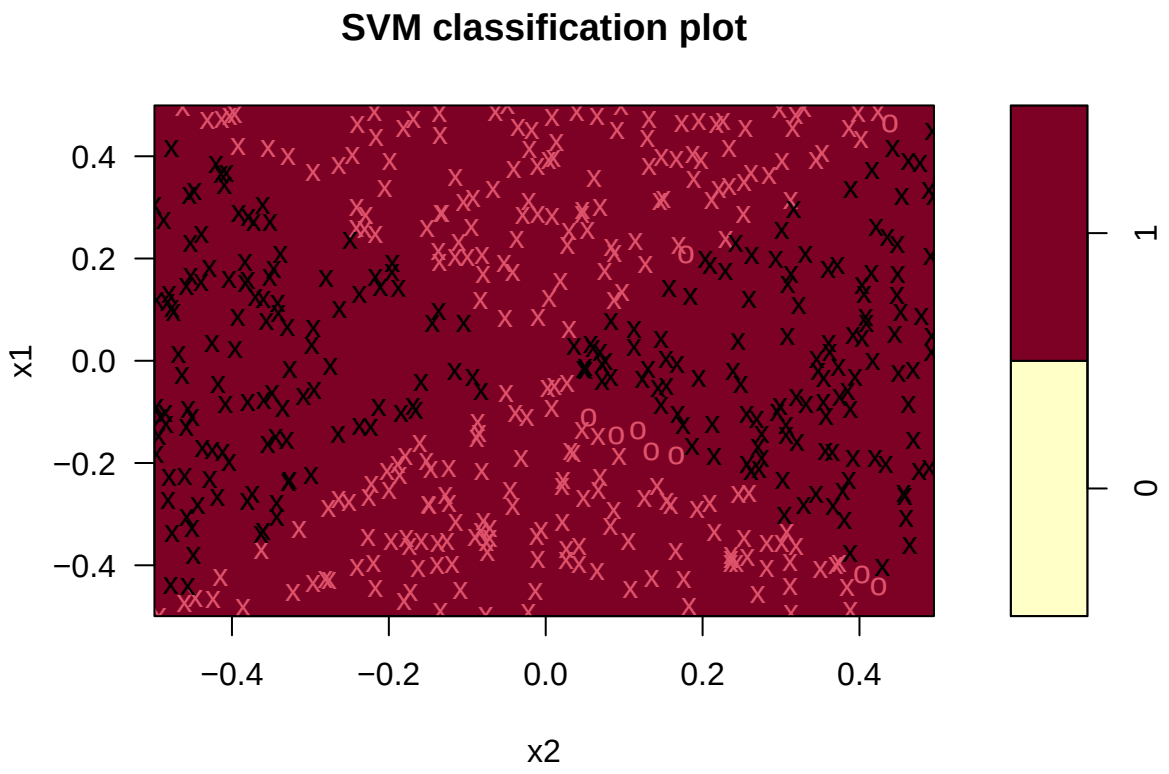
(f) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be obviously non-linear. If it is not, then repeat (a)-(e) until you come up with an example in which the predicted class labels are obviously non-linear.

```
glm.prob=predict(glm.fit.nonlinear,data,type="response")
lm.pred = ifelse(glm.prob > 0.5, 1, 0)
data.1 = data[lm.pred == 1, ]
data.2 = data[lm.pred == 0, ]
plot(data.1$x1, data.1$x2, col = "blue", xlab = "x1", ylab = "x2", pch = 18)
points(data.2$x1, data.2$x2, col = "green", pch = 18)
```



(g) Fit a support vector classifier to the data with X1 and X2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

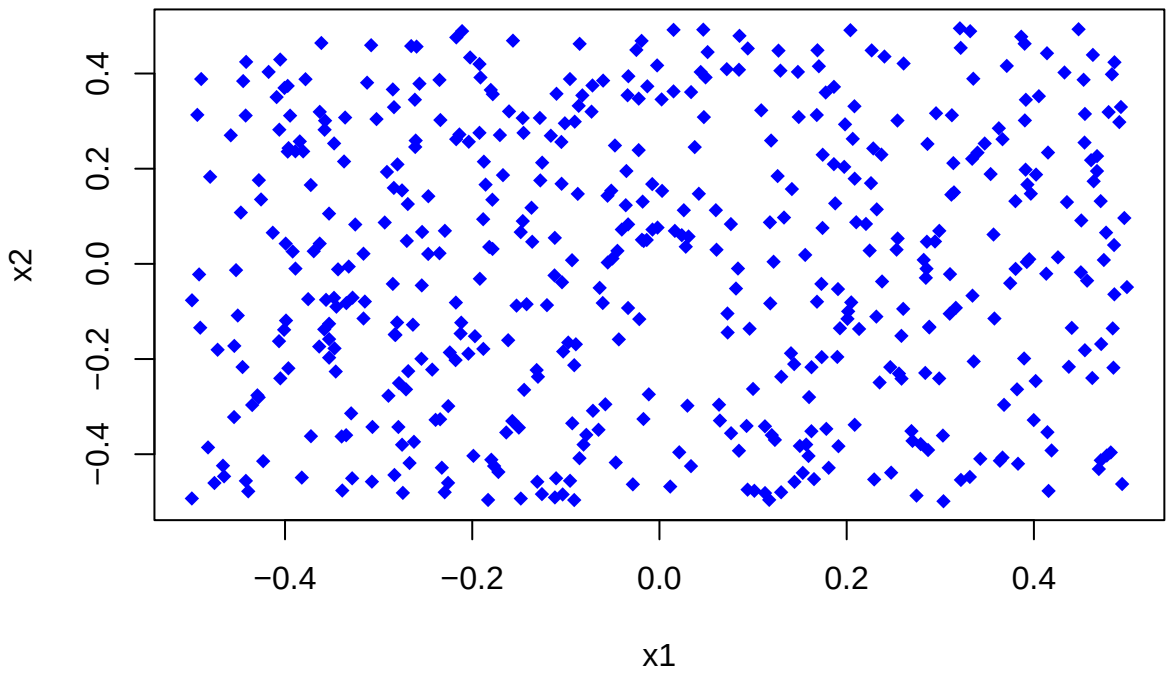
```
library(e1071)

svm.fit <- svm(as.factor(y) ~ x1+x2,data=data, kernel = "linear", cost = 0.1)
plot(svm.fit, data)
```




```
svm.pred <- predict(svm.fit, data)
table(predict = svm.pred, truth = data$y)
```

```
##         truth
## predict   0   1
##       0   0   0
##       1 244 256
```

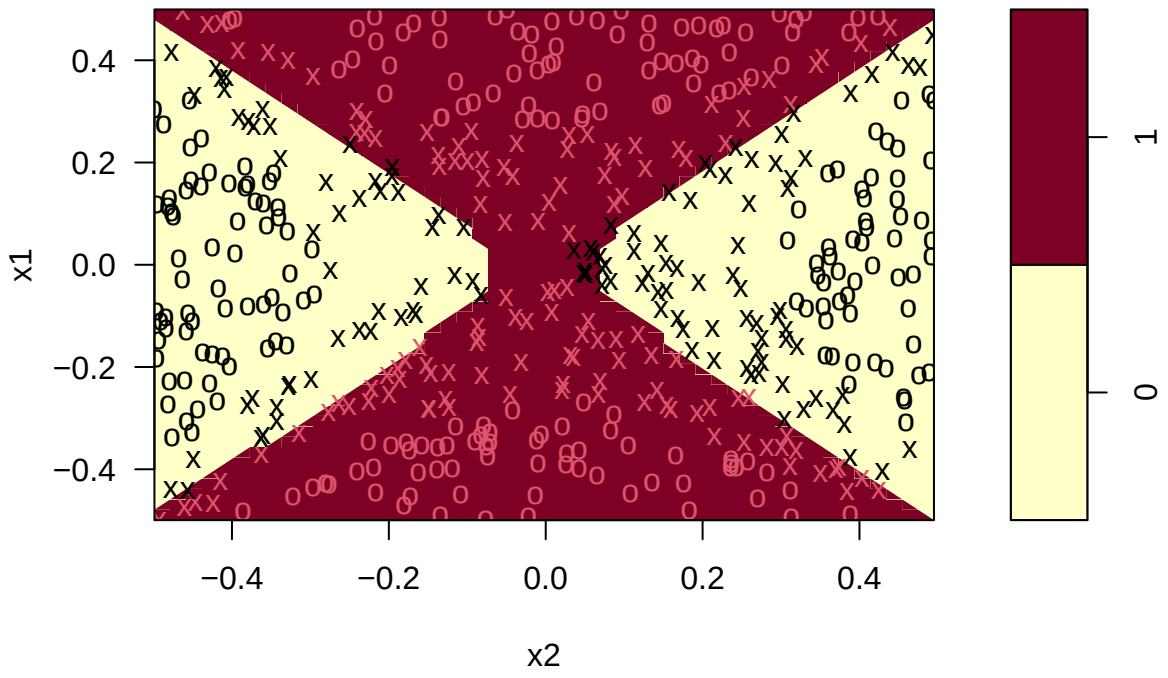
```
data.1 = data[svm.pred == 1, ]
data.2 = data[svm.pred == 0, ]
plot(data.1$x1, data.1$x2, col = "blue", xlab = "x1", ylab = "x2", pch = 18)
points(data.2$x1, data.2$x2, col = "red", pch = 18)
```



The SVM model put x1 x2 into the same classification.

(h) Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

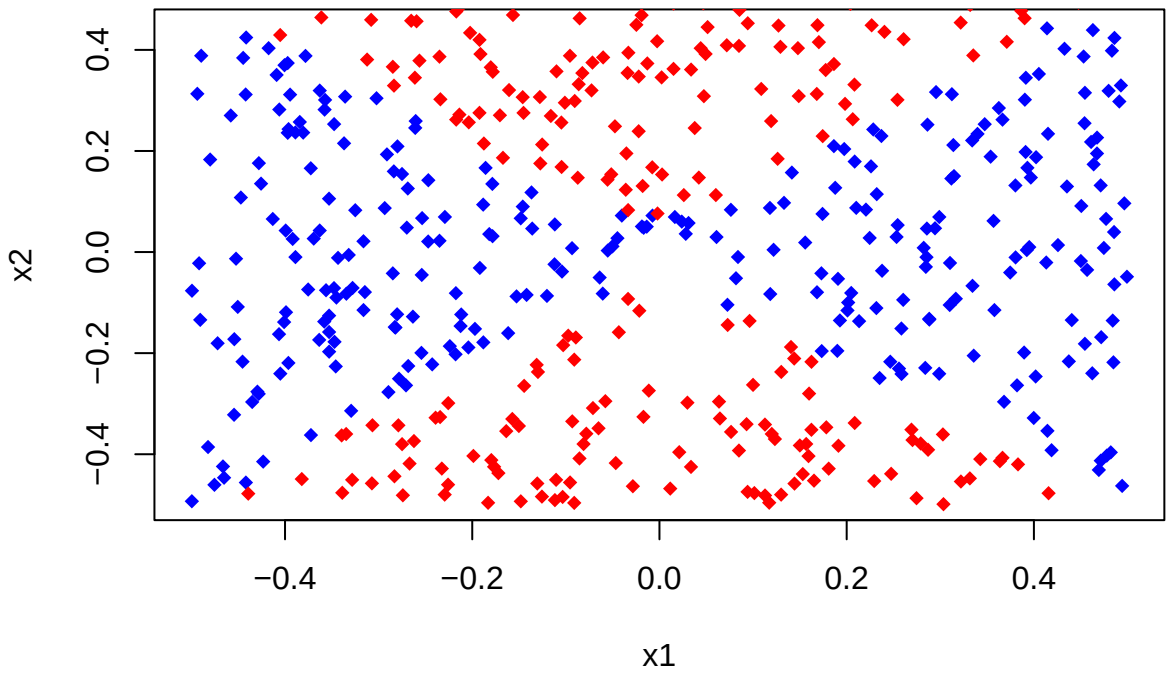
```
#polynomial case
svm.fit.nonlinear <- svm(as.factor(y) ~ x1+x2,data=data, kernel = "polynomial", d=2, cost = 0.1)
plot(svm.fit.nonlinear, data)
```



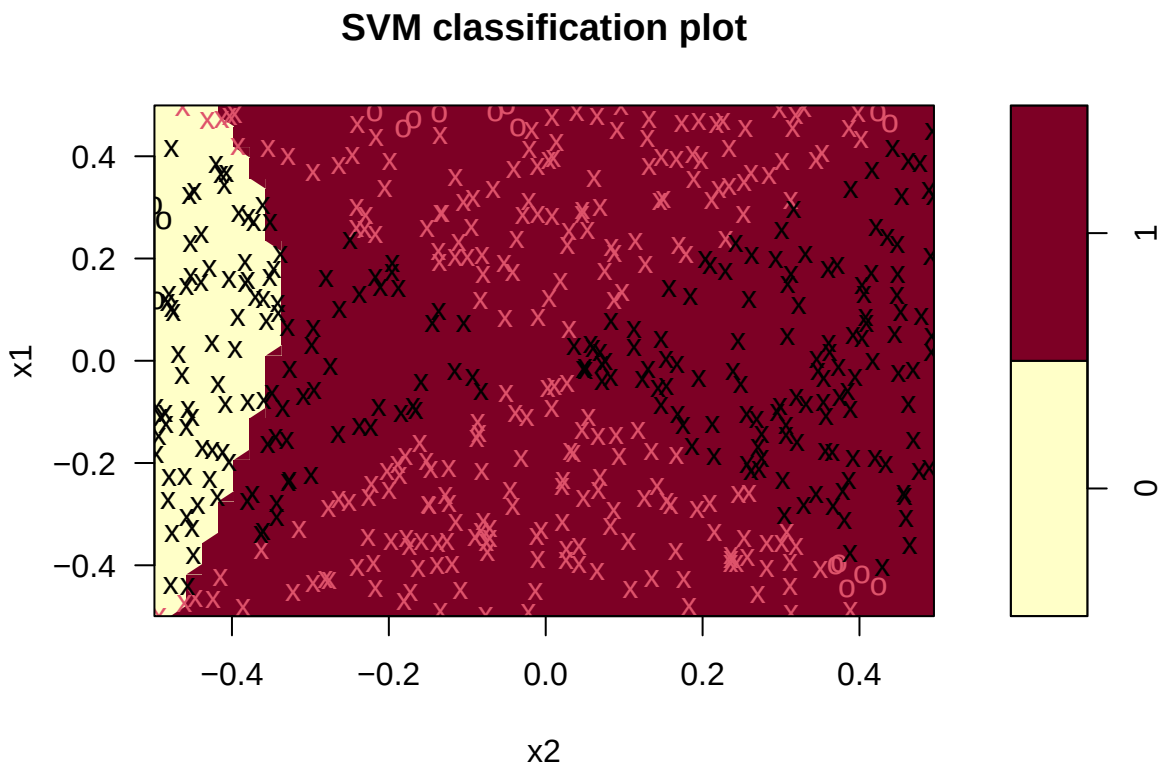

```
svm.pred <- predict(svm.fit.nonlinear, data)
table(predict = svm.pred, truth = data$y)
```

```
##         truth
## predict   0   1
##       0 220   0
##       1  24 256
```

```
data.1 = data[svm.pred == 1, ]
data.2 = data[svm.pred == 0, ]
plot(data.1$x1, data.1$x2, col = "blue", xlab = "x1", ylab = "x2", pch = 18)
points(data.2$x1, data.2$x2, col = "red", pch = 18)
```



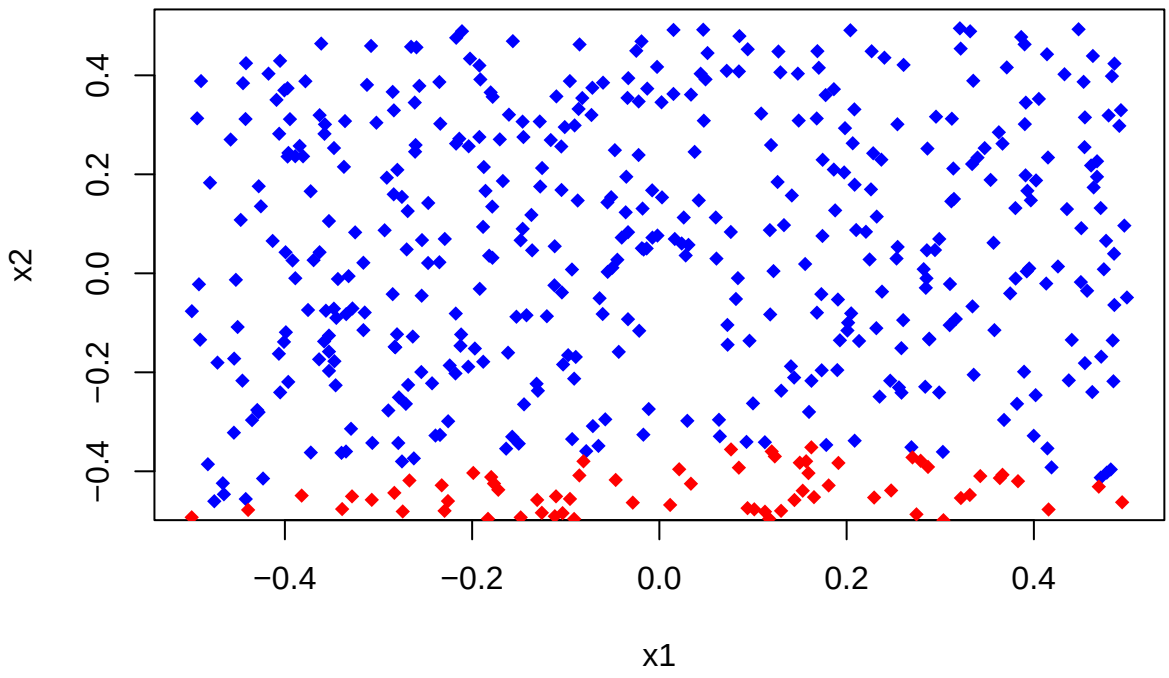

```
#radial case
svm.fit.radial <- svm(as.factor(y) ~ x1+x2,data=data, kernel = "polynomial", gamma=1, cost = 10)
plot(svm.fit.radial, data)
```




```
svm.pred <- predict(svm.fit.radial, data)
table(predict = svm.pred, truth = data$y)
```

```
##         truth
## predict   0   1
##       0  63   3
##       1 181 253
```

```
data.1 = data[svm.pred == 1, ]
data.2 = data[svm.pred == 0, ]
plot(data.1$x1, data.1$x2, col = "blue", xlab = "x1", ylab = "x2", pch = 18)
points(data.2$x1, data.2$x2, col = "red", pch = 18)
```



(i) Comment on your results.

In conclusion, we can see that the non-linear SVM and Logistic Regression Model are useful tools to identify the boundaries. For further investigation, cross validation would be beneficial to find the optimal cost for SVM by tuning the parameters gamma and cost, and generate more accurate boundaries.

Problem 9.7a

In this problem, you will use support vector approaches in order to predict whether a given car gets high or low gas mileage based on the Auto data set.

(a) Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

```
data(Auto)
mpg.median = median(Auto$mpg)
mpg01 = ifelse(Auto$mpg > mpg.median, 1, 0)
Auto$mpg01 = as.factor(mpg01)
```

(b) Fit a support vector classifier to the data with various values of cost, in order to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter. Comment on your results. Note you will need to fit the classifier without the gas mileage variable to produce sensible results.

```
set.seed(123)
tune.out <- tune(svm, mpg01 ~ ., data = Auto, kernel = "linear",
    ranges = list(cost = c(0.001, 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: 0.01025641
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.09935897 0.04521036
## 2 1e-02 0.07634615 0.03928191
## 3 1e-01 0.04333333 0.03191738
## 4 1e+00 0.01025641 0.01792836
## 5 5e+00 0.01538462 0.01792836
## 6 1e+01 0.01788462 0.01727588
## 7 1e+02 0.03320513 0.02720447
```

We found that cost=1 has the lowest error rate, and is the best parameter to use.

(c) Now repeat (b), this time using SVMs with radial and polynomial basis kernels, with different values of gamma and degree and cost. Comment on your results.

```
#polynomial
set.seed(123)
tune.out <- tune(svm, mpg01 ~ ., data = Auto, kernel = "polynomial",
                 ranges = list(cost = c(0.1, 1, 10, 100, 1000),
                 degree = c(2, 3, 4) ))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##  1000      2
##
## - best performance: 0.2471154
##
## - Detailed performance results:
##     cost degree     error dispersion
## 1  1e-01      2 0.5817308 0.04740051
```

```
## 2  1e+00      2 0.5817308 0.04740051
## 3  1e+01      2 0.5714744 0.04575370
## 4  1e+02      2 0.3034615 0.10917787
## 5  1e+03      2 0.2471154 0.09155939
## 6  1e-01      3 0.5817308 0.04740051
## 7  1e+00      3 0.5817308 0.04740051
## 8  1e+01      3 0.5817308 0.04740051
## 9  1e+02      3 0.3521154 0.13782036
## 10 1e+03      3 0.2550641 0.07998902
## 11 1e-01      4 0.5817308 0.04740051
## 12 1e+00      4 0.5817308 0.04740051
## 13 1e+01      4 0.5817308 0.04740051
## 14 1e+02      4 0.5817308 0.04740051
## 15 1e+03      4 0.5791667 0.04605209
```

The best parameters from SVM with polynomial kernels are cost=1000 and degree=2.

```
#radial
set.seed(123)
tune.out <- tune(svm, mpg01 ~ ., data = Auto, kernel = "radial",
                 ranges = list(cost = c(0.1, 1, 10, 100, 1000),
                 gamma = c(0.5, 1, 2, 3, 4) ))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.04576923
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-01   0.5 0.08147436 0.03707182
## 2  1e+00   0.5 0.04576923 0.03903092
## 3  1e+01   0.5 0.05339744 0.03440111
## 4  1e+02   0.5 0.05339744 0.03440111
## 5  1e+03   0.5 0.05339744 0.03440111
## 6  1e-01   1.0 0.58173077 0.04740051
## 7  1e+00   1.0 0.05865385 0.04942437
## 8  1e+01   1.0 0.05608974 0.04595880
## 9  1e+02   1.0 0.05608974 0.04595880
## 10 1e+03   1.0 0.05608974 0.04595880
## 11 1e-01   2.0 0.58173077 0.04740051
## 12 1e+00   2.0 0.11474359 0.06630201
## 13 1e+01   2.0 0.11474359 0.06630201
## 14 1e+02   2.0 0.11474359 0.06630201
## 15 1e+03   2.0 0.11474359 0.06630201
## 16 1e-01   3.0 0.58173077 0.04740051
```

```
## 17 1e+00   3.0 0.42878205 0.17823496
## 18 1e+01   3.0 0.40839744 0.18573046
## 19 1e+02   3.0 0.40839744 0.18573046
## 20 1e+03   3.0 0.40839744 0.18573046
## 21 1e-01   4.0 0.58173077 0.04740051
## 22 1e+00   4.0 0.51538462 0.06959451
## 23 1e+01   4.0 0.50012821 0.07022396
## 24 1e+02   4.0 0.50012821 0.07022396
## 25 1e+03   4.0 0.50012821 0.07022396
```

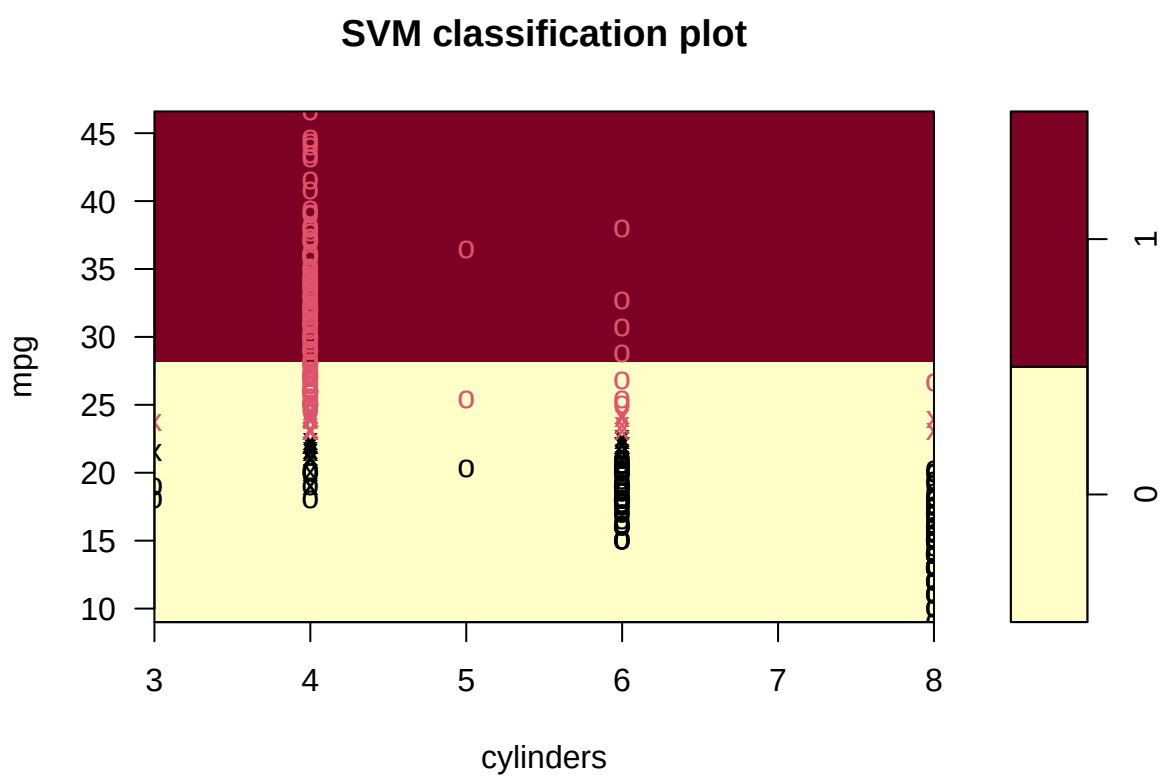
The best parameters from SVM with radial kernels are cost=1 and gamma=0.5.

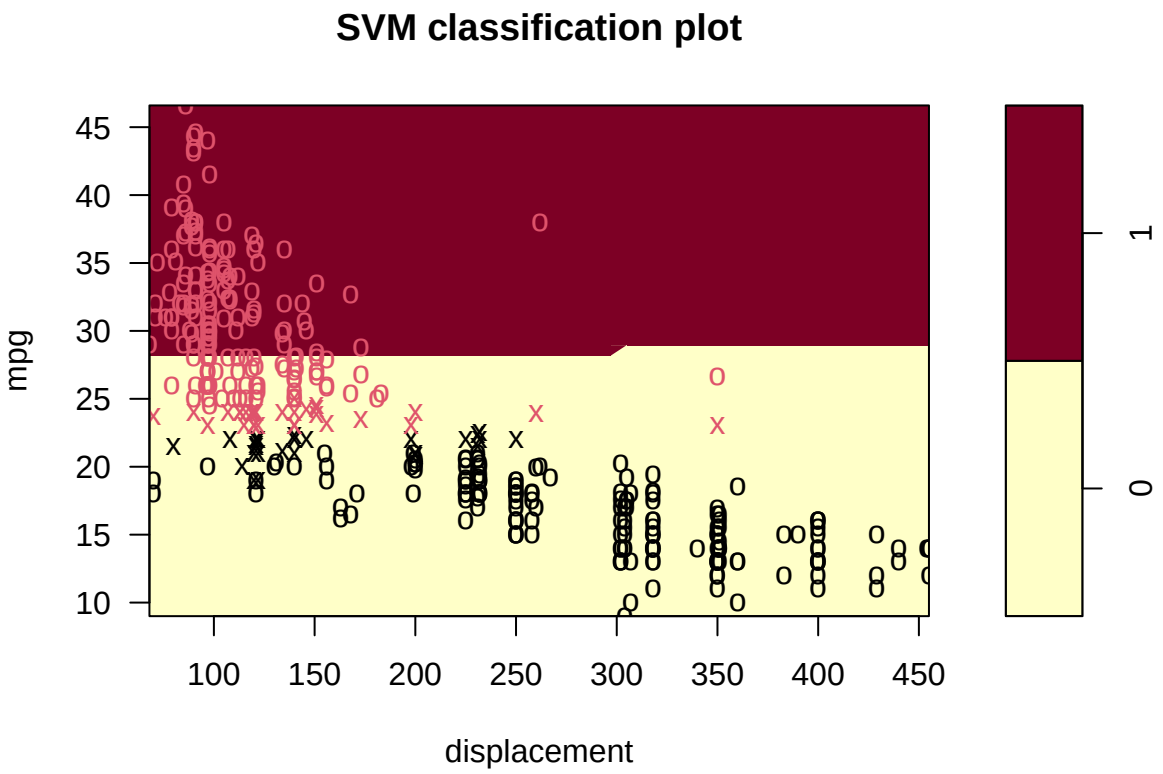
(d) Make some plots to back up your assertions in (b) and (c). Hint: In the lab, we used the plot() function for svm objects only in cases with p = 2. you can use the plot() function to create plots displaying pairs of variables at a time. Essentially, instead of typing where svmfit contains your fitted model and dat is a data frame containing your data, you can type in order to plot just the first and fourth variables. However, you must replace x1 and x4 with the correct variable names.

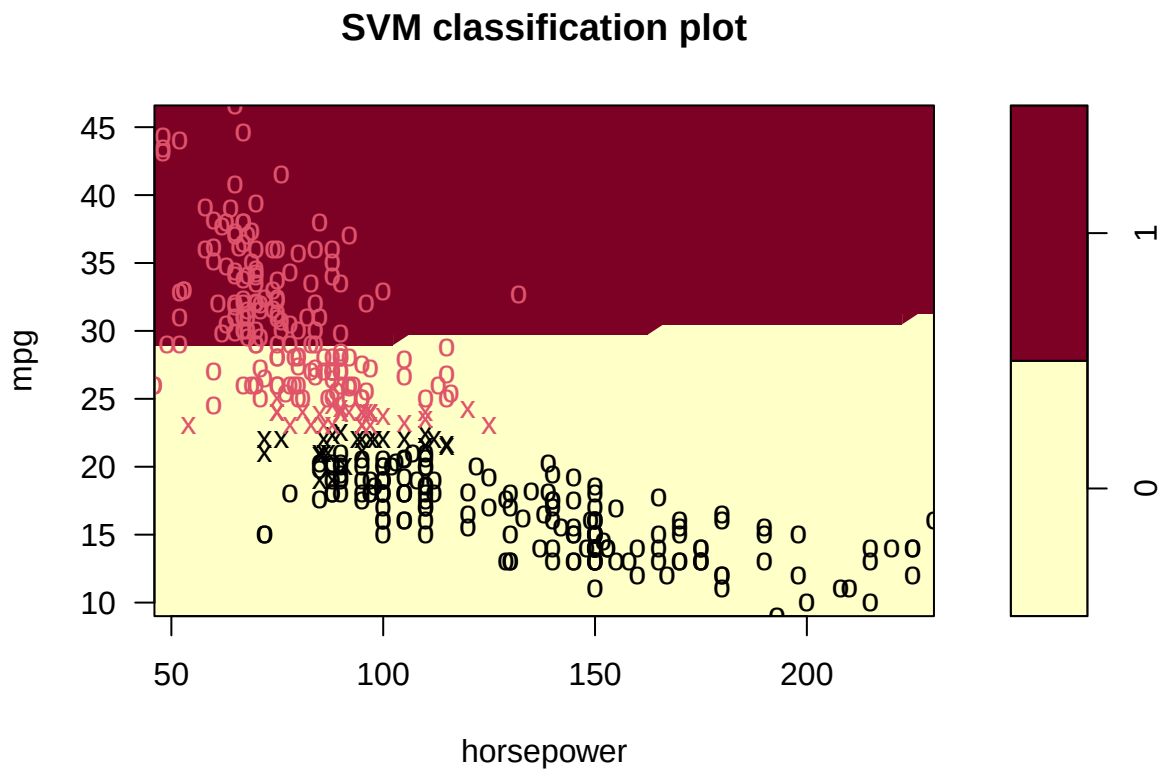
```
svm.linear <- svm(mpg01 ~ ., data = Auto, kernel = "linear", cost = 1)
svm.polynomial <- svm(mpg01 ~ ., data = Auto, kernel = "polynomial", cost = 1000, degree = 2)
svm.radial <- svm(mpg01 ~ ., data = Auto, kernel = "radial", cost = 1, gamma = 0.5)
```

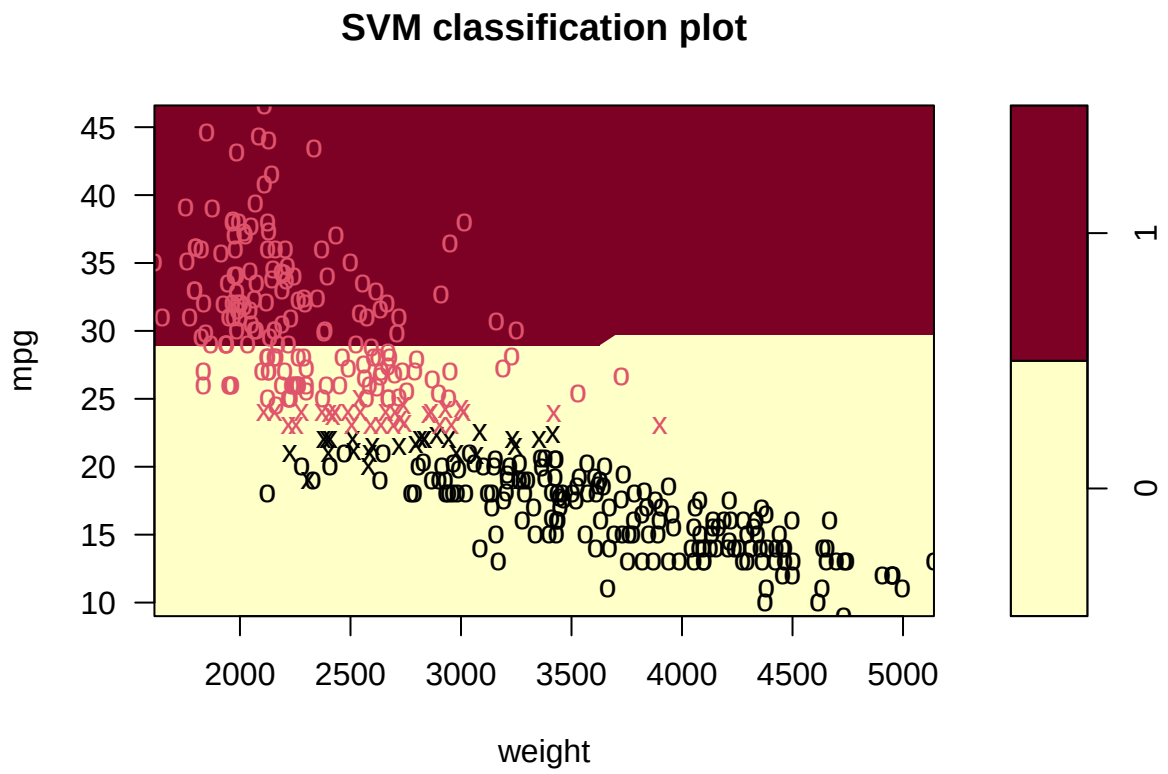
```
#create a function to plot different variables
plotpairs = function(fitted) {
    for (name in names(Auto)[!(names(Auto) %in% c("mpg", "mpg01", "name"))]) {
        plot(fitted, Auto, as.formula(paste("mpg~", name, sep = "")))
    }
}
```

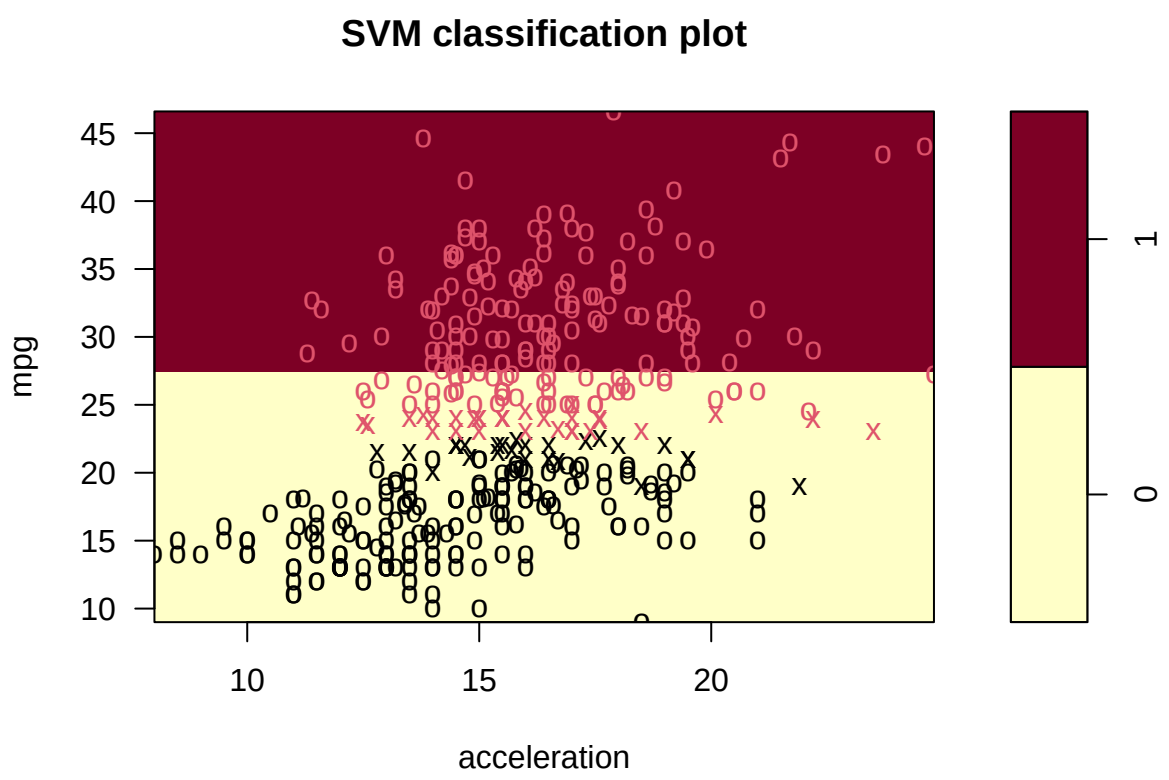
```
plotpairs(svm.linear)
```

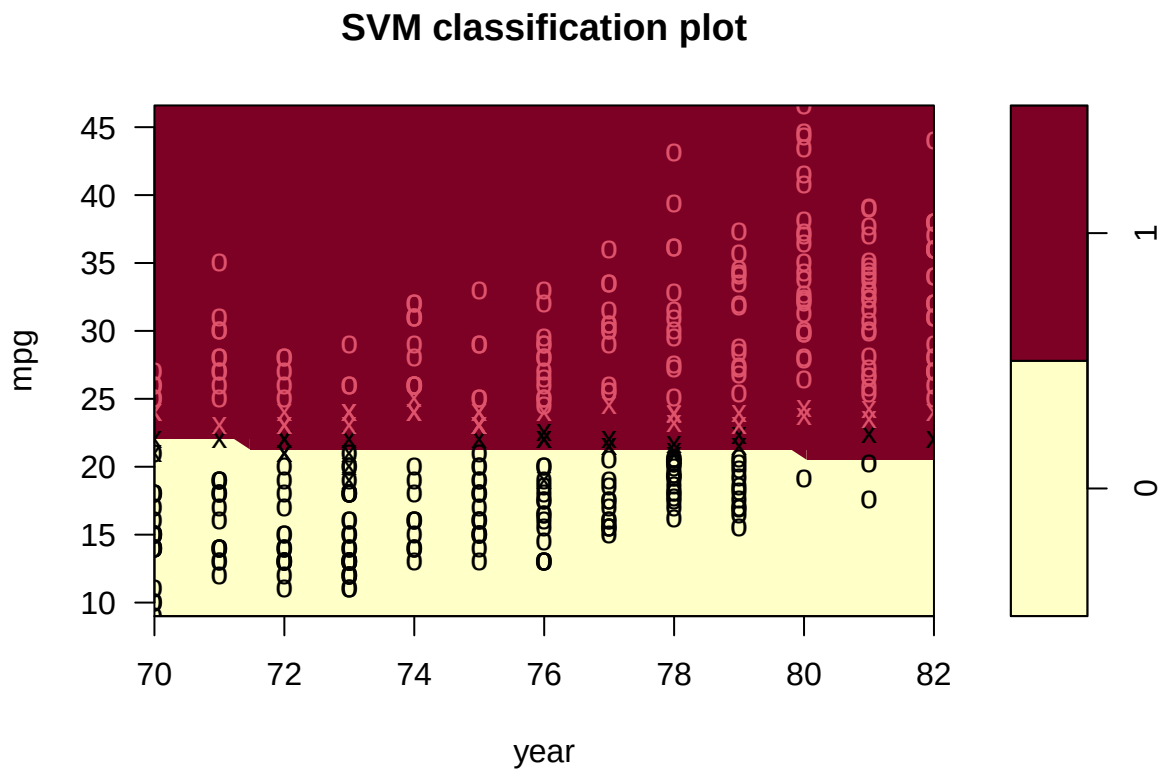

SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0

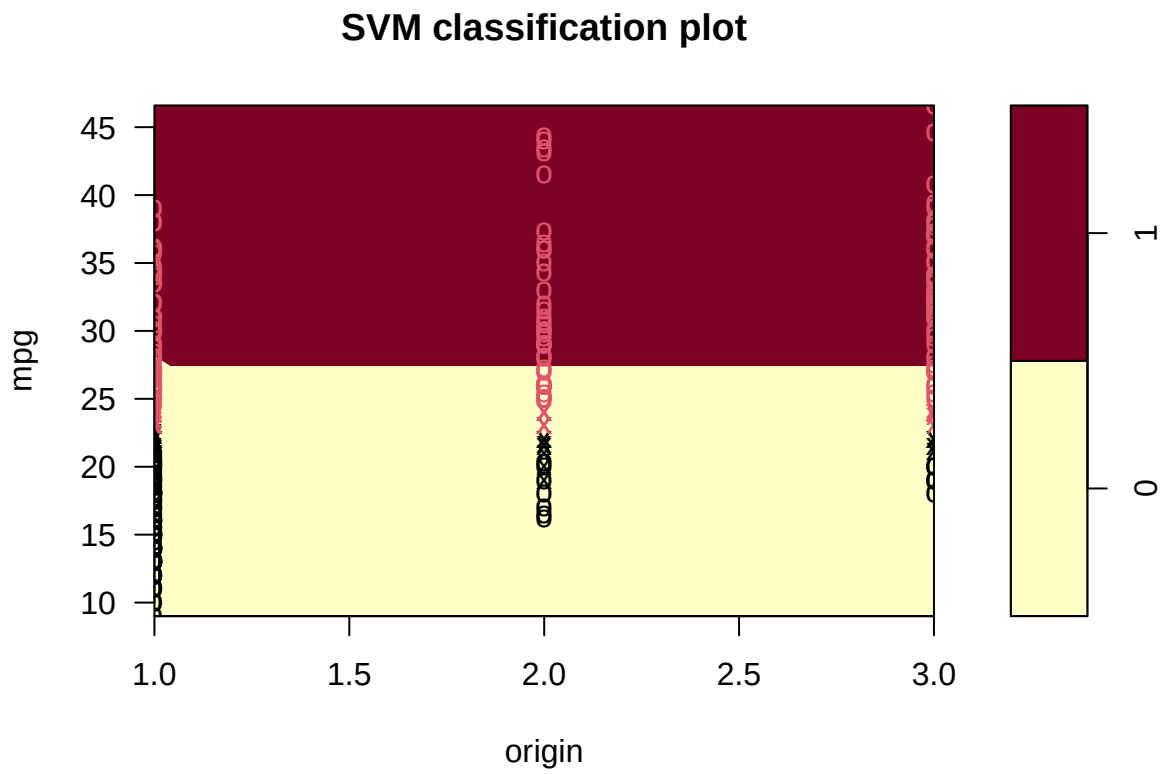

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0


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

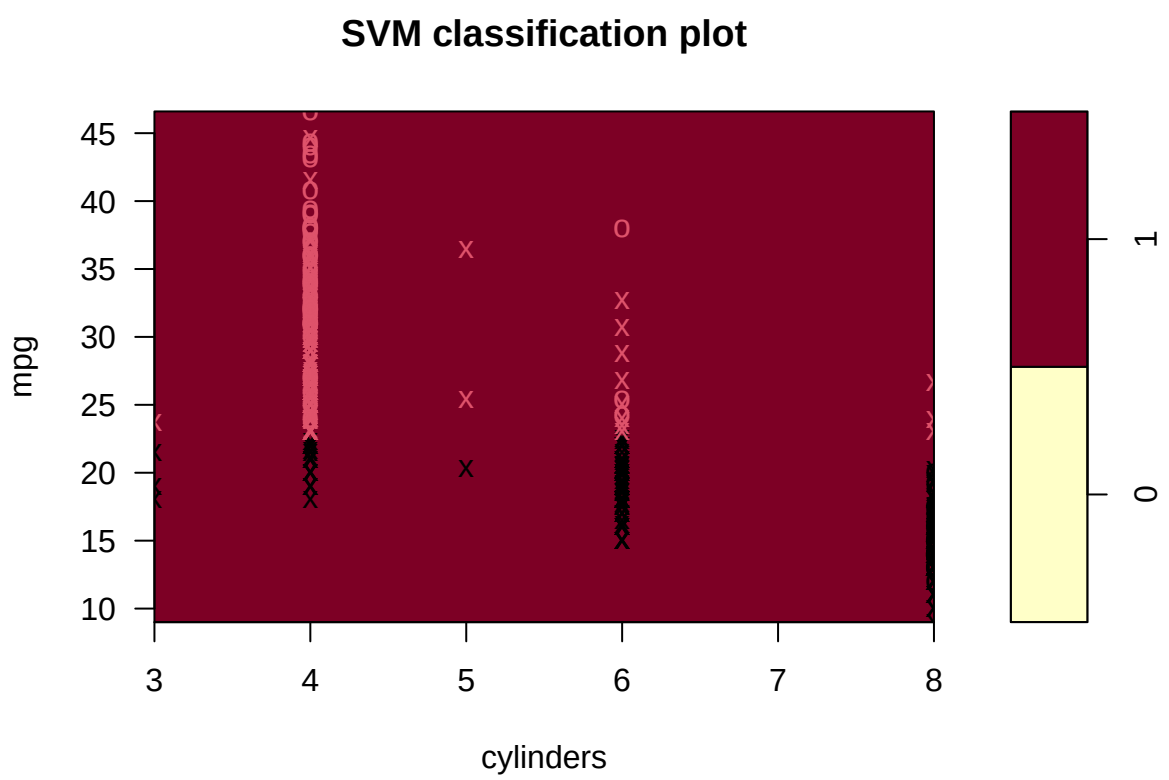

SVM classification plot
mpg
45
40
35
30
25
20
15
10
2000
2500
3000
3500
4000
4500
5000
weight
1
0

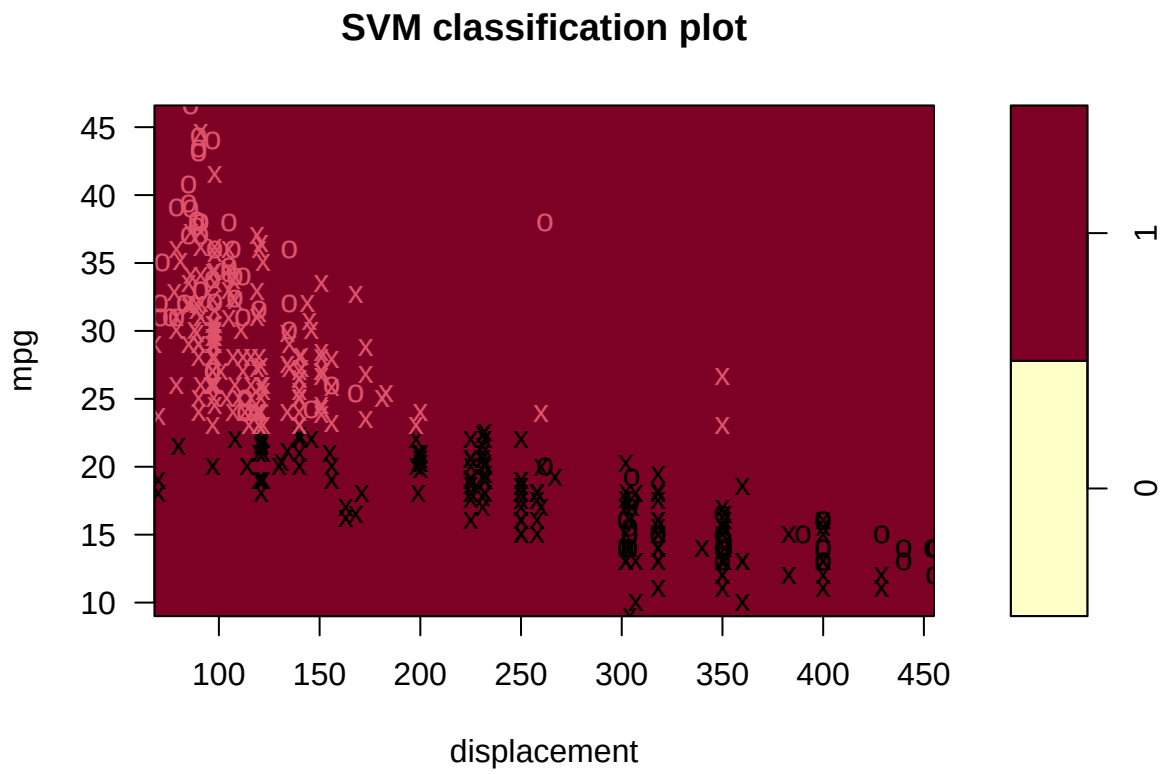

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0

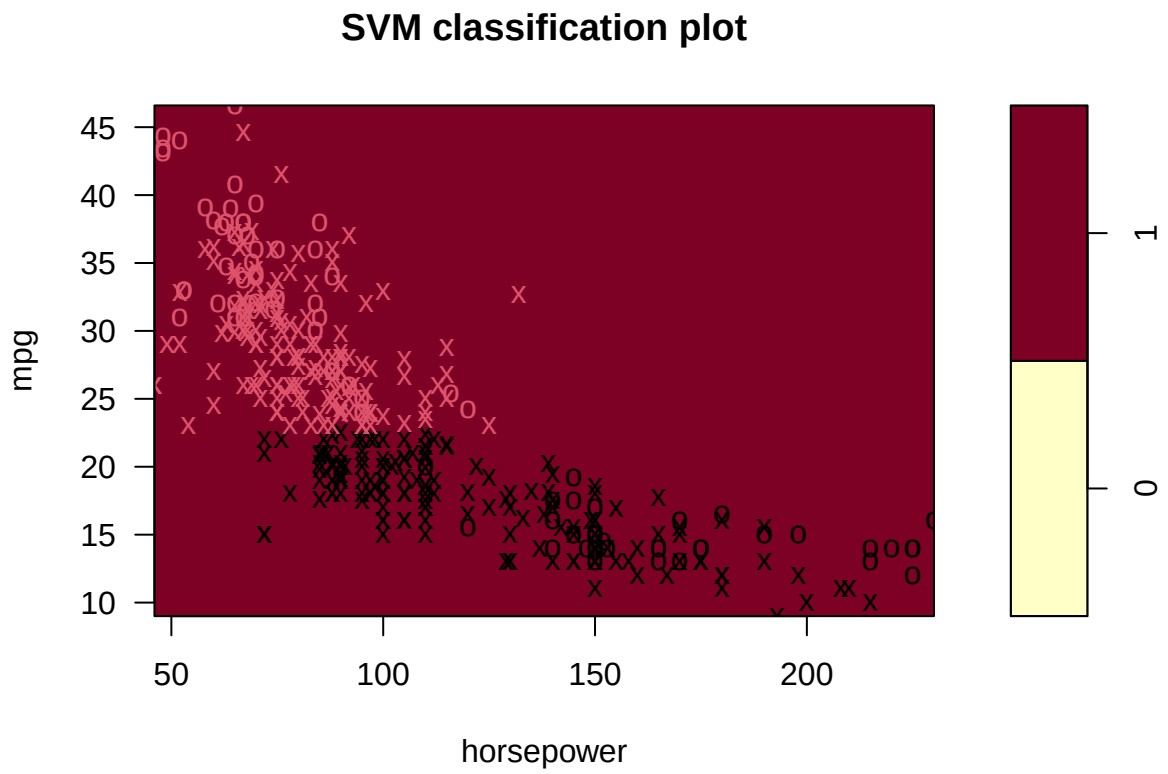

SVM classification plot
mpg
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
year
1
0

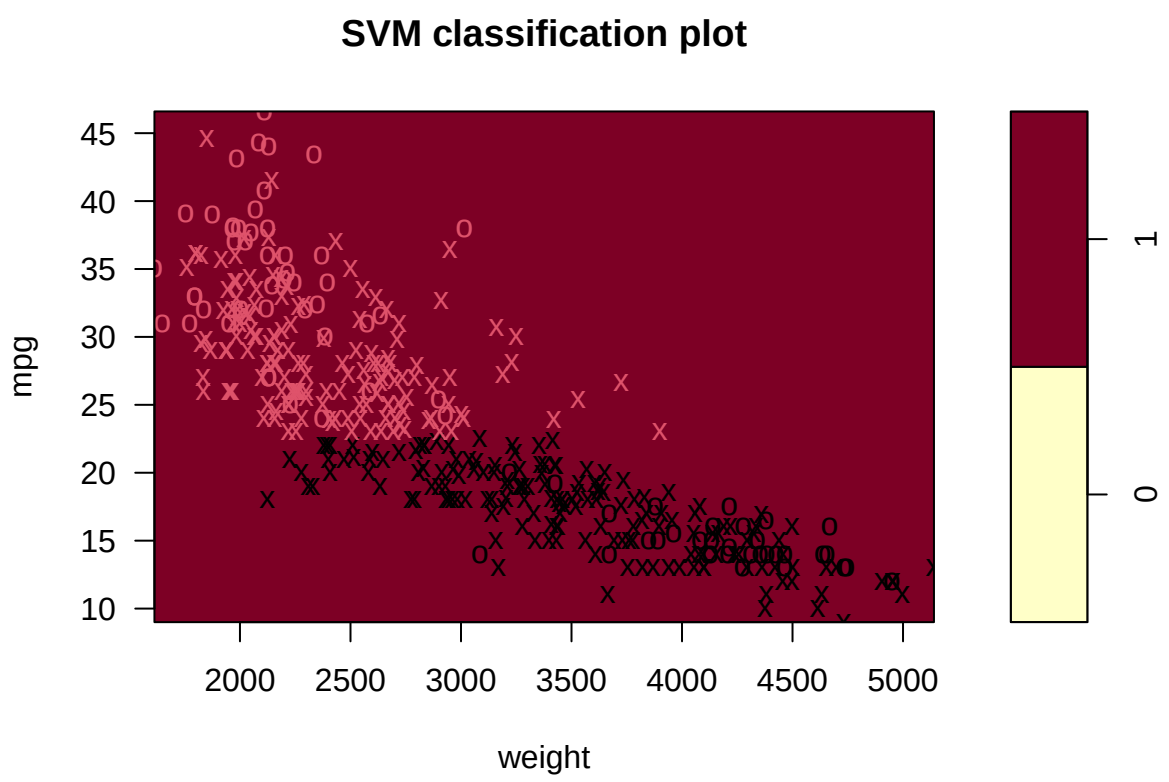


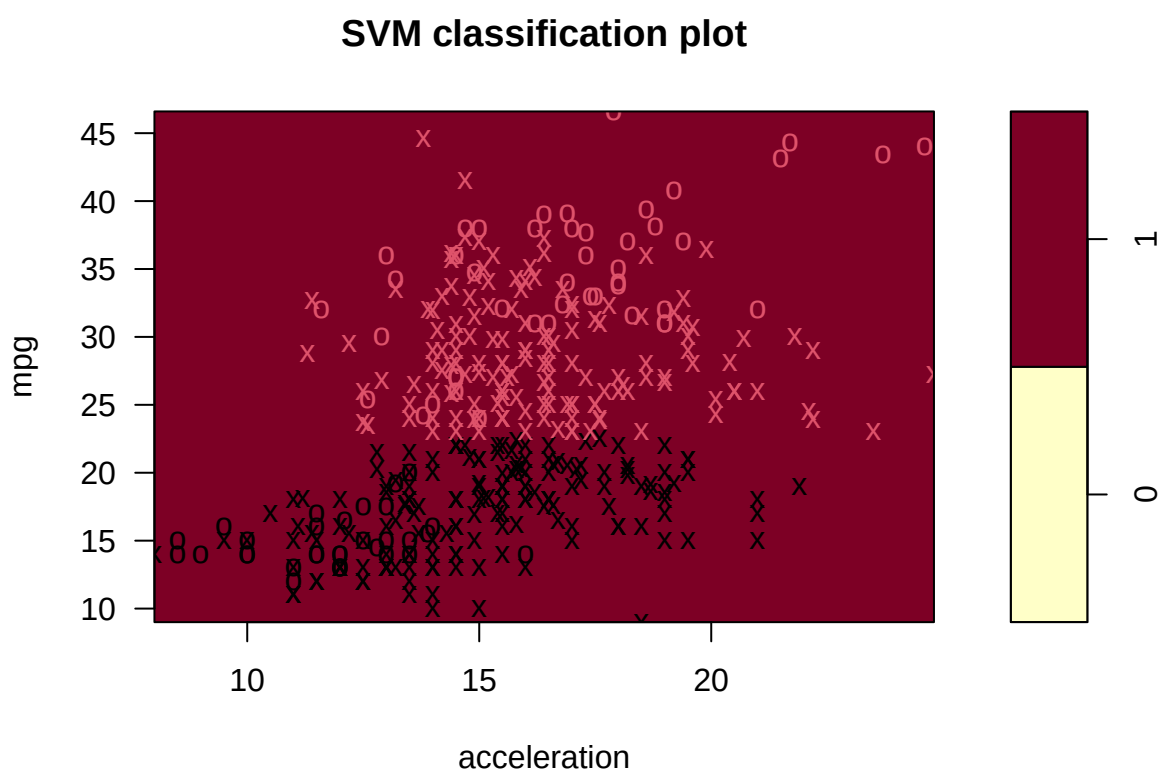

```
plotpairs(svm.polynomial)
```


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

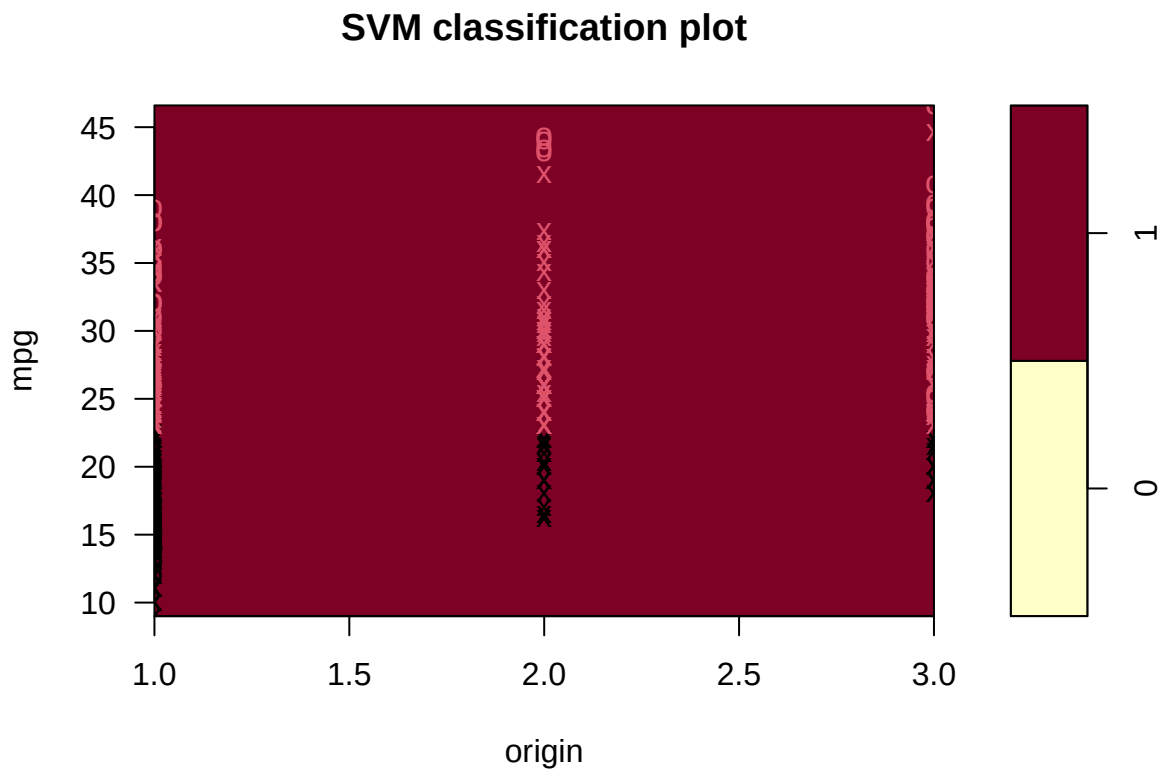

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0


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

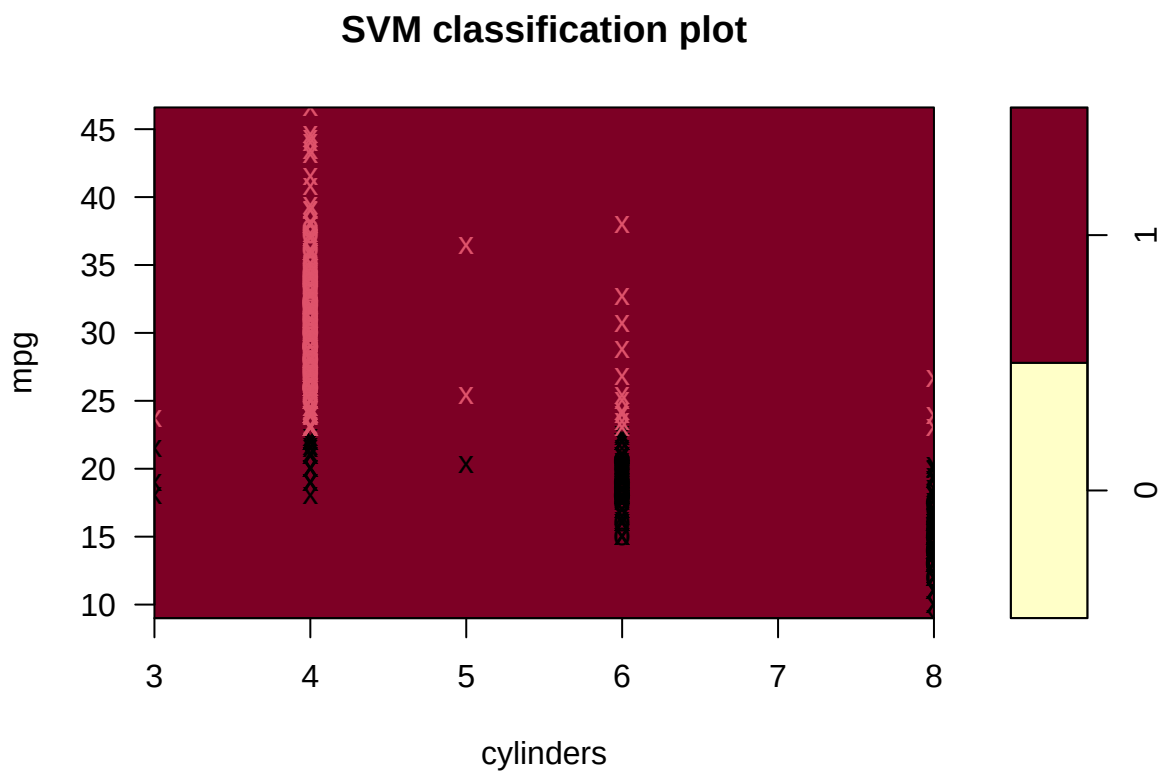

SVM classification plot
mpg
weight
45
40
35
30
25
20
15
10
2000
2500
3000
3500
4000
4500
5000
1
0

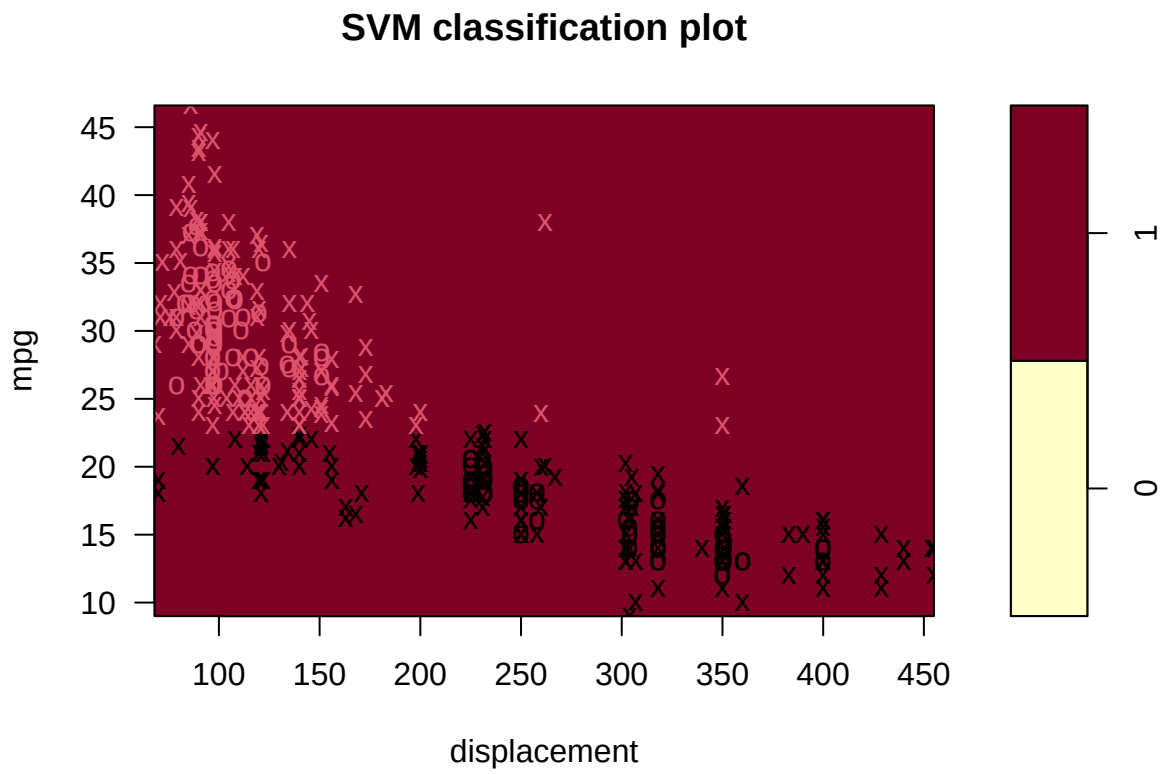

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0

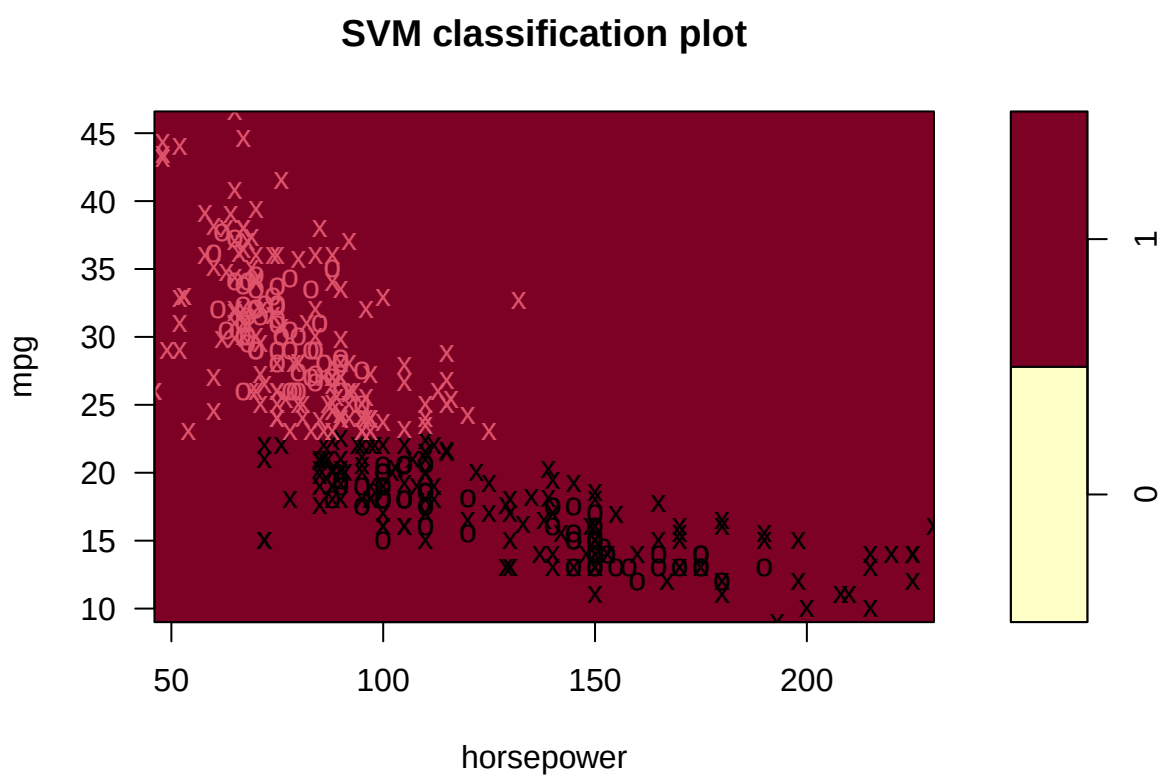

SVM classification plot
mpg
year
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
1
0

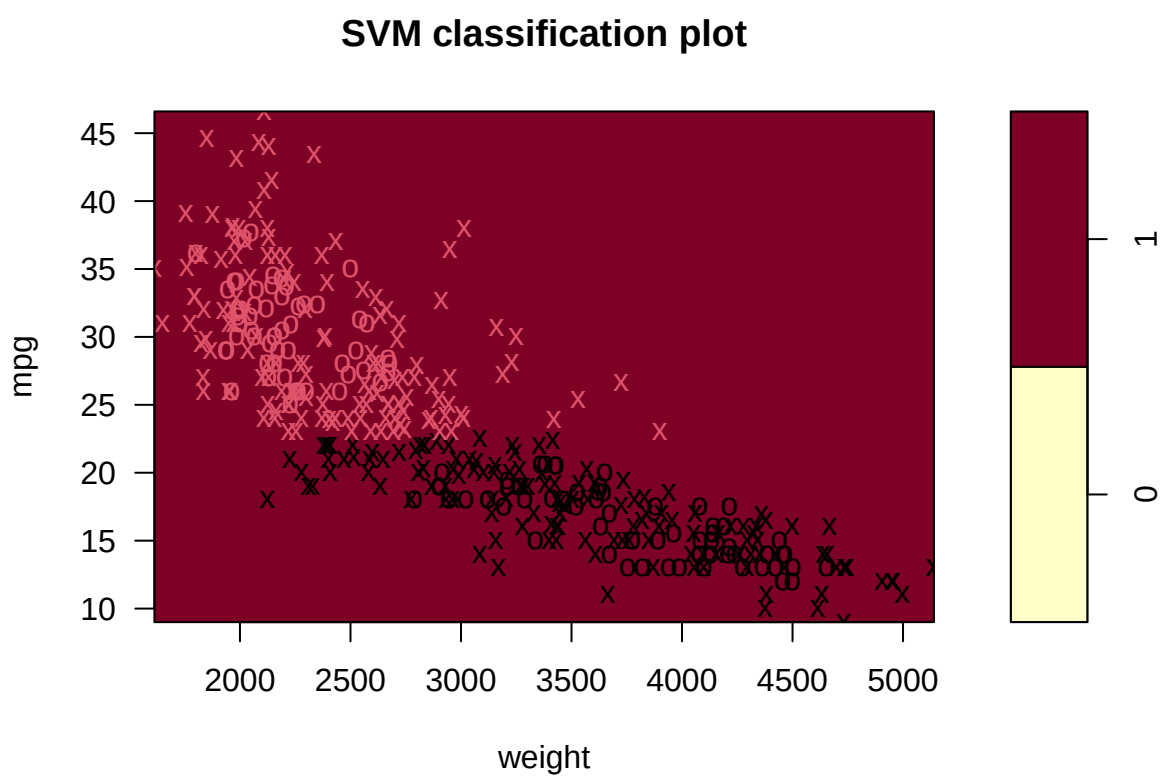


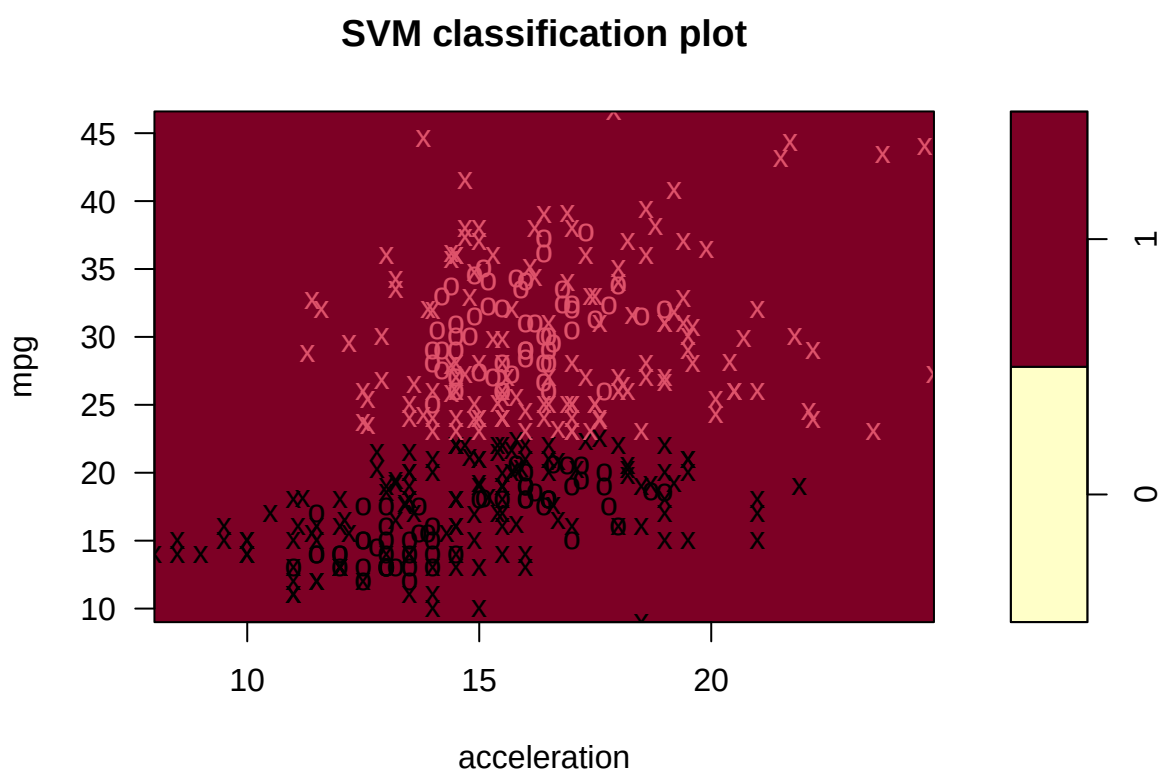

```
plotpairs(svm.radial)
```

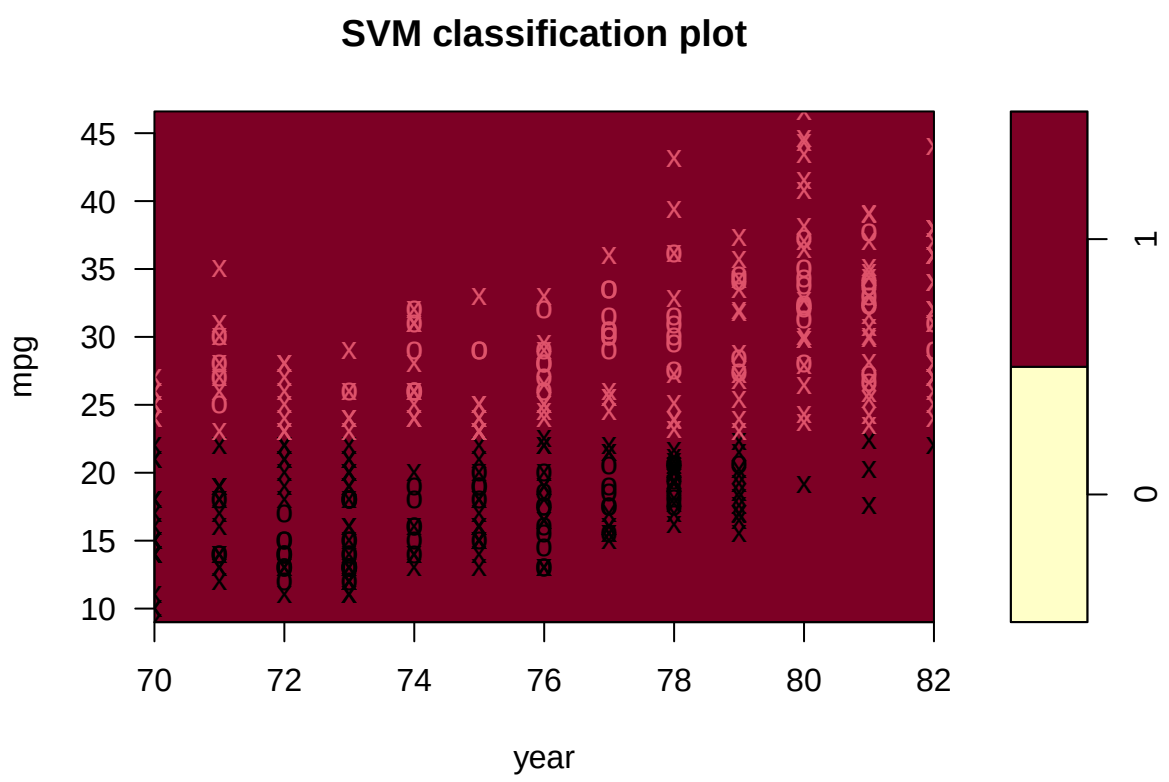

SVM classification plot
mpg
45
40
35
30
25
20
15
10
3
4
5
6
7
8
cylinders
1
0

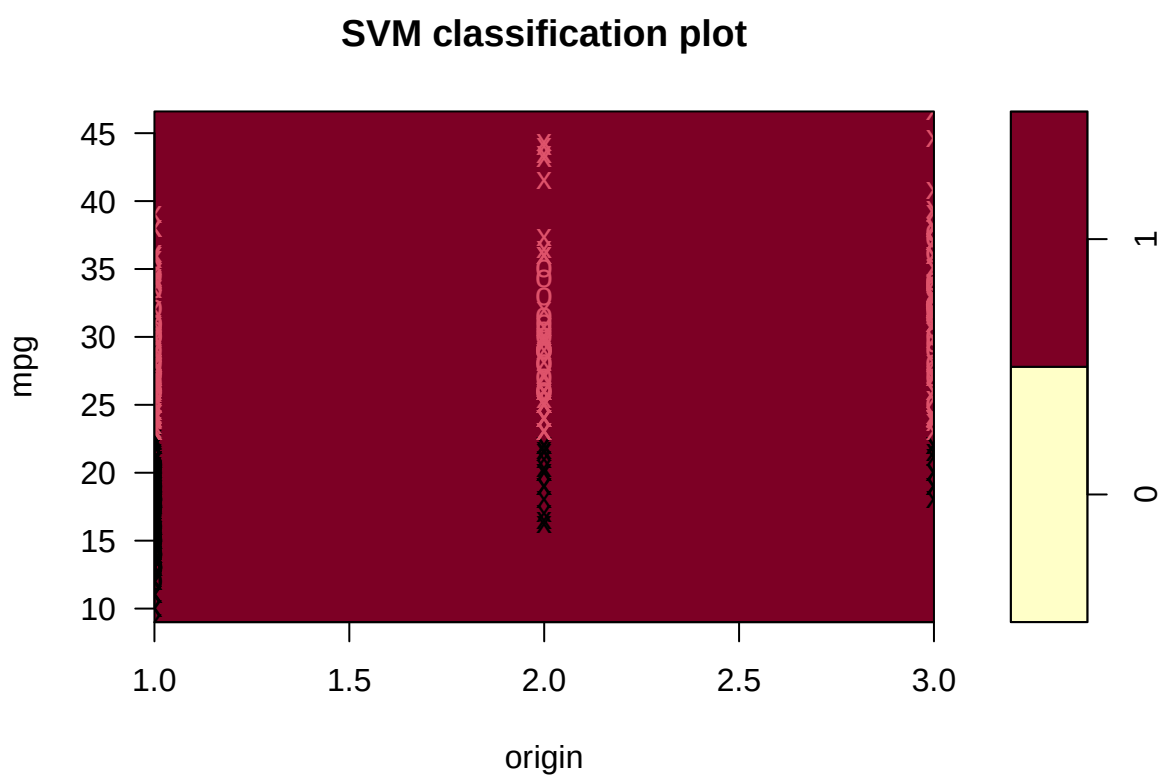

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0


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


SVM classification plot
mpg
weight
45
40
35
30
25
20
15
10
2000
2500
3000
3500
4000
4500
5000
1
0


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


SVM classification plot
mpg
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
year
1
0


SVM classification plot
mpg
45
40
35
30
25
20
15
10
1.0
1.5
2.0
2.5
3.0
origin
1
0