FE590. Assignment #4.

Enter Your Name Here, or "Anonymous" if you want to remain anonymous..

2019-05-08

Instructions

When you have completed the assignment, knit the document into a PDF file, and upload *both* the .pdf and .Rmd files to Canvas.

Note that you must have LaTeX installed in order to knit the equations below. If you do not have it installed, simply delete the questions below.

```
rm(list = ls())
CWID = 10442266 #Place here your Campus wide ID number, this will personalize
#your results, but still maintain the reproduceable nature of using seeds.
#If you ever need to reset the seed in this assignment, use this as your seed
#Papers that use -1 as this CWID variable will earn 0's so make sure you change
#this value before you submit your work.
personal = CWID %% 10000
set.seed(personal)
library(leaps)
library(splines)
library(gam)
library(MASS)
library(tree)
library(randomForest)
library(boot)
library(gbm)
library(class)
library(e1071)
library(glmnet)
```

Question 1:

In this assignment, you will be required to find a set of data to run regression on. This data set should be financial in nature, and of a type that will work with the models we have discussed this semester (hint: we didn't look at time series) You may not use any of the data sets in the ISLR package that we have been looking at all semester. Your data set that you choose should have both qualitative and quantitative variables. (or has variables that you can transform)

Provide a description of the data below, where you obtained it, what the variable names are and what it is describing.

In this project, we try to find some relationship betweeen the sugar futures and other assets including 25 stocks that from food section and sugar producer companies and three major sugar producing countries' currency exchange rates (India,Brazil and Thailand) and the ethanol futures which is the other mojor product using the same raw material of sugar. We try to predict the sugar futures daily return using these assets.

Athough according to the theory that the commodity futures price may be mainly infuluenced by the basic demand and supply situation which is more macro in sence, we can still try to find some formerly indicator to predic the short term price trend.

In practice, the sugar and ethanol futures data are downloaded from Quandl and other exchange ratees and stock prices are from Yahoo Finance. After clearing the data, we combined them to a whole data set (called "all_data") and each column in the data set represents one asset. The first 25 column names are stocks'

codes and next 3 are BRL, INR and THB which are exchange rates and the final two are ethanol and sugar denoting the futures price.

However the asset price data are not stationary. An efficient appraoch is to convert them to log return. Here we present part of our data.

```
# grader can change the direction to the file that contains the data set.
setwd("D:/Grad 2/590/590Assignments/HW4/data")
all_data <- read.csv("all_data.csv",row.names = "X")
all_data <- log(all_data[c(2:nrow(all_data)),]/all_data[c(1:nrow(all_data)-1),])
all_data <- cbind(all_data[c(1:dim(all_data)[1]-1),
                           c(1:dim(all_data)[2]-1)],
                  all_data[c(2:dim(all_data)[1]),
                           dim(all_data)[2]])
colnames(all_data)[ncol(all_data)] <- c("sugar")
print(head(all_data))
```

```
##                     CZZ          MDLZ            TR          HSY
## 2014-02-19 -0.037772637  0.0020453974 -0.0212917015  0.003674036
## 2014-02-20  0.025055263  0.0064009569  0.0003362333  0.012706605
## 2014-02-21  0.023609791 -0.0052341854 -0.0043792534  0.004420264
## 2014-02-24  0.012422383  0.0029114510  0.0026970794  0.003429597
## 2014-02-25 -0.014925431 -0.0131677830 -0.0010104075 -0.012102247
## 2014-02-26 -0.001672216 -0.0008842753  0.0013473418 -0.010639714
##                    RMCF        BRID         CALM         CVGW
## 2014-02-19  0.0086729454  0.00000000 -0.007279822 -0.001368563
## 2014-02-20  0.0008635417  0.00000000  0.012420405  0.034323739
## 2014-02-21  0.0025849661 -0.02268138 -0.003995825  0.015753473
## 2014-02-24  0.0187556692  0.01346473  0.007597323  0.006167890
## 2014-02-25 -0.0136054813  0.00000000 -0.013716998 -0.003241431
## 2014-02-26  0.0111733666 -0.01659789  0.005738220  0.010014578
##                     JVA          FARM          FLO          FDP
## 2014-02-19  0.025479085 -0.0286659011  0.006269856  0.005107162
## 2014-02-20 -0.012658397 -0.0051728779  0.007185741  0.015938087
## 2014-02-21 -0.003189795 -0.0070972622 -0.008628935 -0.017507109
## 2014-02-24  0.032995923  0.0418471119 -0.002893052  0.004698733
## 2014-02-25 -0.023456900 -0.0105287768 -0.005325660 -0.002738009
## 2014-02-26 -0.019169916  0.0009199633 -0.004280029  0.005468709
##                    HAIN         JJSF         JBSS          LANC
## 2014-02-19 -0.007326017 -0.008814437 -0.013239301 -0.0151321618
## 2014-02-20  0.022618089  0.015457723  0.013239301  0.0193015454
## 2014-02-21 -0.001598721 -0.004202218  0.003938059  0.0008992669
## 2014-02-24  0.010685585  0.013101214  0.005226418  0.0112828019
## 2014-02-25  0.011914247 -0.002628871  0.011660751 -0.0081422561
## 2014-02-26 -0.008527902  0.007864815  0.008976183  0.0011192103
##                    MTEX          MKC         NAII         NATR
## 2014-02-19  0.005282379 -0.0104873197 -0.043691659 -0.048506143
## 2014-02-20  0.032143985  0.0103360814  0.024929383  0.028741143
## 2014-02-21  0.010152234 -0.0024223055  0.000000000  0.008974709
## 2014-02-24  0.008048133  0.0042352714  0.001892148 -0.005118527
## 2014-02-25 -0.015652718 -0.0007552476  0.018727139 -0.001926153
## 2014-02-26 -0.110178276 -0.0087995645  0.011070224  0.005767286
##                    POST         RELV         RIBT         STKL
## 2014-02-19  0.007404169 -0.004662013 -0.015810606  0.013015368
## 2014-02-20  0.031175000  0.050124387  0.005958310  0.021322769
```

```
## 2014-02-21 -0.007878899 -0.008928631 -0.017982503 -0.009539023
## 2014-02-24  0.004559864 -0.013544225 -0.036965194  0.036595824
## 2014-02-25 -0.006319154  0.004535155  0.024794659 -0.024949319
## 2014-02-26  0.002110801 -0.018265348  0.002038737  0.000000000
##                     SJM          BRL           INR           THB
## 2014-02-19  0.002515136  0.001803730  0.0067741872  0.0080078031
## 2014-02-20  0.028170844  0.001214634  0.0008033904  0.0042406799
## 2014-02-21 -0.003465490 -0.008153704 -0.0020901527 -0.0017800937
## 2014-02-24  0.007729824 -0.012313858 -0.0019333501  0.0011359318
## 2014-02-25  0.003135899 -0.001838825  0.0009349442 -0.0027038977
## 2014-02-26 -0.004048176  0.001069496 -0.0014510748  0.0006766217
##                  ethanol        sugar
## 2014-02-19  0.0029585820 -0.007929288
## 2014-02-20 -0.0010049361  0.023601701
## 2014-02-21 -0.0014796842  0.040439146
## 2014-02-24  0.0061411471 -0.004028782
## 2014-02-25  0.0114721773 -0.002887672
## 2014-02-26 -0.0009315867  0.008638119
```

Now, we randomly divide them to two sets. Then we can implement different methods on training set and test them in test set.

```
set.seed(1)
train <- sample(nrow(all_data),floor(nrow(all_data)/2))
trainset <- all_data[train,]
testset <- all_data[-train,]
```

Question 2:

Pick a quantitative variable and fit at least four different models in order to predict that variable using the other predictors. Determine which of the models is the best fit. You will need to provide strong reasons as to why the particular model you chose is the best one. You will need to confirm the model you have selected provides the best fit and that you have obtained the best version of that particular model (i.e. subset selection or validation for example). You need to convince the grader that you have chosen the best model.

To conduct regression on the quantitative variable, we use 4 method to do it.

1, Multiple Linear Regression with subset selection and lasso/ridge modification.

2, Polynomial Regression with subset selection.

3, GAM with different model selection.

4, Regression Tree with parameters selected by cross-validation.

Multiple Linear Regression

Now we perform the multiple linear regression. First of all, we do the subset selection to determine which variable is more likely to be included in the model.

```
# Multiple Linear Regression (lasso and ridge)
# subset selection
subsets=regsubsets(sugar~.,data=trainset,method="exhaustive",nvmax=30)
summary(subsets)
```

```
## Subset selection object
## Call: regsubsets.formula(sugar ~ ., data = trainset, method = "exhaustive",
##     nvmax = 30)
```

```
## 29 Variables  (and intercept)
##         Forced in Forced out
## CZZ         FALSE      FALSE
## MDLZ        FALSE      FALSE
## TR          FALSE      FALSE
## HSY         FALSE      FALSE
## RMCF        FALSE      FALSE
## BRID        FALSE      FALSE
## CALM        FALSE      FALSE
## CVGW        FALSE      FALSE
## JVA         FALSE      FALSE
## FARM        FALSE      FALSE
## FLO         FALSE      FALSE
## FDP         FALSE      FALSE
## HAIN        FALSE      FALSE
## JJSF        FALSE      FALSE
## JBSS        FALSE      FALSE
## LANC        FALSE      FALSE
## MTEX        FALSE      FALSE
## MKC         FALSE      FALSE
## NAII        FALSE      FALSE
## NATR        FALSE      FALSE
## POST        FALSE      FALSE
## RELV        FALSE      FALSE
## RIBT        FALSE      FALSE
## STKL        FALSE      FALSE
## SJM         FALSE      FALSE
## BRL         FALSE      FALSE
## INR         FALSE      FALSE
## THB         FALSE      FALSE
## ethanol     FALSE      FALSE
## 1 subsets of each size up to 29
## Selection Algorithm: exhaustive
##           CZZ MDLZ TR  HSY RMCF BRID CALM CVGW JVA FARM FLO FDP HAIN JJSF
## 1  ( 1 )  " " " "  " " " " " "  " "  " "  " " " " "*"  " " " " " " " "  " "
## 2  ( 1 )  " " " "  " " " " " "  " "  "*"  " " " " "*"  " " " " " " " "  " "
## 3  ( 1 )  " " " "  " " " " " "  " "  "*"  " " " " "*"  " " " " " " " "  " "
## 4  ( 1 )  " " " "  " " " " " "  " "  "*"  " " " " "*"  " " " " " " " "  " "
## 5  ( 1 )  " " " "  " " " " " "  " "  "*"  " " " " "*"  " " "*" " " " "  " "
## 6  ( 1 )  " " " "  " " " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 7  ( 1 )  " " " "  " " " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 8  ( 1 )  " " " "  " " " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 9  ( 1 )  " " " "  "*" " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 10  ( 1 ) " " " "  "*" " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 11  ( 1 ) " " " "  "*" " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 12  ( 1 ) " " " "  "*" " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 13  ( 1 ) " " " "  "*" " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 14  ( 1 ) " " " "  "*" " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 15  ( 1 ) " " " "  "*" " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 16  ( 1 ) " " " "  "*" " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 17  ( 1 ) " " " "  "*" " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 18  ( 1 ) " " "*"  "*" " " " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 19  ( 1 ) " " "*"  "*" "*" " "  " "  "*"  " " " " "*"  " " "*" "*" " "  " "
## 20  ( 1 ) " " "*"  "*" "*" " "  " "  "*"  " " " " "*"  "*" "*" "*" " "  " "
```

```
## 21  ( 1 ) "*" "*"  "*" "*" " "  " "  "*"  " "  " " "*"  "*" "*" "*"  " "
## 22  ( 1 ) "*" "*"  "*" "*" " "  " "  "*"  " "  " " "*"  "*" "*" "*"  " "
## 23  ( 1 ) "*" "*"  "*" "*" " "  " "  "*"  "*"  " " "*"  "*" "*" "*"  " "
## 24  ( 1 ) "*" "*"  "*" "*" " "  " "  "*"  "*"  " " "*"  "*" "*" "*"  " "
## 25  ( 1 ) "*" "*"  "*" "*" " "  " "  "*"  "*"  " " "*"  "*" "*" "*"  "*"
## 26  ( 1 ) "*" "*"  "*" "*" " "  "*"  "*"  "*"  " " "*"  "*" "*" "*"  "*"
## 27  ( 1 ) "*" "*"  "*" "*" "*"  "*"  "*"  "*"  " " "*"  "*" "*" "*"  "*"
## 28  ( 1 ) "*" "*"  "*" "*" "*"  "*"  "*"  "*"  " " "*"  "*" "*" "*"  "*"
## 29  ( 1 ) "*" "*"  "*" "*" "*"  "*"  "*"  "*"  "*" "*"  "*" "*" "*"  "*"
##           JBSS LANC MTEX MKC NAII NATR POST RELV RIBT STKL SJM BRL INR THB
## 1  ( 1 )  " "  " "  " "  " " " "  " "  " "  " "  " "  " "  " " " " " " " "
## 2  ( 1 )  " "  " "  " "  " " " "  " "  " "  " "  " "  " "  " " " " " " " "
## 3  ( 1 )  " "  " "  " "  " " " "  "*"  " "  " "  " "  " "  " " " " " " " "
## 4  ( 1 )  " "  " "  " "  " " " "  "*"  " "  " "  " "  " "  " " " " " "*" "
## 5  ( 1 )  " "  " "  " "  " " " "  "*"  " "  " "  " "  " "  " " " " " "*" "
## 6  ( 1 )  " "  " "  " "  " " " "  "*"  " "  " "  " "  " "  " " " " " "*" "
## 7  ( 1 )  " "  " "  " "  " " " "  "*"  " "  " "  "*"  " "  " " " " " "*" "
## 8  ( 1 )  " "  " "  " "  " " "*"  "*"  " "  " "  "*"  " "  " " " " " "*" "
## 9  ( 1 )  " "  " "  " "  " " "*"  "*"  " "  " "  "*"  " "  " " " " " "*" "
## 10  ( 1 ) " "  " "  " "  " " "*"  "*"  " "  " "  "*"  "*"  " " " " " "*" "
## 11  ( 1 ) " "  " "  " "  " " "*"  "*"  " "  " "  "*"  "*"  "*" " " "*" "
## 12  ( 1 ) " "  " "  " "  " " "*"  "*"  " "  " "  "*"  " "  "*" "*" "*" "*"
## 13  ( 1 ) " "  " "  " "  " " "*"  "*"  " "  " "  "*"  "*"  "*" "*" "*" "*"
## 14  ( 1 ) " "  " "  " "  " " "*"  "*"  "*"  " "  "*"  "*"  "*" "*" "*" "*"
## 15  ( 1 ) " "  " "  " "  " " "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 16  ( 1 ) "*"  " "  " "  " " "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 17  ( 1 ) "*"  " "  " "  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 18  ( 1 ) "*"  " "  " "  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 19  ( 1 ) "*"  " "  " "  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 20  ( 1 ) "*"  " "  " "  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 21  ( 1 ) "*"  " "  " "  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 22  ( 1 ) "*"  " "  "*"  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 23  ( 1 ) "*"  " "  "*"  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 24  ( 1 ) "*"  "*"  "*"  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 25  ( 1 ) "*"  "*"  "*"  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 26  ( 1 ) "*"  "*"  "*"  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 27  ( 1 ) "*"  "*"  "*"  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 28  ( 1 ) "*"  "*"  "*"  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
## 29  ( 1 ) "*"  "*"  "*"  "*" "*"  "*"  "*"  "*"  "*"  "*"  "*" "*" "*" "*"
##           ethanol
## 1  ( 1 )  " "
## 2  ( 1 )  " "
## 3  ( 1 )  " "
## 4  ( 1 )  " "
## 5  ( 1 )  " "
## 6  ( 1 )  " "
## 7  ( 1 )  " "
## 8  ( 1 )  " "
## 9  ( 1 )  " "
## 10  ( 1 ) " "
## 11  ( 1 ) " "
## 12  ( 1 ) " "
## 13  ( 1 ) " "
## 14  ( 1 ) " "
```

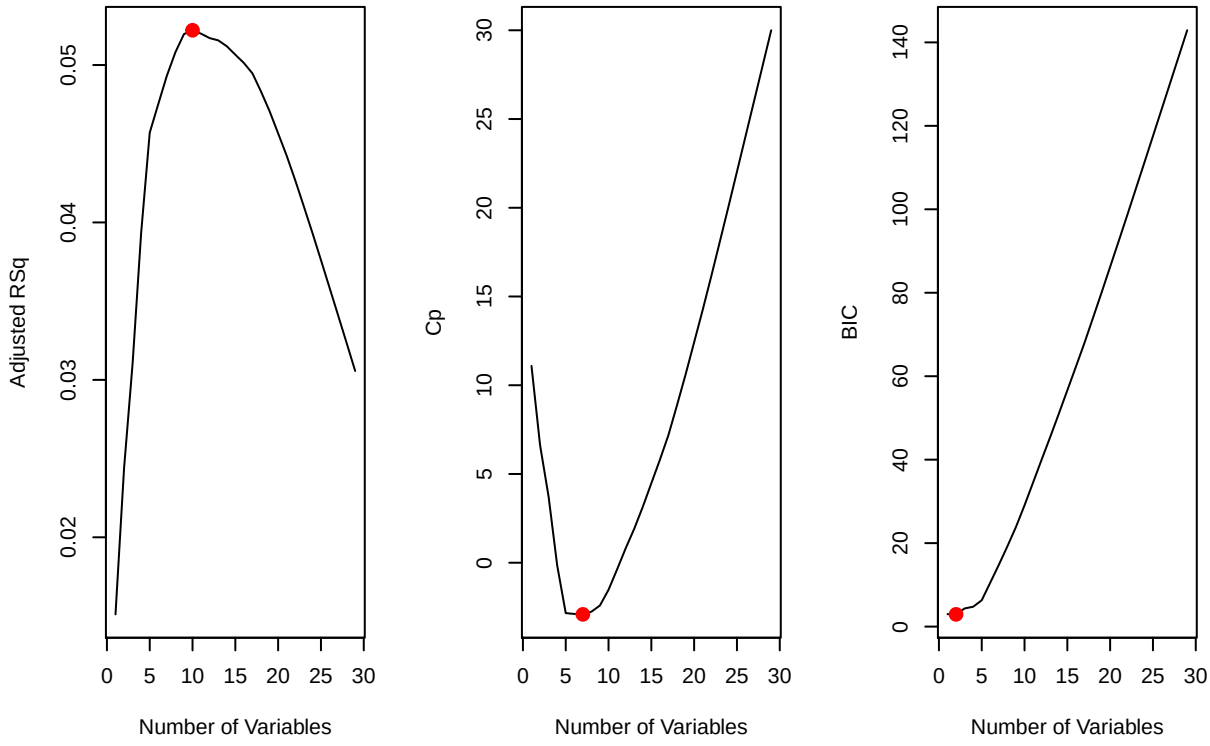
```
## 15  ( 1 ) " "
## 16  ( 1 ) " "
## 17  ( 1 ) " "
## 18  ( 1 ) " "
## 19  ( 1 ) " "
## 20  ( 1 ) " "
## 21  ( 1 ) " "
## 22  ( 1 ) " "
## 23  ( 1 ) " "
## 24  ( 1 ) " "
## 25  ( 1 ) " "
## 26  ( 1 ) " "
## 27  ( 1 ) " "
## 28  ( 1 ) "*"
## 29  ( 1 ) "*"
```

In order to determin the number of variables, we plot several indicators against the number of variables to find the best one.

```
# number of the variable
regfit.summary = summary(subsets)
par(mfrow=c(1,3))
plot(regfit.summary$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="l")
points(which.max(regfit.summary$adjr2)
       ,regfit.summary$adjr2[which.max(regfit.summary$adjr2)]
       ,col="red",cex=2,pch=20)

plot(regfit.summary$cp,xlab="Number of Variables",ylab="Cp",type="l")
points(which.min(regfit.summary$cp)
       ,regfit.summary$cp[which.min(regfit.summary$cp)]
       ,col="red",pch=20,cex=2)

plot(regfit.summary$bic,xlab="Number of Variables",ylab="BIC",type='l')
points(which.min(regfit.summary$bic)
       ,regfit.summary$bic[which.min(regfit.summary$bic)]
       ,col="red",pch=20,cex=2)
```



These plots suggest different results, Here we select 8 variables which is the mean of the first two plot suggest. The 8 varaibles contain 7 stocks and INR/USD exchange rate. After regression, we give the summary of the model and the test MSE.

```
fit1 = lm(sugar~FARM+CALM+NATR+INR+FDP+HAIN+RIBT+NAII,data = trainset)
summary((fit1))
```

```
##
## Call:
## lm(formula = sugar ~ FARM + CALM + NATR + INR + FDP + HAIN +
##     RIBT + NAII, data = trainset)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.166274 -0.012316 -0.000145  0.011629  0.119421
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.0010502  0.0009351  -1.123  0.26188
## FARM         0.1290178  0.0436194   2.958  0.00323 **
## CALM        -0.1310944  0.0430474  -3.045  0.00243 **
## NATR         0.0719882  0.0309433   2.326  0.02035 *
## INR          0.5915361  0.2292914   2.580  0.01014 *
## FDP          0.1061830  0.0554531   1.915  0.05602 .
## HAIN         0.0594723  0.0405754   1.466  0.14328
## RIBT        -0.0367956  0.0259719  -1.417  0.15711
## NAII        -0.0550974  0.0401010  -1.374  0.17000
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02227 on 563 degrees of freedom
## Multiple R-squared:  0.06411,    Adjusted R-squared:  0.05081
## F-statistic: 4.821 on 8 and 563 DF,  p-value: 9.31e-06
```

```
prediction1 = predict(fit1,testset)
mse1 = mean((prediction1-testset$sugar)^2)
```

```
print(mse1)
```

```
## [1] 0.0004866911
```

As we can see from the result, the last 3 variable is not statistical significant and the R^2 is not good enough actually, and the MSE is about 0.000487.

In the rest of this section, we try to use the Lasso and Ridge Regression approacheds to get rid of the subset selection. By using the Lasso and Ridge, we can directly perform a regression on the whole predictors and the model will itself determine which variable to choose. The shrinkage parameter is determined by the CV method, we use the beat λ to do the prediction on test set.

```
# lasso
x_train = as.matrix(trainset[,-ncol(trainset)])
y_train = trainset$sugar
x_test = as.matrix(testset[,-ncol(testset)])
y_test = testset$sugar

lasso.mod = glmnet(x_train,y_train,alpha = 1)
set.seed(1)
cv.out = cv.glmnet(x_train,y_train,alpha=1)
bestlam = cv.out$lambda.min
lasso.pred = predict(lasso.mod,s = bestlam,newx = x_test)
mse11 = mean((lasso.pred-y_test)^2)
lasso.coeff = predict(cv.out,type="coefficients",s = bestlam)
print(lasso.coeff)
```

```
## 30 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -0.001035505
## CZZ          .
## MDLZ         .
## TR           .
## HSY          .
## RMCF         .
## BRID         .
## CALM        -0.068413472
## CVGW         .
## JVA          .
## FARM         0.088377431
## FLO          .
## FDP          0.055342572
## HAIN         0.022980985
## JJSF         .
## JBSS         .
## LANC         .
## MTEX         .
```

```
## MKC           .
## NAII       -0.014430435
## NATR        0.037610493
## POST          .
## RELV          .
## RIBT       -0.009418208
## STKL          .
## SJM           .
## BRL           .
## INR         0.300198252
## THB         0.014492352
## ethanol       .
```

```
print(mse11)
```

```
## [1] 0.0004686722
```

We can see that the Lasso approach gives almost same subset selection with previous method, however the test MSE indeed improve which is 0.000469.

An alternative way to do the same thing is Ridge Regression. We don't present too much detial of this method, and only give the final test MSE.

```
# ridge
ridge.mod = glmnet(x_train,y_train,alpha = 0)
set.seed(1)
cv.out2 = cv.glmnet(x_train,y_train,alpha=0)
bestlam2 = cv.out2$lambda.min
ridge.pred = predict(ridge.mod,s = bestlam2,newx = x_test)
mse12 = mean((ridge.pred-y_test)^2)
print(mse12)
```

```
## [1] 0.0004630361
```

The ridge Reggression gives us the test MSE of 0.000463 which is the best one so far.

Polynomial Regression

To simplify the problem, we only try the degree up to 3. What's more, we use again the subset selection to select the best variables with proper power. We don't present the summary of the subset selection result here because it is too complicated. The final polynomial model is conducted at the final.

```
subsets2=regsubsets(sugar~poly(CZZ,3,raw = T)+poly(MDLZ,3,raw = T)
                    +poly(TR,3,raw = T)+poly(HSY,3,raw = T)+poly(RMCF,3,raw = T)
                    +poly(BRID,3,raw = T)
                    +poly(CALM,3,raw = T)+poly(CVGW,3,raw = T)
                    +poly(JVA,3,raw = T)+poly(FARM,3,raw = T)
                    +poly(FLO,3,raw = T)+poly(FDP,3,raw = T)
                    +poly(HAIN,3,raw = T)+ poly(JJSF,3,raw = T)
                    + poly(JBSS,3,raw = T)+ poly(LANC,3,raw = T)
                    + poly(MTEX,3,raw = T)+ poly(MKC,3,raw = T)
                    + poly(NAII,3,raw = T)+ poly(NATR,3,raw = T)+
                      poly(POST,3,raw = T)+ poly(RELV,3,raw = T)
                    + poly(RIBT,3,raw = T)+ poly(STKL,3,raw = T)
                    + poly(SJM,3,raw = T)+ poly(BRL,3,raw = T)
                    + poly(INR,3,raw = T)+ poly(THB,3,raw = T)
                    + poly(ethanol,3,raw = T),data=trainset,method="forward",nvmax=20)
```

```r
# choose FARM, CALM, CALM^2, NATR, INR, FDP, FDP^2, BRL^2 as the factors.
fit2 = lm(sugar~FARM+poly(CALM,2,raw=T)+NATR+INR+poly(FDP,2,raw = T)+I(BRL^2),data = trainset)
summary(fit2)
```

```
##
## Call:
## lm(formula = sugar ~ FARM + poly(CALM, 2, raw = T) + NATR + INR +
##     poly(FDP, 2, raw = T) + I(BRL^2), data = trainset)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.159561 -0.012567 -0.000067  0.011494  0.120650
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -0.001591   0.001106  -1.438  0.15085
## FARM                      0.134456   0.043219   3.111  0.00196 **
## poly(CALM, 2, raw = T)1 -0.131917   0.042490  -3.105  0.00200 **
## poly(CALM, 2, raw = T)2 -1.698647   0.798902  -2.126  0.03392 *
## NATR                      0.074702   0.030668   2.436  0.01517 *
## INR                       0.578542   0.228049   2.537  0.01145 *
## poly(FDP, 2, raw = T)1   0.167628   0.058009   2.890  0.00400 **
## poly(FDP, 2, raw = T)2   1.323016   0.618303   2.140  0.03280 *
## I(BRL^2)                  7.101492   3.420916   2.076  0.03836 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02215 on 563 degrees of freedom
## Multiple R-squared:  0.07439,    Adjusted R-squared:  0.06124
## F-statistic: 5.656 on 8 and 563 DF,  p-value: 6.332e-07
```

```r
prediction2 = predict(fit2,testset)
mse2 = mean((prediction2-testset$sugar)^2)
print(mse2)
```

```
## [1] 0.0004949486
```

As we can see, the polynomial model improve the performance on the train set. The test MSE is 0.000495 which refers that this model may overfit the train set.

GAM

```r
# GAM (try different combination and local regression)
fit31 = gam(sugar~s(FARM,3)+s(CALM,3)+s(NATR,3)
            +s(INR,3),data=trainset)
fit32 = gam(sugar~s(FARM,3)+s(CALM,3)+s(NATR,3)+s(INR,3)
            +s(FDP,3)+s(BRL,3),data=trainset)
fit33 = gam(sugar~s(FARM,3)+s(CALM,3)+s(CALM^2,3)+s(NATR,3)
            +s(INR,3),data=trainset)
fit34 = gam(sugar~s(FARM,3)+s(CALM,3)+s(CALM^2,3)+s(NATR,3)
            +s(INR,3)+s(FDP,3)+s(FDP^2,3),data=trainset)
fit35 = gam(sugar~s(FARM,3)+s(CALM,3)+s(CALM^2,3)+s(NATR,3)
            +s(INR,3)+s(FDP,3)+s(FDP^2,3)+s(BRL^2,3),data=trainset)
fit36 = gam(sugar~s(FARM,3)+s(CALM,3)+s(NATR,3)+s(INR,3)
```

```
            +s(FDP,3)+s(BRL,3)+lo(CALM,span = 0.5)+lo(FDP,span = 0.5),data=trainset)
anova(fit31,fit32,fit33,fit34,fit35,fit36)
```

```
## Analysis of Deviance Table
##
## Model 1: sugar ~ s(FARM, 3) + s(CALM, 3) + s(NATR, 3) + s(INR, 3)
## Model 2: sugar ~ s(FARM, 3) + s(CALM, 3) + s(NATR, 3) + s(INR, 3) + s(FDP,
##     3) + s(BRL, 3)
## Model 3: sugar ~ s(FARM, 3) + s(CALM, 3) + s(CALM^2, 3) + s(NATR, 3) +
##     s(INR, 3)
## Model 4: sugar ~ s(FARM, 3) + s(CALM, 3) + s(CALM^2, 3) + s(NATR, 3) +
##     s(INR, 3) + s(FDP, 3) + s(FDP^2, 3)
## Model 5: sugar ~ s(FARM, 3) + s(CALM, 3) + s(CALM^2, 3) + s(NATR, 3) +
##     s(INR, 3) + s(FDP, 3) + s(FDP^2, 3) + s(BRL^2, 3)
## Model 6: sugar ~ s(FARM, 3) + s(CALM, 3) + s(NATR, 3) + s(INR, 3) + s(FDP,
##     3) + s(BRL, 3) + lo(CALM, span = 0.5) + lo(FDP, span = 0.5)
##   Resid. Df Resid. Dev       Df   Deviance Pr(>Chi)
## 1    559.00    0.27942
## 2    553.00    0.27164  6.00031  0.0077818  0.01533 *
## 3    556.00    0.27860 -3.00005 -0.0069575  0.00283 **
## 4    549.50    0.27271  6.50150  0.0058878  0.08275 .
## 5    542.88    0.26872  6.61406  0.0039937  0.28938
## 6    543.72    0.26799 -0.83444  0.0007280
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

After trying different model, the fourth model gives the best performance. We use it on the test set to see the accuracy.

```
prediction34 = predict(fit34,testset)
mse34 = mean((prediction34-testset$sugar)^2)
print(mse34)
```

```
## [1] 0.0004983242
```

From the result, we can tell that the MSE of test set using this GAM mthod is 0.000498.

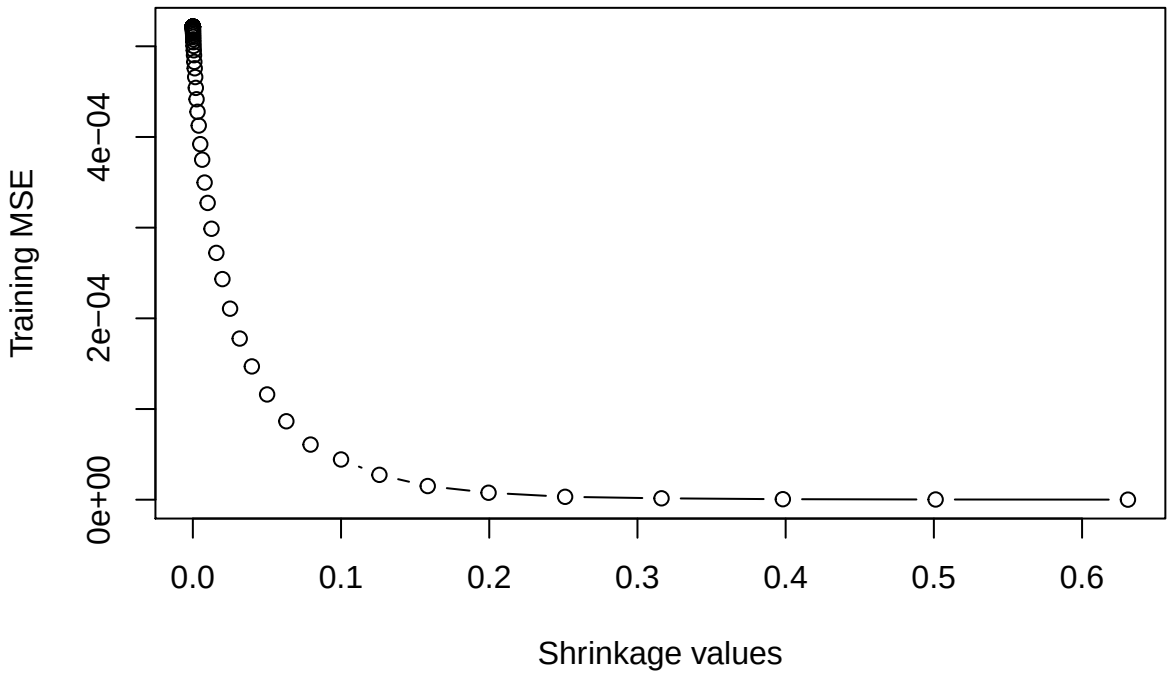
Regression Tree

In this section, we try to use different desicion tree to predict the sugar futures return.

```
# normal tree
fit4 = tree(sugar~.,data = trainset)
prediction4 = predict(fit4,trainset)
mse_train_41 = mean((prediction4-trainset$sugar)^2)

# boosting
means_43=rep(0,99)
set.seed(1)
pows <- seq(-10, -0.2, by = 0.1)
lambdas <- 10^pows
for (i in 1:99){
  fit43=gbm(sugar~.,data = trainset,distribution = "gaussian"
                , n.trees = 1000, interaction.depth = 2,shrinkage = lambdas[i])
  pred43=predict(fit43,newdata=trainset,n.trees = 1000)
  means_43[i] = mean((trainset$sugar-pred43)^2)
```

```
}
plot(lambdas,means_43,type='b',xlab = "Shrinkage values", ylab = "Training MSE")
```




```
fit43 = gbm(sugar~.,data = trainset,distribution = "gaussian"
            , n.trees = 1000, interaction.depth = 2,shrinkage = 0.2)
prediction43 = predict(fit43,trainset,n.trees = 1000)
mse_train_43 = mean((prediction43-trainset$sugar)^2)

# bagging
set.seed(1)
fit44 = randomForest(sugar~.,data = trainset,mtry = 29,importance = T)
prediction44 = predict(fit44,trainset)
mse_train_44 = mean((prediction44-trainset$sugar)^2)
```

```
print(mse_train_41)
```

```
## [1] 0.0003718905
```

```
print(mse_train_43)
```

```
## [1] 7.959163e-06
```

```
print(mse_train_44)
```

```
## [1] 9.919596e-05
```

However, we find the boosting method tree is the best in the train set. Note that we use corss-validation to determine the shrinkage parameter λ which is 0.2 here. We determin it from the plot when shrinkage is 0.2,

it first hit the relatively low level.

After using boosting and bagging, then we use the boosting method tree in the test set.

```
pred_tree = predict(fit43,testset,n.trees = 1000)
mse41 = mean((pred_tree-testset$sugar)^2)
```

Conclusion

```
conclusion1 = data.frame(c(mse1,mse11,mse12,mse2,mse34,mse41),row.names = c("linear regression","Lasso"
colnames(conclusion1) = "test MSE"
print(conclusion1)
```

```
##                       test MSE
## linear regression 0.0004866911
## Lasso             0.0004686722
## Ridge             0.0004630361
## Polynomial        0.0004949486
## GAM               0.0004983242
## boosted tree      0.0006747402
```

As we can see, the Ridge regression is the best model which has the lowest test MSE.

Question 3:

Do the same approach as in question 2, but this time for a qualitative variable.

Now we convert the previously data to the qualitative variable, we convert the sugar data to be either "up" or "down". When the return is negetive, the variable will be down, else will be up.

```
clas_data <- all_data
sugar_der <- rep(1,nrow(clas_data))
sugar_der[which(clas_data$sugar>=0)] = "up"
sugar_der[which(clas_data$sugar<0)] = "down"
sugar_der <- as.factor(sugar_der)
clas_data$sugar <- sugar_der
clas_train <- sample(nrow(clas_data),floor(nrow(clas_data)/2))
clas_trainset <- clas_data[clas_train,]
clas_testset <- clas_data[-clas_train,]
```

In this section, we use several appraoches to do the classificaiton problem.

They are:

1, Logistic Regression

2, LDA/QDA

3, KNN with different k

4, SVM with different kernel and parameters.

Logistic Regression

```
logfit=glm(sugar~.,data=clas_trainset,family=binomial)
result1 = predict(logfit,clas_testset,type = "response")
test_predict1 = rep(1,nrow(clas_testset))
test_predict1[which(result1>=0.5)] <- "up"
```

```
test_predict1[which(result1<0.5)] <- "down"
res_table1 = table(test_predict1,clas_testset$sugar)
acc1 = (res_table1[1,1]+res_table1[2,2])/sum(res_table1)
print(res_table1)
```

```
##
## test_predict1 down  up
##          down  168 139
##          up    133 132
```

```
print(acc1)
```

```
## [1] 0.5244755
```

The accuracy of this regression on the test set is 0.5245. Note that we include all the variable into the model, an alternative way is to use the lasso for logistic regression, however we didn't cover that issue in the course. The results are not good enough, we try to caliberate this model by select the variable that is significant.

```
logfit12=glm(sugar~JVA+FARM+FDP+MKC+NATR+RIBT+ethanol,data=clas_trainset,family=binomial)
result12 = predict(logfit12,clas_testset,type = "response")
test_predict12 = rep(1,nrow(clas_testset))
test_predict12[which(result12>=0.5)] <- "up"
test_predict12[which(result12<0.5)] <- "down"
res_table12 = table(test_predict12,clas_testset$sugar)
acc12 = (res_table12[1,1]+res_table12[2,2])/sum(res_table12)
print(res_table12)
```

```
##
## test_predict12 down  up
##           down  174 140
##           up    127 131
```

After selecting the significant variables, we indeed improve the logistic regression's performance.

LDA/QDA

```
#LDA/QDA
ldafit=lda(sugar~.,data=clas_trainset)
result2 = predict(ldafit,clas_testset,type = "response")$class
res_table2 = table(result2,clas_testset$sugar)
acc21 = (res_table2[1,1]+res_table2[2,2])/sum(res_table2)
print(acc21)
```

```
## [1] 0.5262238
```

Using the LDA method we can get the accuracy of 0.5262 on the test set.

```
qdafit = qda(sugar~.,data=clas_trainset)
result3 = predict(qdafit,clas_testset,type = "response")$class
res_table3 = table(result3,clas_testset$sugar)
acc22 = (res_table3[1,1]+res_table3[2,2])/sum(res_table3)
print(acc22)
```

```
## [1] 0.4807692
```

Using the QDA method we can get the accuracy of 0.4808 on the test set which is not better than the LDA method.

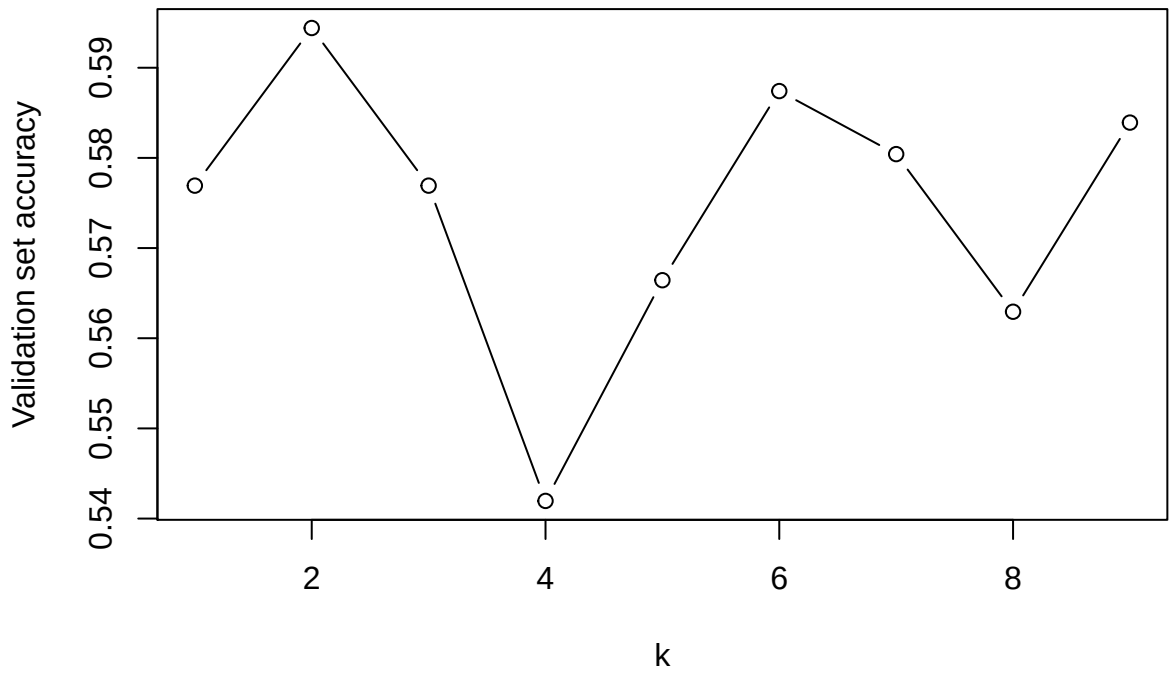
KNN

```
# KNN (different k)
set.seed(1)
clas_train <- sample(nrow(clas_trainset),floor(nrow(clas_trainset)/2))
clas_trainset_t <- clas_trainset[clas_train,]
clas_trainset_v<- clas_trainset[-clas_train,]
acc3_table = rep(1,9)
for (k_ in c(1:9)){
  knnfit1=knn(clas_trainset_t[,c(1:ncol(clas_trainset_t)-1)],
              clas_trainset_v[,c(1:ncol(clas_trainset_v)-1)],clas_trainset_v$sugar,k=k_)
  res_table4 = table(knnfit1,clas_trainset_v$sugar)
  acc3_table[k_] = (res_table4[1,1]+res_table4[2,2])/sum(res_table4)
}

print(acc3_table)
```

```
## [1] 0.5769231 0.5944056 0.5769231 0.5419580 0.5664336 0.5874126 0.5804196
## [8] 0.5629371 0.5839161
```

```
bestk = which.max(acc3_table)
plot(c(1:9),acc3_table,type='b',xlab = "k", ylab = "Validation set accuracy")
```



Note that we use the cross-validation method to determine the best "k", and we here use simply cross-validation. We may get a high accuracy by using k = 2, because we can get 0.5944 accuracy on the validation set which is the best version of this knn model. Now we use this parameter and model on the test set.

```
knnfit1=knn(clas_trainset[,c(1:ncol(clas_trainset)-1)],
            clas_testset[,c(1:ncol(clas_testset)-1)],clas_testset$sugar,k=bestk)
res_table5 = table(knnfit1,clas_testset$sugar)
acc3 = (res_table5[1,1]+res_table5[2,2])/sum(res_table5)
print(acc3)
```

```
## [1] 0.513986
```

When we use the whole train set to predict the test set, we will get an accuracy of 0.5297.

SVM

In this method we try to have different kernel and parameter to get the best SVM model.

```
# SVM (different kernel)
set.seed(1)
tune.out = tune(svm,sugar~.,data = clas_trainset,kernel="linear",
                ranges = list(cost=c(0.001,0.01,0.1,1.5,10,100)))

bestmod41 = tune.out$best.model
result41 = predict(bestmod41,clas_testset)
res_table61 = table(result41,clas_testset$sugar)
acc41 = (res_table61[1,1]+res_table61[2,2])/sum(res_table61)

set.seed(1)
tune.out = tune(svm,sugar~.,data = clas_trainset,kernel="polynomial",
                ranges = list(cost=c(0.001,0.01,0.1,1.5,10,100),degree = c(2,3,4)))

bestmod42 = tune.out$best.model
result42 = predict(bestmod42,clas_testset)
res_table62 = table(result42,clas_testset$sugar)
acc42 = (res_table62[1,1]+res_table62[2,2])/sum(res_table61)

set.seed(1)
tune.out = tune(svm,sugar~.,data = clas_trainset,kernel="radial",
                ranges = list(cost=c(0.001,0.01,0.1,1.5,10,100),gamma = c(0.01,0.1,1,3,5,10,100)))

bestmod43 = tune.out$best.model
result43 = predict(bestmod43,clas_testset)
res_table63 = table(result43,clas_testset$sugar)
acc43 = (res_table63[1,1]+res_table63[2,2])/sum(res_table63)

print(acc41)
```

```
## [1] 0.5262238
```

```
print(acc42)
```

```
## [1] 0.5087413
```

```
print(acc43)
```

```
## [1] 0.5262238
```

The first one and the last one give the same best results. which is the accuracy of 52.62%.

Conclusion

Here we present the final results of model above.

```
conclusion2 = data.frame(c(acc12,acc21,acc22,acc3,acc41),row.names = c("Logistic regression","LDA","QDA
colnames(conclusion2) = "test accuracy"
print(conclusion2)
```

```
##                      test accuracy
## Logistic regression     0.5332168
## LDA                     0.5262238
## QDA                     0.4807692
## KNN k=2                 0.5139860
## SVM linear & radial     0.5262238
```

So the Logistic regression will give us the highest test accuracy result. And the best version of the Logistic regression is that doing a regression on a subset of variables.

Question 4:

(Based on ISLR Chapter 9 #7) 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.

```
rm(list = ls())
CWID = 10442266
personal = CWID %% 10000
library(ISLR)
auto_data = Auto
bi_variable = rep(1,nrow(auto_data))
bi_variable[which(auto_data$mpg>median(auto_data$mpg))] <- 1
bi_variable[which(auto_data$mpg<=median(auto_data$mpg))] <- 0
auto_data$bi_variable = as.factor(bi_variable)
```

(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.

```
library(e1071)
set.seed(personal)
tune.out2 = tune(svm,bi_variable~.,data = auto_data,kernel = "linear",range = list(cost = c(0.01,0.1,1,5
summary(tune.out2)
```

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

```
##     1
##
## - best performance: 0.01025641
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-02 0.07410256 0.03924557
## 2 1e-01 0.04352564 0.03433256
## 3 1e+00 0.01025641 0.01792836
## 4 5e+00 0.01788462 0.01727588
## 5 1e+01 0.02044872 0.02020886
## 6 1e+02 0.04096154 0.02481661
```

When the cost equals to 1, the error reaches the lowest level.

(c)

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

```
set.seed(personal)
tune.out3 = tune(svm,bi_variable~.,data = auto_data,kernel="radial",ranges = list(cost=c(0.01,0.1,1,5,1
summary(tune.out3)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100  0.01
##
## - best performance: 0.01794872
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-02 1e-02 0.54461538 0.17293816
## 2  1e-01 1e-02 0.09198718 0.03884572
## 3  1e+00 1e-02 0.07410256 0.03924557
## 4  5e+00 1e-02 0.05115385 0.02716416
## 5  1e+01 1e-02 0.01794872 0.01730637
## 6  1e+02 1e-02 0.01794872 0.02716282
## 7  1e-02 1e-01 0.20974359 0.09161699
## 8  1e-01 1e-01 0.07923077 0.04451342
## 9  1e+00 1e-01 0.05878205 0.02729369
## 10 5e+00 1e-01 0.03333333 0.03636247
## 11 1e+01 1e-01 0.03076923 0.03375798
## 12 1e+02 1e-01 0.03070513 0.03376941
## 13 1e-02 1e+00 0.53711538 0.19569627
## 14 1e-01 1e+00 0.53711538 0.19569627
## 15 1e+00 1e+00 0.06397436 0.03681395
## 16 5e+00 1e+00 0.06134615 0.03852328
## 17 1e+01 1e+00 0.06134615 0.03852328
## 18 1e+02 1e+00 0.06134615 0.03852328
```

```
## 19 1e-02 3e+00 0.57961538 0.07504643
## 20 1e-01 3e+00 0.57961538 0.07504643
## 21 1e+00 3e+00 0.42205128 0.18381539
## 22 5e+00 3e+00 0.40935897 0.18871482
## 23 1e+01 3e+00 0.40935897 0.18871482
## 24 1e+02 3e+00 0.40935897 0.18871482
## 25 1e-02 5e+00 0.57961538 0.07504643
## 26 1e-01 5e+00 0.57961538 0.07504643
## 27 1e+00 5e+00 0.53096154 0.06728568
## 28 5e+00 5e+00 0.52839744 0.06958864
## 29 1e+01 5e+00 0.52839744 0.06958864
## 30 1e+02 5e+00 0.52839744 0.06958864
## 31 1e-02 1e+01 0.58461538 0.06452921
## 32 1e-01 1e+01 0.58461538 0.06452921
## 33 1e+00 1e+01 0.55916667 0.07611641
## 34 5e+00 1e+01 0.55147436 0.07560845
## 35 1e+01 1e+01 0.55147436 0.07560845
## 36 1e+02 1e+01 0.55147436 0.07560845
## 37 1e-02 1e+02 0.58711538 0.06014602
## 38 1e-01 1e+02 0.58711538 0.06014602
## 39 1e+00 1e+02 0.58711538 0.06014602
## 40 5e+00 1e+02 0.58711538 0.06014602
## 41 1e+01 1e+02 0.58711538 0.06014602
## 42 1e+02 1e+02 0.58711538 0.06014602
```

For radial kernel, when cost=100 and gamma = 0.01, the error will get the lowest level.

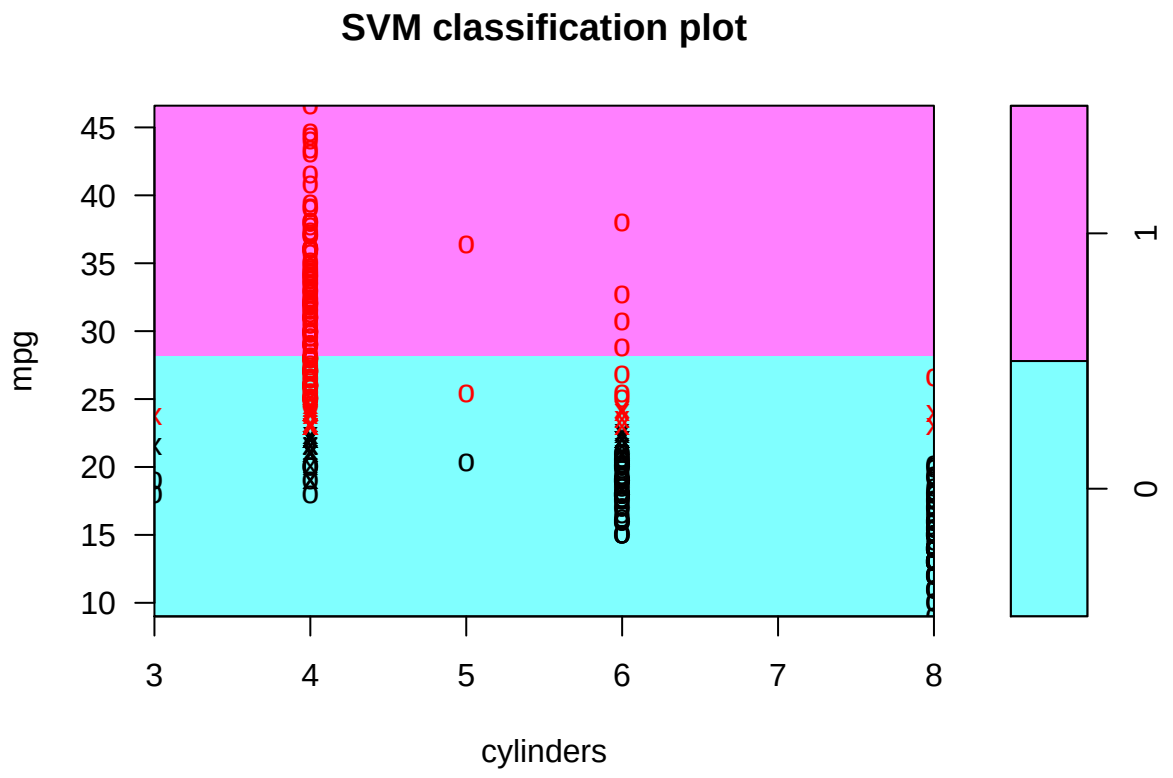
```
set.seed(personal)
tune.out4 = tune(svm,bi_variable~.,data = auto_data,kernel="polynomial",ranges = list(cost=c(0.01,0.1,1
summary(tune.out4)
```

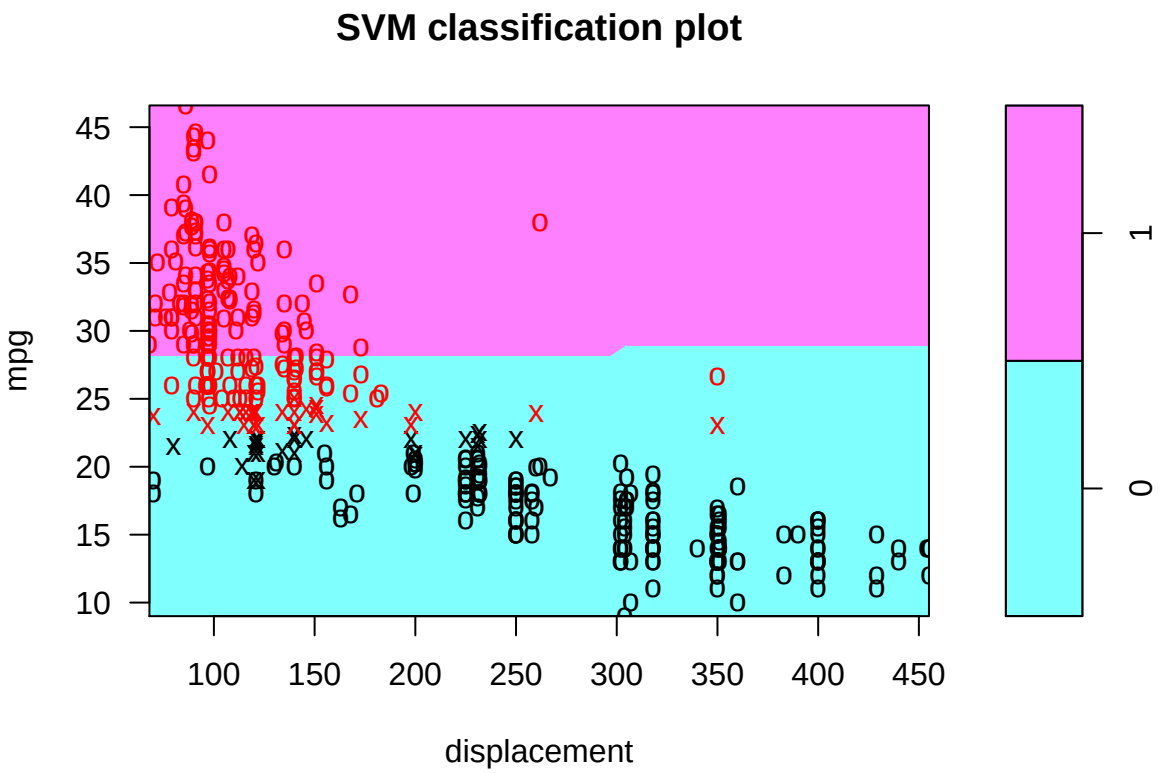
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      2
##
## - best performance: 0.3217308
##
## - Detailed performance results:
##     cost degree     error dispersion
## 1  1e-02      2 0.5846154 0.06452921
## 2  1e-01      2 0.5846154 0.06452921
## 3  1e+00      2 0.5846154 0.06452921
## 4  5e+00      2 0.5846154 0.06452921
## 5  1e+01      2 0.5436538 0.11402029
## 6  1e+02      2 0.3217308 0.06863895
## 7  1e-02      3 0.5796154 0.07504643
## 8  1e-01      3 0.5796154 0.07504643
## 9  1e+00      3 0.5796154 0.07504643
## 10 5e+00      3 0.5796154 0.07504643
## 11 1e+01      3 0.5796154 0.07504643
```

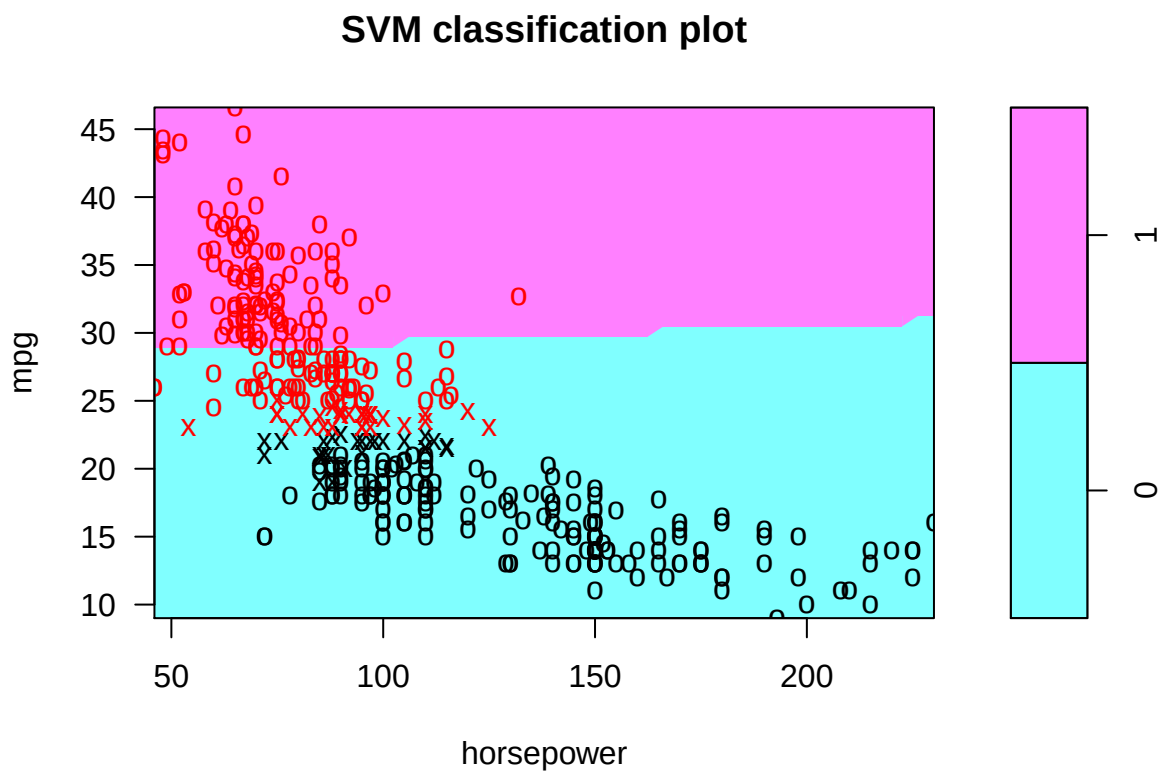
```
## 12 1e+02      3 0.3340385 0.13488457
## 13 1e-02      4 0.5871154 0.06014602
## 14 1e-01      4 0.5871154 0.06014602
## 15 1e+00      4 0.5871154 0.06014602
## 16 5e+00      4 0.5871154 0.06014602
## 17 1e+01      4 0.5871154 0.06014602
## 18 1e+02      4 0.5871154 0.06014602
```

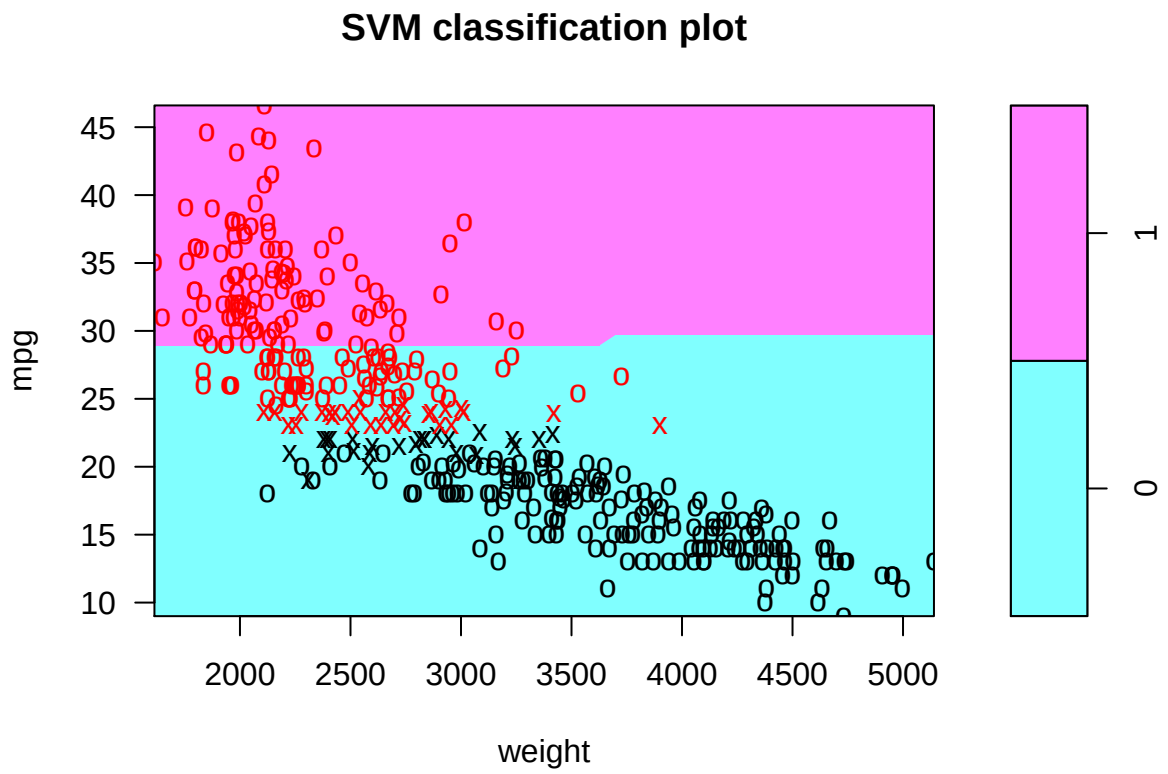
When using the polynomial basis kernel, the cv error reaches its lowest level for cost equals to 100 and dgree equals to 2. ##(d) Make some plots to back up your assertions in (b) and (c). Hint: In the lab, we used the plot() function for svm objects only in cases with p=2 When p>2,you can use the plot() function to create plots displaying pairs of variables at a time. Essentially, instead of typing plot(svmfit , dat) where svmfit contains your fitted model and dat is a data frame containing your data, you can type plot(svmfit , dat, x1~x4) in order to plot just the first and fourth variables. However, you must replace x1 and x4 with the correct variable names. To find out more, type ?plot.svm.

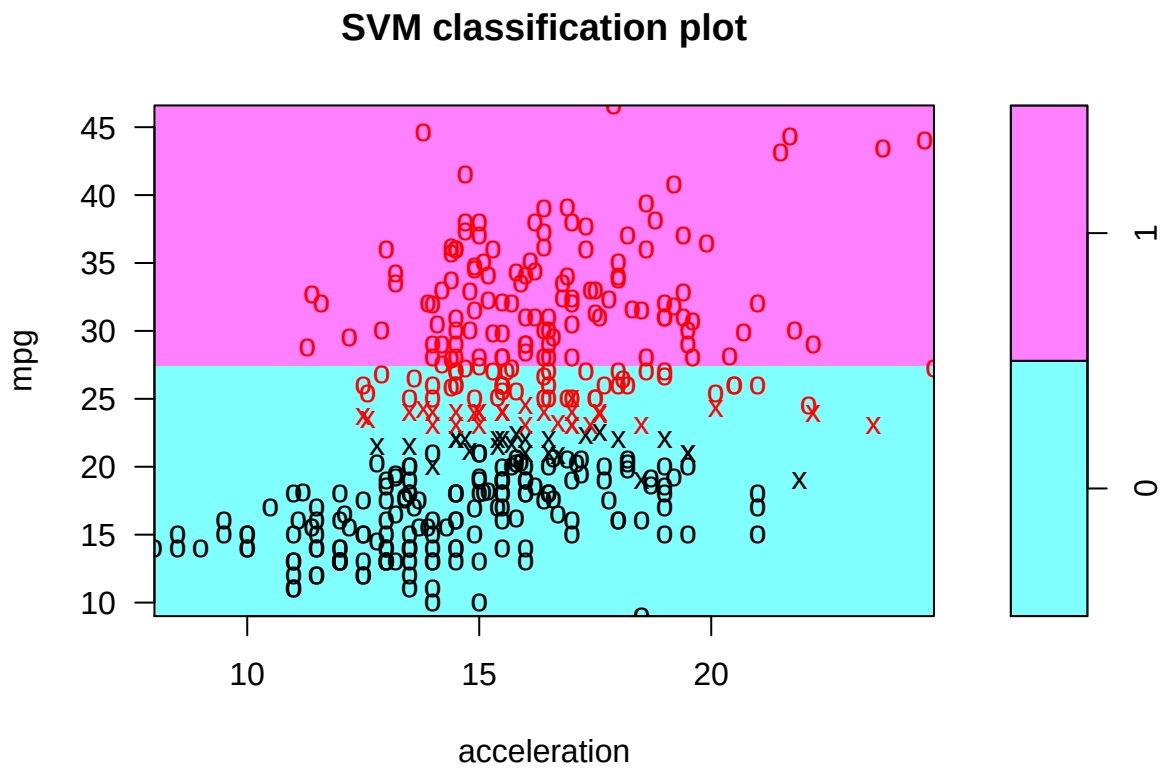
```
bestmod.lin = tune.out2$best.model
bestmod.rad = tune.out3$best.model
bestmod.pol = tune.out4$best.model
for (name in c("cylinders","displacement","horsepower","weight","acceleration","year","origin")){
  plot(bestmod.lin,auto_data,as.formula(paste("mpg~",name,sep="")))
}
```

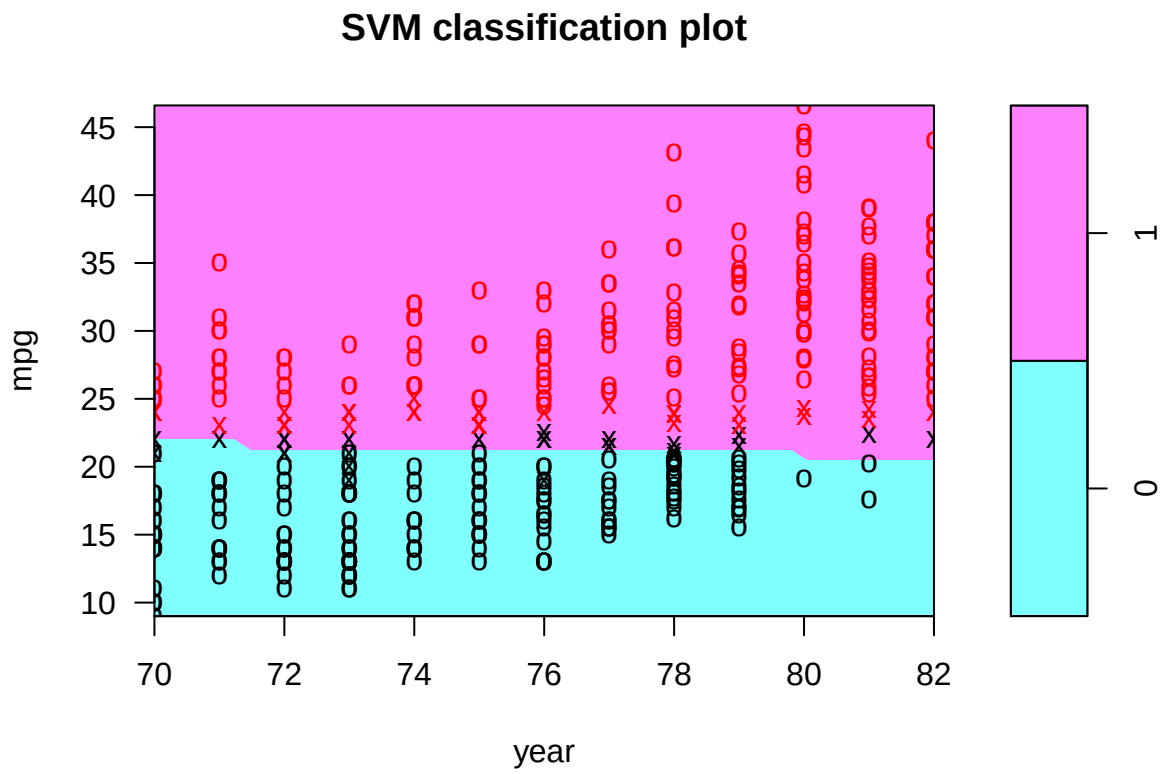


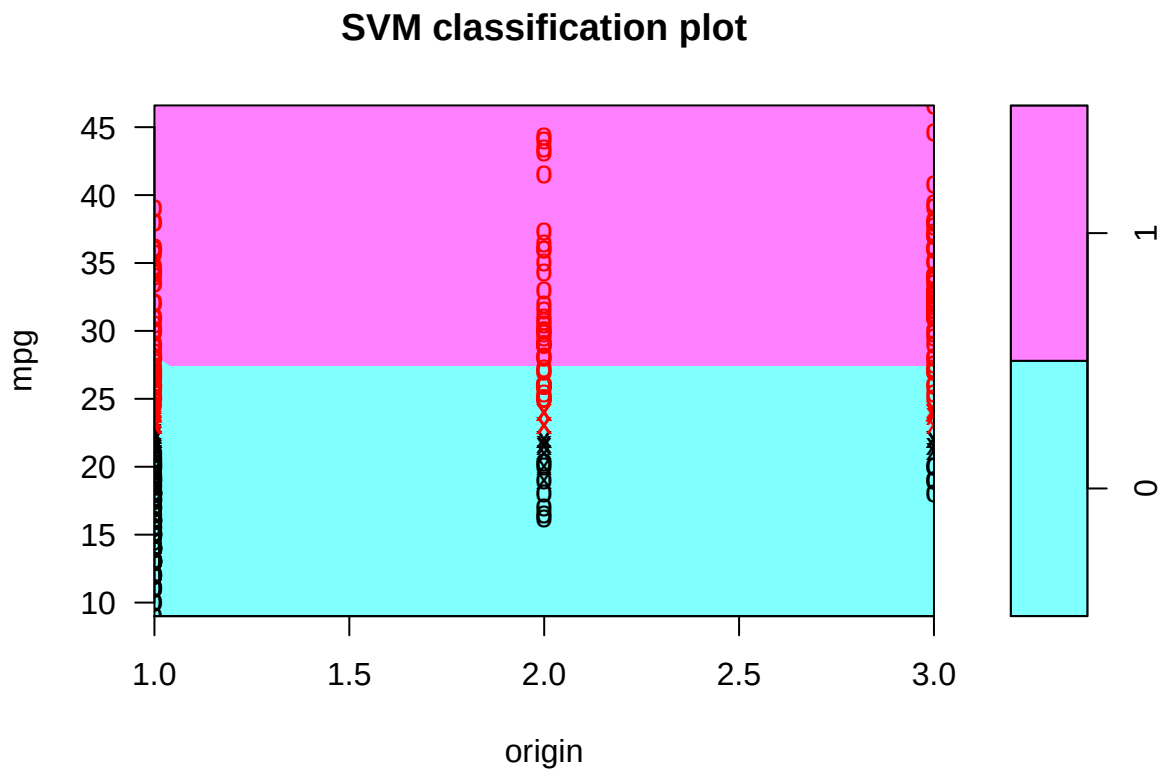

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0


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

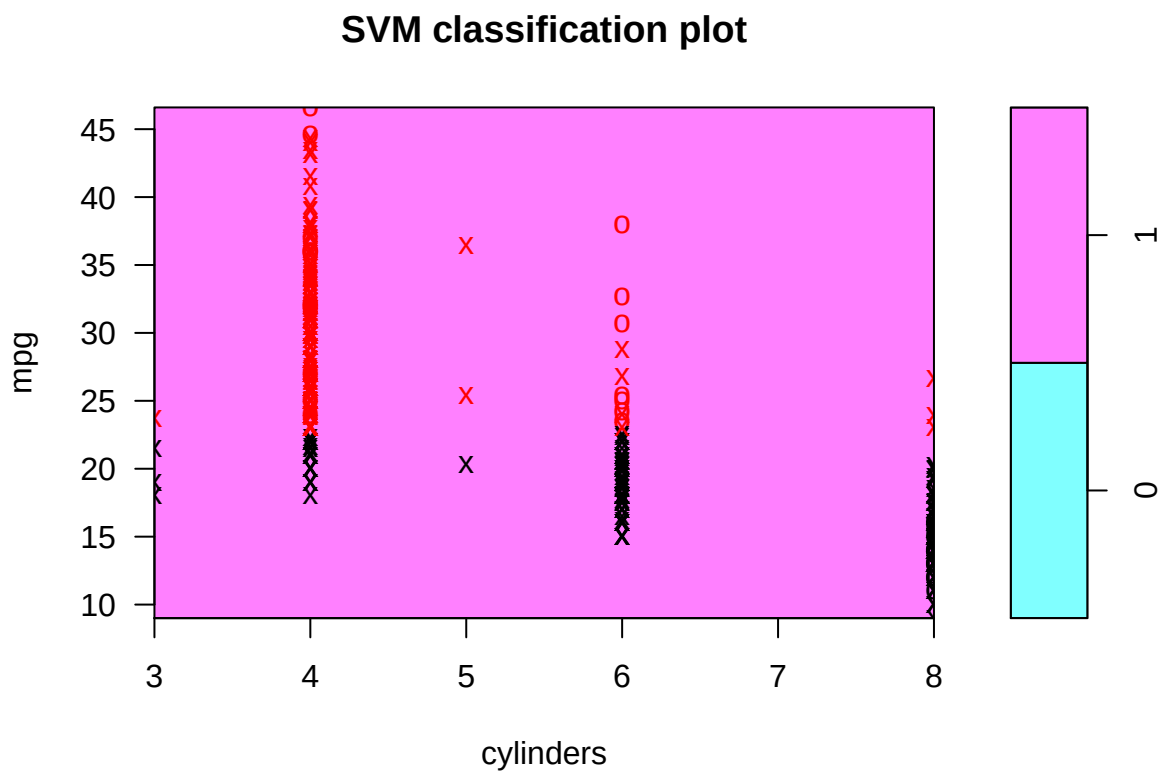

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0

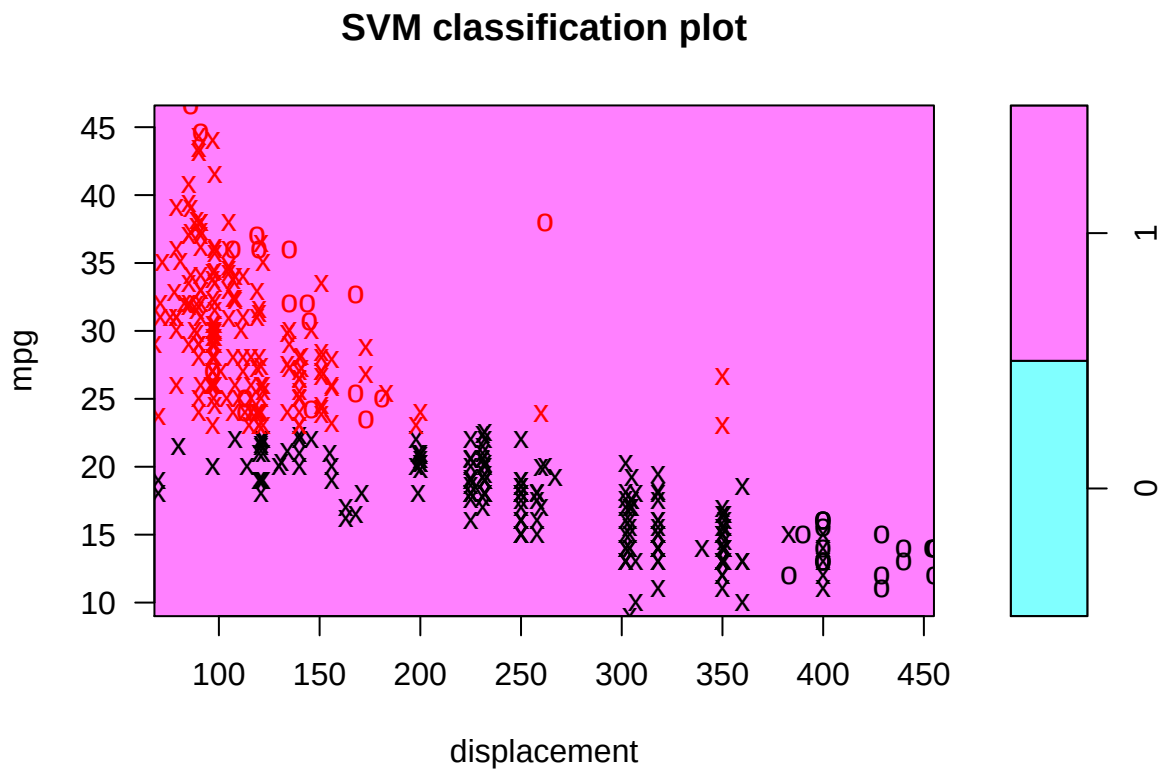

SVM classification plot
mpg
year
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
1
0

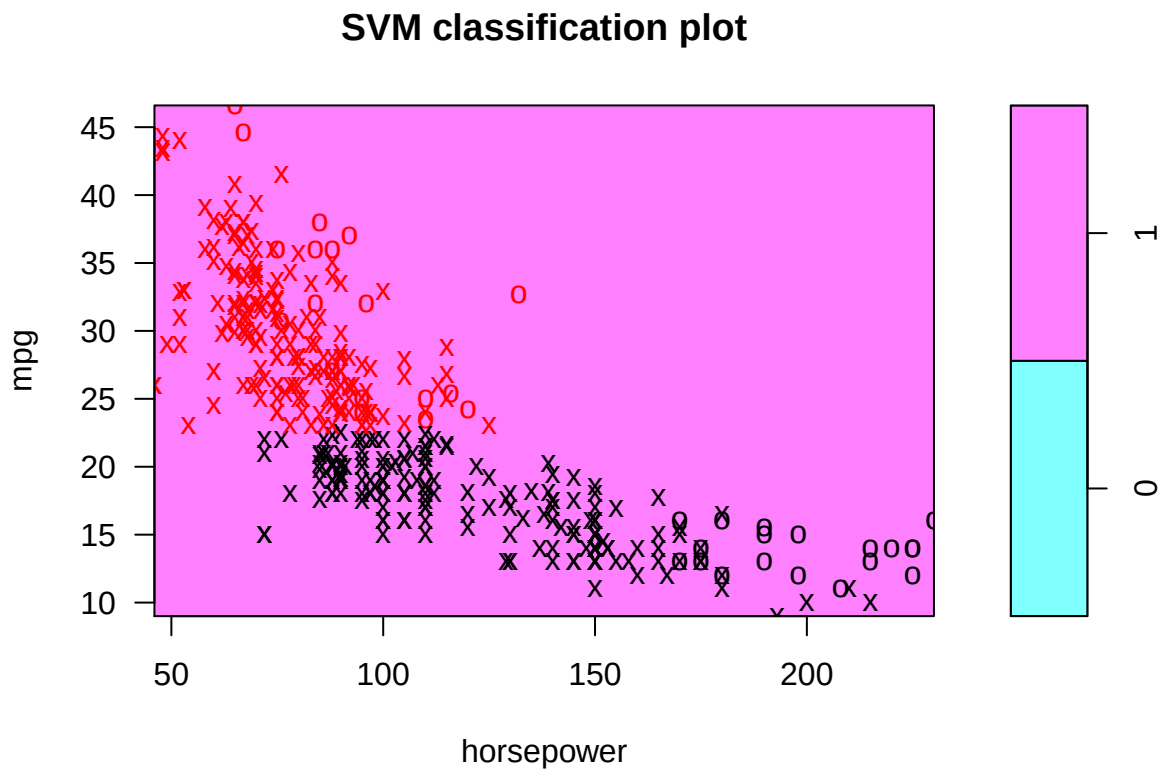


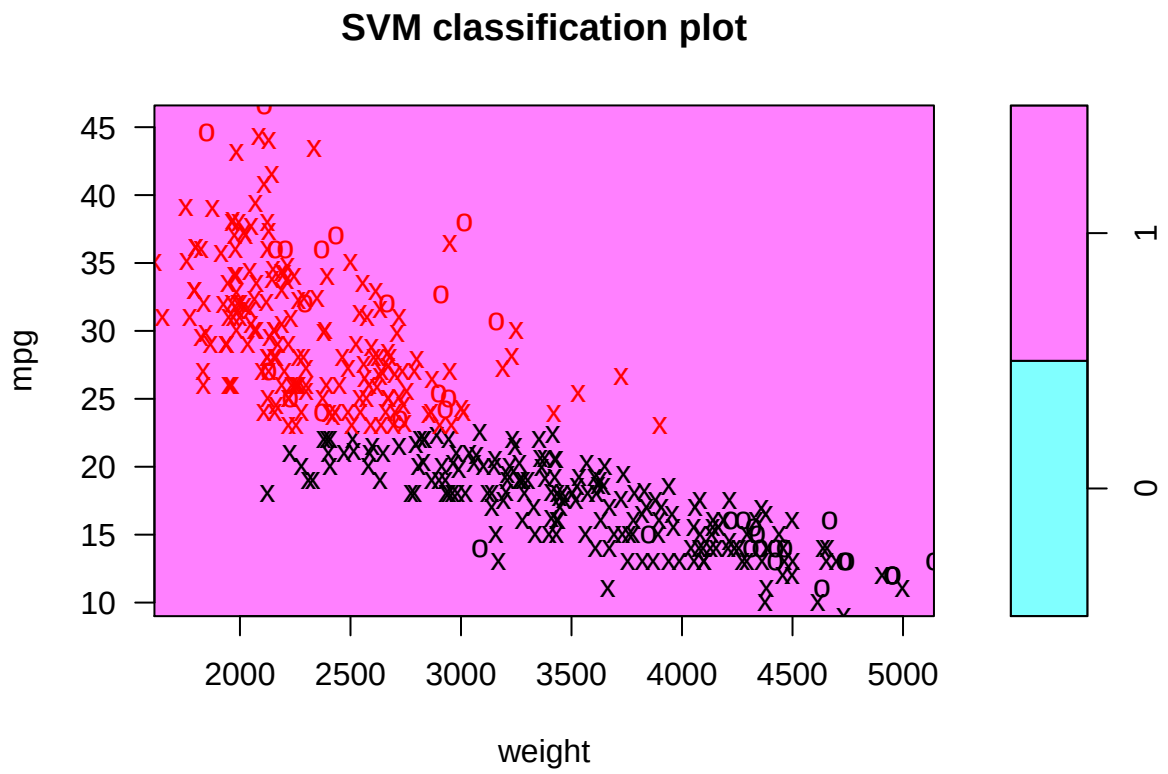
This is the graphs that using the linear kernel.

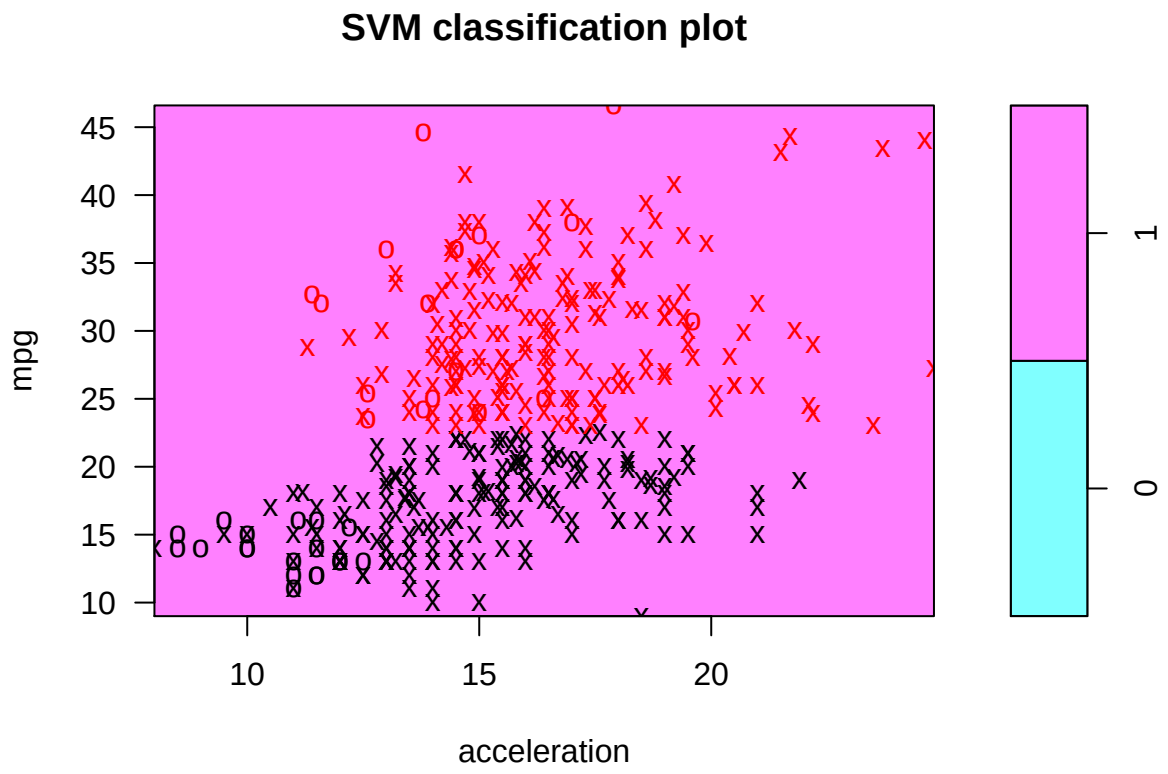
```
for (name in c("cylinders","displacement","horsepower","weight","acceleration","year","origin")){
  plot(bestmod.pol,auto_data,as.formula(paste("mpg~",name,sep="")))
}
```

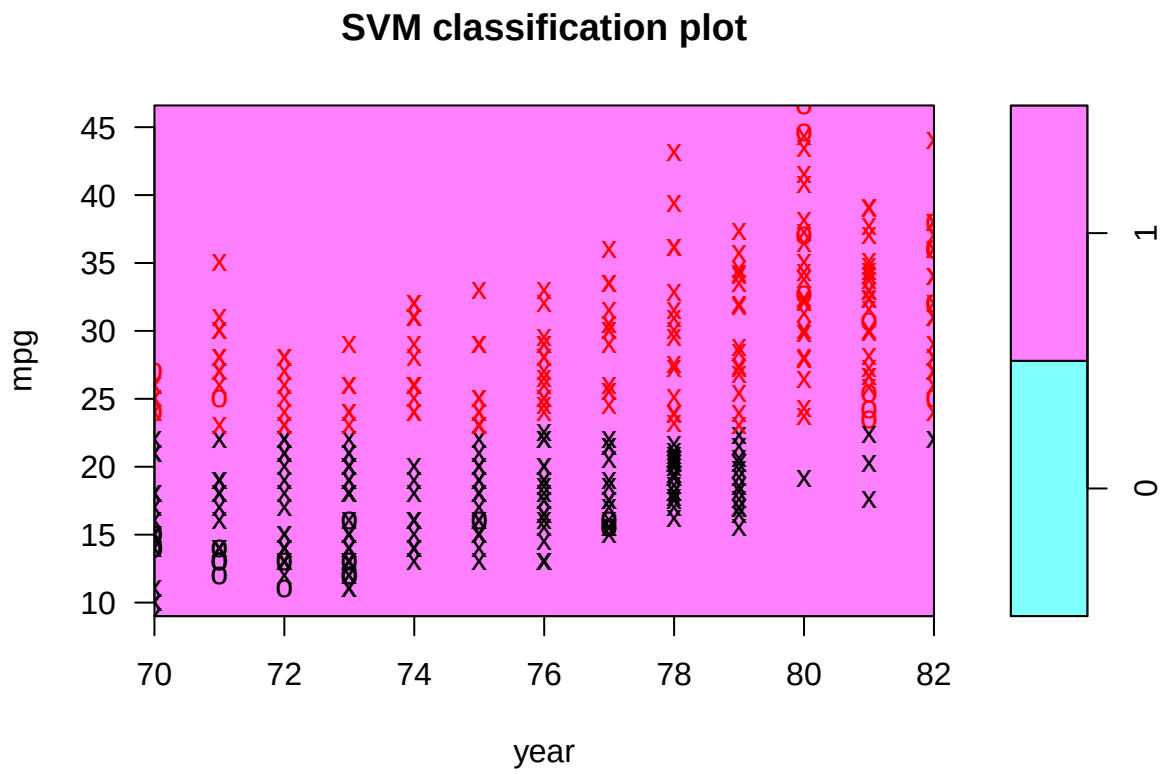

SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0

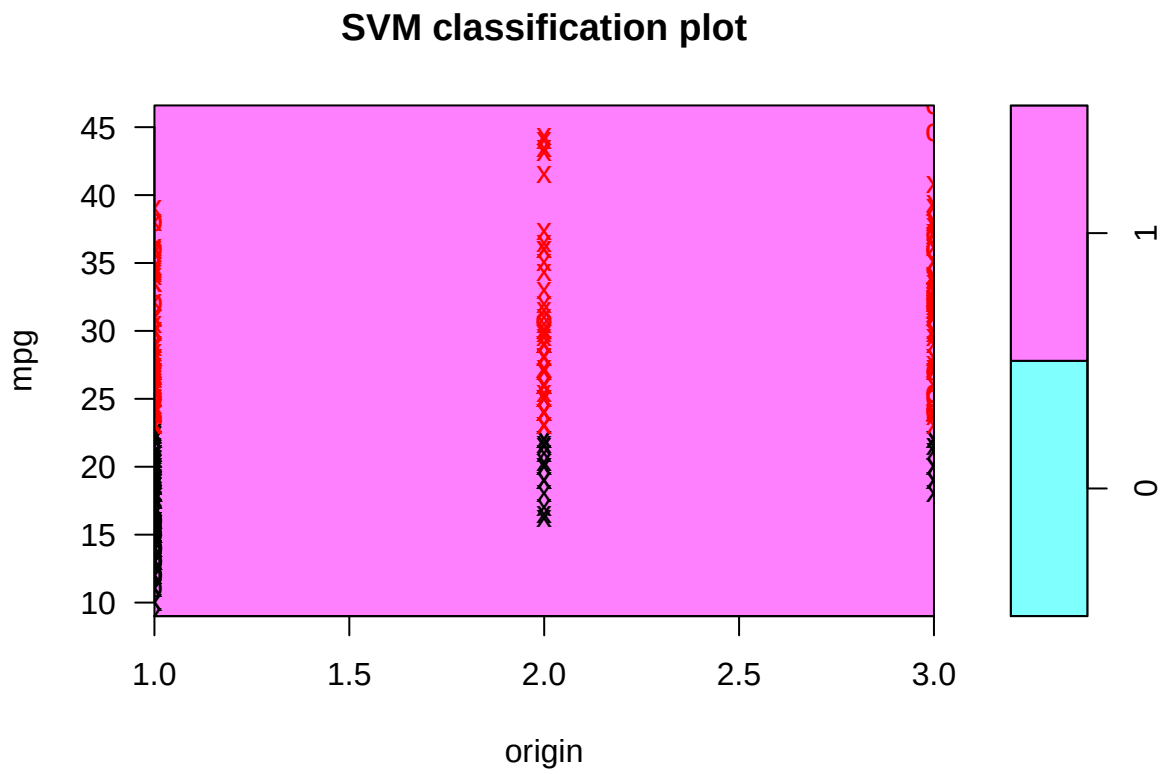

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0


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

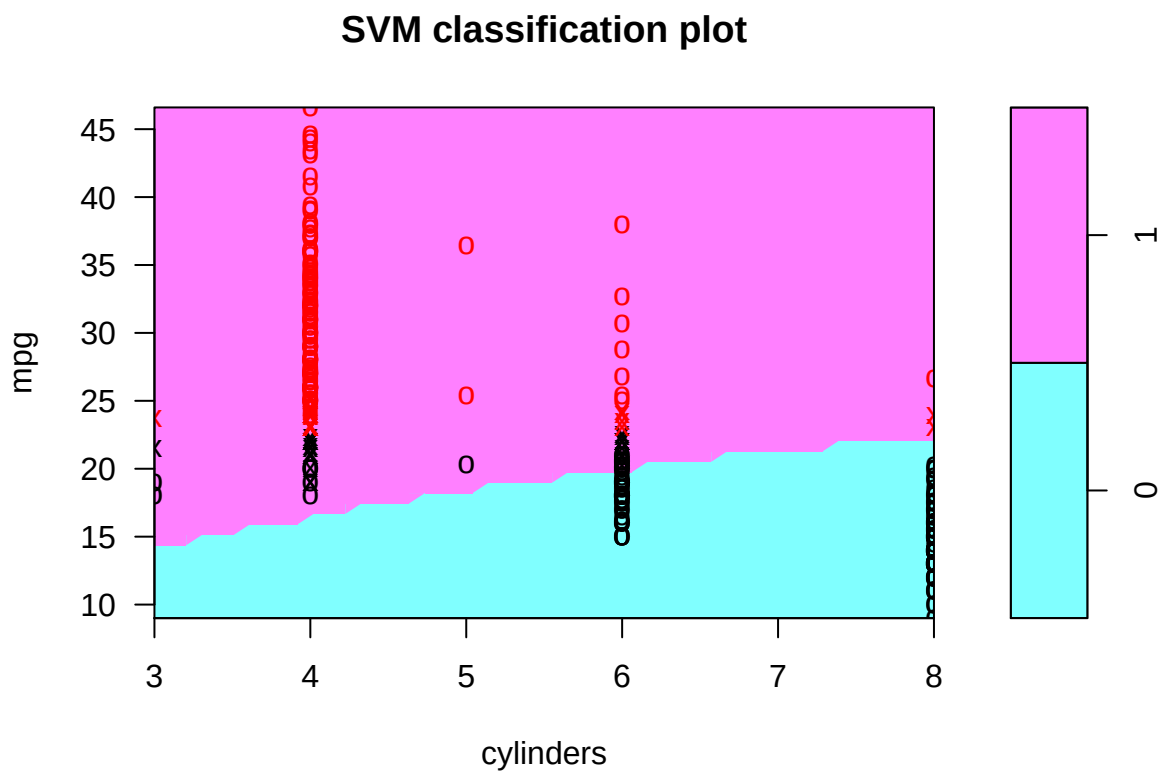

SVM classification plot
mpg
45
40
35
30
25
20
15
10
10
15
20
acceleration
1
0

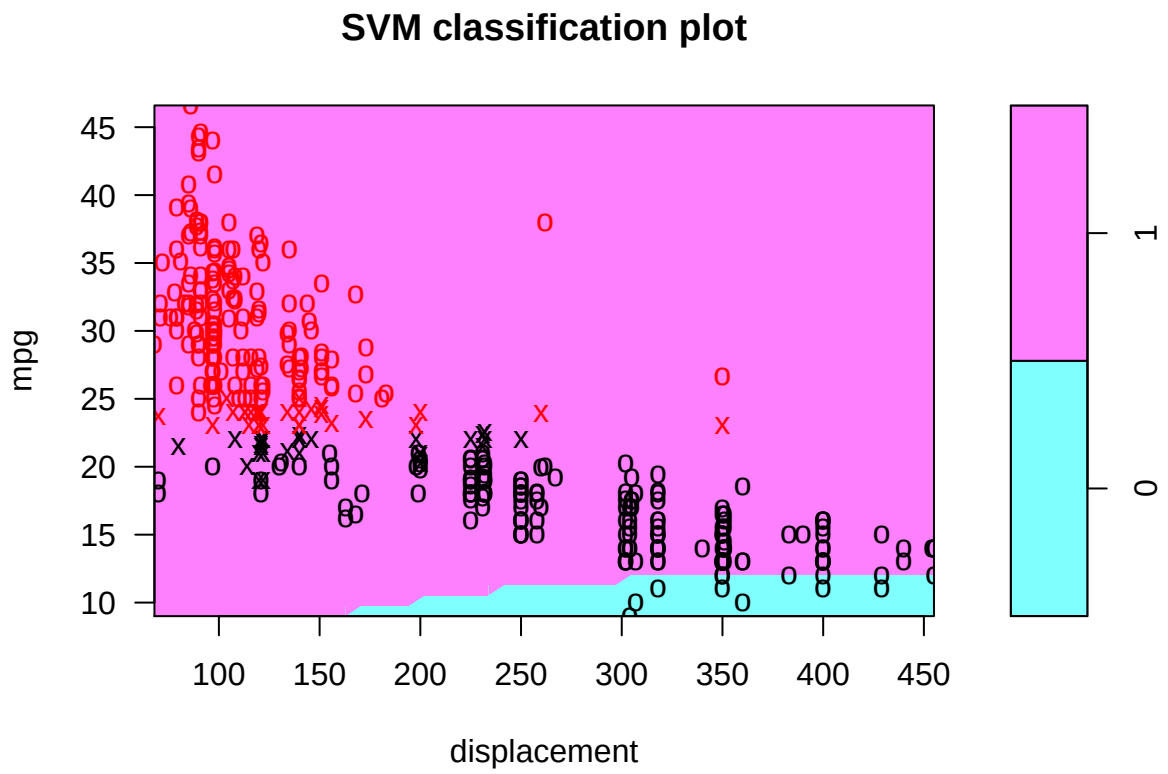

SVM classification plot
mpg
year
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
1
0

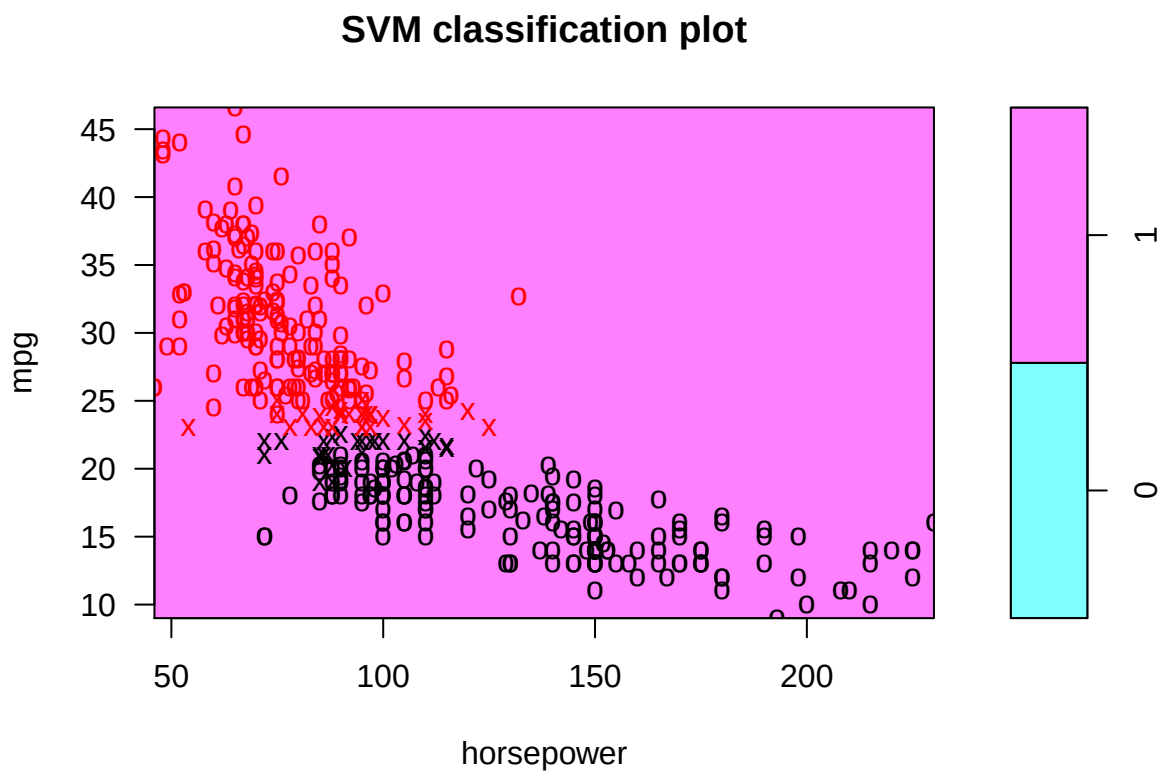


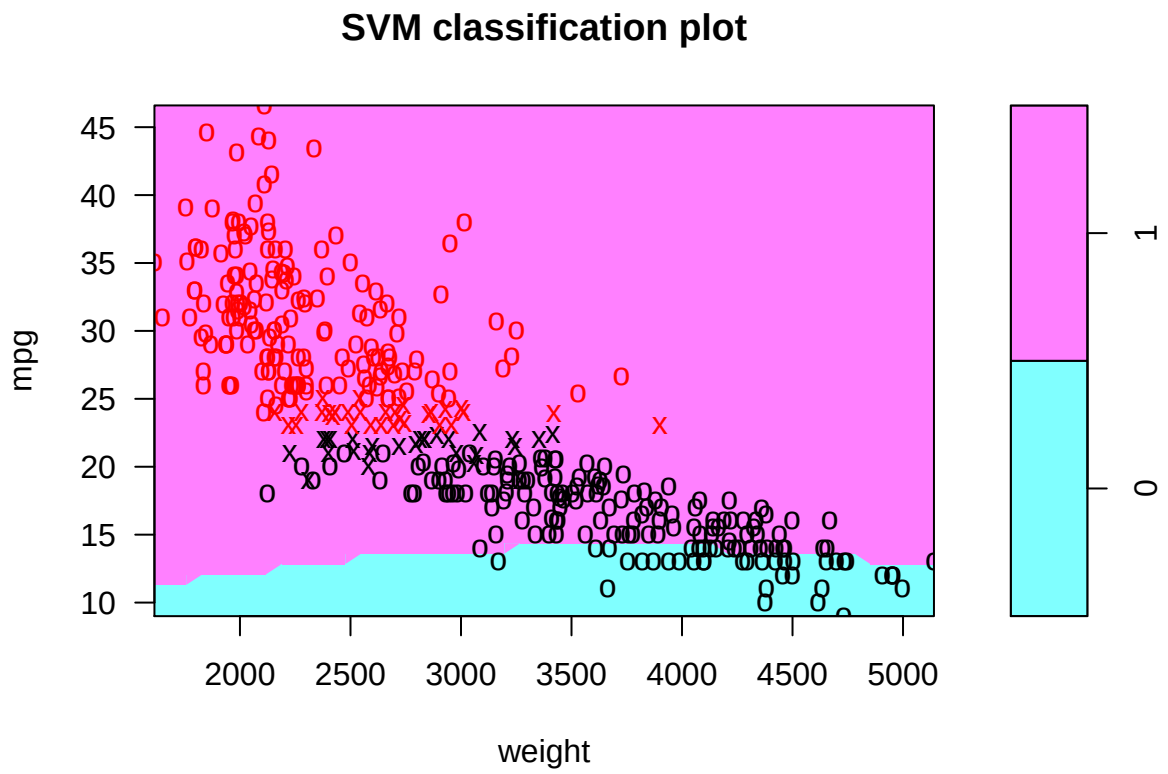
The above is the graphs that using the polynomial kernel.

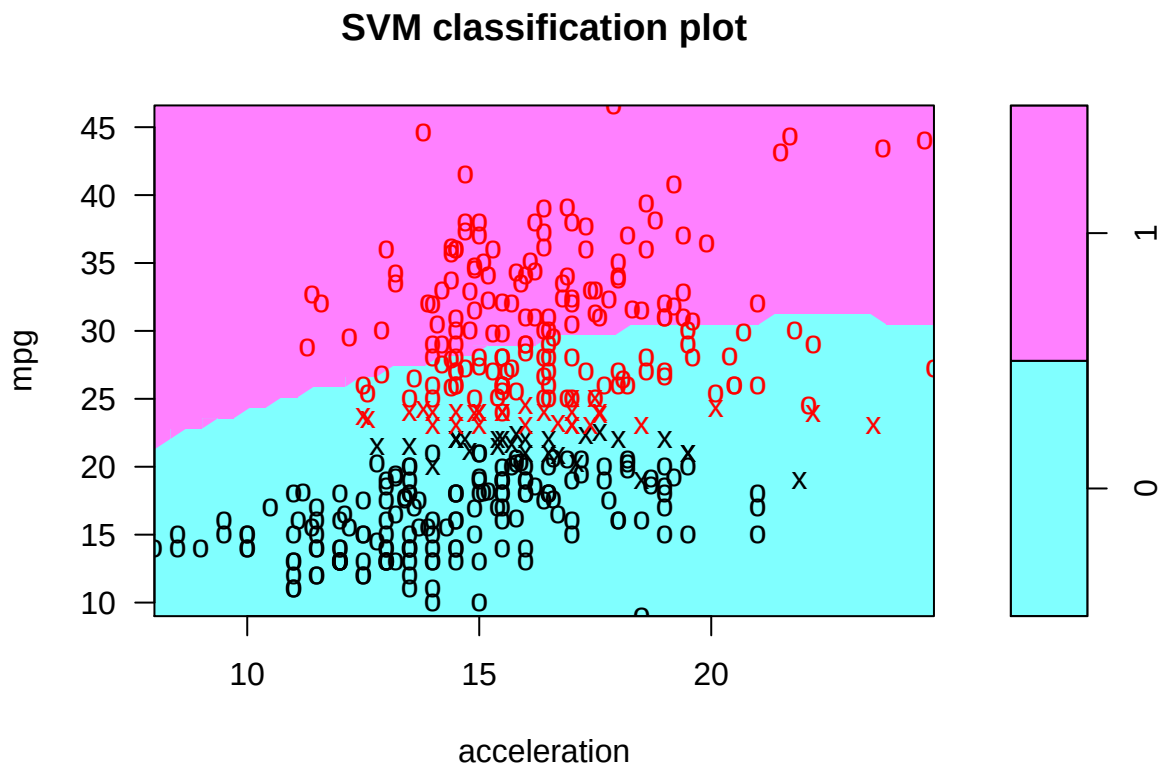
```
for (name in c("cylinders","displacement","horsepower","weight","acceleration","year","origin")){
  plot(bestmod.rad,auto_data,as.formula(paste("mpg~",name,sep="")))
}
```

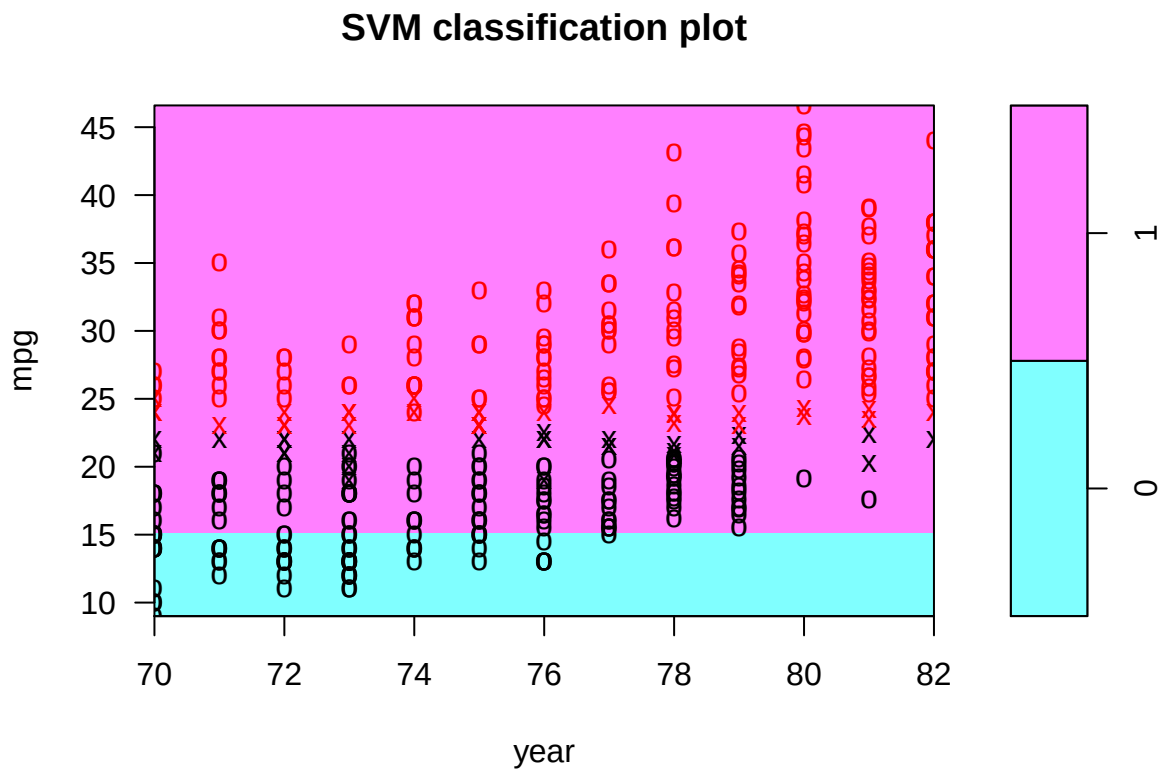

SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0

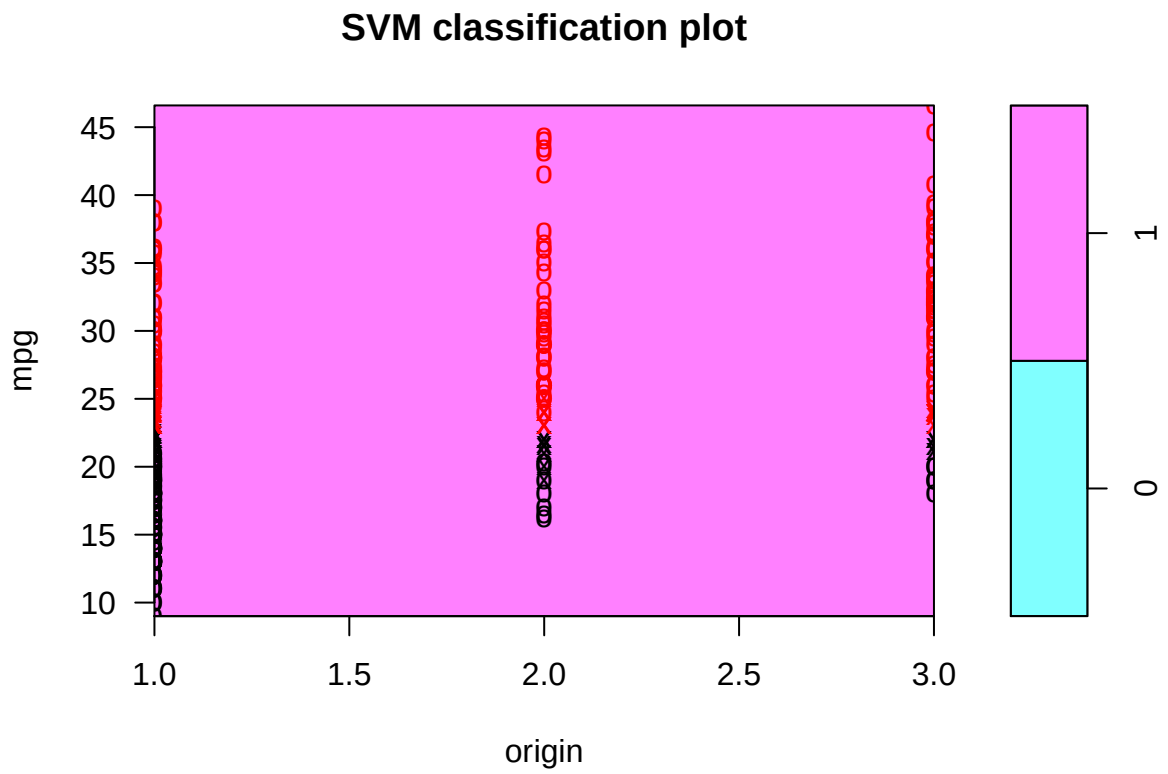

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0


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



The above is the graphs that using the radial kernel.