Homework 4.2 - OJ dataset

Hamed

2/26/2020

# This problem involves the OJ data set which is part of the ISLR package.

library(ISLR)  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(kernlab)

##   
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':  
##   
## alpha

library(ROCR)

## Loading required package: gplots

##   
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':  
##   
## lowess

library(e1071)  
library(tidyverse)

## -- Attaching packages ----------------------------------------- tidyverse 1.3.0 --

## <U+2713> tibble 2.1.3 <U+2713> dplyr 0.8.3  
## <U+2713> tidyr 1.0.0 <U+2713> stringr 1.4.0  
## <U+2713> readr 1.3.1 <U+2713> forcats 0.4.0  
## <U+2713> purrr 0.3.3

## -- Conflicts -------------------------------------------- tidyverse\_conflicts() --  
## x kernlab::alpha() masks ggplot2::alpha()  
## x purrr::cross() masks kernlab::cross()  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()  
## x purrr::lift() masks caret::lift()

OJ=as.data.frame(OJ)  
str(OJ)

## 'data.frame': 1070 obs. of 18 variables:  
## $ Purchase : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 1 1 1 1 1 ...  
## $ WeekofPurchase: num 237 239 245 227 228 230 232 234 235 238 ...  
## $ StoreID : num 1 1 1 1 7 7 7 7 7 7 ...  
## $ PriceCH : num 1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...  
## $ PriceMM : num 1.99 1.99 2.09 1.69 1.69 1.99 1.99 1.99 1.99 1.99 ...  
## $ DiscCH : num 0 0 0.17 0 0 0 0 0 0 0 ...  
## $ DiscMM : num 0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...  
## $ SpecialCH : num 0 0 0 0 0 0 1 1 0 0 ...  
## $ SpecialMM : num 0 1 0 0 0 1 1 0 0 0 ...  
## $ LoyalCH : num 0.5 0.6 0.68 0.4 0.957 ...  
## $ SalePriceMM : num 1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...  
## $ SalePriceCH : num 1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...  
## $ PriceDiff : num 0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...  
## $ Store7 : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 2 2 ...  
## $ PctDiscMM : num 0 0.151 0 0 0 ...  
## $ PctDiscCH : num 0 0 0.0914 0 0 ...  
## $ ListPriceDiff : num 0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...  
## $ STORE : num 1 1 1 1 0 0 0 0 0 0 ...

## (a) Create a training set containing a random sample of 60% of the observations,and a test set containing the remaining observations.

set.seed(123)  
smp\_size <-floor(0.60 \* nrow(OJ))  
train\_ind <-sample(seq\_len(nrow(OJ)), size = smp\_size)  
train.OJ = OJ[train\_ind,]  
test.OJ = OJ[-train\_ind,]  
dim(train.OJ)

## [1] 642 18

dim(test.OJ)

## [1] 428 18

## (b) Fit a support vector classifier with a linear kernel to the training data using cost=0.01, with Purchase as the response and the other variables as predictors. Describe the results obtained.

library(kernlab)  
svm1 = train(Purchase~.,data=train.OJ,method='svmLinear' ,cost=0.01)  
svm1$results #accuracy

## C Accuracy Kappa AccuracySD KappaSD  
## 1 1 0.8190212 0.6157996 0.02052438 0.04380648

svm1$finalModel #error

## Support Vector Machine object of class "ksvm"   
##   
## SV type: C-svc (classification)   
## parameter : cost C = 1   
##   
## Linear (vanilla) kernel function.   
##   
## Number of Support Vectors : 259   
##   
## Objective Function Value : -252.7715   
## Training error : 0.155763

## (c) What are the training and test error rates?

# Training error rates  
svm1.train.pred = predict(svm1,newdata=train.OJ)  
confusion.mat=table(obs=train.OJ$Purchase,pred=svm1.train.pred)  
confusion.mat

## pred  
## obs CH MM  
## CH 347 44  
## MM 56 195

# The above object is a confusion matrix thus grabbing the false positive and negative elements will give the error of the prediction  
error\_rate=(confusion.mat[1,2]+confusion.mat[2,1])/nrow(train.OJ)  
error\_rate #confirmed its equal to the value gotten earlier from the model

## [1] 0.1557632

# Test error rates  
svm1.test.pred = predict(svm1,newdata=test.OJ)  
confusion.Mat=table(obs=test.OJ$Purchase,pred=svm1.test.pred)  
confusion.Mat

## pred  
## obs CH MM  
## CH 229 33  
## MM 39 127

# The above object is a confusion matrix thus grabbing the false positive and negative elements will give the error of the prediction  
test\_error\_linear =(confusion.Mat[1,2]+confusion.Mat[2,1])/nrow(test.OJ)  
test\_error\_linear

## [1] 0.1682243

## (d) Use a tuning grid in caret to select an optimal cost. Consider values in the range 0.01 to 10.

svm1.tune = tune(svm,Purchase~.,data=train.OJ,  
 ranges=list(cost=c(.01,.02,.05,.1,.2,.5,1,2,5,10)),kernel='linear')  
summary(svm1.tune)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 5  
##   
## - best performance: 0.1650962   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 0.01 0.1712740 0.03928411  
## 2 0.02 0.1728365 0.03892928  
## 3 0.05 0.1712500 0.03913674  
## 4 0.10 0.1666106 0.03575720  
## 5 0.20 0.1697115 0.03521432  
## 6 0.50 0.1697115 0.03817148  
## 7 1.00 0.1697115 0.03952538  
## 8 2.00 0.1666106 0.03860857  
## 9 5.00 0.1650962 0.03459290  
## 10 10.00 0.1697837 0.03988164

## (e) Compute the training and test error rates using this new value for cost.

# Train error rates  
svm1.best.train.pred = predict(svm1.tune$best.model,newdata=train.OJ)  
confusion.Mat=table(obs=train.OJ$Purchase,pred=svm1.best.train.pred)  
confusion.Mat

## pred  
## obs CH MM  
## CH 347 44  
## MM 55 196

# The above object is a confusion matrix thus grabbing the false positive and negative elements will give the error of the prediction  
error\_rate=(confusion.Mat[1,2]+confusion.Mat[2,1])/nrow(train.OJ)  
error\_rate

## [1] 0.1542056

# Test error rates  
svm1.best.test.pred=predict(svm1.tune$best.model,newdata=test.OJ)  
confusion.Mat=table(obs=test.OJ$Purchase,pred=svm1.best.test.pred)  
# The above object is a confusion matrix thus grabbing the false positive and negative elements will give the error of the prediction  
test\_error\_linearTune=(confusion.Mat[1,2]+confusion.Mat[2,1])/nrow(test.OJ)  
test\_error\_linearTune

## [1] 0.1682243

## (f) Repeat parts (b) through (e) using a support vector machine with aradial kernel. Use a grid of values for gamma but use default for cost

svm2.tune = tune(svm , Purchase~.,data=train.OJ,ranges=list(  
 cost=c(.01,.02,.05,.1,.2,.5,1,2,5,10),gamma=c(.001,.002,.005,  
 .01,.02,.05,.1,.2,.5,1,2,5,10)),kernel='radial')  
summary(svm2.tune)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost gamma  
## 10 0.02  
##   
## - best performance: 0.1542067   
##   
## - Detailed performance results:  
## cost gamma error dispersion  
## 1 0.01 1e-03 0.3910096 0.05724320  
## 2 0.02 1e-03 0.3910096 0.05724320  
## 3 0.05 1e-03 0.3910096 0.05724320  
## 4 0.10 1e-03 0.3910096 0.05724320  
## 5 0.20 1e-03 0.3910096 0.05724320  
## 6 0.50 1e-03 0.3723317 0.05612687  
## 7 1.00 1e-03 0.2070673 0.05982902  
## 8 2.00 1e-03 0.1698077 0.06697378  
## 9 5.00 1e-03 0.1620673 0.06540129  
## 10 10.00 1e-03 0.1589423 0.05579695  
## 11 0.01 2e-03 0.3910096 0.05724320  
## 12 0.02 2e-03 0.3910096 0.05724320  
## 13 0.05 2e-03 0.3910096 0.05724320  
## 14 0.10 2e-03 0.3910096 0.05724320  
## 15 0.20 2e-03 0.3878846 0.05995102  
## 16 0.50 2e-03 0.2117067 0.05549826  
## 17 1.00 2e-03 0.1729087 0.05926683  
## 18 2.00 2e-03 0.1744952 0.06255354  
## 19 5.00 2e-03 0.1589183 0.05482543  
## 20 10.00 2e-03 0.1651442 0.05735803  
## 21 0.01 5e-03 0.3910096 0.05724320  
## 22 0.02 5e-03 0.3910096 0.05724320  
## 23 0.05 5e-03 0.3910096 0.05724320  
## 24 0.10 5e-03 0.3785577 0.05909060  
## 25 0.20 5e-03 0.2241587 0.04918009  
## 26 0.50 5e-03 0.1744471 0.05974632  
## 27 1.00 5e-03 0.1667067 0.05730161  
## 28 2.00 5e-03 0.1667067 0.05824072  
## 29 5.00 5e-03 0.1620192 0.06099699  
## 30 10.00 5e-03 0.1604327 0.06001482  
## 31 0.01 1e-02 0.3910096 0.05724320  
## 32 0.02 1e-02 0.3910096 0.05724320  
## 33 0.05 1e-02 0.3910096 0.05724320  
## 34 0.10 1e-02 0.2630288 0.06730119  
## 35 0.20 1e-02 0.1791346 0.05876734  
## 36 0.50 1e-02 0.1682452 0.05377438  
## 37 1.00 1e-02 0.1729567 0.06295153  
## 38 2.00 1e-02 0.1651442 0.06406048  
## 39 5.00 1e-02 0.1635337 0.05734617  
## 40 10.00 1e-02 0.1588702 0.04620466  
## 41 0.01 2e-02 0.3910096 0.05724320  
## 42 0.02 2e-02 0.3910096 0.05724320  
## 43 0.05 2e-02 0.3442548 0.05721327  
## 44 0.10 2e-02 0.1869231 0.05499787  
## 45 0.20 2e-02 0.1729327 0.06338199  
## 46 0.50 2e-02 0.1636058 0.06460900  
## 47 1.00 2e-02 0.1697837 0.06491671  
## 48 2.00 2e-02 0.1635337 0.05342804  
## 49 5.00 2e-02 0.1588702 0.05278174  
## 50 10.00 2e-02 0.1542067 0.04839176  
## 51 0.01 5e-02 0.3910096 0.05724320  
## 52 0.02 5e-02 0.3910096 0.05724320  
## 53 0.05 5e-02 0.2367067 0.06298546  
## 54 0.10 5e-02 0.1760096 0.05875510  
## 55 0.20 5e-02 0.1697596 0.04976314  
## 56 0.50 5e-02 0.1603606 0.04771303  
## 57 1.00 5e-02 0.1618990 0.05452785  
## 58 2.00 5e-02 0.1588462 0.05452049  
## 59 5.00 5e-02 0.1588702 0.04722420  
## 60 10.00 5e-02 0.1651683 0.04429509  
## 61 0.01 1e-01 0.3910096 0.05724320  
## 62 0.02 1e-01 0.3910096 0.05724320  
## 63 0.05 1e-01 0.2679567 0.05930776  
## 64 0.10 1e-01 0.1931490 0.05637180  
## 65 0.20 1e-01 0.1682212 0.05222205  
## 66 0.50 1e-01 0.1697356 0.04918089  
## 67 1.00 1e-01 0.1759615 0.04829996  
## 68 2.00 1e-01 0.1666587 0.04262113  
## 69 5.00 1e-01 0.1728846 0.04771213  
## 70 10.00 1e-01 0.1807212 0.05359143  
## 71 0.01 2e-01 0.3910096 0.05724320  
## 72 0.02 2e-01 0.3910096 0.05724320  
## 73 0.05 2e-01 0.3676202 0.06018210  
## 74 0.10 2e-01 0.2336538 0.05364462  
## 75 0.20 2e-01 0.1931490 0.04393120  
## 76 0.50 2e-01 0.1822596 0.03881644  
## 77 1.00 2e-01 0.1822356 0.03508709  
## 78 2.00 2e-01 0.1837740 0.03324331  
## 79 5.00 2e-01 0.1901202 0.04054573  
## 80 10.00 2e-01 0.1931971 0.04793516  
## 81 0.01 5e-01 0.3910096 0.05724320  
## 82 0.02 5e-01 0.3910096 0.05724320  
## 83 0.05 5e-01 0.3910096 0.05724320  
## 84 0.10 5e-01 0.3006731 0.06291180  
## 85 0.20 5e-01 0.2227885 0.04739823  
## 86 0.50 5e-01 0.1962981 0.03074088  
## 87 1.00 5e-01 0.1885337 0.02737690  
## 88 2.00 5e-01 0.1885577 0.05045319  
## 89 5.00 5e-01 0.1901442 0.04502948  
## 90 10.00 5e-01 0.2102885 0.05294405  
## 91 0.01 1e+00 0.3910096 0.05724320  
## 92 0.02 1e+00 0.3910096 0.05724320  
## 93 0.05 1e+00 0.3910096 0.05724320  
## 94 0.10 1e+00 0.3816587 0.05857455  
## 95 0.20 1e+00 0.2507692 0.05951684  
## 96 0.50 1e+00 0.2010577 0.04242638  
## 97 1.00 1e+00 0.1900962 0.03470371  
## 98 2.00 1e+00 0.1994231 0.03827365  
## 99 5.00 1e+00 0.2165144 0.04839505  
## 100 10.00 1e+00 0.2243029 0.04706050  
## 101 0.01 2e+00 0.3910096 0.05724320  
## 102 0.02 2e+00 0.3910096 0.05724320  
## 103 0.05 2e+00 0.3910096 0.05724320  
## 104 0.10 2e+00 0.3910096 0.05724320  
## 105 0.20 2e+00 0.2975000 0.05988051  
## 106 0.50 2e+00 0.2118750 0.03482180  
## 107 1.00 2e+00 0.2119471 0.04094387  
## 108 2.00 2e+00 0.2228125 0.03999232  
## 109 5.00 2e+00 0.2258894 0.04842781  
## 110 10.00 2e+00 0.2352644 0.04932202  
## 111 0.01 5e+00 0.3910096 0.05724320  
## 112 0.02 5e+00 0.3910096 0.05724320  
## 113 0.05 5e+00 0.3910096 0.05724320  
## 114 0.10 5e+00 0.3910096 0.05724320  
## 115 0.20 5e+00 0.3582692 0.04648295  
## 116 0.50 5e+00 0.2555288 0.03887680  
## 117 1.00 5e+00 0.2321635 0.03532260  
## 118 2.00 5e+00 0.2337260 0.03000008  
## 119 5.00 5e+00 0.2431010 0.03526006  
## 120 10.00 5e+00 0.2477885 0.04313217  
## 121 0.01 1e+01 0.3910096 0.05724320  
## 122 0.02 1e+01 0.3910096 0.05724320  
## 123 0.05 1e+01 0.3910096 0.05724320  
## 124 0.10 1e+01 0.3910096 0.05724320  
## 125 0.20 1e+01 0.3816587 0.05716832  
## 126 0.50 1e+01 0.2585577 0.04718962  
## 127 1.00 1e+01 0.2492788 0.03860750  
## 128 2.00 1e+01 0.2445913 0.04060172  
## 129 5.00 1e+01 0.2461779 0.04189463  
## 130 10.00 1e+01 0.2539423 0.03568328

svm2.tune$best.performance

## [1] 0.1542067

# Compute the training and test error rates using this new value for cost.  
# Train error rates  
svm2.best.train.pred = predict(svm2.tune$best.model,newdata=train.OJ)  
confusion.Mat=table(obs=train.OJ$Purchase,pred=svm2.best.train.pred)  
confusion.Mat

## pred  
## obs CH MM  
## CH 362 29  
## MM 52 199

# The above object is a confusion matrix thus grabbing the false positive and negative elements will give the error of the prediction  
error\_rate=(confusion.Mat[1,2]+confusion.Mat[2,1])/nrow(train.OJ)  
error\_rate

## [1] 0.1261682

# Test error rates  
svm2.best.test.pred = predict(svm2.tune$best.model,newdata=test.OJ)  
confusion.Mat=table(obs=test.OJ$Purchase,pred=svm2.best.test.pred)  
confusion.Mat

## pred  
## obs CH MM  
## CH 231 31  
## MM 46 120

# The above object is a confusion matrix thus grabbing the false positive and negative elements will give the error of the prediction  
test\_error\_radialTune =(confusion.Mat[1,2]+confusion.Mat[2,1])/nrow(test.OJ)  
test\_error\_radialTune

## [1] 0.1799065

## (g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Tune degree and cost using a grid.

# Fit the model on the training set  
set.seed(123)  
model <- train(Purchase ~., data = train.OJ, method = "svmPoly",  
 trControl = trainControl("cv", number = 10),  
 tuneGrid = expand.grid(C=c(.01,.02,.05,.1,.2,.5,1,2,5,10)  
 ,degree=c(1:5),scale=c(0.01:1)),  
 preProcess = c("center","scale"),  
 tuneLength = 4)  
  
# Print the best tuning parameter sigma and C that  
# maximizes model accuracy  
model$bestTune

## degree scale C  
## 34 4 0.01 1

model$finalModel #training error

## Support Vector Machine object of class "ksvm"   
##   
## SV type: C-svc (classification)   
## parameter : cost C = 1   
##   
## Polynomial kernel function.   
## Hyperparameters : degree = 4 scale = 0.01 offset = 1   
##   
## Number of Support Vectors : 287   
##   
## Objective Function Value : -251.8879   
## Training error : 0.140187

# Compute the training and test error rates using this new value for cost.  
# Train error rates  
svm3.best.train.pred <- model %>% predict(train.OJ)  
confusion.Mat=table(obs=train.OJ$Purchase,pred=svm3.best.train.pred)  
confusion.Mat

## pred  
## obs CH MM  
## CH 356 35  
## MM 55 196

# The above object is a confusion matrix thus grabbing the false positive and negative elements will give the error of the prediction  
error\_rate=(confusion.Mat[1,2]+confusion.Mat[2,1])/nrow(train.OJ)  
error\_rate #training error after tuning

## [1] 0.1401869

# Test error rates  
# Make predictions on the test data  
svm3.best.test.pred <- model %>% predict(test.OJ)  
confusion.Mat=table(obs=test.OJ$Purchase,pred=svm3.best.test.pred)  
confusion.Mat

## pred  
## obs CH MM  
## CH 229 33  
## MM 42 124

# The above object is a confusion matrix thus grabbing the false positive and negative elements will give the error of the prediction  
test\_error\_polyTune=(confusion.Mat[1,2]+confusion.Mat[2,1])/nrow(test.OJ)  
test\_error\_polyTune #test error

## [1] 0.1752336

## (h) Overall, which approach seems to give the best results on this data?

#The model with the lowest test error would be the best approach  
test\_error\_linear

## [1] 0.1682243

test\_error\_linearTune

## [1] 0.1682243

test\_error\_radialTune

## [1] 0.1799065

test\_error\_polyTune

## [1] 0.1752336