Homework-3 ML 562

Sagar Kalauni

2023-11-28

1. Let’s explore the maximal margin classifier on a toy data set. We are given n = 7 observations in p = 2 dimensions. For each observation, there is an associated class label Y .

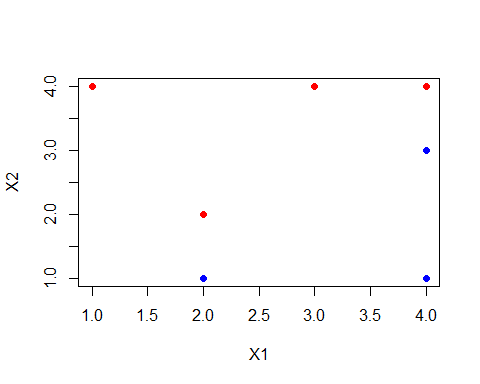
set.seed(12321)  
X1 <- c(3,2,4,1,2,4,4)  
X2 <- c(4,2,4,4,1,3,1)  
Y <- c(rep("Red", 4),rep("Blue", 3))  
mydf <- data.frame(X1, X2, Y)  
  
mydf

## X1 X2 Y  
## 1 3 4 Red  
## 2 2 2 Red  
## 3 4 4 Red  
## 4 1 4 Red  
## 5 2 1 Blue  
## 6 4 3 Blue  
## 7 4 1 Blue

set.seed(12321)  
#install.packages("tidyverse")  
library(ggplot2)  
#install.packages("e1071")  
library(e1071)

1. Sketch the observations.

set.seed(12321)  
# Creating the scatter plot  
plot(mydf$X1, mydf$X2, col=Y, pch=19, xlab = "X1", ylab = "X2")

 Observation: From the above plot we can say that they are linearly seperable.

1. Sketch the optimal separating hyperplane.

set.seed(12321)  
library(e1071)  
mydf$Y=as.factor(mydf$Y)  
  
# Fitting our model with some random cost  
fit.svm = svm(Y ~ ., data = mydf, kernel = "linear", cost = 10, scale = FALSE)  
fit.svm$index

## [1] 2 3 6

-I believe to sketch the optimal separating hyperplane, my model should also have to be the one with the best fit(among the tested cost values), So I will first find the best fitting model and then draw Optimal separating hyperplane with the help of that.

* The optimal separating hyperplane refers to the decision boundary that maximally separates different classes in the feature space.

-Here I want to do cross-validation to find the best model but tune function by default take 10-fold cross validation and my sample size was not enough to do that so I need to do the tunecontrol and preform cross-validation.

# Cross-validation to find the best fit model

set.seed(12321)  
# perform cross-validation  
tune.out <- tune(  
 svm, # SVM function  
 Y ~ ., # Formula for the model  
 data = mydf, # my data frame  
 kernel = "linear", # Linear kernel  
 ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), # Range of cost values(she did not mention in particular which value to take so I am taking of my wish)  
 tunecontrol = tune.control(sampling = "cross", cross = 2) # 2-fold cross-validation  
)  
  
summary(tune.out)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 2-fold cross validation   
##   
## - best parameters:  
## cost  
## 5  
##   
## - best performance: 0.25   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.5833333 0.1178511  
## 2 1e-02 0.5833333 0.1178511  
## 3 1e-01 0.5833333 0.1178511  
## 4 1e+00 0.5833333 0.1178511  
## 5 5e+00 0.2500000 0.3535534  
## 6 1e+01 0.2500000 0.3535534  
## 7 1e+02 0.2500000 0.3535534

Observation: -Here 5e+00 means , so we can see the error is minimum when cost is 5. So our model will be best when cost=5 (amoung the given cost values)

Now we have clear idea which cost will give us our best model, so let’s find our best model

best.mod=svm(Y ~ ., data = mydf, kernel = "linear", cost = 5, scale = FALSE, )  
best.mod

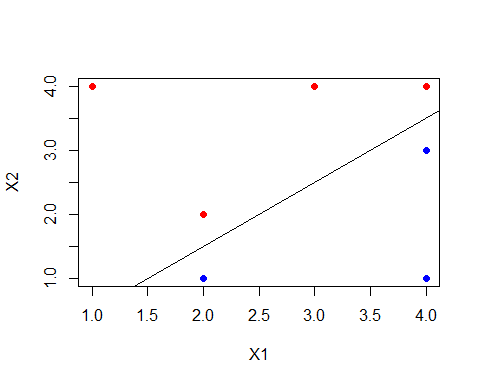
##   
## Call:  
## svm(formula = Y ~ ., data = mydf, kernel = "linear", cost = 5, ,   
## scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 5   
##   
## Number of Support Vectors: 3

set.seed(12321)  
summary(tune.out$best.model)

##   
## Call:  
## best.tune(METHOD = svm, train.x = Y ~ ., data = mydf, ranges = list(cost = c(0.001,   
## 0.01, 0.1, 1, 5, 10, 100)), tunecontrol = tune.control(sampling = "cross",   
## cross = 2), kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 5   
##   
## Number of Support Vectors: 4  
##   
## ( 2 2 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## Blue Red

Now using this best model to sketch the optimal separating hyperplane.

set.seed(12321)  
# Extract beta\_0 and beta\_1  
beta0 = best.mod$rho  
beta = drop(t(best.mod$coefs) %\*% as.matrix(mydf[best.mod$index,1:2]))  
  
# Replot, this time with the solid line representing the optimal(maximal) margin plane.  
plot(X1, X2, col=Y, pch=19, data=mydf)  
abline(beta0/beta[2], -beta[1]/beta[2])

 Here we got our optimal seperating hyperplane In code the a, b arguments above in abline() represent the intercept and slope, single values in the plot functions.

1. Provide the equation for this hyperplane. Describe the classification rule. It should be something along the lines of ?Classify to Red if ANSWER: The equation of the given hyperplane is: , where $\_0 = -1.00041, \_1=-1.999846,\_2=1.999693 $

Hence the exect equation of the hyperplane is: which on simplification became:

set.seed(12321)  
paste("Intercept: ", round(beta0/beta[2],1), ", Slope: ", round(-beta[1]/beta[2],1), sep="")

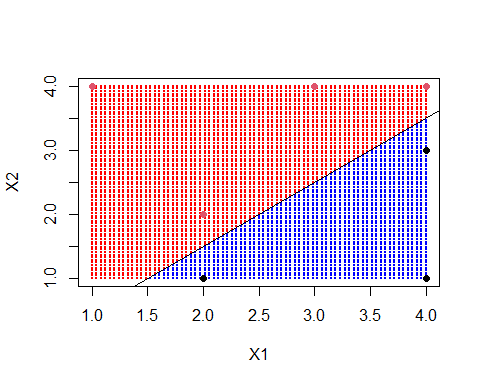
## [1] "Intercept: -0.5, Slope: 1"

If the Values were rounded then the equation becomes:

-The Classification Rule is any point that lies below hyperplane(lower half space) will be classified as blue and any point that lies above the hyperplane(upper half space) will be classified as Red.

Mathematically, any point lies in classified as RED otherwise BLUE

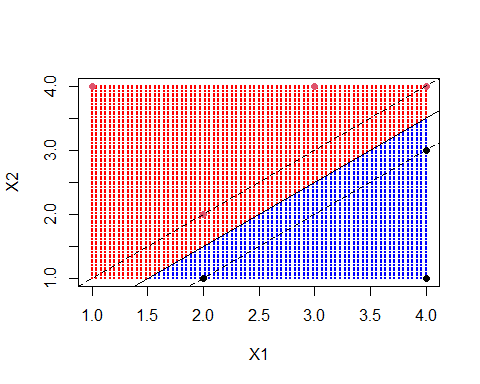
set.seed(12321)  
# Making better plot  
make.grid = function(x, n = 75) {  
 grange = apply(x, 2, range)  
 x1 = seq(from = grange[1,1], to = grange[2,1], length = n)  
 x2 = seq(from = grange[1,2], to = grange[2,2], length = n)  
 expand.grid(X1 = x1, X2 = x2)  
 }  
xgrid = make.grid(mydf)  
ygrid = predict(best.mod, xgrid)  
  
plot(xgrid, col = c("blue","Red")[as.numeric(ygrid)], pch = 20, cex = .2)  
points(mydf, col =mydf$Y, pch = 19)  
#points(mydf[best.mod$index,1:2], pch = 5, cex = 2)  
  
  
### Add the margins  
## you have to do some work to get back the linear coefficients  
beta = t(best.mod$coefs)%\*%as.matrix(mydf[best.mod$index,1:2])  
beta0 = best.mod$rho  
  
abline(beta0 / beta[2], -beta[1] / beta[2])



#abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)  
#abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)

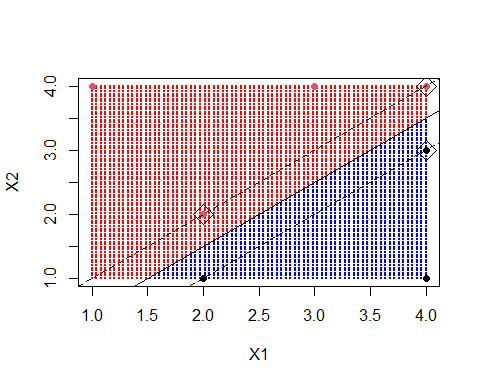
1. On your sketch, indicate the margin for the maximal margin hyperplane.

set.seed(12321)  
# Making better plot  
make.grid = function(x, n = 75) {  
 grange = apply(x, 2, range)  
 x1 = seq(from = grange[1,1], to = grange[2,1], length = n)  
 x2 = seq(from = grange[1,2], to = grange[2,2], length = n)  
 expand.grid(X1 = x1, X2 = x2)  
 }  
xgrid = make.grid(mydf)  
ygrid = predict(best.mod, xgrid)  
  
plot(xgrid, col = c("blue","Red")[as.numeric(ygrid)], pch = 20, cex = .2)  
points(mydf, col =mydf$Y, pch = 19)  
#points(mydf[best.mod$index,1:2], pch = 5, cex = 2)  
  
  
### Add the margins  
## you have to do some work to get back the linear coefficients  
beta = t(best.mod$coefs)%\*%as.matrix(mydf[best.mod$index,1:2])  
beta0 = best.mod$rho  
  
abline(beta0 / beta[2], -beta[1] / beta[2])  
abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)  
abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)

 Observation: -To find the margin length we compute the smallest distance from any training observation to the given separating hyperplane. This is the same as computing the distance from the dashed margin line to the solid hyperplane.The margin width is from the solid line to either of the dashed lines

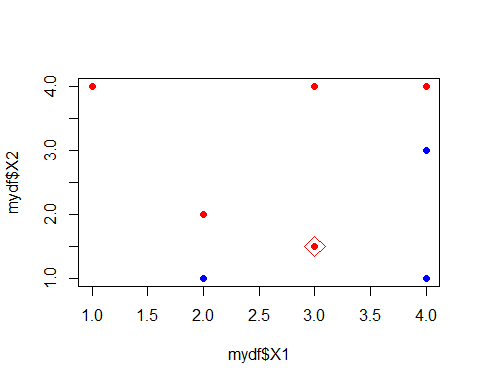
1. Indicate the support vectors for the maximal margin classifier.

set.seed(12321)  
# Making better plot  
make.grid = function(x, n = 75) {  
 grange = apply(x, 2, range)  
 x1 = seq(from = grange[1,1], to = grange[2,1], length = n)  
 x2 = seq(from = grange[1,2], to = grange[2,2], length = n)  
 expand.grid(X1 = x1, X2 = x2)  
 }  
xgrid = make.grid(mydf)  
ygrid = predict(best.mod, xgrid)  
  
plot(xgrid, col = c("blue","Red")[as.numeric(ygrid)], pch = 20, cex = .2)  
points(mydf, col =mydf$Y, pch = 19)  
points(mydf[best.mod$index,1:2], pch = 5, cex = 2)  
  
  
### Add the margins  
## you have to do some work to get back the linear coefficients  
beta = t(best.mod$coefs)%\*%as.matrix(mydf[best.mod$index,1:2])  
beta0 = best.mod$rho  
  
abline(beta0 / beta[2], -beta[1] / beta[2])  
abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)  
abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)

 Observation: -Support vectors are indicated by the square box around them.

1. Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane. Answer: So In order to make data no longer separable by a hyperplane, I just need to add one point (at least, I can add more also) in the opposite side of the halfspace determined by our hyperplane. If our hyperplane classify all point to be blue in the lower halfspace , I will add one Red point over there, then hyperplane can not seperate them. I need to keep in mind that, newly added point should be outside of the margin also

set.seed(12321)  
plot(mydf$X1, mydf$X2, col=Y, pch=19)  
points(3, 1.5, col="Red", pch=19)  
points(3, 1.5, col="red", pch=5, cex=2)



Observation: This newly added data point which is red point with red squre around make the data point linearly inseperable that means we can not seperate our two classes using linear classifier like hyperplane.

1. In this problem, you will use support vector approaches in order to predict Purchase based on the OJ data set.
2. Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

set.seed(100)  
library(ISLR2) #Loading the ISLR2 library in the R working environment

## Warning: package 'ISLR2' was built under R version 4.3.2

set.seed(100)  
# Load the OJ dataset  
data(OJ)  
dim(OJ)

## [1] 1070 18

# Spliting the data into training and testing set  
set.seed(100)   
Index=sample(1:nrow(OJ), 800) # we take 800 data for training set   
train=OJ[Index,]  
test=OJ[-Index,]

1. Fit a linear SVM to the training data using cost=0.01, with Purchase as the response and the other variables as predictors. Describe the results obtained.

set.seed(100)  
library(e1071)  
  
# Fitting a linear model with cost=0.01  
OJ.fit.svm = svm(Purchase ~ ., data =train, kernel = "linear", cost = 0.01, scale = FALSE)  
OJ.fit.svm

##   
## Call:  
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01,   
## scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 0.01   
##   
## Number of Support Vectors: 623

set.seed(100)  
summary(OJ.fit.svm)

##   
## Call:  
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01,   
## scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 0.01   
##   
## Number of Support Vectors: 623  
##   
## ( 312 311 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

Observation: Summary tells us that, the linear kernel was used with cost=0.01 and that there were 623 support vectors, out of which 312 belongs to one class and 311 belongs to the other class. Number of classes are two with levels CH and MM

1. What are the training and test error rates?

set.seed(100)  
# Prediciting the class for our training dataset  
pred\_train=predict(OJ.fit.svm, train)  
pred\_train[1:10] # Looking at the first 10 prediction made by our model in the training dataset

## 503 985 1004 919 470 823 838 903 1031 183   
## CH CH CH CH CH CH MM CH CH CH   
## Levels: CH MM

set.seed(100)  
# Confusion matrix  
table(pred\_train, train$Purchase)

##   
## pred\_train CH MM  
## CH 466 177  
## MM 22 135

Observation: -Looking at the confusion matrix we see that the training error rate is: (177+22)/800= 0.24875 i.e 24.875%

Now predicting the class for our test data set using our model

set.seed(100)  
pred\_test=predict(OJ.fit.svm, test)  
pred\_test[1:10] # Looking at the first 10 prediction made by our model in the Test dataset

## 3 5 7 8 20 25 27 29 33 36   
## CH CH CH CH CH CH CH CH MM CH   
## Levels: CH MM

set.seed(100)  
# Confusion Matrix  
table(pred\_test, test$Purchase)

##   
## pred\_test CH MM  
## CH 157 63  
## MM 8 42

Observation: -Looking at the confusion matrix we see that the test error rate is: (63+8)/270= 0.262963 i.e 26.2963%

1. Tune the linear SVM with various values of cost. Report the cross-validation errors associated with different values of this parameter. Select an optimal cost. Compute the training and test error rates using this new cost value. Comment on your findings.

set.seed(100)  
# perform cross-validation  
OJ.tune.out <- tune(  
 svm, # SVM function  
 Purchase~., # Formula for the model  
 data = train, # my training data frame  
 kernel = "linear", # Linear kernel  
 ranges = list(cost = seq(0.01, 10, length.out = 20)) # Range of cost values(she did not mention in particular which value to take so I am taking of my wish)  
)  
OJ.tune.out

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 6.845263  
##   
## - best performance: 0.17

summary(OJ.tune.out)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 6.845263  
##   
## - best performance: 0.17   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 0.0100000 0.17500 0.04639804  
## 2 0.5357895 0.17500 0.03908680  
## 3 1.0615789 0.17500 0.03908680  
## 4 1.5873684 0.17250 0.03525699  
## 5 2.1131579 0.17125 0.03230175  
## 6 2.6389474 0.17125 0.03438447  
## 7 3.1647368 0.17375 0.03251602  
## 8 3.6905263 0.17250 0.03476109  
## 9 4.2163158 0.17250 0.03476109  
## 10 4.7421053 0.17125 0.03729108  
## 11 5.2678947 0.17125 0.03729108  
## 12 5.7936842 0.17125 0.03729108  
## 13 6.3194737 0.17125 0.03729108  
## 14 6.8452632 0.17000 0.03782269  
## 15 7.3710526 0.17000 0.03782269  
## 16 7.8968421 0.17000 0.03782269  
## 17 8.4226316 0.17000 0.03782269  
## 18 8.9484211 0.17000 0.03782269  
## 19 9.4742105 0.17000 0.03782269  
## 20 10.0000000 0.17000 0.03782269

Observation: -Here we can see the error is minimum when cost is 6.845263. So our model will be best when cost=6.845263 So the best performance model can be obtained using cost=0.01(depending upon the cost which we have tried on , can not say in general)

OJ.best.mod=svm(Purchase~ ., data = train, kernel = "linear", cost = 6.845263, scale = FALSE, )  
OJ.best.mod

##   
## Call:  
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 6.845263,   
## , scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 6.845263   
##   
## Number of Support Vectors: 323

set.seed(100)  
# Prediciting the class for our training dataset using this new best model after cross-validation  
B\_pred\_train=predict(OJ.best.mod, train)  
B\_pred\_train[1:10] # Looking at the first 10 prediction made by our new best model in the training dataset

## 503 985 1004 919 470 823 838 903 1031 183   
## CH CH MM CH CH CH MM CH CH CH   
## Levels: CH MM

set.seed(100)  
# Confusion matrix  
table(B\_pred\_train, train$Purchase)

##   
## B\_pred\_train CH MM  
## CH 429 65  
## MM 59 247

Observation: -Looking at the confusion matrix we see that the training error rate is: (65+59)/800= 0.155 i.e 15.5% for this new best fit model.

Now predicting the class for our test data set using this new best model

set.seed(100)  
B\_pred\_test=predict(OJ.best.mod, test)  
B\_pred\_test[1:10] # Looking at the first 10 prediction made by new best model in the Test dataset

## 3 5 7 8 20 25 27 29 33 36   
## CH CH CH CH CH CH CH CH MM CH   
## Levels: CH MM

# Confusion matrix  
table(B\_pred\_test, test$Purchase)

##   
## B\_pred\_test CH MM  
## CH 144 28  
## MM 21 77

Observation: -Looking at the confusion matrix we see that the test error rate is: (28+21)/270= 0.1814815 i.e 18.14815% for this new best fit model.

Conclusion: This is kind of interesting observation, the training error rate goes down from 24.875% (i) to 15.5% (ii) when using the best model and test error rate goes down from 26.2963% (i) to 18.14815%. So we can say that by doing model tuning we make our model really nice compared the original one.

1. Now repeat (d), with radial basis kernels, with different values of gamma and cost. Comment on your results. Which approach seems to give the better results on this data?

# Radial

set.seed(100)  
library(e1071)  
  
# Fitting a linear model with cost=0.01  
Radial.OJ.svm = svm(Purchase ~ ., data =train, kernel = "radial", gamma=0.5, cost = 5, scale = FALSE)  
summary(Radial.OJ.svm)

##   
## Call:  
## svm(formula = Purchase ~ ., data = train, kernel = "radial", gamma = 0.5,   
## cost = 5, scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 5   
##   
## Number of Support Vectors: 451  
##   
## ( 245 206 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

Summary tells us that, the radial kernel was used with cost=5, gamma=0.5 and that there were 451 support vectors, out of which 245 belongs to one class and 206 belongs to the other class. Number of classes are two with levels CH and MM

set.seed(100)  
# Prediciting the class for our training dataset with radial kernel  
R\_pred\_train=predict(Radial.OJ.svm, train)  
R\_pred\_train[1:10] # Looking at the first 10 prediction made by our model in the training dataset with radial kernel

## 503 985 1004 919 470 823 838 903 1031 183   
## CH CH MM MM CH CH MM CH CH CH   
## Levels: CH MM

set.seed(100)  
# Confusion matrix  
table(R\_pred\_train, train$Purchase)

##   
## R\_pred\_train CH MM  
## CH 459 46  
## MM 29 266

Observation: -Looking at the confusion matrix we see that the training error rate is: (46+29)/800= 0.09375 i.e 9.375%

now predicting for the test data

set.seed(100)  
R\_pred\_test=predict(Radial.OJ.svm, test)  
R\_pred\_test[1:10] # Looking at the first 10 prediction made by our model in the Test dataset

## 3 5 7 8 20 25 27 29 33 36   
## CH CH CH MM CH CH CH CH MM MM   
## Levels: CH MM

set.seed(100)  
# Confusion Matrix  
table(R\_pred\_test, test$Purchase)

##   
## R\_pred\_test CH MM  
## CH 133 38  
## MM 32 67

Observation: -Looking at the confusion matrix we see that the test error rate is: (38+32)/270= 0.2592593 i.e 25.92593% with radial kernel.

Now lets try to find the best model with radial kernel by trying different values of cost and gamma

set.seed(100)  
# perform cross-validation  
R.OJ.tune <- tune(  
 svm, # SVM function  
 Purchase~., # Formula for the model  
 data = train, # my training data frame  
 kernel = "radial", # radial kernel is used  
 ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100),gamma=c(0.5,1,2,3,4)) # Range of cost values and gamma values  
)  
summary(R.OJ.tune)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost gamma  
## 1 0.5  
##   
## - best performance: 0.1825   
##   
## - Detailed performance results:  
## cost gamma error dispersion  
## 1 1e-03 0.5 0.39000 0.03809710  
## 2 1e-02 0.5 0.39000 0.03809710  
## 3 1e-01 0.5 0.30000 0.03864008  
## 4 1e+00 0.5 0.18250 0.04005205  
## 5 5e+00 0.5 0.20375 0.03682259  
## 6 1e+01 0.5 0.20875 0.03775377  
## 7 1e+02 0.5 0.21750 0.03395258  
## 8 1e-03 1.0 0.39000 0.03809710  
## 9 1e-02 1.0 0.39000 0.03809710  
## 10 1e-01 1.0 0.34250 0.04090979  
## 11 1e+00 1.0 0.19375 0.03784563  
## 12 5e+00 1.0 0.21375 0.03606033  
## 13 1e+01 1.0 0.21125 0.03747684  
## 14 1e+02 1.0 0.22750 0.03425801  
## 15 1e-03 2.0 0.39000 0.03809710  
## 16 1e-02 2.0 0.39000 0.03809710  
## 17 1e-01 2.0 0.37375 0.04267529  
## 18 1e+00 2.0 0.21125 0.04059026  
## 19 5e+00 2.0 0.22625 0.03458584  
## 20 1e+01 2.0 0.22625 0.03356689  
## 21 1e+02 2.0 0.23375 0.03335936  
## 22 1e-03 3.0 0.39000 0.03809710  
## 23 1e-02 3.0 0.39000 0.03809710  
## 24 1e-01 3.0 0.38375 0.03729108  
## 25 1e+00 3.0 0.22375 0.03972562  
## 26 5e+00 3.0 0.23125 0.01692508  
## 27 1e+01 3.0 0.23625 0.02389938  
## 28 1e+02 3.0 0.24000 0.03425801  
## 29 1e-03 4.0 0.39000 0.03809710  
## 30 1e-02 4.0 0.39000 0.03809710  
## 31 1e-01 4.0 0.38625 0.03653860  
## 32 1e+00 4.0 0.22625 0.03304563  
## 33 5e+00 4.0 0.23250 0.02220485  
## 34 1e+01 4.0 0.23250 0.03016160  
## 35 1e+02 4.0 0.24625 0.03634805

Observation: -Here we can see the error is minimum when cost is 1 and gamma=0.5. So our model will be best when cost=1 and gamma=0.5(depending upon the cost which we have tried on , can not say in general)

R.OJ.best.mod=svm(Purchase~ ., data = train, kernel = "radial", cost = 1, gamma=0.5, scale = FALSE, )  
summary(R.OJ.best.mod)

##   
## Call:  
## svm(formula = Purchase ~ ., data = train, kernel = "radial", cost = 1,   
## gamma = 0.5, , scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
##   
## Number of Support Vectors: 544  
##   
## ( 287 257 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

set.seed(100)  
# Prediciting the class for our training dataset with radial kernel and best model  
tune\_R\_pred\_train=predict(R.OJ.best.mod, train)  
tune\_R\_pred\_train[1:10] # Looking at the first 10 prediction made by our model in the training dataset with radial kernel and best model

## 503 985 1004 919 470 823 838 903 1031 183   
## MM CH MM MM CH CH MM CH CH CH   
## Levels: CH MM

set.seed(100)  
# Confusion matrix  
table(tune\_R\_pred\_train, train$Purchase)

##   
## tune\_R\_pred\_train CH MM  
## CH 449 88  
## MM 39 224

Observation: -Looking at the confusion matrix we see that the training error rate is: (88+39)/800= 0.15875 i.e 15.875% with radial kernel and best model.

Now predicting the test data set using this new best model with radial kernel

set.seed(100)  
tune\_R\_pred\_test=predict(R.OJ.best.mod, test)  
tune\_R\_pred\_test[1:10] # Looking at the first 10 prediction made by our model in the Test dataset

## 3 5 7 8 20 25 27 29 33 36   
## CH CH CH MM CH CH CH CH MM MM   
## Levels: CH MM

set.seed(100)  
# Confusion matrix  
table(tune\_R\_pred\_test, test$Purchase)

##   
## tune\_R\_pred\_test CH MM  
## CH 137 47  
## MM 28 58

Observation: -Looking at the confusion matrix we see that the test error rate is: (47+28)/270= 0.2777778 i.e 27.77778% with radial kernel and best model.

Conclusion: From above we see that the training error rate went up from 9.375% (i)to 15.875% (ii) when using the best model and test error rate went up from 25.92593% (i)to 27.77778%. So we can say that by doing model tuning we did not get our new model as good model for predicting the test set compared to original.

# Comprasion

comparing the best linear and best radial model, we conclude that best linear model was more nicer then best radial for predicting this test dataset because best linear model has test error rate: 18.14815% only but the best radial model has the test error rate of 27.77778%

1. Now repeat again, with polynomial basis kernels, with different values of degree and cost. Comment on your results. Which approach (kernel) seems to give the best results on this data?

# Polynomial

set.seed(100)  
library(e1071)  
  
# Fitting a polynomial model with cost=5 and degree=3  
Poly.OJ.svm = svm(Purchase ~ ., data =train, kernel = "polynomial", degree=3, cost = 5, scale = FALSE)  
summary(Poly.OJ.svm)

##   
## Call:  
## svm(formula = Purchase ~ ., data = train, kernel = "polynomial",   
## degree = 3, cost = 5, scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: polynomial   
## cost: 5   
## degree: 3   
## coef.0: 0   
##   
## Number of Support Vectors: 226  
##   
## ( 115 111 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

Summary tells us that, the polynomial kernel was used with cost=5, degree=3 and that there were 226 support vectors, out of which 115 belongs to one class and 111 belongs to the other class. Number of classes are two with levels CH and MM

set.seed(100)  
# Prediciting the class for our training dataset with polynomial kernel  
poly\_pred\_train=predict(Poly.OJ.svm, train)  
poly\_pred\_train[1:10] # Looking at the first 10 prediction made by our model in the training dataset with polynomial kernel

## 503 985 1004 919 470 823 838 903 1031 183   
## CH CH MM CH CH CH MM CH CH CH   
## Levels: CH MM

set.seed(100)  
# Confusion matrix  
table(poly\_pred\_train, train$Purchase)

##   
## poly\_pred\_train CH MM  
## CH 429 71  
## MM 59 241

Observation: -Looking at the confusion matrix we see that the training error rate is: (71+59)/800= 0.1625 i.e 16.25%

now predicting for the test data

set.seed(100)  
poly\_pred\_test=predict(Poly.OJ.svm, test)  
poly\_pred\_test[1:10] # Looking at the first 10 prediction made by our model in the Test dataset

## 3 5 7 8 20 25 27 29 33 36   
## CH CH CH CH CH CH CH CH MM CH   
## Levels: CH MM

set.seed(100)  
# Confusion Matrix  
table(poly\_pred\_test, test$Purchase)

##   
## poly\_pred\_test CH MM  
## CH 143 26  
## MM 22 79

Observation: -Looking at the confusion matrix we see that the test error rate is: (26+22)/270= 0.1777778 i.e 17.77778% with radial kernel.

Now lets try to find the best model with polynomial kernel by trying different values of cost and degree

set.seed(100)  
# perform cross-validation  
poly.OJ.tune <- tune(  
 svm, # SVM function  
 Purchase~., # Formula for the model  
 data = train, # my training data frame  
 kernel = "polynomial", # polynomial kernel is used  
 ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,50,100),degree=c(0.25,0.33,0.5,1,2,3,4)) # Range of cost values and degree values  
)  
summary(poly.OJ.tune)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost degree  
## 1 1  
##   
## - best performance: 0.16625   
##   
## - Detailed performance results:  
## cost degree error dispersion  
## 1 1e-03 0.25 0.39000 0.03809710  
## 2 1e-02 0.25 0.39000 0.03809710  
## 3 1e-01 0.25 0.39000 0.03809710  
## 4 1e+00 0.25 0.39000 0.03809710  
## 5 5e+00 0.25 0.39000 0.03809710  
## 6 1e+01 0.25 0.39000 0.03809710  
## 7 5e+01 0.25 0.39000 0.03809710  
## 8 1e+02 0.25 0.39000 0.03809710  
## 9 1e-03 0.33 0.39000 0.03809710  
## 10 1e-02 0.33 0.39000 0.03809710  
## 11 1e-01 0.33 0.39000 0.03809710  
## 12 1e+00 0.33 0.39000 0.03809710  
## 13 5e+00 0.33 0.39000 0.03809710  
## 14 1e+01 0.33 0.39000 0.03809710  
## 15 5e+01 0.33 0.39000 0.03809710  
## 16 1e+02 0.33 0.39000 0.03809710  
## 17 1e-03 0.50 0.39000 0.03809710  
## 18 1e-02 0.50 0.39000 0.03809710  
## 19 1e-01 0.50 0.39000 0.03809710  
## 20 1e+00 0.50 0.39000 0.03809710  
## 21 5e+00 0.50 0.39000 0.03809710  
## 22 1e+01 0.50 0.39000 0.03809710  
## 23 5e+01 0.50 0.39000 0.03809710  
## 24 1e+02 0.50 0.39000 0.03809710  
## 25 1e-03 1.00 0.39000 0.03809710  
## 26 1e-02 1.00 0.38750 0.03908680  
## 27 1e-01 1.00 0.17000 0.03827895  
## 28 1e+00 1.00 0.16625 0.04411554  
## 29 5e+00 1.00 0.17375 0.03928617  
## 30 1e+01 1.00 0.17375 0.03928617  
## 31 5e+01 1.00 0.17125 0.03438447  
## 32 1e+02 1.00 0.17125 0.03729108  
## 33 1e-03 2.00 0.39000 0.03809710  
## 34 1e-02 2.00 0.38875 0.03972562  
## 35 1e-01 2.00 0.31875 0.04686342  
## 36 1e+00 2.00 0.19375 0.03019037  
## 37 5e+00 2.00 0.17875 0.03866254  
## 38 1e+01 2.00 0.17750 0.04031129  
## 39 5e+01 2.00 0.17250 0.03574602  
## 40 1e+02 2.00 0.17875 0.03910900  
## 41 1e-03 3.00 0.39000 0.03809710  
## 42 1e-02 3.00 0.37375 0.04387878  
## 43 1e-01 3.00 0.28625 0.04226652  
## 44 1e+00 3.00 0.18375 0.04210189  
## 45 5e+00 3.00 0.17250 0.03525699  
## 46 1e+01 3.00 0.17500 0.03333333  
## 47 5e+01 3.00 0.19125 0.02503470  
## 48 1e+02 3.00 0.20250 0.02874698  
## 49 1e-03 4.00 0.39000 0.03809710  
## 50 1e-02 4.00 0.37375 0.04387878  
## 51 1e-01 4.00 0.31500 0.04479893  
## 52 1e+00 4.00 0.22625 0.04803428  
## 53 5e+00 4.00 0.20250 0.04158325  
## 54 1e+01 4.00 0.19875 0.03508422  
## 55 5e+01 4.00 0.19875 0.03087272  
## 56 1e+02 4.00 0.19375 0.02841288

(Funny event, I have to wait approx 3 min to run this code) Observation: -Here we can see the error is minimum when cost is 1 and degree=1. So our model will be best when cost=1 and gamma=0.5(depending upon the cost which we have tried on , can not say in general)

poly.OJ.best.mod=svm(Purchase~ ., data = train, kernel = "polynomial", cost = 1, degree=1, scale = FALSE, )  
summary(poly.OJ.best.mod)

##   
## Call:  
## svm(formula = Purchase ~ ., data = train, kernel = "polynomial",   
## cost = 1, degree = 1, , scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: polynomial   
## cost: 1   
## degree: 1   
## coef.0: 0   
##   
## Number of Support Vectors: 484  
##   
## ( 242 242 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

set.seed(100)  
# Prediciting the class for our training dataset with polynomial kernel and best model  
tune\_poly\_pred\_train=predict(poly.OJ.best.mod, train)  
tune\_poly\_pred\_train[1:10] # Looking at the first 10 prediction made by our model in the training dataset with polynomial kernel and best model

## 503 985 1004 919 470 823 838 903 1031 183   
## CH CH MM CH CH CH MM CH CH CH   
## Levels: CH MM

set.seed(100)  
# Confusion matrix  
table(tune\_poly\_pred\_train, train$Purchase)

##   
## tune\_poly\_pred\_train CH MM  
## CH 429 87  
## MM 59 225

Observation: -Looking at the confusion matrix we see that the training error rate is: (87+59)/800= 0.1825 i.e 18.25% with polynomial kernel and best model.

Now predicting the test data set using this new best model with polynomial kernel

set.seed(100)  
tune\_poly\_pred\_test=predict(poly.OJ.best.mod, test)  
tune\_poly\_pred\_test[1:10] # Looking at the first 10 prediction made by our model in the Test dataset

## 3 5 7 8 20 25 27 29 33 36   
## CH CH CH CH CH CH CH CH MM CH   
## Levels: CH MM

set.seed(100)  
# Confusion matrix  
table(tune\_poly\_pred\_test, test$Purchase)

##   
## tune\_poly\_pred\_test CH MM  
## CH 147 23  
## MM 18 82

Observation: -Looking at the confusion matrix we see that the test error rate is: (23+18)/270= 0.1518519 i.e 15.18519% with polynomial kernel and best model.

Conclusion: From above we see that the training error rate went up from 16.25% (i)to 18.25% (ii) when using the best model and test error rate went down from 17.77778% (i)to 15.18519%. So we can say that by doing model tuning we did get our new model as good model for predicting the test set.

# Comprasion

comparing the best linear and best radial model and best polynomial mode, we conclude that best linear model was more nicer then best radial for predicting this test dataset because only but the

-best linear model has test error rate: 18.14815% -best radial model has the test error rate of 27.77778% -best polynomial model has the test error rate of 15.18519%

(among my given values of cost, gamma, degree)We can say polonomial kernel is best, linear is second best and radial goes last for predicting our test data.

1. Perform gradient boost (using gbm function in R) on the training set with 1,000 trees for a chosen values of the shrinkage parameter. You may experiment with a range of values of the shrinkage parameter. Answer: Before applying gradient boost, we will first convert data type of our purchase variable[The reference for this is book page no. 174, chapter 4(for exam)]

contrasts(OJ$Purchase)

## MM  
## CH 0  
## MM 1

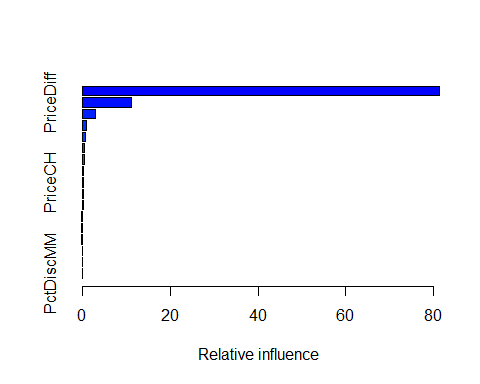
library(gbm)

## Warning: package 'gbm' was built under R version 4.3.2

## Loaded gbm 2.1.8.1

# Converted all my training data set to binary response  
OJ.train = train  
OJ.train$Purchase = factor(OJ.train$Purchase, levels=c("CH","MM"), labels=c(0,1))  
OJ.train$Purchase = as.integer(OJ.train$Purchase)-1  
  
# Converted all my Testing data set to binary response  
OJ.test = test  
OJ.test$Purchase = factor(OJ.test$Purchase, levels=c("CH","MM"), labels=c(0,1))  
OJ.test$Purchase = as.integer(OJ.test$Purchase)-1  
  
# What I did here is that I first convert the Purchase variable to factor 0,1 from factor CH, MM.  
# After that I changed Purchase to numeric so it become 1,2 but I need 0,1 so subtracted 1 from both

set.seed(100)  
#Trying learning rate of 0.001(shrinkage paremeter)  
boost.OJ\_1 = gbm(Purchase ~ ., data=OJ.train, distribution="bernoulli",  
 n.trees=1000, interaction.depth=4, shrinkage=0.001)  
  
#boost.OJ.pred = predict(boost.OJ, newdata=OJ.boost[test.id, ], n.trees=5000, type="response")  
summary(boost.OJ\_1)

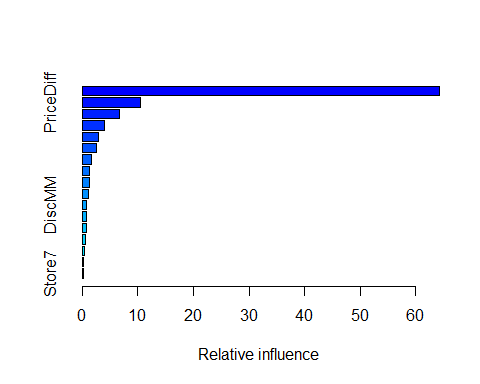


## var rel.inf  
## LoyalCH LoyalCH 81.31581802  
## PriceDiff PriceDiff 11.13510966  
## ListPriceDiff ListPriceDiff 3.09025031  
## StoreID StoreID 1.01439548  
## SalePriceMM SalePriceMM 0.76137059  
## WeekofPurchase WeekofPurchase 0.54789484  
## STORE STORE 0.49470665  
## SpecialCH SpecialCH 0.32338489  
## PriceCH PriceCH 0.27845690  
## SalePriceCH SalePriceCH 0.26143073  
## PriceMM PriceMM 0.21336105  
## DiscCH DiscCH 0.13650070  
## Store7 Store7 0.13593927  
## DiscMM DiscMM 0.12808395  
## SpecialMM SpecialMM 0.09607851  
## PctDiscCH PctDiscCH 0.04290508  
## PctDiscMM PctDiscMM 0.02431337

Observation: We see that LoyalCH and PriceDiff are the most important variables.

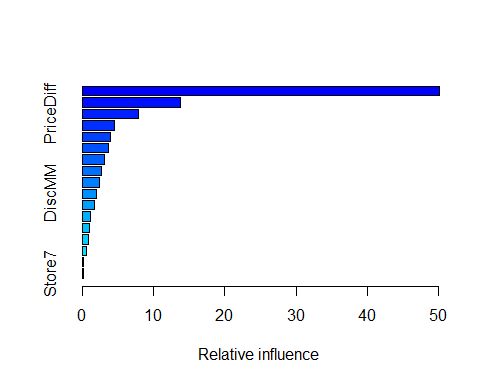
Trying different values of the learning rate(Shrinkage parameter)

#Trying learning rate of 0.01(shrinkage paremeter)  
boost.OJ\_2=gbm(Purchase~., data = OJ.train, n.trees = 1000, distribution ="bernoulli", interaction.depth =4, shrinkage = 0.01)  
  
summary(boost.OJ\_2)



## var rel.inf  
## LoyalCH LoyalCH 64.2716329  
## PriceDiff PriceDiff 10.5343962  
## WeekofPurchase WeekofPurchase 6.6289727  
## ListPriceDiff ListPriceDiff 4.0015017  
## StoreID StoreID 2.9709493  
## SalePriceMM SalePriceMM 2.6095022  
## STORE STORE 1.7452825  
## SalePriceCH SalePriceCH 1.3418927  
## PriceCH PriceCH 1.2258143  
## PriceMM PriceMM 1.0903073  
## DiscMM DiscMM 0.8312014  
## SpecialCH SpecialCH 0.7394305  
## SpecialMM SpecialMM 0.6875707  
## DiscCH DiscCH 0.4919267  
## PctDiscMM PctDiscMM 0.3511397  
## PctDiscCH PctDiscCH 0.2580347  
## Store7 Store7 0.2204444

#Trying learning rate of 0.01(shrinkage paremeter)  
boost.OJ\_3=gbm(Purchase~., data = OJ.train, n.trees = 1000, distribution ="bernoulli", interaction.depth =4, shrinkage = 0.1)  
  
summary(boost.OJ\_3)



## var rel.inf  
## LoyalCH LoyalCH 50.1068807  
## WeekofPurchase WeekofPurchase 13.7551499  
## PriceDiff PriceDiff 7.8611662  
## ListPriceDiff ListPriceDiff 4.5610340  
## StoreID StoreID 4.0045121  
## SalePriceMM SalePriceMM 3.6932731  
## STORE STORE 3.0898383  
## PriceCH PriceCH 2.7594595  
## PriceMM PriceMM 2.4742854  
## DiscMM DiscMM 2.0401289  
## SalePriceCH SalePriceCH 1.7385279  
## SpecialMM SpecialMM 1.1263269  
## SpecialCH SpecialCH 1.0208753  
## DiscCH DiscCH 0.8781028  
## PctDiscMM PctDiscMM 0.5774586  
## PctDiscCH PctDiscCH 0.1658227  
## Store7 Store7 0.1471577

[Ask her: Can I say as the learning rate increase other variable then LoyalCH are also becoming more and more important each time?]

1. Which variables appear to be the most important predictors in the boost model? Answer:- LoyalCH variable appears to be the most important predictor from the boost model.
2. Use the boosting model to predict the response on the test data. Form a confusion matrix. How does this compare with the result SVM obtained? Answer: for predicting we use the modle with learning rate 0.1

set.seed(100)  
glm.probs=predict(boost.OJ\_3 , OJ.test, type = "response")

## Using 1000 trees...

glm.probs[1:10]

## [1] 0.007514068 0.023404826 0.008723721 0.099361581 0.141581807 0.002850720  
## [7] 0.003545146 0.002283526 0.368595711 0.021847529

glm.pred <- rep("CH", 270)  
glm.pred[glm.probs > .5] = "MM"

table(glm.pred, OJ.test$Purchase)

##   
## glm.pred 0 1  
## CH 137 27  
## MM 28 78

Observation: -Looking at the confusion matrix we see that the test error rate is: (27+28)/270= 0.2037037 i.e 20.37037%

# Comparing this boosting model to SVM model

-The bosting model with learning rate 0.1 has thas the test error rate= 20.37037% -best linear model has test error rate: 18.14815% -best radial model has the test error rate of 27.77778% -best polynomial model has the test error rate of 15.18519%

So this boosting model only did better as compared to svm model with radial kernal in our test dataset. With other kernal smv model did much better then 20.370.7%

———————————————–THE END—————————————————-