489 HW6

Nicholas Thompson

11/4/2021

#Below is Question 1 in writing. Maximal margin denoted with 1b

A picture containing letter

Description automatically generated

set.seed(333)  
attach(OJ)  
  
train=sample(1:nrow(OJ),800)  
  
OJ.tr=OJ[train,]  
OJ.te=OJ[-train,]  
nrow(OJ)

## [1] 1070

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 ...

#2b  
svm2b = svm(Purchase ~ ., data = OJ.tr, kernel = "linear", cost = 0.01)  
summary(svm2b)

##   
## Call:  
## svm(formula = Purchase ~ ., data = OJ.tr, kernel = "linear", cost = 0.01)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 0.01   
##   
## Number of Support Vectors: 452  
##   
## ( 226 226 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

#The support vector classifier created 440 support vectors from our training data, 220 for each class of the response Purchase (CH and MM)   
  
#2c  
#training error  
pred2c.tr=predict(svm2b,OJ.tr)  
table(OJ.tr$Purchase, pred2c.tr)

## pred2c.tr  
## CH MM  
## CH 421 57  
## MM 79 243

mean(pred2c.tr != OJ.tr$Purchase)

## [1] 0.17

#0.17  
  
#testing error  
pred2c.te=predict(svm2b,OJ.te)  
table(OJ.te$Purchase,pred2c.te)

## pred2c.te  
## CH MM  
## CH 157 18  
## MM 20 75

mean(pred2c.te != OJ.te$Purchase)

## [1] 0.1407407

#0.1407407  
  
#2d  
tune.out2d=tune(svm,Purchase~.,data=OJ.tr,kernel="linear",ranges=list(cost=10^seq(-2,1,by=0.20)))  
summary(tune.out2d)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 0.01584893  
##   
## - best performance: 0.1775   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 0.01000000 0.18125 0.05597929  
## 2 0.01584893 0.17750 0.06032320  
## 3 0.02511886 0.17875 0.05529278  
## 4 0.03981072 0.18250 0.05779514  
## 5 0.06309573 0.17875 0.05804991  
## 6 0.10000000 0.18375 0.05864500  
## 7 0.15848932 0.18125 0.05781015  
## 8 0.25118864 0.18375 0.05864500  
## 9 0.39810717 0.18250 0.05986095  
## 10 0.63095734 0.18375 0.06265259  
## 11 1.00000000 0.18125 0.06187184  
## 12 1.58489319 0.18125 0.05781015  
## 13 2.51188643 0.17875 0.05714565  
## 14 3.98107171 0.17875 0.05560588  
## 15 6.30957344 0.17875 0.05434266  
## 16 10.00000000 0.18000 0.05210833

#Based on a parameter tuning comparing 16 values for cost, the optimal cost is 0.1 with the lowest cross validation error of 0.17375  
  
svm2d=svm(Purchase ~ ., data = OJ.tr, kernel = "linear", cost = 0.1)  
#lets see how many support vectors this produces just for fun  
summary(svm2d)

##   
## Call:  
## svm(formula = Purchase ~ ., data = OJ.tr, kernel = "linear", cost = 0.1)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 0.1   
##   
## Number of Support Vectors: 365  
##   
## ( 184 181 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

#365 support vectors, 184 to class CH and 181 to class MM  
  
#training error  
pred2d.tr=predict(svm2d,OJ.tr)  
table(OJ.tr$Purchase, pred2d.tr)

## pred2d.tr  
## CH MM  
## CH 418 60  
## MM 75 247

mean(pred2d.tr != OJ.tr$Purchase)

## [1] 0.16875

#0.16875  
#testing error  
pred2d.te=predict(svm2d,OJ.te)  
table(OJ.te$Purchase,pred2d.te)

## pred2d.te  
## CH MM  
## CH 158 17  
## MM 21 74

mean(pred2d.te != OJ.te$Purchase)

## [1] 0.1407407

#same as the original svm  
  
#linear svm with cost of 0.1 produces lower training error but the exact same testing error rate. Curious  
  
#2e  
#let's find an optimal radial basis kernel  
#for sake of time, we are only testing 7 costs and 5 gamma values  
tune.radker=tune(svm,Purchase~.,data=OJ.tr,kernel="radial",ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100),gamma=c(0.5,1,2,3,4)))  
summary(tune.radker)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost gamma  
## 1 0.5  
##   
## - best performance: 0.19375   
##   
## - Detailed performance results:  
## cost gamma error dispersion  
## 1 1e-03 0.5 0.40250 0.04518481  
## 2 1e-02 0.5 0.40250 0.04518481  
## 3 1e-01 0.5 0.28875 0.04427267  
## 4 1e+00 0.5 0.19375 0.04218428  
## 5 5e+00 0.5 0.20000 0.03773077  
## 6 1e+01 0.5 0.20750 0.04684490  
## 7 1e+02 0.5 0.24125 0.04372023  
## 8 1e-03 1.0 0.40250 0.04518481  
## 9 1e-02 1.0 0.40250 0.04518481  
## 10 1e-01 1.0 0.34375 0.04649149  
## 11 1e+00 1.0 0.19750 0.03670453  
## 12 5e+00 1.0 0.20500 0.04257347  
## 13 1e+01 1.0 0.22250 0.05096295  
## 14 1e+02 1.0 0.25250 0.04958158  
## 15 1e-03 2.0 0.40250 0.04518481  
## 16 1e-02 2.0 0.40250 0.04518481  
## 17 1e-01 2.0 0.38875 0.04767147  
## 18 1e+00 2.0 0.21125 0.03653860  
## 19 5e+00 2.0 0.23000 0.05041494  
## 20 1e+01 2.0 0.23375 0.05402224  
## 21 1e+02 2.0 0.25500 0.05109903  
## 22 1e-03 3.0 0.40250 0.04518481  
## 23 1e-02 3.0 0.40250 0.04518481  
## 24 1e-01 3.0 0.39500 0.04684490  
## 25 1e+00 3.0 0.22375 0.05185785  
## 26 5e+00 3.0 0.24500 0.05309844  
## 27 1e+01 3.0 0.25375 0.05138701  
## 28 1e+02 3.0 0.25875 0.05172376  
## 29 1e-03 4.0 0.40250 0.04518481  
## 30 1e-02 4.0 0.40250 0.04518481  
## 31 1e-01 4.0 0.39625 0.04966904  
## 32 1e+00 4.0 0.23000 0.05688683  
## 33 5e+00 4.0 0.25375 0.05172376  
## 34 1e+01 4.0 0.25750 0.04972145  
## 35 1e+02 4.0 0.26750 0.05688683

#from parameter tuning comparing 35 combinations, a cost of 1 and a gamma of 0.5 produce the lowest cv error  
  
svm.radker = svm(Purchase ~ ., data = OJ.tr, kernel = "radial",gamma=0.5, cost = 1, scale = FALSE)  
summary(svm.radker)

##   
## Call:  
## svm(formula = Purchase ~ ., data = OJ.tr, kernel = "radial", gamma = 0.5,   
## cost = 1, scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
##   
## Number of Support Vectors: 554  
##   
## ( 293 261 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

#554 support vectors , 293 to CH and 261 to MM  
  
#training error  
predker.tr=predict(svm.radker,OJ.tr)  
table(OJ.tr$Purchase, predker.tr)

## predker.tr  
## CH MM  
## CH 436 42  
## MM 93 229

mean(predker.tr != OJ.tr$Purchase)

## [1] 0.16875

#0.16875  
#testing error  
predker.te=predict(svm.radker,OJ.te)  
table(OJ.te$Purchase,predker.te)

## predker.te  
## CH MM  
## CH 152 23  
## MM 45 50

mean(predker.te != OJ.te$Purchase)

## [1] 0.2518519

#0.2518519  
  
#radial kernal has same training error as optimal linear svm but higher testing error. Based on the lower testing error, I believe the optimal linear svm is the best approach, further it has the benefit of having the lowest training error out of the 3 svms computed.  
  
#2f  
#polynomial kernel tuning  
tune.poly=tune(svm,Purchase~.,data=OJ.tr,kernel="polynomial",ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100),degree=c(1:5)))  
summary(tune.poly)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost degree  
## 1 1  
##   
## - best performance: 0.18125   
##   
## - Detailed performance results:  
## cost degree error dispersion  
## 1 1e-03 1 0.40250 0.07355081  
## 2 1e-02 1 0.39375 0.08705243  
## 3 1e-01 1 0.18250 0.04090979  
## 4 1e+00 1 0.18125 0.04535738  
## 5 5e+00 1 0.18375 0.04489571  
## 6 1e+01 1 0.18750 0.04750731  
## 7 1e+02 1 0.18125 0.04573854  
## 8 1e-03 2 0.40250 0.07355081  
## 9 1e-02 2 0.38250 0.07481459  
## 10 1e-01 2 0.33250 0.07642171  
## 11 1e+00 2 0.19875 0.05252314  
## 12 5e+00 2 0.18625 0.05118390  
## 13 1e+01 2 0.18125 0.04759858  
## 14 1e+02 2 0.18750 0.04564355  
## 15 1e-03 3 0.40250 0.07355081  
## 16 1e-02 3 0.37625 0.07827347  
## 17 1e-01 3 0.31000 0.07564537  
## 18 1e+00 3 0.20125 0.04730589  
## 19 5e+00 3 0.20125 0.04267529  
## 20 1e+01 3 0.19500 0.04048319  
## 21 1e+02 3 0.20750 0.04297932  
## 22 1e-03 4 0.40250 0.07355081  
## 23 1e-02 4 0.37625 0.07827347  
## 24 1e-01 4 0.31750 0.07799573  
## 25 1e+00 4 0.25375 0.05434266  
## 26 5e+00 4 0.21000 0.06313566  
## 27 1e+01 4 0.21375 0.05541823  
## 28 1e+02 4 0.21500 0.04816061  
## 29 1e-03 5 0.37625 0.07827347  
## 30 1e-02 5 0.37875 0.07638763  
## 31 1e-01 5 0.32625 0.06857933  
## 32 1e+00 5 0.25375 0.04604120  
## 33 5e+00 5 0.22625 0.05252314  
## 34 1e+01 5 0.22750 0.05296750  
## 35 1e+02 5 0.21750 0.04005205

#optimal polynomial kernel: cost=10,degree=1  
  
svm.poly <-svm(Purchase~ ., kernel="polynomial",data=OJ.tr,cost=10, degree=1)  
summary(svm.poly)

##   
## Call:  
## svm(formula = Purchase ~ ., data = OJ.tr, kernel = "polynomial",   
## cost = 10, degree = 1)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: polynomial   
## cost: 10   
## degree: 1   
## coef.0: 0   
##   
## Number of Support Vectors: 349  
##   
## ( 175 174 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CH MM

#175 support vectors of class CH, 174 support vectors of class MM  
  
#training error  
predpoly.tr=predict(svm.poly,OJ.tr)  
table(OJ.tr$Purchase, predpoly.tr)

## predpoly.tr  
## CH MM  
## CH 420 58  
## MM 78 244

mean(predpoly.tr != OJ.tr$Purchase)

## [1] 0.17

#0.17  
#testing error  
predpoly.te=predict(svm.poly,OJ.te)  
table(OJ.te$Purchase,predpoly.te)

## predpoly.te  
## CH MM  
## CH 158 17  
## MM 21 74

mean(predpoly.te != OJ.te$Purchase)

## [1] 0.1407407

#0.1407407  
  
#This polynomial kernel produces a lower amount of support vectors than the first linear svm and yet it gives the same training and testing error.  
#If we are comparing only kernels here, then the polynomial is the best by far compared to radial. If we compare to the other two svms, then the optimal linear svm is still our best support vector machine.