Homework 4

Team 3

4/26/2021

## ISTM660 Homework 4 - Team 3

Pre-requisites:

# install.packages("e1071")  
# install.packages("randomForest")  
# install.packages("rstudioapi")  
# install.packages("pROC")

Predicting with Random Forests (10 points) a - Create a classification random forest with the training data specifying 500 trees and using all the variables (bagging).

# Get active path, clear environment variables, clear console, load data & packages  
library(rstudioapi)  
library(randomForest)

## Warning: package 'randomForest' was built under R version 4.0.5

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

current\_path = rstudioapi::getActiveDocumentContext()$path   
setwd(dirname(current\_path ))  
rm(list=ls())  
cat("\014")

load("transaction.rdata")  
  
#Create training and testing data  
set.seed(25)  
indx = sample(1:nrow(trans), nrow(trans)/2)  
  
train = trans[indx,]  
test = trans[-indx,]  
  
#Perform bagging with mtry set as total number of variables  
trans.bag=randomForest(class~.,data=train,mtry=13,importance=TRUE, ntree=500)  
trans.bag

##   
## Call:  
## randomForest(formula = class ~ ., data = train, mtry = 13, importance = TRUE, ntree = 500)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 13  
##   
## OOB estimate of error rate: 6.38%  
## Confusion matrix:  
## 0 1 class.error  
## 0 412 9 0.02137767  
## 1 21 28 0.42857143

yhat = predict(trans.bag, newdata = test)

b - Assess the accuracy of this model.

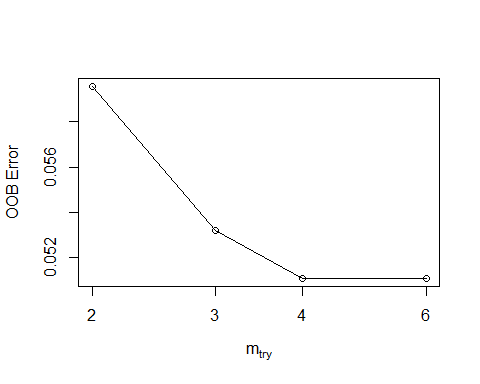
paste("Bagging Accuracy = ",mean(yhat==test$class))

## [1] "Bagging Accuracy = 0.953191489361702"

c - Tune your random forest model on the number of variables. What is your best model? How accurate is it?

x <- test[,1:13]  
y <- test[,14]  
  
set.seed(25)  
bestmtry <- tuneRF(x, y, stepFactor=1.5, improve=1e-5, ntree=500)

## mtry = 3 OOB error = 5.32%   
## Searching left ...  
## mtry = 2 OOB error = 5.96%   
## -0.12 1e-05   
## Searching right ...  
## mtry = 4 OOB error = 5.11%   
## 0.04 1e-05   
## mtry = 6 OOB error = 5.11%   
## 0 1e-05



print(bestmtry)

## mtry OOBError  
## 2.OOB 2 0.05957447  
## 3.OOB 3 0.05319149  
## 4.OOB 4 0.05106383  
## 6.OOB 6 0.05106383

#As seen from the plot, best mtry value is 4.  
tuned.rf=randomForest(class~.,data=train,mtry=4,importance=TRUE, ntree=500)  
tuned.rf

##   
## Call:  
## randomForest(formula = class ~ ., data = train, mtry = 4, importance = TRUE, ntree = 500)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 4  
##   
## OOB estimate of error rate: 6.6%  
## Confusion matrix:  
## 0 1 class.error  
## 0 415 6 0.01425178  
## 1 25 24 0.51020408

tuned.yhat = predict(tuned.rf,newdata=test)  
paste("Accuracy of model with 4 predictors (tuned) = ",mean(tuned.yhat==test$class))

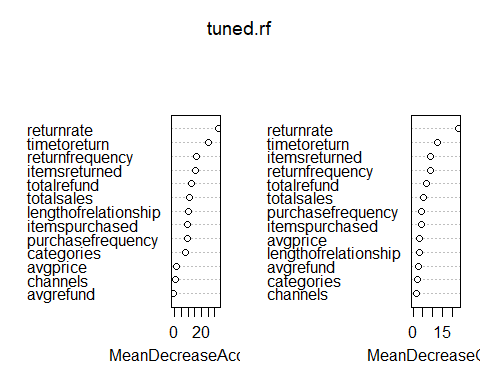
## [1] "Accuracy of model with 4 predictors (tuned) = 0.936170212765957"

d - Assess variable importance. What is the least important variable?

importance(tuned.rf)

## 0 1 MeanDecreaseAccuracy  
## lengthofrelationship 10.6800035 -1.1033553 10.3431229  
## itemsreturned 12.0503690 10.7553256 15.2889558  
## itemspurchased 8.9380316 2.1237003 9.5185741  
## totalsales 10.5962885 1.5780842 11.2466765  
## totalrefund 11.1963373 5.4356570 12.6558940  
## categories 8.7390162 -2.3750297 8.1838829  
## timetoreturn 17.0516356 22.6452710 25.3472552  
## channels 3.1464004 -3.7650260 0.8395294  
## avgprice 2.3289807 -1.7159431 1.1870156  
## purchasefrequency 9.4742162 -0.2484333 9.4462731  
## returnfrequency 14.6603463 8.1525404 16.1959993  
## avgrefund -0.4954901 -1.7772890 -1.1522472  
## returnrate 21.7919841 31.9050902 32.9692125  
## MeanDecreaseGini  
## lengthofrelationship 3.257082  
## itemsreturned 8.916360  
## itemspurchased 4.092214  
## totalsales 5.085536  
## totalrefund 6.837320  
## categories 2.409855  
## timetoreturn 12.551012  
## channels 1.959910  
## avgprice 3.313799  
## purchasefrequency 4.449147  
## returnfrequency 8.781700  
## avgrefund 2.929150  
## returnrate 23.373964

varImpPlot(tuned.rf)



The least important variables are channels and avgprice. We will drop these now.

e - Does dropping the least important variables (based on %IncMSE column in the #generated importance table) improve your best model from c? #Which variables would you drop and why?

trans.rf=randomForest(class~.-channels-avgprice,data=train,mtry=10,ntree=500,importance=TRUE)  
trans.rf

##   
## Call:  
## randomForest(formula = class ~ . - channels - avgprice, data = train, mtry = 10, ntree = 500, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 10  
##   
## OOB estimate of error rate: 7.02%  
## Confusion matrix:  
## 0 1 class.error  
## 0 410 11 0.02612827  
## 1 22 27 0.44897959

dropped.yhat = predict(trans.rf, newdata = test)  
paste("Accuracy of model after dropping least important variables = ", mean(dropped.yhat==test$class))

## [1] "Accuracy of model after dropping least important variables = 0.951063829787234"

Dropping the 2 variables improved the accuracy when compared to tuned algorithm, but it slightly became worse in comparison to the bagging algorithm.

In the end, accuracy is as follows: Bagging > Dropping 2 variables > Tuned (4 predictors)

Predicting with Support Vector Machines (10 points)

a - Fit a support vector classifier to the training data, tuning on the cost parameter for values of 0.1, 1, 5, 10, 50, and 100. What is the best tuned value for cost? How many support vectors are associated with this model? What is the test error rate?

# Get active path, clear environment variables, clear console, load data & packages  
library(rstudioapi)  
library(e1071)

## Warning: package 'e1071' was built under R version 4.0.5

library(randomForest)  
current\_path = rstudioapi::getActiveDocumentContext()$path   
setwd(dirname(current\_path ))  
rm(list=ls())  
cat("\014")

load("transaction.rdata")  
  
#Create training and testing data  
set.seed(25)  
indx = sample(1:nrow(trans), nrow(trans)/2)  
  
train = trans[indx,]  
test = trans[-indx,]  
  
#Fitting a support vector classifier and perform tuning on cost parameter  
set.seed(25)  
linear= tune(svm,class~.,data=train,kernel="linear",ranges=list(cost=c(0.1,1,5,10,50,100)))  
summary(linear)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 1  
##   
## - best performance: 0.08510638   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 0.1 0.08936170 0.04231612  
## 2 1.0 0.08510638 0.02836879  
## 3 5.0 0.09361702 0.02872122  
## 4 10.0 0.08936170 0.03139850  
## 5 50.0 0.09148936 0.03017312  
## 6 100.0 0.09148936 0.03017312

best.linear = linear$best.model  
summary(best.linear)

##   
## Call:  
## best.tune(method = svm, train.x = class ~ ., data = train, ranges = list(cost = c(0.1,   
## 1, 5, 10, 50, 100)), kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
##   
## Number of Support Vectors: 80  
##   
## ( 43 37 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

#Predict and state accuracy  
best.yhat=predict(best.linear,test)  
table(best.yhat,test$class)

##   
## best.yhat 0 1  
## 0 417 22  
## 1 10 21

mean(best.yhat==test$class)

## [1] 0.9319149

b - Redo your analysis in part a. with support vector machines using a radial kernel and tuning with respect to values of gamma=0.5, 1, 2, 3, and 4.

#Redo using kernal set to "radial"  
set.seed(25)  
radial=tune(svm,class~.,data=train,kernel="radial",ranges=list(cost=c(0.1,1,5,10,50,100),gamma=c(0.5,1,2,3,4)))  
summary(radial)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost gamma  
## 1 0.5  
##   
## - best performance: 0.1   
##   
## - Detailed performance results:  
## cost gamma error dispersion  
## 1 0.1 0.5 0.1042553 0.04423395  
## 2 1.0 0.5 0.1000000 0.04261225  
## 3 5.0 0.5 0.1042553 0.04535681  
## 4 10.0 0.5 0.1042553 0.04535681  
## 5 50.0 0.5 0.1042553 0.04535681  
## 6 100.0 0.5 0.1042553 0.04535681  
## 7 0.1 1.0 0.1042553 0.04423395  
## 8 1.0 1.0 0.1042553 0.04423395  
## 9 5.0 1.0 0.1021277 0.04348854  
## 10 10.0 1.0 0.1021277 0.04348854  
## 11 50.0 1.0 0.1021277 0.04348854  
## 12 100.0 1.0 0.1021277 0.04348854  
## 13 0.1 2.0 0.1042553 0.04423395  
## 14 1.0 2.0 0.1042553 0.04423395  
## 15 5.0 2.0 0.1042553 0.04423395  
## 16 10.0 2.0 0.1042553 0.04423395  
## 17 50.0 2.0 0.1042553 0.04423395  
## 18 100.0 2.0 0.1042553 0.04423395  
## 19 0.1 3.0 0.1042553 0.04423395  
## 20 1.0 3.0 0.1042553 0.04423395  
## 21 5.0 3.0 0.1042553 0.04423395  
## 22 10.0 3.0 0.1042553 0.04423395  
## 23 50.0 3.0 0.1042553 0.04423395  
## 24 100.0 3.0 0.1042553 0.04423395  
## 25 0.1 4.0 0.1042553 0.04423395  
## 26 1.0 4.0 0.1042553 0.04423395  
## 27 5.0 4.0 0.1042553 0.04423395  
## 28 10.0 4.0 0.1042553 0.04423395  
## 29 50.0 4.0 0.1042553 0.04423395  
## 30 100.0 4.0 0.1042553 0.04423395

best.radial = radial$best.model  
summary(best.radial)

##   
## Call:  
## best.tune(method = svm, train.x = class ~ ., data = train, ranges = list(cost = c(0.1,   
## 1, 5, 10, 50, 100), gamma = c(0.5, 1, 2, 3, 4)), kernel = "radial")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
##   
## Number of Support Vectors: 257  
##   
## ( 208 49 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

#Predict and state accuracy  
best.yhat = predict(best.radial,test)  
table(best.yhat,test$class)

##   
## best.yhat 0 1  
## 0 426 42  
## 1 1 1

mean(best.yhat==test$class)

## [1] 0.9085106

c - Redo your analysis in part a. with support vector machines using a polynomial kernel and tuning with respect to values of degree = 2, 3, and 4.

#Redo using kernal set to "polynomial"  
set.seed(25)  
poly = tune(svm,class~.,data=train,kernel="polynomial",ranges=list(cost=c(0.1, 1,5,10,50,100),degree=c(2,3,4)))  
summary(poly)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost degree  
## 1 2  
##   
## - best performance: 0.0787234   
##   
## - Detailed performance results:  
## cost degree error dispersion  
## 1 0.1 2 0.08297872 0.03393897  
## 2 1.0 2 0.07872340 0.02663118  
## 3 5.0 2 0.08723404 0.03539000  
## 4 10.0 2 0.08297872 0.03242306  
## 5 50.0 2 0.08297872 0.03242306  
## 6 100.0 2 0.08723404 0.04189803  
## 7 0.1 3 0.08510638 0.03171728  
## 8 1.0 3 0.09574468 0.03364125  
## 9 5.0 3 0.10212766 0.03296156  
## 10 10.0 3 0.10000000 0.03179647  
## 11 50.0 3 0.08723404 0.02737625  
## 12 100.0 3 0.08723404 0.03242306  
## 13 0.1 4 0.09361702 0.04159682  
## 14 1.0 4 0.10000000 0.04815389  
## 15 5.0 4 0.08936170 0.04574331  
## 16 10.0 4 0.08936170 0.03858573  
## 17 50.0 4 0.08936170 0.03588400  
## 18 100.0 4 0.08936170 0.02415516

best.poly = poly$best.model  
summary(best.poly)

##   
## Call:  
## best.tune(method = svm, train.x = class ~ ., data = train, ranges = list(cost = c(0.1,   
## 1, 5, 10, 50, 100), degree = c(2, 3, 4)), kernel = "polynomial")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: polynomial   
## cost: 1   
## degree: 2   
## coef.0: 0   
##   
## Number of Support Vectors: 92  
##   
## ( 53 39 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

#Predict and state accuracy  
best.yhat = predict(best.poly, test)  
table(best.yhat,test$class)

##   
## best.yhat 0 1  
## 0 420 28  
## 1 7 15

mean(best.yhat == test$class)

## [1] 0.9255319

d - After reviewing your results for parts a-c of this question, what is your best model? Accuracy of models: Linear - 0.9319149 Radial - 0.9085106 Polynomial - 0.9255319

As Linear kernel has the highest accuracy among the three, our best model by this parameter is Linear. The order would be Linear > Polynomial > Radial.

e - Plot an ROC curve for your best model using the ROCR package. Is your best model a good model for predicting return abuse? Justify your answer.

library(pROC)

## Warning: package 'pROC' was built under R version 4.0.4

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

library(ROCR)

## Warning: package 'ROCR' was built under R version 4.0.4

#Function for ROC curve  
rocplot=function(pred, truth, ...){  
 predob = prediction(pred, truth)  
 perf = performance(predob, "tpr", "fpr")  
 auc<-performance(predob,measure="auc")@y.values[[1]]  
 title<-paste(...,"AUC = ",round(auc, digits=3))  
 plot(perf,main=title)  
}  
  
#Plot ROC  
svmfit.opt=svm(class~., data=train, kernel="polynomial",cost=5,degree=2,decision.values=T)  
fitted=as.numeric(attributes(predict(svmfit.opt,test,decision.values=T))$decision.values)  
rocplot(fitted,test$class,main="Test Data")

