Assignment-3\_lab2.R

axelj

2022-12-02

library(caret)  
library(matlib)  
library(ggplot2)  
#Task 1  
data = read.csv('communities.csv', header=TRUE) #Header is true because we have correct column names  
set.seed(12345)  
  
df = as.data.frame(data)  
df1= df[, -ncol(df)]  
params=preProcess(df1)  
df1S = predict(params, df1) #Scale data except last column  
dfM = data.matrix(df1S)  
S=(1/ncol(df1S))\*t(dfM)%\*%(dfM) #Covariance matrix  
S = as.data.frame(S)  
  
e = eigen(S)   
#Eigen values  
values = e$values #Eigen values of S  
#Sum of values  
sum = sum(values) #Sum of all eigen values  
percent = 0  
count = 0  
for (i in values) { #For.loop to calculate how many PCs are needed for 95% variance  
 percent = percent + i/sum  
 count = count + 1  
 if (percent > 0.95)  
 break  
}  
count #35 components are needed for variance of at least 95%

## [1] 35

allVariance = values/sum  
PCOneTwo = allVariance[1:2]  
PCOneTwo #Proportion of variance for first two is 25% and 17%

## [1] 0.2501699 0.1693597

#Task 2  
df1S$Fat=c()  
res = princomp(df1S)  
lambda=res$sdev^2  
sprintf("%2.3f",lambda/sum(lambda)\*100)

screeplot(res)

scores = as.data.frame(res$scores[,1:2])  
U = res$loadings  
  
dfg = data.frame("x" = c(scores$Comp.1), "y"=c(scores$Comp.2), "Crimes" = c(df$ViolentCrimesPerPop))  
ggplot(dfg, aes(x=x, y=y, col=Crimes)) +  
 geom\_point() +  
 labs(colour = "Violent Crimes", x="Comp. 1", y="Comp. 2", title="PC Scores")

plot(U[,1], main="Traceplot for PC1", ylab="PC1")

eigen\_vectors = res$loadings[,1] #Get the eigen vectors from princomp  
ordered\_vectors = eigen\_vectors[order(abs(eigen\_vectors),decreasing = TRUE)] #Order the vectors in decreasing order  
top\_five = ordered\_vectors[1:5] #Pick out the top five vectors  
top\_five

## medFamInc medIncome PctKids2Par pctWInvInc PctPopUnderPov   
## -0.1833080 -0.1819830 -0.1755423 -0.1748683 0.1737978

plot(abs(ordered\_vectors),main="Ordered contribution to PC1", ylab="Contribution to PC1") #Plot all eigen vectors in decreasing order

#Task 3  
data = read.csv('communities.csv', header=TRUE) #Header is true because we have correct column names  
  
n = nrow(data)  
set.seed(12345)  
id = sample(1:n, floor(n\*0.5))  
train= data.frame(data[id,]) #Split data  
test = data.frame(data[-id,]) #Split data  
  
params = preProcess(train)  
trainS = predict(params, train) #Scale train  
testS = predict(params, test) #Scale test  
  
m1 = lm(ViolentCrimesPerPop ~ ., data=trainS) #Train model  
summary(m1)

##   
## Call:  
## lm(formula = ViolentCrimesPerPop ~ ., data = trainS)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -2.17350 -0.31062 -0.03815 0.21430 3.07074   
##   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.5534 on 896 degrees of freedom  
## Multiple R-squared: 0.7245, Adjusted R-squared: 0.6938   
## F-statistic: 23.56 on 100 and 896 DF, p-value: < 2.2e-16

p1 = predict(m1, trainS, type="response")  
p2 = predict(m1, testS, type="response")  
  
MSEtest = mean((p2 - testS$ViolentCrimesPerPop)^2) #Mean square error for test data  
MSEtest

## [1] 0.4248011

MSEtrain = mean((p1 - trainS$ViolentCrimesPerPop)^2) #Mean square error for train data  
MSEtrain

## [1] 0.2752071

#Task 4  
TestE=rep(0,0)  
TrainE=rep(0,0)  
k=0  
  
costFunction = function(theta, train, test, p1, p2) {  
 trainHat = train %\*% theta  
 testHat = test %\*% theta  
   
 MSETrain = mean((p1- trainHat)^2)  
 MSETest = mean((p2 - testHat)^2)  
   
 .GlobalEnv$k= .GlobalEnv$k+1  
 .GlobalEnv$TrainE[[k]]=MSETrain  
 .GlobalEnv$TestE[[k]]=MSETest  
   
 return(MSETrain)  
   
}  
  
n = ncol(trainS)-1 #Nr of columns except intercept  
trainX = as.matrix(trainS[,1:n])  
testX = as.matrix(testS[,1:n])  
theta = as.matrix(rep(0,n)) #Theta=0, repeat 0 n times  
  
p1 = as.matrix(trainS[c('ViolentCrimesPerPop')]) #Values of ViolentCrimesPerPop for training data  
p2 = as.matrix(testS[c('ViolentCrimesPerPop')]) #Values of ViolentCrimesPerPop for test data  
  
set.seed(12345)  
res = optim(par=theta, fn=costFunction, train = trainX, test = testX, p1 = p1, p2 = p2, method = "BFGS")  
  
TrainOptMSE = res$value #MSE for training data  
TrainOptMSE

## [1] 0.2752213

TestOptMSE = mean((p2 - (testX%\*%res$par))^2) #MSE for test data  
TestOptMSE

## [1] 0.424518

removedTrainData = TrainE[c(500:length(TrainE))]  
removedTestData = TestE[c(500:length(TestE))]  
  
TestMin = which.min(TestE)  
TrainMin = which.min(TrainE)  
TestMin

## [1] 2182

TrainMin

## [1] 19914

plot(removedTrainData, xlim=c(500, 10000), ylim=c(0.1,1.3), col = "black", main="MSE for training in black and test in red", xlab="Iterations 500 - 10 000",ylab="MSE")  
points(removedTestData,pch=1, col="red")  
points(which.min(TestE),TestE[which.min(TestE)], pch=19, col="blue")