hw3.R

Muhammad

2021-09-08

library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5 v purrr 0.3.4  
## v tibble 3.1.4 v dplyr 1.0.7  
## v tidyr 1.1.3 v stringr 1.4.0  
## v readr 2.0.1 v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

library(caret)

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

library(kernlab)

##   
## Attaching package: 'kernlab'

## The following object is masked from 'package:purrr':  
##   
## cross

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

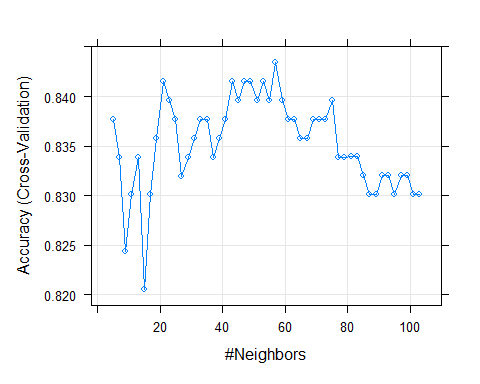
setwd("C:/Users/Muhammad/ISYE/hw3")  
data<-read.table("credit\_card\_data-headers.txt", header = TRUE)  
data<-as.data.frame(data)  
  
#Q3.1.a  
set.seed(100)  
#splitting data for training/validation and testing. 80% training data set  
indxTrain <- createDataPartition(y = data$R1,p = 0.8,list = FALSE)  
training <- data[indxTrain,]  
testing <- data[-indxTrain,]  
  
#Pre-processing data i.e scaling for Knn  
trainX <- training[,names(training) != "R1"]  
Proccessed\_Values <- preProcess(x = trainX,method = c("center", "scale"))  
  
#I am choosing k-fold value of 5 beacause our dataset is too small  
  
control <- trainControl(method="cv",number=5)  
#training data set  
knnFit <- train(as.factor(R1) ~ ., data = training, method = "knn", trControl = control, preProcess = c("center","scale"), tuneLength = 50)  
  
#finding the value of kappa where accuracy maximizes;  
index\_where\_max<-which.max(knnFit$results$Accuracy)  
knnFit$results$Accuracy[index\_where\_max]

## [1] 0.8434249

k=knnFit$results$k[index\_where\_max]  
k

## [1] 57

plot(knnFit)



cat("For final model Accuracy =", knnFit$results$Accuracy[index\_where\_max], "and k =",k)

## For final model Accuracy = 0.8434249 and k = 57

#For final model, function chose K=23 with highest accuracy of 83.97%  
#Testing our model on test dataset  
knnPredict <- predict(knnFit, newdata = testing)  
con<-confusionMatrix(knnPredict, as.factor(testing$R1))  
con

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 63 16  
## 1 6 45  
##   
## Accuracy : 0.8308   
## 95% CI : (0.7551, 0.8908)  
## No Information Rate : 0.5308   
## P-Value [Acc > NIR] : 6.438e-13   
##   
## Kappa : 0.657   
##   
## Mcnemar's Test P-Value : 0.05501   
##   
## Sensitivity : 0.9130   
## Specificity : 0.7377   
## Pos Pred Value : 0.7975   
## Neg Pred Value : 0.8824   
## Prevalence : 0.5308   
## Detection Rate : 0.4846   
## Detection Prevalence : 0.6077   
## Balanced Accuracy : 0.8254   
##   
## 'Positive' Class : 0   
##

overall\_accuracy<-con$overall["Accuracy"]  
cat("Accuracy on test data set =", overall\_accuracy\*100)

## Accuracy on test data set = 83.07692

#Q3.1.b  
  
#Sampling and splitting data for training and testing  
  
index\_sample<- sample(1:nrow(data), as.integer(0.75\*nrow(data)))  
train\_data <- data[index\_sample,]  
  
#Assigning validation and test data  
valid\_test\_data <- data[-index\_sample,]  
index\_sample2<- sample(1:nrow(valid\_test\_data), as.integer(0.50\*nrow(valid\_test\_data)))  
validation\_data<-valid\_test\_data[index\_sample2,]  
testing\_data<- valid\_test\_data[-index\_sample2,]  
  
#c\_value <- seq(1, 100, by=5)  
c\_value= seq(.1,1,by=0.1)  
accuracy1= rep(1,10)  
#looping for accuracy for svm model for different c value with increment of 0.1, tried 1 to 100 by 5 increment already  
for (i in 1:10) {  
   
 #training model on training data  
 svm\_model = ksvm(as.matrix(train\_data[,1:10]),as.factor(train\_data[,11]), type = "C-svc",kernel = "vanilladot",C = c\_value[i],scaled=TRUE)   
 #testing it on validation data  
 prediction1 = predict(svm\_model,validation\_data[,1:10])  
 accuracy1[i] = (sum(prediction1 == validation\_data$R1) / nrow(validation\_data))\*100  
}

## Setting default kernel parameters   
## Setting default kernel parameters   
## Setting default kernel parameters   
## Setting default kernel parameters   
## Setting default kernel parameters   
## Setting default kernel parameters   
## Setting default kernel parameters   
## Setting default kernel parameters   
## Setting default kernel parameters   
## Setting default kernel parameters

#which model maximizes   
max\_ind<-which.max(accuracy1)  
max(accuracy1)

## [1] 86.58537

c\_max<-c\_value[max\_ind]  
cat("C value which maximizes accuracy =",c\_max,"\n")

## C value which maximizes accuracy = 0.1

#retraining model on training dataset with parameter c=0.1,   
#(in this case it really doesn't matter because we got same accuracy with the c values in last sequence I tried.  
#however just to show that I know generally we have to rerun our model on training set with the parameter we found for   
#maximizing accuracy in validation data set)  
  
svm\_model\_test = ksvm(as.matrix(train\_data[,1:10]),as.factor(train\_data[,11]), type = "C-svc",kernel = "vanilladot",C =0.1,scaled=TRUE)

## Setting default kernel parameters

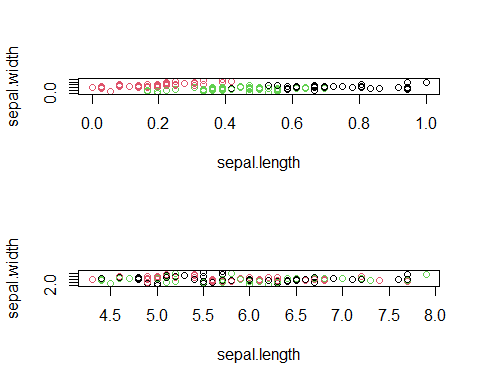
Accuracy\_on\_test = sum(predict(svm\_model\_test,testing\_data[,1:10])==testing\_data[,11])/nrow(testing\_data)  
cat("Accuracy on test data set =", Accuracy\_on\_test)

## Accuracy on test data set = 0.8292683

#Q4.2  
library(datasets)  
data(iris)  
names(iris) <- tolower(names(iris))  
iris\_category <- unique(iris[,5])  
  
#Scaling all data set  
preproc2 <- preProcess(iris[,1:4], method=c("range"))  
  
normalized\_data <- predict(preproc2, iris[,1:4])  
  
model\_kmean<-kmeans(normalized\_data[,1:4], 3, nstart = 20)  
model\_kmean

## K-means clustering with 3 clusters of sizes 39, 50, 61  
##   
## Cluster means:  
## sepal.length sepal.width petal.length petal.width  
## 1 0.7072650 0.4508547 0.79704476 0.82478632  
## 2 0.1961111 0.5950000 0.07830508 0.06083333  
## 3 0.4412568 0.3073770 0.57571548 0.54918033  
##   
## Clustering vector:  
## [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
## [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 1 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
## [75] 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 1 1 1 3 1 1 1 1  
## [112] 1 1 3 1 1 1 1 1 3 1 3 1 3 1 1 3 3 1 1 1 1 1 3 3 1 1 1 3 1 1 1 3 1 1 1 3 1  
## [149] 1 3  
##   
## Within cluster sum of squares by cluster:  
## [1] 2.073324 1.829062 3.079830  
## (between\_SS / total\_SS = 83.0 %)  
##   
## Available components:  
##   
## [1] "cluster" "centers" "totss" "withinss" "tot.withinss"  
## [6] "betweenss" "size" "iter" "ifault"

#There is not really a way to tell accuracy since clustering is unsupervised learning  
#We could look at the compactness which is 76.7% or how similar to members within a group are  
#We can look at purity but I don't think that is expected here  
#More over anything over 3 dimensions would be really hard to determine manually  
#We could loop over multiple values of K but since we already know that there are 3 classes it's not necessary,  
#however we could gain some extra information but we only have 150 observation so that isn't a good idea since  
#it will over-fit and accuracy will probably go down   
  
# We could visualize for different combination of attributes and their graphs and see how two attributes  
# are placed in a category depending on their values, but as we can see after just 3 dimensions we have  
#difficulty visualizing, hence usually we could just look at individual cluster means for group and try  
#understand intuitively or we could look at visualize 2 or 3 features of interest at a time to study further.  
  
#i.e:  
par(mfrow=c(2,1))  
plot(normalized\_data[c(1,2)], col=model\_kmean$cluster)  
plot(iris[c(1,2)], col=iris\_category)



#I think for something like