FML\_Assignment\_2

Chakri

2023-09-28

library(class)  
library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(e1071)

#importing the data from the directory

uni\_bank.df <- read.csv("H:\\Kent Sem-1\\FML\\FML\_class\\UniversalBank.csv")  
#uni\_bank.df

#idnetifying the rows and columns and head and tail of the data

dim(uni\_bank.df)

## [1] 5000 14

t(t(names(uni\_bank.df)))

## [,1]   
## [1,] "ID"   
## [2,] "Age"   
## [3,] "Experience"   
## [4,] "Income"   
## [5,] "ZIP.Code"   
## [6,] "Family"   
## [7,] "CCAvg"   
## [8,] "Education"   
## [9,] "Mortgage"   
## [10,] "Personal.Loan"   
## [11,] "Securities.Account"  
## [12,] "CD.Account"   
## [13,] "Online"   
## [14,] "CreditCard"

head(uni\_bank.df)

## ID Age Experience Income ZIP.Code Family CCAvg Education Mortgage  
## 1 1 25 1 49 91107 4 1.6 1 0  
## 2 2 45 19 34 90089 3 1.5 1 0  
## 3 3 39 15 11 94720 1 1.0 1 0  
## 4 4 35 9 100 94112 1 2.7 2 0  
## 5 5 35 8 45 91330 4 1.0 2 0  
## 6 6 37 13 29 92121 4 0.4 2 155  
## Personal.Loan Securities.Account CD.Account Online CreditCard  
## 1 0 1 0 0 0  
## 2 0 1 0 0 0  
## 3 0 0 0 0 0  
## 4 0 0 0 0 0  
## 5 0 0 0 0 1  
## 6 0 0 0 1 0

tail(uni\_bank.df)

## ID Age Experience Income ZIP.Code Family CCAvg Education Mortgage  
## 4995 4995 64 40 75 94588 3 2.0 3 0  
## 4996 4996 29 3 40 92697 1 1.9 3 0  
## 4997 4997 30 4 15 92037 4 0.4 1 85  
## 4998 4998 63 39 24 93023 2 0.3 3 0  
## 4999 4999 65 40 49 90034 3 0.5 2 0  
## 5000 5000 28 4 83 92612 3 0.8 1 0  
## Personal.Loan Securities.Account CD.Account Online CreditCard  
## 4995 0 0 0 1 0  
## 4996 0 0 0 1 0  
## 4997 0 0 0 1 0  
## 4998 0 0 0 0 0  
## 4999 0 0 0 1 0  
## 5000 0 0 0 1 1

#Drop unnecessary rows like ID and Zip

uni\_bank.df <- uni\_bank.df[,-c(1,5)]

#Categorical varables will be converted as factor(ie,. Education) as mentioned in the question

uni\_bank.df$Education <- as.factor(uni\_bank.df$Education)

#now converting the Education into dummy variables

groups <- dummyVars(~., data = uni\_bank.df)  
#the new data frame is named as modified universal bank data  
uni\_bank.m.df <- as.data.frame(predict(groups,uni\_bank.df))

#splitting the data for training data(60%) and validation data(remaining 40%)

#it is important to ensure that that we get the same sample if we return the code multiple times  
set.seed(7)  
#dividing the data into 60% training data and remaining data to validation  
train.data <- sample(row.names(uni\_bank.m.df),0.6\*dim(uni\_bank.m.df)[1])  
valid.data <- setdiff(row.names(uni\_bank.m.df),train.data)  
  
#apply the model  
#all the variables are taking in after the seperated comma  
train.df <- uni\_bank.m.df[train.data,]  
valid.df <- uni\_bank.m.df[valid.data,]  
t(t(names(train.df)))

## [,1]   
## [1,] "Age"   
## [2,] "Experience"   
## [3,] "Income"   
## [4,] "Family"   
## [5,] "CCAvg"   
## [6,] "Education.1"   
## [7,] "Education.2"   
## [8,] "Education.3"   
## [9,] "Mortgage"   
## [10,] "Personal.Loan"   
## [11,] "Securities.Account"  
## [12,] "CD.Account"   
## [13,] "Online"   
## [14,] "CreditCard"

#normalizing the data

train.norm.df <- train.df[,-10]  
valid.norm.df <- valid.df[,-10]#personal income is the 10th variable  
  
norm.values <- preProcess(train.df[,-10], method = c("center","scale"))  
  
#mean and standard deviation is coming from the train data  
  
train.norm.df <- predict(norm.values,train.df[,-10])  
valid.norm.df <- predict(norm.values,valid.df[,-10])

#1. Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education\_1 = 0, Education\_2 =1, Education\_3 = 0, Mortgage = 0, Securities Account = 0, CD Account = 0, Online = 1, and Credit Card = 1. Perform a k-NN classification with all predictors except ID and ZIP code using k = 1. Remember to transform categorical predictors with more than two categories into dummy variables first. Specify the success class as 1 (loan acceptance), and use the default cutoff value of 0.5. How would this customer be classified?

#creating the new customer and with knn-prediction

new\_customer <- data.frame(Age = 40,   
Experience = 10,   
Income = 84,   
Family = 2,   
CCAvg = 2,   
Education.1 = 0,   
Education.2 = 1,   
Education.3 = 0,   
Mortgage = 0,   
Securities.Account = 0,   
CD.Account = 0,   
Online = 1,  
CreditCard = 1  
)

#now normalize the data for the new customer

new.cust.norm <- new\_customer  
new.cust.norm <- predict(norm.values,new.cust.norm)

knn.pred1 <- class::knn(train= train.norm.df,   
 test = new.cust.norm,   
 cl= train.df$Personal.Loan, k=1 )  
knn.pred1

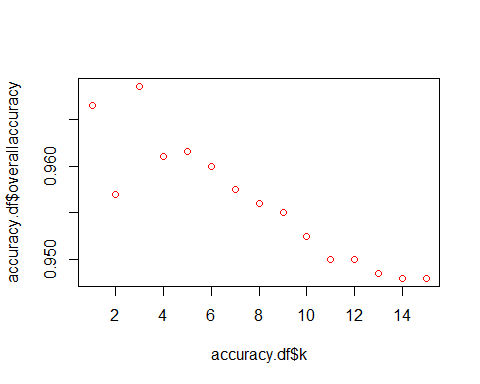
## [1] 0  
## Levels: 0 1

#2.What is a choice of k that balances between overfitting and ignoring the predictor information?

accuracy.df <- data.frame(k= seq(1,15,1), overallaccuracy= rep(0,15))  
for(i in 1:15){  
 knn.pred <- class::knn(train=train.norm.df, test = valid.norm.df, cl= train.df$Personal.Loan, k=i)  
 accuracy.df[i,2] <- confusionMatrix(knn.pred, as.factor(valid.df$Personal.Loan), positive= "1")$overall[1]  
}  
  
which(accuracy.df[,2]== max(accuracy.df[,2]))

## [1] 3

plot(accuracy.df$k,accuracy.df$overallaccuracy,col="red")



#3. Show the confusion matrix for the validation data that results from using the best k?

knn.pred2 <- class::knn(train = train.norm.df, test = valid.norm.df, cl= train.df$Personal.Loan, k=3)  
con.mat <-confusionMatrix(table(knn.pred2,as.factor(valid.df$Personal.Loan)))  
con.mat

## Confusion Matrix and Statistics  
##   
##   
## knn.pred2 0 1  
## 0 1814 53  
## 1 10 123  
##   
## Accuracy : 0.9685   
## 95% CI : (0.9599, 0.9757)  
## No Information Rate : 0.912   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.7794   
##   
## Mcnemar's Test P-Value : 1.213e-07   
##   
## Sensitivity : 0.9945   
## Specificity : 0.6989   
## Pos Pred Value : 0.9716   
## Neg Pred Value : 0.9248   
## Prevalence : 0.9120   
## Detection Rate : 0.9070   
## Detection Prevalence : 0.9335   
## Balanced Accuracy : 0.8467   
##   
## 'Positive' Class : 0   
##

#4. Consider the following customer: Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education\_1 = 0, Education\_2 = 1, Education\_3 = 0, Mortgage = 0, Securities Account = 0, CD Account = 0, Online = 1 and Credit Card = 1. Classify the customer using the best k.

new\_customer1 <- data.frame(Age = 40,   
 Experience = 10,   
 Income = 84,   
 Family = 2,   
 CCAvg = 2,   
 Education.1 = 0,   
 Education.2 = 1,   
 Education.3 = 0,   
 Mortgage = 0,   
 Securities.Account = 0,   
 CD.Account = 0,   
 Online = 1,  
 CreditCard = 1)  
  
new.cust.norm1 <- new\_customer1  
new.cust.norm1 <- predict(norm.values,new.cust.norm1)  
  
knn.pred3 <- class::knn(train = train.norm.df, test = new.cust.norm1, cl= train.df$Personal.Loan,k= 3)  
knn.pred3

## [1] 0  
## Levels: 0 1

#5.Repartition the data, this time into training, validation, and test sets (50% : 30% : 20%). Apply the k-NN method with the k chosen above. Compare the confusion matrix of the test set with that of the training and validation sets. Comment on the differences and their reason.

#it is important to ensure that that we get the same sample if we return the code multiple times #splitting the data to training(50%), validation(30%) and testing(remaining 20%)

set.seed(26)  
train.data1 <- sample(row.names(uni\_bank.m.df),0.5\*dim(uni\_bank.m.df)[1])  
train.df1 <- uni\_bank.m.df[as.numeric(train.data1),]  
  
valid.data0 <- setdiff(row.names(uni\_bank.m.df),train.data1)  
valid.df0 <- uni\_bank.m.df[as.numeric(valid.data0),]  
valid.data1 <- sample(row.names(valid.df0),0.6\*dim(valid.df0)[1])  
test.data1 <- setdiff(row.names(valid.df0),valid.data1)  
  
valid.df1 <- uni\_bank.m.df[valid.data1,]  
test.df1 <- uni\_bank.m.df[test.data1,]  
  
t(t(names(train.df1)))

## [,1]   
## [1,] "Age"   
## [2,] "Experience"   
## [3,] "Income"   
## [4,] "Family"   
## [5,] "CCAvg"   
## [6,] "Education.1"   
## [7,] "Education.2"   
## [8,] "Education.3"   
## [9,] "Mortgage"   
## [10,] "Personal.Loan"   
## [11,] "Securities.Account"  
## [12,] "CD.Account"   
## [13,] "Online"   
## [14,] "CreditCard"

#normalizing the data

train.norm.df1 <- train.df1[,-10]  
valid.norm.df1 <- valid.df1[,-10]  
test.norm.df1 <- test.df1[,-10]  
  
norm.values1 <- preProcess(train.df1[,-10], method = c("center","scale"))  
  
train.norm.df1 <- predict(norm.values1,train.df1[,-10])  
valid.norm.df1 <- predict(norm.values,valid.df1[,-10])  
test.norm.df1 <- predict(norm.values1,test.df1[,-10])

#confusion matrix for the training data

knn.pred4 <- class::knn(train = train.norm.df1,  
 test = train.norm.df1,  
 cl= train.df1$Personal.Loan, k= 3)  
knn.pred4

## [1] 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0  
## [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0  
## [75] 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0  
## [112] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [149] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0  
## [186] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [223] 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0  
## [260] 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0  
## [297] 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [334] 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [371] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0  
## [408] 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0  
## [445] 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0  
## [482] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0  
## [519] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [556] 0 1 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [667] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0  
## [704] 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [741] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0  
## [778] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0  
## [815] 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [852] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [889] 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0  
## [926] 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0  
## [963] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0  
## [1000] 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0  
## [1037] 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0  
## [1074] 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1111] 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0  
## [1148] 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0  
## [1185] 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0  
## [1222] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0  
## [1259] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
## [1296] 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0  
## [1333] 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0  
## [1370] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1407] 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0  
## [1444] 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0  
## [1481] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0  
## [1518] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1555] 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0  
## [1592] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1629] 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0  
## [1666] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0  
## [1703] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0  
## [1740] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0  
## [1777] 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0  
## [1814] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0  
## [1851] 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1888] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1925] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1962] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1999] 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0  
## [2036] 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1  
## [2073] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0  
## [2110] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0  
## [2147] 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [2184] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0  
## [2221] 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0  
## [2258] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0  
## [2295] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1  
## [2332] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0  
## [2369] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [2406] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0  
## [2443] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1  
## [2480] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## Levels: 0 1

con.mat4 <- confusionMatrix(knn.pred4, as.factor(train.df1$Personal.Loan))  
con.mat4

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 2254 64  
## 1 3 179  
##   
## Accuracy : 0.9732   
## 95% CI : (0.9661, 0.9792)  
## No Information Rate : 0.9028   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.828   
##   
## Mcnemar's Test P-Value : 2.299e-13   
##   
## Sensitivity : 0.9987   
## Specificity : 0.7366   
## Pos Pred Value : 0.9724   
## Neg Pred Value : 0.9835   
## Prevalence : 0.9028   
## Detection Rate : 0.9016   
## Detection Prevalence : 0.9272   
## Balanced Accuracy : 0.8676   
##   
## 'Positive' Class : 0   
##

From the training data shows the model ability to learn from the training data with highest accuracy and sensitivity results in overfitting.

#confusion matrix for the validation data

knn.pred5 <- class::knn(train = train.norm.df1,  
 test = valid.norm.df1,  
 cl= train.df1$Personal.Loan, k= 3)  
knn.pred5

## [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0  
## [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0  
## [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0  
## [112] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0  
## [149] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0  
## [186] 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0  
## [223] 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [260] 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [297] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [371] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [408] 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [445] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [482] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [519] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0  
## [556] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0  
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0  
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0  
## [667] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0  
## [704] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [741] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0  
## [778] 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0  
## [815] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0  
## [852] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0  
## [889] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1  
## [926] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [963] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1000] 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0  
## [1037] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0  
## [1074] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1111] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1148] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1185] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0  
## [1222] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1259] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0  
## [1296] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1333] 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0  
## [1370] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0  
## [1407] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1444] 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1481] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## Levels: 0 1

con.mat5 <- confusionMatrix(knn.pred5, as.factor(valid.df1$Personal.Loan))  
con.mat5

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 1354 57  
## 1 3 86  
##   
## Accuracy : 0.96   
## 95% CI : (0.9488, 0.9693)  
## No Information Rate : 0.9047   
## P-Value [Acc > NIR] : 2.751e-16   
##   
## Kappa : 0.721   
##   
## Mcnemar's Test P-Value : 7.795e-12   
##   
## Sensitivity : 0.9978   
## Specificity : 0.6014   
## Pos Pred Value : 0.9596   
## Neg Pred Value : 0.9663   
## Prevalence : 0.9047   
## Detection Rate : 0.9027   
## Detection Prevalence : 0.9407   
## Balanced Accuracy : 0.7996   
##   
## 'Positive' Class : 0   
##

From the validation data where the model performs with other than training data/unseen data maintaining with high accuracy but slightly reduced specificity.

#confusion matrix for the testing data

knn.pred6 <- class::knn(train = train.norm.df1,  
 test = test.norm.df1,  
 cl= train.df1$Personal.Loan, k= 3)  
knn.pred6

## [1] 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0  
## [38] 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0  
## [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [112] 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0  
## [149] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [186] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [223] 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0  
## [260] 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0  
## [297] 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0  
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0  
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1  
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0  
## [482] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0  
## [519] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0  
## [556] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [593] 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0  
## [630] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [667] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [704] 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [741] 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [778] 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0  
## [815] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0  
## [852] 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [889] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0  
## [926] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [963] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
## [1000] 0  
## Levels: 0 1

con.mat6 <- confusionMatrix(knn.pred6, as.factor(test.df1$Personal.Loan))  
con.mat6

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 901 27  
## 1 5 67  
##   
## Accuracy : 0.968   
## 95% CI : (0.9551, 0.978)  
## No Information Rate : 0.906   
## P-Value [Acc > NIR] : 1.465e-14   
##   
## Kappa : 0.7901   
##   
## Mcnemar's Test P-Value : 0.0002054   
##   
## Sensitivity : 0.9945   
## Specificity : 0.7128   
## Pos Pred Value : 0.9709   
## Neg Pred Value : 0.9306   
## Prevalence : 0.9060   
## Detection Rate : 0.9010   
## Detection Prevalence : 0.9280   
## Balanced Accuracy : 0.8536   
##   
## 'Positive' Class : 0   
##

From the test data where it confirms the real-world scenarios with high accuracy and sensitivity eventhough the specificity slightly reduced.

Finally, the above model demonstrates good balance between overfitting and ignoring predictor information. it fits the training data and the model can take the knowledge from the training data and applying it to unseen data, and also the model is not only memorizing the data but instead learning the patterns that can be applied in real world scenarios