Assignment 2

R Markdown

##Read the Csv and create a data frame

```
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
library(e1071)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
Bankdata <- read.csv("UniversalBank.csv")
```

##Exploratory data analysis–checking structure summary data distribution –plot

```
str(Bankdata) # To check the structure of the data
```

```
## 'data.frame':    5000 obs. of  14 variables:
##  $ ID                : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Age               : int  25 45 39 35 35 37 53 50 35 34 ...
##  $ Experience        : int  1 19 15 9 8 13 27 24 10 9 ...
##  $ Income            : int  49 34 11 100 45 29 72 22 81 180 ...
##  $ ZIP.Code          : int  91107 90089 94720 94112 91330 92121 91711 93943 90089 93023 ...
##  $ Family            : int  4 3 1 1 4 4 2 1 3 1 ...
##  $ CCAvg             : num  1.6 1.5 1 2.7 1 0.4 1.5 0.3 0.6 8.9 ...
##  $ Education         : int  1 1 1 2 2 2 2 3 2 3 ...
##  $ Mortgage          : int  0 0 0 0 0 155 0 0 104 0 ...
##  $ Personal.Loan     : int  0 0 0 0 0 0 0 0 0 1 ...
##  $ Securities.Account: int  1 1 0 0 0 0 0 0 0 0 ...
##  $ CD.Account        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Online            : int  0 0 0 0 0 1 1 0 1 0 ...
##  $ CreditCard        : int  0 0 0 0 1 0 0 1 0 0 ...
```

```
colSums(is.na(Bankdata)) # To check Missing values in the dataset
```

```
##                  ID                 Age          Experience              Income
##                   0                   0                   0                   0
##            ZIP.Code              Family               CCAvg           Education
##                   0                   0                   0                   0
##            Mortgage      Personal.Loan Securities.Account         CD.Account
##                   0                   0                   0                   0
##              Online          CreditCard
##                   0                   0
```

```
summary(Bankdata)
```

```
##        ID              Age            Experience       Income          ZIP.Code
##  Min.   :   1   Min.   :23.00   Min.   :-3.0   Min.   :  8.00   Min.   : 9307
##  1st Qu.:1251   1st Qu.:35.00   1st Qu.:10.0   1st Qu.: 39.00   1st Qu.:91911
##  Median :2500   Median :45.00   Median :20.0   Median : 64.00   Median :93437
##  Mean   :2500   Mean   :45.34   Mean   :20.1   Mean   : 73.77   Mean   :93152
##  3rd Qu.:3750   3rd Qu.:55.00   3rd Qu.:30.0   3rd Qu.: 98.00   3rd Qu.:94608
##  Max.   :5000   Max.   :67.00   Max.   :43.0   Max.   :224.00   Max.   :96651
##      Family           CCAvg          Education        Mortgage
##  Min.   :1.000   Min.   : 0.000   Min.   :1.000   Min.   :  0.0
##  1st Qu.:1.000   1st Qu.: 0.700   1st Qu.:1.000   1st Qu.:  0.0
##  Median :2.000   Median : 1.500   Median :2.000   Median :  0.0
##  Mean   :2.396   Mean   : 1.938   Mean   :1.881   Mean   : 56.5
##  3rd Qu.:3.000   3rd Qu.: 2.500   3rd Qu.:3.000   3rd Qu.:101.0
##  Max.   :4.000   Max.   :10.000   Max.   :3.000   Max.   :635.0
##  Personal.Loan   Securities.Account   CD.Account         Online
##  Min.   :0.000   Min.   :0.0000     Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.000   1st Qu.:0.0000     1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.000   Median :0.0000     Median :0.0000   Median :1.0000
##  Mean   :0.096   Mean   :0.1044     Mean   :0.0604   Mean   :0.5968
##  3rd Qu.:0.000   3rd Qu.:0.0000     3rd Qu.:0.0000   3rd Qu.:1.0000
##  Max.   :1.000   Max.   :1.0000     Max.   :1.0000   Max.   :1.0000
##    CreditCard
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.294
##  3rd Qu.:1.000
##  Max.   :1.000
```

#Transforming variables and introducing dummy variables. Sample usage of dummy to check the implementation

```
library(dummies)
```

```
## dummies-1.5.6 provided by Decision Patterns
```

```
library(dplyr)
Bankdata$Education = as.factor(Bankdata$Education) # splitting the education into 3 parts

dummybank <- dummy.data.frame(select(Bankdata,-c(ZIP.Code,ID))) # removing the ZIP.Code and ID from the
```

```
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts = FALSE):
## non-list contrasts argument ignored
```

```
dummybank$Personal.Loan <- as.factor(dummybank$Personal.Loan) ## as.factor is used when you want to con
```

##Split the data into training and validation

```
set.seed(123)

Train_index <- createDataPartition(dummybank$Personal.Loan, p=.6,
                                   list = FALSE,
                                   times = 1)

Training.df=dummybank[Train_index,] #Assigning the Train_index to the training data frame

Validation.df=dummybank[-Train_index,]  #Assigning the rest(Validation_index) to the validation data fr

Conditions = data.frame(Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education1 = 0, E


Normal <- preProcess(Training.df,method=c("center","scale")) #normalizing the data
Training.df <- predict(Normal,Training.df)    # prediction using normalized data into training model
Validation.df <-predict(Normal,Validation.df) ## predicting normalized data with validation data frame
Conditions = predict(Normal,Conditions)  # predicting normalized data with conditions which are given
```

```
library(caret)
library(class)
library(ISLR)
K1 <- knn(train = Training.df[,-c(10)],test = Conditions, cl = Training.df[,c(10)],k=1, prob=TRUE) # ap

Knnattributes <- attributes(K1)  #determining the attributes
Knnattributes[1]
```

```
## $levels
## [1] "0" "1"
```

```
Knnattributes[3]
```

```
## $prob
## [1] 1
```

Q2) What is a choice of k that balances between overfitting and ignoring the predictor information?

```
accuracy.df <- data.frame(k = seq(1,5,1), accuracy = rep(0,5)) # data frame accuracy to check the k val
for(i in 1:5)  #i in 1:5, is a recurssive login from 1 to 5.
{
K2 <- knn(train = Training.df[,-10],test = Validation.df[,-10], cl = Training.df[,10],
k=i, prob=TRUE)
accuracy.df[i, 2] <- confusionMatrix(K2, Validation.df[,10])$overall[1] # for loop to generate accuracy
}
accuracy.df # k=1 has the highest accuracy
```

```
##   k accuracy
## 1 1   0.9645
## 2 2   0.9605
## 3 3   0.9635
## 4 4   0.9635
## 5 5   0.9595
```

Q3) Show the confusion matrix for the validation data that results from using the best k.

```
K3<- knn(train = Training.df[,-10],test = Validation.df[,-10], cl = Training.df[,10],
k=1, prob=TRUE) # using validation data we are showing confusion matrix with 96 % accuracy
confusionMatrix(K3, Validation.df[,10])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1793   56
##          1   15  136
##
##                Accuracy : 0.9645
##                  95% CI : (0.9554, 0.9722)
##     No Information Rate : 0.904
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7739
##
##  Mcnemar's Test P-Value : 2.063e-06
##
##             Sensitivity : 0.9917
##             Specificity : 0.7083
##          Pos Pred Value : 0.9697
##          Neg Pred Value : 0.9007
##              Prevalence : 0.9040
##          Detection Rate : 0.8965
##    Detection Prevalence : 0.9245
##       Balanced Accuracy : 0.8500
##
##        'Positive' Class : 0
##
```

Q4) 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.

```
Customer123 =data.frame(Age = (40), Experience = (10), Income = (84), Family
= (2), CCAvg = (2), Education1 = (0), Education2 = (1), Education3 = (0),
Mortgage = (0), Securities.Account = (0), CD.Account = (0), Online = (1),
CreditCard = (1))
K4 <- knn(train = Training.df[,-10],test = Customer123, cl = Training.df[,10], k=3,
prob=TRUE) # best value of K is 3

Knnattributes <- attributes(K4)
Knnattributes[3]
```

```
## $prob
## [1] 0.6666667
```

```
K4
```

```
## [1] 1
## attr(,"prob")
## [1] 0.6666667
## Levels: 0 1
```

Q5) 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.

```
set.seed(1123)
Train_index1 <- sample(rownames(dummybank), 0.5*dim(dummybank)[1]) ## 50% of the data partition
set.seed(123)
valid.index <- sample(setdiff(rownames(dummybank),Train_index1),0.3*dim(dummybank)[1]) #30 % validation
test.index = setdiff(rownames(dummybank), union(Train_index1, valid.index)) #remaining 20 % in test dat


# loading index values to respective data frame.
Training.df1 <- dummybank[Train_index1, ]
Validation.df1 <- dummybank[valid.index, ]
test.df <- dummybank[test.index, ]

Normalized <- preProcess(Training.df1, method=c("center", "scale"))
Training.df1 <- predict(Normalized, Training.df1) #predicting train data with nomalized data
Validation.df1 <- predict(Normalized, Validation.df1) #predicting Valid data with nomalized data
test.df <- predict(Normalized, test.df) # predicting Test data with nomalized data
```

```
#applying Knn Algorithm for test, train, valid sets
TestingKnnAlg <- knn(train = Training.df1[,-c(10)],test = test.df[,-c(10)], cl =
Training.df1[,10], k=6, prob=TRUE)

ValidatingKnnAlg <- knn(train = Training.df1[,-c(10)],test = Validation.df1[,-c(10)], cl = Training.df1

TrainKnnAlg <- knn(train = Training.df1[,-c(10)],test = Training.df1[,-c(10)], cl = Training.df1[,10], k
```

confusion matrix for test, train,valid which has knn algorithm applied to it

```
# Matrix for predicted values and actual values for Testing
confusionMatrix(TestingKnnAlg, test.df[,10])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 909  38
##          1   4  49
##
##                Accuracy : 0.958
##                  95% CI : (0.9436, 0.9696)
##     No Information Rate : 0.913
##     P-Value [Acc > NIR] : 2.109e-08
##
##                   Kappa : 0.6788
##
##  Mcnemar's Test P-Value : 3.543e-07
##
##             Sensitivity : 0.9956
##             Specificity : 0.5632
##          Pos Pred Value : 0.9599
##          Neg Pred Value : 0.9245
##              Prevalence : 0.9130
##          Detection Rate : 0.9090
##    Detection Prevalence : 0.9470
##       Balanced Accuracy : 0.7794
##
##        'Positive' Class : 0
##
```

```
# Matrix for predicted values and actual values for validation
confusionMatrix(ValidatingKnnAlg, Validation.df1[,10])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1344   67
##          1    6   83
##
##                Accuracy : 0.9513
##                  95% CI : (0.9392, 0.9617)
##     No Information Rate : 0.9
##     P-Value [Acc > NIR] : 2.502e-13
##
##                   Kappa : 0.67
##
```

```
##  Mcnemar's Test P-Value : 2.180e-12
##
##             Sensitivity : 0.9956
##             Specificity : 0.5533
##          Pos Pred Value : 0.9525
##          Neg Pred Value : 0.9326
##              Prevalence : 0.9000
##          Detection Rate : 0.8960
##    Detection Prevalence : 0.9407
##       Balanced Accuracy : 0.7744
##
##        'Positive' Class : 0
##
```

```
# Matrix for predicted values and actual values for Training
confusionMatrix(TrainKnnAlg, Training.df1[,10])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2251   76
##          1    6  167
##
##                Accuracy : 0.9672
##                  95% CI : (0.9594, 0.9738)
##     No Information Rate : 0.9028
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7855
##
##  Mcnemar's Test P-Value : 2.541e-14
##
##             Sensitivity : 0.9973
##             Specificity : 0.6872
##          Pos Pred Value : 0.9673
##          Neg Pred Value : 0.9653
##              Prevalence : 0.9028
##          Detection Rate : 0.9004
##    Detection Prevalence : 0.9308
##       Balanced Accuracy : 0.8423
##
##        'Positive' Class : 0
##
```