Assignment

Ananth Kumar

29/09/2021

install.packages(“ISLR”) install.packages(“caret”) install.packages(“tidyverse”) install.packages(“dplyr”) install.packages(“thePackage”) library(thePackage) library(dplyr) install.packages(“class”) library(class) library(ISLR) library(caret) install.packages(“FNN”) library(“dummies”) install.packages(“ROCR”) library(tidyverse)

tinytex::reinstall_tinytex()

library(“dplyr”) library(“tidyr”)

.libPaths(“C:\Users\Ananth\OneDrive\Desktop\MSBA Kent\Fall 2021\Fundamentals of Machine Learning\Assignment\Ass 2”) ## R Markdown

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. 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?

```
setwd("C:\\Users\\Ananth\\OneDrive\\Desktop\\MSBA Kent\\Fall 
2021\\Fundamentals of Machine Learning\\Assignment\\Ass 2")
rm(list=ls())
Bank <- read.csv("UniversalBank.csv")
head(Bank)
```

```
##   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
```

```
#dummy variables

Bank$Education = as.factor(Bank$Education) # as.factor is used when you want
to convert the data type of a variable to a factor/categorical variable.
library(dummies)

## Warning: package 'dummies' was built under R version 4.0.3

## dummies-1.5.6 provided by Decision Patterns

library(dplyr)

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

##
## 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

dummy_cat <-dummy.data.frame(select(Bank,-c(ZIP.Code,ID)))  # removing the
ZIP.Code and ID from the dummy data

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

dummy_cat$Personal.Loan = as.factor(dummy_cat$Personal.Loan) # Converting
personal loan to categorical
dummy_cat$CCAvg = as.integer(dummy_cat$CCAvg)  #Converting CCAvg loan to
categorical

library(class)

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

library(ISLR)

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

library(dplyr)
library(caret)

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

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
## Loading required package: lattice
```

```
set.seed(124)
train.index <- sample(row.names(dummy_cat), 0.6*dim(dummy_cat)[1])  # 60 % of
the data into training set
valid.index <- setdiff(row.names(dummy_cat), train.index) #the remaining data
into validation index
train.df <- dummy_cat[train.index, ] # storing the values from index into
train data frame
valid.df <- dummy_cat[valid.index, ] # storing the values from index into
valid data frame

condition = 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) # Given
conditions


normal <- preProcess(train.df[,-c(10)], method=c("center", "scale")) #
normalizing the data
train.df[, -c(10)] <- predict(normal, train.df[, -c(10)]) # prediction using
normalized data into training model
valid.df[, -c(10)] <- predict(normal, valid.df[, -c(10)])# predicting
normalized data with valid data frame
condition <- predict(normal, condition) # predicting normalized data and with
the given condition

K1 <- knn(train = train.df[,-c(10)],test = condition, cl = train.df[,10],
k=1, prob=TRUE) # applying knn alorithm
knn.attributes <- attributes(K1)

knn.attributes[1]
```

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

```
knn.attributes[3]
```

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

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 values from 1 to 5

for(i in 1:5) { #i in 1:5, is a recurssive login from 1 to 5.
  k2 <- knn(train = train.df[,-10],test = valid.df[,-10], cl = train.df[,10],
k=i, prob=TRUE)
  accuracy.df[i, 2] <- confusionMatrix(k2, valid.df[,10])$overall[1] # for
```

```
loop to generate accuracy for k values from 1 to 5
}
accuracy.df # k=1 has the highest accuracy
```

```
##   k accuracy
## 1 1   0.9635
## 2 2   0.9580
## 3 3   0.9615
## 4 4   0.9605
## 5 5   0.9600
```

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

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1804   53
##          1   20  123
##
##                Accuracy : 0.9635
##                  95% CI : (0.9543, 0.9713)
##     No Information Rate : 0.912
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7516
##
##  Mcnemar's Test P-Value : 0.0001802
##
##             Sensitivity : 0.9890
##             Specificity : 0.6989
##          Pos Pred Value : 0.9715
##          Neg Pred Value : 0.8601
##              Prevalence : 0.9120
##          Detection Rate : 0.9020
##    Detection Prevalence : 0.9285
##       Balanced Accuracy : 0.8439
##
##        'Positive' Class : 0
##
```

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 = train.df[,-10],test = Customer123, cl = train.df[,10], k=3,
prob=TRUE) # best K is 3 as it

K4

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

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

# Loading index values to respective data frame.
train.df <- dummy_cat[train.index, ]
valid.df <- dummy_cat[valid.index, ]
test.df <- dummy_cat[test.index, ]

normal <- preProcess(train.df, method=c("center", "scale"))
train.df <- predict(normal, train.df) #predicting train data with nomalized
data
valid.df <- predict(normal, valid.df) #predicting Valid data with nomalized
data
test.df <- predict(normal, test.df)  # predicting Test data with nomalized
data


#applying Knn for test, train, valid
testk <- knn(train = train.df[,-c(10)],test = test.df[,-c(10)], cl =
train.df[,10], k=5, prob=TRUE)
validk <- knn(train = train.df[,-c(10)],test = valid.df[,-c(10)], cl =
train.df[,10], k=3, prob=TRUE)
traink <- knn(train = train.df[,-c(10)],test = train.df[,-c(10)], cl =
train.df[,10], k=4, prob=TRUE)
```

```r
# confusion matrix for test, train,valid which has knn algo applied to it
confusionMatrix(testk, test.df[,10])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 908  36
##          1   5  51
##
##                Accuracy : 0.959
##                  95% CI : (0.9448, 0.9704)
##     No Information Rate : 0.913
##     P-Value [Acc > NIR] : 9.501e-09
##
##                   Kappa : 0.6923
##
##  Mcnemar's Test P-Value : 2.797e-06
##
##             Sensitivity : 0.9945
##             Specificity : 0.5862
##          Pos Pred Value : 0.9619
##          Neg Pred Value : 0.9107
##              Prevalence : 0.9130
##          Detection Rate : 0.9080
##    Detection Prevalence : 0.9440
##       Balanced Accuracy : 0.7904
##
##        'Positive' Class : 0
##
```

```r
confusionMatrix(validk, valid.df[,10])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1346   57
##          1    4   93
##
##                Accuracy : 0.9593
##                  95% CI : (0.9481, 0.9688)
##     No Information Rate : 0.9
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.732
##
##  Mcnemar's Test P-Value : 2.777e-11
##
```

```
##             Sensitivity : 0.9970
##             Specificity : 0.6200
##          Pos Pred Value : 0.9594
##          Neg Pred Value : 0.9588
##              Prevalence : 0.9000
##          Detection Rate : 0.8973
##    Detection Prevalence : 0.9353
##       Balanced Accuracy : 0.8085
##
##        'Positive' Class : 0
##
```

```r
confusionMatrix(traink, train.df[,10])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2249   64
##          1    8  179
##
##                Accuracy : 0.9712
##                  95% CI : (0.9639, 0.9774)
##     No Information Rate : 0.9028
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8171
##
##  Mcnemar's Test P-Value : 9.063e-11
##
##             Sensitivity : 0.9965
##             Specificity : 0.7366
##          Pos Pred Value : 0.9723
##          Neg Pred Value : 0.9572
##              Prevalence : 0.9028
##          Detection Rate : 0.8996
##    Detection Prevalence : 0.9252
##       Balanced Accuracy : 0.8665
##
##        'Positive' Class : 0
##
```