DSCI5340_HW3_Group10

4/7/2024

***k*-NN Classification using Universal Bank data**

```
pacman::p_load(dplyr, caret, data.table, ggplot2, FNN, dummies)
knitr::opts_chunk$set(echo = TRUE, fig.width=12, fig.height=6, fig.path = 'Figs/')
theme_set(theme_classic())
options(digits = 3)
```

```
# adjusting the seed for reproducibility
set.seed(42)
```

Partition the data

```
bank.df <- read.csv("UniversalBank.csv", stringsAsFactors = TRUE)
colnames(bank.df)
```

```
##  [1] "ID"                 "Age"                "Experience"
##  [4] "Income"             "ZIP.Code"           "Family"
##  [7] "CCAvg"              "Education"          "Mortgage"
## [10] "Personal.Loan"      "Securities.Account" "CD.Account"
## [13] "Online"             "CreditCard"
```

```
bank.df$Education <- as.factor(bank.df$Education)

# Perform one-hot encoding using model.matrix
encoded_education <- model.matrix(~ Education - 1, data = bank.df)

# (creating) New data frame with the one-hot encoded columns
encoded_education_df <- data.frame(encoded_education)
colnames(encoded_education_df) <- sub("^Education", "Education_", colnames(encoded_education_d
f))

bank.df <- subset(bank.df, select = -Education)

# Combine the original data frame with the one-hot encoded columns
bank.df <- cbind(bank.df, encoded_education_df)

# Resulting Data Frame
str(bank.df)
```

```
## 'data.frame':    5000 obs. of  16 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 ...
##  $ 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 ...
##  $ Education_1       : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ Education_2       : num  0 0 0 1 1 1 1 0 1 0 ...
##  $ Education_3       : num  0 0 0 0 0 0 0 1 0 1 ...
```

1. Partition the data into training (75%) and validation (25%) sets

```
train.index <- sample(row.names(bank.df), 0.75*dim(bank.df)[1])
valid.index <- setdiff(row.names(bank.df), train.index)
train.df <- bank.df[train.index, ]
valid.df <- bank.df[valid.index, ]
```

2. Consider the following customer for classification: Age = 40,Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education_1 = 0, Education_2 = 1, Education_3 = 0, Mortgage = 0, Securities Account = 1, CD Account = 1, Online = 1, and Credit Card = 1.

```
new_customer.df <- 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 = 1, CD.Account = 1, Online = 1,
CreditCard = 1)
colnames(new_customer.df)
```

```
##  [1] "Age"                 "Experience"          "Income"
##  [4] "Family"              "CCAvg"               "Education_1"
##  [7] "Education_2"         "Education_3"         "Mortgage"
## [10] "Securities.Account" "CD.Account"          "Online"
## [13] "CreditCard"
```

```
colnames(train.df[, c(2:4, 6:8, 10:16)])
```

```
##  [1] "Age"                 "Experience"          "Income"
##  [4] "Family"              "CCAvg"               "Mortgage"
##  [7] "Securities.Account" "CD.Account"          "Online"
## [10] "CreditCard"          "Education_1"         "Education_2"
## [13] "Education_3"
```

Preprocess the data

```
# creating copies or duplicating the datasets
train.norm.df <- train.df
valid.norm.df <- valid.df
bank.norm.df <- bank.df
```

3. Standardize all the data sets using mean and standard deviations.

```
# preprocess() estimates the required parameter
norm.values <- preProcess(train.df[, c(2:4, 6:8, 10:16)], method=c("center", "scale"))
# predict is used to preprocess the data using the parameters above
train.norm.df[, c(2:4, 6:8, 10:16)] <- predict(norm.values, train.df[, c(2:4, 6:8, 10:16)])
valid.norm.df[, c(2:4, 6:8, 10:16)] <- predict(norm.values, valid.df[, c(2:4, 6:8, 10:16)])
bank.norm.df[, c(2:4, 6:8, 10:16)] <- predict(norm.values, bank.df[, c(2:4, 6:8, 10:16)])
new.norm.df <- predict(norm.values, new_customer.df)
```

4. Perform a k-NN classification with all predictors except ID and ZIP code using k = 1. How would this customer be classified?

```
nn <- knn(train = train.norm.df[, c(2:4, 6:8, 10:16)], test = new.norm.df,
          cl = train.norm.df[, 9], k = 1)
  # Nearest-neighbor Index (ratio of observed distance divided by the expected distance)
row.names(train.norm.df)[attr(nn, "nn.index")]
```

```
## [1] "3557"
```

The customer can be classified as unlikely to take a personal loan as the customer with ID 3557 did not take a personal loan according to the data set.

5. Now find the optimal value of k using the validation data set. What is the optimal k?

```
  # Initialize a data frame with two columns: k and accuracy
accuracy.df <- data.frame(k = seq(1, 16, 1), accuracy = rep(0, 16))

train.norm.df[, 9] <- factor(train.norm.df[, 9])
valid.norm.df[, 9] <- factor(valid.norm.df[, 9])



#finding optimum value of k
for(i in 1:16) {
  knn.pred <- knn(train.norm.df[, c(2:4, 6:8, 10:16)], valid.norm.df[, c(2:4, 6:8, 10:16)],
                  cl = train.norm.df[, 9], k = i)
 accuracy.df[i, 2] <- confusionMatrix(knn.pred, valid.norm.df[, 9])$overall[1]
}


accuracy.df
```

```
##     k accuracy
## 1   1    0.962
## 2   2    0.952
## 3   3    0.961
## 4   4    0.954
## 5   5    0.960
## 6   6    0.955
## 7   7    0.960
## 8   8    0.955
## 9   9    0.957
## 10 10    0.950
## 11 11    0.953
## 12 12    0.950
## 13 13    0.947
## 14 14    0.946
## 15 15    0.947
## 16 16    0.945
```

K = 1 has the highest accuracy, hence we can take 1 as the optimal k

```
  # Re-run KNN with all observations
    # using k = 1 as it is giving the highest accuracy of 0.962

knn.pred.optimal <- knn(train = train.norm.df[, c(2:4, 6:8, 10:16)],
                        test = valid.norm.df[, c(2:4, 6:8, 10:16)],
                        cl = train.norm.df[, 9], k = 1)



row.names(bank.norm.df)[attr(knn.pred.optimal, "nn.index")]
```

```
##   [1] "301"  "539"  "2008" "38"   "681"  "3509" "3109" "2512" "1772" "847" 
##  [11] "2234" "1715" "386"  "210"  "3061" "1408" "234"  "1047" "1389" "136" 
##  [21] "1559" "1819" "2180" "3195" "887"  "54"   "3155" "701"  "2174" "1619"
##  [31] "2563" "3590" "1848" "175"  "705"  "2679" "3750" "3071" "3043" "39"  
##  [41] "26"   "1654" "2455" "617"  "3498" "1416" "1817" "1703" "1856" "1714"
##  [51] "2049" "1857" "666"  "2939" "684"  "1058" "324"  "2738" "289"  "2387"
##  [61] "1307" "3358" "3061" "227"  "10"   "1791" "3733" "2837" "662"  "2235"
##  [71] "3362" "502"  "2106" "2896" "963"  "2137" "3411" "3137" "200"  "1346"
##  [81] "1519" "3362" "1039" "3562" "1360" "308"  "731"  "984"  "248"  "3412"
##  [91] "1057" "2944" "1294" "1642" "859"  "267"  "3586" "2626" "3578" "3461"
## [101] "1692" "1649" "2881" "895"  "2755" "567"  "687"  "1522" "1987" "1255"
## [111] "1302" "2003" "1629" "466"  "3356" "2051" "2868" "2356" "1492" "3265"
## [121] "2650" "1247" "461"  "762"  "1592" "2555" "1486" "1419" "294"  "551" 
## [131] "464"  "333"  "3434" "1154" "1368" "2545" "2147" "1499" "3560" "3090"
## [141] "3522" "3344" "1548" "1579" "708"  "2584" "3664" "2928" "51"   "3678"
## [151] "1710" "2674" "2263" "697"  "3054" "778"  "3151" "812"  "1462" "660" 
## [161] "1367" "2469" "305"  "55"   "1012" "1912" "2929" "2365" "2916" "817" 
## [171] "682"  "288"  "1113" "2362" "2201" "1646" "1013" "1014" "3737" "2026"
## [181] "2625" "1187" "1099" "1445" "319"  "3225" "730"  "235"  "3323" "1006"
## [191] "2465" "1734" "1907" "621"  "1657" "186"  "2012" "3055" "2909" "2322"
## [201] "3460" "1331" "755"  "764"  "2127" "926"  "687"  "1772" "1115" "1909"
## [211] "2067" "1859" "1092" "3236" "1432" "2371" "1611" "3245" "2490" "2112"
## [221] "1029" "1529" "2035" "2133" "1938" "3378" "3509" "1569" "2170" "3523"
## [231] "1095" "3656" "2982" "2761" "841"  "1594" "3607" "3743" "1704" "3354"
## [241] "1013" "2726" "1029" "3676" "2571" "3666" "1827" "313"  "1770" "2388"
## [251] "2565" "2784" "594"  "1492" "826"  "1842" "1149" "3690" "3464" "1369"
## [261] "323"  "1249" "2650" "1202" "1165" "1171" "3272" "1605" "721"  "1068"
## [271] "693"  "1189" "2088" "2805" "898"  "2313" "3665" "3219" "170"  "33"  
## [281] "1827" "1079" "408"  "1060" "2602" "153"  "3671" "3191" "238"  "1621"
## [291] "1992" "2520" "2320" "3190" "2054" "1697" "404"  "1686" "1342" "3145"
## [301] "2490" "941"  "1416" "1725" "2636" "3609" "1978" "2012" "826"  "2884"
## [311] "2434" "2540" "3640" "3608" "2157" "3664" "568"  "942"  "2650" "2303"
## [321] "1134" "167"  "2083" "1433" "317"  "3520" "731"  "1397" "1759" "3054"
## [331] "1681" "3292" "2717" "1798" "2626" "1198" "2153" "3410" "2996" "1000"
## [341] "1194" "2546" "1634" "1956" "288"  "1906" "2151" "142"  "2110" "1444"
## [351] "1248" "1721" "3008" "3107" "482"  "1614" "677"  "1402" "2654" "159" 
## [361] "2794" "636"  "2568" "457"  "467"  "2576" "2643" "3278" "427"  "3670"
## [371] "824"  "2978" "3596" "1814" "1024" "2584" "2359" "1061" "2475" "1269"
## [381] "318"  "178"  "181"  "2459" "2272" "3068" "3292" "2921" "480"  "1714"
## [391] "2059" "3700" "3005" "925"  "1268" "2521" "3210" "200"  "1169" "1901"
## [401] "1074" "71"   "1405" "3175" "822"  "3372" "2230" "3054" "3217" "1759"
## [411] "2442" "3263" "3638" "335"  "3296" "3022" "3040" "1929" "2916" "1183"
## [421] "2716" "835"  "2876" "2860" "3661" "1010" "1555" "370"  "811"  "909" 
## [431] "1759" "3411" "3314" "1117" "328"  "2075" "3455" "1980" "1730" "476" 
## [441] "1885" "2364" "1922" "589"  "1133" "2854" "693"  "1774" "2893" "107" 
## [451] "743"  "1163" "3680" "1302" "809"  "1772" "1124" "3220" "3726" "1930"
## [461] "2687" "2562" "2328" "2019" "3166" "1164" "447"  "1513" "2293" "3265"
## [471] "2033" "1751" "91"   "2089" "691"  "136"  "3389" "136"  "3134" "1766"
## [481] "3722" "3336" "3505" "1720" "427"  "3494" "3454" "2836" "931"  "2465"
## [491] "883"  "761"  "1073" "20"   "1234" "965"  "2080" "73"   "3257" "189" 
## [501] "15"   "3090" "3214" "1185" "3271" "1589" "645"  "231"  "678"  "1377"
## [511] "3332" "1601" "3659" "106"  "2340" "3275" "1557" "3455" "89"   "1520"
```

```
##  [521] "3693" "3093" "3519" "1751" "30"   "1758" "3045" "2517" "3370" "2581"
##  [531] "643"  "3426" "3598" "2622" "2702" "2604" "614"  "1952" "1825" "1416"
##  [541] "3514" "2016" "447"  "2872" "845"  "1372" "3308" "3455" "920"  "896"
##  [551] "149"  "1792" "2814" "3080" "363"  "1693" "2986" "1968" "1742" "1282"
##  [561] "3087" "1193" "937"  "2674" "308"  "2364" "2563" "343"  "246"  "1133"
##  [571] "721"  "3335" "544"  "1293" "836"  "993"  "522"  "822"  "2970" "3644"
##  [581] "28"   "3017" "1062" "821"  "685"  "541"  "1839" "772"  "2957" "1282"
##  [591] "1571" "2327" "1006" "3233" "1936" "2448" "1364" "1372" "1562" "3140"
##  [601] "3532" "1267" "829"  "2999" "1512" "3368" "1224" "665"  "1169" "76"
##  [611] "3273" "904"  "802"  "3381" "1019" "3210" "400"  "2355" "2077" "1501"
##  [621] "3682" "3735" "515"  "3370" "397"  "3016" "3112" "2504" "2025" "263"
##  [631] "738"  "1289" "1162" "2930" "2210" "3191" "2061" "2012" "885"  "2619"
##  [641] "1976" "3037" "1230" "1778" "551"  "1305" "3050" "3426" "1016" "3596"
##  [651] "2241" "1445" "1121" "3004" "934"  "3036" "2602" "280"  "2059" "2443"
##  [661] "3598" "2129" "274"  "3571" "237"  "1415" "3690" "3150" "311"  "1911"
##  [671] "2020" "1677" "3543" "2349" "2253" "2520" "2632" "3322" "3314" "2884"
##  [681] "2163" "3709" "1270" "3194" "2857" "2932" "1862" "3657" "2083" "1222"
##  [691] "2608" "3042" "2906" "717"  "816"  "2034" "1822" "3707" "3636" "1844"
##  [701] "3659" "781"  "3461" "2266" "2902" "1229" "3462" "535"  "559"  "2358"
##  [711] "1992" "3476" "2521" "1475" "2960" "3531" "447"  "1684" "3656" "2068"
##  [721] "2442" "3553" "1621" "933"  "2879" "866"  "2979" "937"  "2290" "2574"
##  [731] "934"  "1968" "3239" "1015" "1381" "2014" "1420" "2193" "2096" "1039"
##  [741] "937"  "3215" "704"  "2675" "1946" "1558" "2590" "2034" "1410" "3102"
##  [751] "2686" "2296" "847"  "1085" "1986" "926"  "310"  "1557" "2560" "726"
##  [761] "1757" "1766" "2750" "830"  "1319" "2341" "3337" "362"  "2345" "1784"
##  [771] "2946" "2856" "1520" "293"  "966"  "2802" "3512" "3615" "527"  "255"
##  [781] "1927" "953"  "202"  "377"  "1274" "2076" "3437" "1978" "353"  "2857"
##  [791] "347"  "1709" "1892" "2818" "3015" "1543" "1893" "1836" "2080" "829"
##  [801] "1190" "387"  "3070" "3576" "2112" "215"  "3061" "2897" "3486" "2154"
##  [811] "1583" "2736" "3005" "3139" "2999" "1715" "2439" "3227" "1378" "948"
##  [821] "1090" "3577" "2579" "288"  "3597" "2982" "3387" "917"  "1046" "935"
##  [831] "2080" "3066" "3318" "1513" "3220" "639"  "765"  "2209" "3682" "738"
##  [841] "2092" "1646" "2164" "3559" "3294" "1805" "963"  "297"  "1937" "2658"
##  [851] "444"  "1617" "1007" "1898" "1938" "1379" "3742" "1991" "447"  "410"
##  [861] "3295" "1344" "953"  "5"    "454"  "1908" "3520" "28"   "793"  "3226"
##  [871] "3136" "2111" "484"  "2948" "3062" "2645" "755"  "2468" "288"  "3024"
##  [881] "2546" "2768" "2564" "2283" "1241" "3667" "3384" "805"  "2691" "1939"
##  [891] "2232" "3518" "2333" "2160" "1029" "3426" "921"  "3331" "1895" "38"
##  [901] "56"   "418"  "3002" "2540" "763"  "2024" "3198" "2205" "2"    "225"
##  [911] "1508" "2618" "115"  "1874" "1735" "1667" "3739" "1308" "413"  "3064"
##  [921] "2312" "2902" "1151" "3696" "3301" "2381" "355"  "398"  "834"  "3661"
##  [931] "2077" "1987" "1133" "1630" "2619" "3101" "3735" "3378" "1367" "421"
##  [941] "2432" "3621" "1365" "1319" "979"  "1656" "1939" "3360" "1068" "643"
##  [951] "1396" "1224" "3015" "373"  "2570" "1218" "2986" "2110" "2487" "3134"
##  [961] "3239" "644"  "1002" "2786" "3177" "3359" "1219" "771"  "1217" "908"
##  [971] "1775" "1931" "771"  "2462" "3085" "1557" "1900" "276"  "1643" "3583"
##  [981] "1985" "347"  "2609" "1750" "2004" "3264" "2261" "259"  "2449" "1309"
##  [991] "328"  "2635" "2040" "1437" "933"  "19"   "1652" "3257" "1700" "2235"
## [1001] "109"  "1517" "3172" "206"  "1929" "3242" "917"  "3151" "834"  "3387"
## [1011] "1222" "407"  "1784" "1076" "2863" "1377" "2059" "1935" "2341" "1475"
## [1021] "90"   "408"  "3525" "1968" "1654" "529"  "2878" "1355" "2005" "3550"
## [1031] "2427" "2419" "769"  "1140" "571"  "1180" "3181" "764"  "3572" "1079"
```

```
## [1041] "947"  "3083" "2783" "386"  "2565" "1044" "3528" "3370" "3226" "123"
## [1051] "809"  "3171" "1005" "808"  "916"  "3387" "2935" "817"  "2015" "2626"
## [1061] "3677" "534"  "1508" "683"  "3343" "944"  "1158" "2795" "1019" "653"
## [1071] "1"    "828"  "1024" "1079" "1894" "313"  "2834" "884"  "1520" "2705"
## [1081] "199"  "786"  "3412" "1239" "97"   "2814" "2923" "680"  "2312" "580"
## [1091] "3189" "426"  "3325" "2605" "3637" "649"  "2776" "2137" "3140" "2986"
## [1101] "2209" "659"  "897"  "426"  "490"  "2903" "1605" "2124" "2366" "3569"
## [1111] "2465" "139"  "3325" "1968" "461"  "2520" "3374" "670"  "3260" "105"
## [1121] "399"  "2023" "1282" "3568" "2520" "1515" "1547" "1703" "436"  "2172"
## [1131] "2014" "779"  "2574" "824"  "529"  "2053" "2278" "3547" "618"  "3700"
## [1141] "705"  "3169" "105"  "2579" "567"  "223"  "1783" "2015" "2531" "2001"
## [1151] "3610" "3598" "470"  "1413" "872"  "2815" "3263" "2437" "607"  "3263"
## [1161] "3279" "2185" "1901" "2487" "38"   "296"  "1227" "3619" "436"  "917"
## [1171] "430"  "109"  "345"  "3064" "2971" "2899" "1978" "2444" "193"  "1650"
## [1181] "3709" "3362" "1840" "2154" "865"  "3576" "3324" "1334" "852"  "361"
## [1191] "519"  "951"  "1828" "259"  "151"  "33"   "2943" "3510" "382"  "1407"
## [1201] "2755" "130"  "136"  "2442" "1065" "65"   "1742" "200"  "2163" "1689"
## [1211] "787"  "536"  "3042" "3034" "377"  "3318" "2397" "1842" "3663" "2637"
## [1221] "1366" "3351" "2894" "56"   "842"  "1369" "1212" "3742" "544"  "1045"
## [1231] "2095" "1796" "2001" "1607" "1779" "194"  "3571" "1500" "1748" "1689"
## [1241] "1297" "3663" "1419" "2443" "3061" "1062" "3543" "1348" "3387" "1707"
```

```
length(knn.pred.optimal)
```

```
## [1] 1250
```

```
length(valid.norm.df[, 9])
```

```
## [1] 1250
```

6.Using ConfusionMatrix() function from the caret package, print the confusion matrix for the validation data that results from using the optimal k.

```
confusion_matrix <- confusionMatrix(knn.pred.optimal, valid.norm.df[, 9])
confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1114   34
##          1   14   88
##
##                Accuracy : 0.962
##                  95% CI : (0.949, 0.972)
##     No Information Rate : 0.902
##     P-Value [Acc > NIR] : 1.74e-15
##
##                   Kappa : 0.765
##
##  Mcnemar's Test P-Value : 0.0061
##
##             Sensitivity : 0.988
##             Specificity : 0.721
##          Pos Pred Value : 0.970
##          Neg Pred Value : 0.863
##              Prevalence : 0.902
##          Detection Rate : 0.891
##    Detection Prevalence : 0.918
##       Balanced Accuracy : 0.854
##
##        'Positive' Class : 0
##
```

7. Classify the customer specified in Question 2 using the best k.

```
knn.pred.new <- knn(bank.norm.df[, c(2:4, 6:8, 10:16)], new.norm.df,
                    cl = bank.norm.df[, 9], k = 1)


row.names(bank.norm.df)[attr(knn.pred.new, "nn.index")]
```

```
## [1] "4336"
```

The above mentioned customer index 4336 has not taken any personal loan previously. so, with k=1 ,we can predict that the new customer may not likely to take loan

8. Now repartition the data into three parts: training, validation, and test sets (50%, 30%, and 20%).

```
# Repartitioning data
train2.index <- sample(row.names(bank.df), 0.50 * dim(bank.df)[1])  # 50% for training
valid2.index <- sample(setdiff(row.names(bank.df), train2.index), 0.30 * dim(bank.df)[1])  # 30%
for validation
test2.index <- setdiff(row.names(bank.df), c(train2.index, valid2.index))  # 20% for test

train2.df <- bank.df[train2.index, ]
valid2.df <- bank.df[valid2.index, ]
test2.df <- bank.df[test2.index, ]
head(train2.df)
```

```
##        ID Age Experience Income ZIP.Code Family CCAvg Mortgage Personal.Loan
## 1910 1910  56         30    101    90048      3   1.7        0             0
## 3425 3425  44         19     45    94539      4   0.0        0             0
## 2346 2346  65         40     89    90291      1   4.1      299             1
## 4235 4235  50         24     91    93555      1   0.8        0             0
## 2977 2977  33          8     82    95747      1   2.6        0             0
## 920   920  51         27     88    91116      1   2.6        0             0
##      Securities.Account CD.Account Online CreditCard Education_1 Education_2
## 1910                  0          0      0          1           0           1
## 3425                  0          0      1          1           0           1
## 2346                  0          1      1          0           1           0
## 4235                  0          0      1          0           0           0
## 2977                  1          1      1          1           0           1
## 920                   1          0      0          1           0           1
##      Education_3
## 1910           0
## 3425           0
## 2346           0
## 4235           1
## 2977           0
## 920            0
```

```
head(valid2.df)
```

```
##        ID Age Experience Income ZIP.Code Family CCAvg Mortgage Personal.Loan
## 2433 2433  54         30     45    92182      4  0.90        0             0
## 262   262  42         16    111    93106      2  1.20      251             1
## 4486 4486  35          9     50    92182      4  2.20        0             0
## 2054 2054  58         32     85    92110      2  2.00      161             0
## 866   866  60         34     22    92037      3  0.30      139             0
## 3929 3929  57         33     61    92115      3  2.67        0             0
##      Securities.Account CD.Account Online CreditCard Education_1 Education_2
## 2433                  0          0      0          1           0           1
## 262                   0          0      1          0           0           0
## 4486                  0          0      0          0           0           1
## 2054                  1          1      1          1           1           0
## 866                   0          0      1          1           0           0
## 3929                  0          0      1          0           1           0
##      Education_3
## 2433           0
## 262            1
## 4486           0
## 2054           0
## 866            1
## 3929           0
```

```
head(test2.df)
```

```
##    ID Age Experience Income ZIP.Code Family CCAvg Mortgage Personal.Loan
## 3   3  39         15     11    94720      1   1.0        0             0
## 4   4  35          9    100    94112      1   2.7        0             0
## 6   6  37         13     29    92121      4   0.4      155             0
## 9   9  35         10     81    90089      3   0.6      104             0
## 10 10  34          9    180    93023      1   8.9        0             1
## 17 17  38         14    130    95010      4   4.7      134             1
##    Securities.Account CD.Account Online CreditCard Education_1 Education_2
## 3                   0          0      0          0           1           0
## 4                   0          0      0          0           0           1
## 6                   0          0      1          0           0           1
## 9                   0          0      1          0           0           1
## 10                  0          0      0          0           0           0
## 17                  0          0      0          0           0           0
##    Education_3
## 3            0
## 4            0
## 6            0
## 9            0
## 10           1
## 17           1
```

```
# create copies of the datasets
train2.norm.df <- train2.df
valid2.norm.df <- valid2.df
test2.norm.df <- test2.df

# Standardize data using preProcess() from CARET
set.seed(42)
  # preprocess() estimates the required parameter
norm2.values <- preProcess(train2.df[, c(2:4, 6:8, 10:16)], method=c("center", "scale"))
  # predict is used to preprocess the data using the parameters above
train2.norm.df[, c(2:4, 6:8, 10:16)] <- predict(norm2.values, train2.df[, c(2:4, 6:8, 10:16)])
valid2.norm.df[, c(2:4, 6:8, 10:16)] <- predict(norm2.values, valid2.df[, c(2:4, 6:8, 10:16)])
bank.norm.df[, c(2:4, 6:8, 10:16)] <- predict(norm.values, bank.df[, c(2:4, 6:8, 10:16)])
new.norm.df <- predict(norm.values, new_customer.df)
```

9. Apply the k-NN method with the optimal k chosen above.

```
train2.df[, 9] <- factor(train2.df[, 9])
valid2.df[, 9] <- factor(valid2.df[, 9])
test2.df[, 9] <- factor(test2.df[, 9])

knn.pred_train2 <- knn(train = train2.df[, c(2:4, 6:8, 10:16)], test = train2.df[, c(2:4, 6:8, 1
0:16)],
                      cl = train2.df[, 9], k = 3)#here K=3 (second highest K value ) is chosen i
n order to avoid model overfitting

# Apply k-NN with the best k on the validation set
knn.pred_valid2 <- knn(train = train2.df[, c(2:4, 6:8, 10:16)], test = valid2.df[, c(2:4, 6:8, 1
0:16)],
                      cl = train2.df[, 9], k = 3)

# Apply k-NN with the best k on the test set
knn.pred_test2 <- knn(train = train2.df[, c(2:4, 6:8, 10:16)], test = test2.df[, c(2:4, 6:8, 10:
16)],
                     cl = train2.df[, 9], k = 3)
```

10. Compare the confusion matrix of the test set with that of the training and validation sets. Comment on the differences and their reason

```
confusion_matrix_train2 <- confusionMatrix(knn.pred_train2, train2.df[, 9])
confusion_matrix_valid2 <- confusionMatrix(knn.pred_valid2, valid2.df[, 9])
confusion_matrix_test2 <- confusionMatrix(knn.pred_test2, test2.df[, 9])

confusion_matrix_train2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2221   81
##          1   37  161
##
##                Accuracy : 0.953
##                  95% CI : (0.944, 0.961)
##     No Information Rate : 0.903
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.706
##
##  Mcnemar's Test P-Value : 7.54e-05
##
##             Sensitivity : 0.984
##             Specificity : 0.665
##          Pos Pred Value : 0.965
##          Neg Pred Value : 0.813
##              Prevalence : 0.903
##          Detection Rate : 0.888
##    Detection Prevalence : 0.921
##       Balanced Accuracy : 0.824
##
##        'Positive' Class : 0
##
```

```
confusion_matrix_valid2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1303   96
##          1   49   52
##
##                Accuracy : 0.903
##                  95% CI : (0.887, 0.918)
##     No Information Rate : 0.901
##     P-Value [Acc > NIR] : 0.418643
##
##                   Kappa : 0.367
##
##  Mcnemar's Test P-Value : 0.000133
##
##             Sensitivity : 0.964
##             Specificity : 0.351
##          Pos Pred Value : 0.931
##          Neg Pred Value : 0.515
##              Prevalence : 0.901
##          Detection Rate : 0.869
##    Detection Prevalence : 0.933
##       Balanced Accuracy : 0.658
##
##        'Positive' Class : 0
##
```

```
confusion_matrix_test2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 882  59
##          1  28  31
##
##                Accuracy : 0.913
##                  95% CI : (0.894, 0.93)
##     No Information Rate : 0.91
##     P-Value [Acc > NIR] : 0.3966
##
##                   Kappa : 0.371
##
##  Mcnemar's Test P-Value : 0.0013
##
##             Sensitivity : 0.969
##             Specificity : 0.344
##          Pos Pred Value : 0.937
##          Neg Pred Value : 0.525
##              Prevalence : 0.910
##          Detection Rate : 0.882
##    Detection Prevalence : 0.941
##       Balanced Accuracy : 0.657
##
##        'Positive' Class : 0
##
```

These are results for k=1 Accuracy for confusion_matrix_train2 = 1 Accuracy for validation. = 0.916 Accuracy for testing = 0.902 The above Accuracy Results are Good Kappa values = 0.454,0.414 indicates that the model may be overfit

Results for k =3 (second highest K value) Accuracy for confusion_matrix_train2 = 0.953 Accuracy for validation. = 0.903
Accuracy for testing = 0.913 The above Accuracy Results are Good Kappa values = 0.367,0.371 by taking k=3 we an avoid an overfit model