Assignment_2

wliu16

2022-09-30

1. Classify a customer with k = 1

```r
### The program is as follows
## 1) partition 2) normalize 3) create model and apply 4) compute knn k=1
## 5) find best k 6) rerun with best k on whole data

m_bank_dummy<- bank.df[-c(1, 5, 10)] #subset the data

# Create dummy variable
# All the categorical columns are already in 1s and 0s
# Convert to factor but no need to convert to dummy variables
m_bank_dummy$Personal.Loan <- as.factor(bank.df$Personal.Loan)

#Partition 80% of random rows for training and rest for validation
set.seed(115)
Index_Train <- createDataPartition(m_bank_dummy$Personal.Loan,
                                   p=0.80,
                                   list = FALSE)
Train <- m_bank_dummy[Index_Train, ]
Validate <- m_bank_dummy[-Index_Train, ]
TrainVal <- m_bank_dummy

# Create customer data point with 11 dim
New <-data.frame(Age = 40, Experience = 10, Income = 84, Family = 2,
                      CCAvg = 2, Education = 1, Mortgage = 0,
                      Securities.Account = 0, CD.Account = 0, Online = 1,
                      CreditCard = 1)

train.norm.df <- Train
valid.norm.df <- Validate
trainVal.norm.df <- TrainVal
new.norm.df <- New

# Create normalized model
norm_var <-  c("Age", "Experience", "Income", "Family", "CCAvg",
                     "Education", "Mortgage",
                     "Securities.Account", "CD.Account", "Online","CreditCard")
train.norm.df[, norm_var] <- Train[, norm_var]
valid.norm.df[, norm_var] <- Validate[, norm_var]

norm.values <- preProcess(Train[, norm_var], method = c("center", "scale"))
train.norm.df[, norm_var] <- predict(norm.values, Train[, norm_var])
valid.norm.df[, norm_var] <- predict(norm.values, Validate[, norm_var])
```

```r
new.norm.df <- predict(norm.values, new.norm.df)

# Check normalized values
#summary(train.norm.df) #dim 12
#summary(valid.norm.df) #dim 12
#summary(new.norm.df) #dim 11
```

```r
#use knn to compute knn
nn  <- knn(train = train.norm.df[, norm_var],
           test = new.norm.df,
           cl = train.norm.df[, 12],
           k = 1)
row.names(m_bank_dummy)[attr(nn, "nn.index")]
```

```
## [1] "2739"
```

```r
nn
```

```
## [1] 0
## attr(,"nn.index")
##      [,1]
## [1,] 2739
## attr(,"nn.dist")
##           [,1]
## [1,] 0.4515491
## Levels: 0
```

```r
# Out of the 12 columns, Personal.Loan is the classifier and rest are predictors
# Compute KNN where K=1
Train_Predictors <- Train[, -12]
Validate_Predictors <- Validate[, -12]
TrainVal_Predictors <- TrainVal[, -12]

Train_Labels <- Train[, 12]
Validate_Labels <- Validate[, 12]
TrainVal_Labels <- TrainVal[, 12]

Predicted_Test_Labels <- knn(Train_Predictors,
                             new.norm.df,
                             cl= Train_Labels,
                             k = 1)
str(Predicted_Test_Labels)
```

```
##  Factor w/ 1 level "0": 1
##  - attr(*, "nn.index")= int [1, 1] 3060
##  - attr(*, "nn.dist")= num [1, 1] 26.8
```

```r
head(Predicted_Test_Labels)
```

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

Comments: The customer will be classfied as 0 which is “No” for Personal Loan for k = 1

2. Find the best k that balances between overfitting and underfitting

```r
#find the best k
set.seed(123)
model1 <- train(Personal.Loan~.,
               data = train.norm.df,
               method ="knn")
model1
```

```
## k-Nearest Neighbors
##
## 4000 samples
##   11 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 4000, 4000, 4000, 4000, 4000, 4000, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.9516989  0.6771440
##   7  0.9501946  0.6563369
##   9  0.9493864  0.6452656
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```r
#expand the search grid and test the best k
#set.seed(101)
searchGrid <- expand.grid(k=c(2, 5, 7, 9, 13, 15, 17, 19, 21, 25, 29))
model2 <- train(Personal.Loan~.,
                data = train.norm.df,
                method= "knn",
                tuneGrid=searchGrid)
model2
```

```
## k-Nearest Neighbors
##
## 4000 samples
##   11 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 4000, 4000, 4000, 4000, 4000, 4000, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    2  0.9482119  0.6704605
##    5  0.9493430  0.6601338
##    7  0.9498001  0.6543967
##    9  0.9488766  0.6428867
##   13  0.9471121  0.6238308
##   15  0.9457016  0.6091845
##   17  0.9445914  0.5972688
##   19  0.9431131  0.5828690
##   21  0.9419418  0.5707584
##   25  0.9401163  0.5516503
##   29  0.9384562  0.5339672
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```
best_k <- model2$bestTune[[1]] # saves the best k
best_k #print the best k
```

```
## [1] 7
```

Comment: the best k is 7

```
# Calculate the probability using the best k
Predicted_Validate_labels <- knn(Train_Predictors,
                             Validate_Predictors,
                             cl= Train_Labels,
                             k=best_k,
                             prob = TRUE)
class_prob <- attr(Predicted_Validate_labels, "prob")
head(class_prob)
```

```
## [1] 1.0000000 1.0000000 1.0000000 1.0000000 0.8571429 0.5714286
```

3. Create confusion matrix using the best K

```
CrossTable(x= Validate_Labels,
           y= Predicted_Validate_labels,
           prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  1000
##
##
##                 | Predicted_Validate_labels
## Validate_Labels |         0 |         1 | Row Total |
## ----------------|-----------|-----------|-----------|
##               0 |       882 |        22 |       904 |
##                 |     0.976 |     0.024 |     0.904 |
##                 |     0.934 |     0.393 |           |
##                 |     0.882 |     0.022 |           |
## ----------------|-----------|-----------|-----------|
##               1 |        62 |        34 |        96 |
##                 |     0.646 |     0.354 |     0.096 |
##                 |     0.066 |     0.607 |           |
##                  |     0.062 |     0.034 |           |
## -----------------|-----------|-----------|-----------|
##     Column Total |       944 |        56 |      1000 |
##                  |     0.944 |     0.056 |           |
## -----------------|-----------|-----------|-----------|
##
##
```

Comments: there is a total of 25+62 = 87 misclassification cases Recall/TPR = TP/(TP+FN) = 34/(34+62) = 35.41% Precision = TP/(TP+FP) = 34/(34+25) = 57.63% Specificity/TNR = 1- FPR= 1- FP/(FP+TN)= 1-25/(25+879)= 97.23%

Therefore, this model is not good at correctly identifying positive cases (recall) and out of all the identified positive cases identified, slightly more than half of the times the model is right (precision). The model is much better at correctly identifying negative cases (high specificity). The reason is that the dataset is extremely unbalanced (9.6% yes/Personal.Loan) and a small training dataset greatly affects the model capability to correctly identify positive cases.

4. Classify the customer using the best K

```
# Compute KNN where K=1
nn <- knn(train = Train_Predictors,
          test = new.norm.df,
          cl= Train_Labels,
          k = best_k)
row.names(m_bank_dummy)[attr(nn, "nn.index")]
```

```
## [1] "3060" "3603" "2638" "3960" "3775" "3000" "1701"
```

```
nn
```

```
## [1] 0
## attr(,"nn.index")
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,] 3060 3603 2638 3960 3775 3000 1701
## attr(,"nn.dist")
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
## [1,] 26.76496 28.07898 28.98389 29.21908 29.37744 29.56627 30.15765
## Levels: 0
```

Comments: Customer is still classfied as 0 which means no to personal loan based on the best k

5. Re-partition the data into 5:3:2 with the K chosen above. Compare confusion matrix of the test and with training + validation.

```
#Re-partition 50% of random rows for training, 30% for validation and 20%
#for testing
set.seed(111)
Index_Train <- createDataPartition(m_bank_dummy$Personal.Loan,
                                   p=0.50,
                                   list = FALSE)
RTrain <- m_bank_dummy[Index_Train, ]
TotalValidate <- m_bank_dummy[-Index_Train, ]
Index_Validate <- createDataPartition(TotalValidate$Personal.Loan,
                                      p = 0.60,
                                      list = FALSE)
RTest <- TotalValidate[-Index_Validate, ]
RValidate <- TotalValidate[Index_Validate,]
RTrainVal <- rbind(RTrain, RValidate)
```

```r
r.train.norm.df <- RTrain
r.valid.norm.df <- RValidate
r.trainVal.norm.df <- RTrainVal
r.test.norm.df <- RTest

# Create normalized model
norm_var <-  c("Age", "Experience", "Income", "Family", "CCAvg",
                      "Education", "Mortgage",
                      "Securities.Account", "CD.Account", "Online","CreditCard")
r.train.norm.df[, norm_var] <- RTrain[, norm_var]
r.valid.norm.df[, norm_var] <- RValidate[, norm_var]
r.test.norm.df[, norm_var] <- RTest[, norm_var]
r.trainVal.norm.df[, norm_var] <- RTrainVal[, norm_var]

norm.values <- preProcess(RTrain[, norm_var], method = c("center", "scale"))
r.train.norm.df[, norm_var] <- predict(norm.values, RTrain[, norm_var])
r.valid.norm.df[, norm_var] <- predict(norm.values, RValidate[, norm_var])
r.test.norm.df[, norm_var] <- predict(norm.values, RTest[, norm_var])
r.trainVal.norm.df[, norm_var] <- predict(norm.values, RTrainVal[, norm_var])

# Check normalized values
#dim(r.train.norm.df) # dim 2500 12
#dim(r.valid.norm.df) # dim 1500 12
#dim(r.test.norm.df) #dim 1000 12
#dim(r.trainVal.norm.df) #4000 12
```

```r
RTrain_Predictors <- RTrain[, -12]
RValidate_Predictors <- RValidate[, -12]
RTrainVal_Predictors <- RTrainVal[, -12]
RTest_Predictors <- RTest[, -12]

RTrain_Labels <- RTrain[, 12]
RValidate_Labels <- RValidate[, 12]
RTrainVal_Labels <- RTrainVal[, 12]
RTest_Labels <- RTest[, 12]

Predicted_Test_Labels <- knn(RTrain_Predictors,
                             RTest_Predictors,
                             cl= RTrain_Labels,
                             k = best_k)

# Calculate test confusion matrix
CrossTable(x= r.test.norm.df[, 12],
           y= Predicted_Test_Labels,
           prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  1000
##
##
##                      | Predicted_Test_Labels
## r.test.norm.df[, 12] |         0 |         1 | Row Total |
## ---------------------|-----------|-----------|-----------|
##                    0 |       880 |        24 |       904 |
##                      |     0.973 |     0.027 |     0.904 |
##                      |     0.936 |     0.400 |           |
##                      |     0.880 |     0.024 |           |
## ---------------------|-----------|-----------|-----------|
##                    1 |        60 |        36 |        96 |
##                      |     0.625 |     0.375 |     0.096 |
##                      |     0.064 |     0.600 |           |
##                      |     0.060 |     0.036 |           |
## ---------------------|-----------|-----------|-----------|
##         Column Total |       940 |        60 |      1000 |
##                      |     0.940 |     0.060 |           |
## ---------------------|-----------|-----------|-----------|
##
##
```

```
# Calculate the training and validation dataset confusion matrix
Predicted_RTrainVal_labels <- knn(RTrain_Predictors,
                              RTrainVal_Predictors,
                              cl= RTrain_Labels,
                              k=best_k,
                              prob = TRUE)

CrossTable(x= RTrainVal_Labels,
           y= Predicted_RTrainVal_labels,
           prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  4000
##
##
##                  | Predicted_RTrainVal_labels
## RTrainVal_Labels |         0 |         1 | Row Total |
## -----------------|-----------|-----------|-----------|
##                0 |      3533 |        83 |      3616 |
##                  |     0.977 |     0.023 |     0.904 |
##                  |     0.940 |     0.346 |           |
##                  |     0.883 |     0.021 |           |
## -----------------|-----------|-----------|-----------|
##                1 |       227 |       157 |       384 |
##                  |     0.591 |     0.409 |     0.096 |
##                  |     0.060 |     0.654 |           |
##                  |     0.057 |     0.039 |           |
## -----------------|-----------|-----------|-----------|
##     Column Total |      3760 |       240 |      4000 |
##                  |     0.940 |     0.060 |           |
## -----------------|-----------|-----------|-----------|
##
##
```

Comments: The Confusion matrix on Test data (1st graph) has 26+64= 90 misclassification cases out of 1000 cases (9% misclassfication rate). The Confusion matrix on TrainVal data (2nd graph) has 85+204=289 misclassification cases out of 4000 cases (7.2% misclassification rate). Recall1= 32/96 = 33% Recall2 = 180/384 = 47% Precision1 = TP/(TP+FP) = 32/58 = 55% Precision2 = TP/(TP+FP) = 180/265 = 68%

Both models will have high specification rates but the model trained on training and validation data will have a better recall and precision rate given that the training data is unbalanced and a larger data will be much more accurate