Universal Bank

Abhinav Reddy

2024-02-25

Loading the required libraries and reading a CSV file named "UniversalBank.csv" and it stores the data in a variable called data.

```
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
```

```
library(class)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
library(readr)
data <- read_csv("C:/Users/Abhinav Reddy/Desktop/Assignment 2/UniversalBank.csv")
```

```
## Rows: 5000 Columns: 14

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (14): ID, Age, Experience, Income, ZIP Code, Family, CCAvg, Education, M...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

- This below R code partitions the dataset into a training set comprising 60% of the data, amounting to 3000 observations, and a validation set comprising 40% of the data, totaling 2000 observations.

```r
# Spliting the data into training (60%) and validation (40%) sets
train_Index <- createDataPartition(data$`Personal Loan`, p = 0.6, list = FALSE, times = 1)
training_data <- data[train_Index, ]
validation_data <- data[-train_Index, ]


dim(training_data)
```

```
## [1] 3000   14
```

```r
dim(validation_data)
```

```
## [1] 2000   14
```

Question 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?

- Answer 1. This below R code performs k-nearest neighbors classification to predict whether a new customer will accept a loan offer. It prepares the data, creates dummy variables for education, defines the attributes of a new customer, and predicts their classification using k = 1. Finally, it prints the classification result.

```r
class_lbl <- data$"Personal Loan"

head(class_lbl)
```

```
## [1] 0 0 0 0 0 0
```

```r
# Convert Education into dummy variables
data$Education_1 <- ifelse(data$Education == 1, 1, 0)
data$Education_2 <- ifelse(data$Education == 2, 1, 0)
data$Education_3 <- ifelse(data$Education == 3, 1, 0)
data$"Personal Loan" <- as.factor(data$"Personal Loan")


predictor <- data[, c("Age", "Experience", "Income", "Family", "CCAvg",
                      "Mortgage", "Securities Account", "CD Account",
                      "Online", "CreditCard", "Education_1",
                      "Education_2", "Education_3")]


new__customer <- data.frame(
  Age = 40,
  Experience = 10,
  Income = 84,
  Family = 2,
  CCAvg = 2,
```

```
  Mortgage = 0,
  `Securities Account` = 0,
  `CD Account` = 0,
  Online = 1,
  CreditCard = 1,
  Education_1 = 0,
  Education_2 = 1,
  Education_3 = 0
)

#  k-NN classification with k = 1
knnmodel <- knn(train = predictor, test = new__customer, cl = class_lbl, k = 1)

#classification of the new customer
if (knnmodel == 1) {
  print("The customer is classified as accepting the loan offer.\n")
} else {
  print("The customer is classified as not accepting the loan offer.\n")
}
```

```
## [1] "The customer is classified as not accepting the loan offer.\n"
```

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

- Answer 2. To balance between overfitting and ignoring predictor information in kNN:

Always start with the low k values (e.g., 1, 3) to capture fine-grained patterns, but be cautious of overfitting.

Gradually increase k until you find a sweet spot where the model generalizes well without oversimplifying. Avoid very high k values, as they may lead to underfitting by oversmoothing the data.

Validate model performance using techniques like cross-validation and choose the k value that maximizes performance metrics on a validation set. For example:

- Example R code:

```
k_value_s <- seq(1, 20, by = 2)


training_control <- trainControl(method = "cv", number = 10)
suppressWarnings({
#finding the optimal k
knn_grid <- expand.grid(k = k_value_s)
knnmodel <- train(
  x = predictor,
  y = class_lbl,
  method = "knn",
  trControl = training_control,
  tuneGrid = knn_grid
)

print(knnmodel)})
```

```
## k-Nearest Neighbors
##
## 5000 samples
##   13 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4500, 4500, 4500, 4500, 4500, 4500, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    1  0.3040282  0.1892570  0.0930000
##    3  0.2546233  0.2894622  0.1000500
##    5  0.2495083  0.2961805  0.1057200
##    7  0.2475955  0.3000724  0.1096107
##    9  0.2468193  0.3010845  0.1121107
##   11  0.2471921  0.2990264  0.1141379
##   13  0.2463867  0.3020733  0.1150088
##   15  0.2472880  0.2969477  0.1165258
##   17  0.2475432  0.2956257  0.1171098
##   19  0.2473827  0.2962592  0.1175822
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 13.
```

Question 3: Show the confusion matrix for the validation data that results from using the best k?

- Answer 3. The code performs cross-validation to find the best k value for k-nearest neighbors (k-NN) classification. It then trains the final k-NN model with the optimal k and evaluates its performance on validation data using a confusion matrix. The confusion matrix shows the model's accuracy, precision, recall, and F1 score.

```r
library(caret)

#k values to try
k_value_s <- seq(1, 20, by = 2)

training_control <- trainControl(method = "cv", number = 10)

suppressWarnings({
knn_grid <- expand.grid(k = k_value_s)
knnmodel <- train(
  x = predictor,
  y = class_lbl,
  method = "knn",
  trControl = training_control,
  tuneGrid = knn_grid
)
})
```

```r
optimal_k <- knnmodel$bestTune$k

final_knnmodel <- knn(train = as.matrix(predictor),
                      test = as.matrix(predictor),
                      cl = class_lbl,
                      k = optimal_k)


prediction <- as.factor(final_knnmodel)

class_lbl <- as.factor(class_lbl)

confusion_matrix <- confusionMatrix(prediction, class_lbl)


print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4451  321
##          1   69  159
##
##                Accuracy : 0.922
##                  95% CI : (0.9142, 0.9293)
##     No Information Rate : 0.904
##     P-Value [Acc > NIR] : 4.938e-06
##
##                   Kappa : 0.4128
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9847
##             Specificity : 0.3312
##          Pos Pred Value : 0.9327
##          Neg Pred Value : 0.6974
##              Prevalence : 0.9040
##          Detection Rate : 0.8902
##    Detection Prevalence : 0.9544
##       Balanced Accuracy : 0.6580
##
##        'Positive' Class : 0
##
```

Question 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?

- Answer 4. Considering 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.

- The provided code segment predicts whether a new customer will accept or reject a loan offer using a k-NN model trained on existing customer data.

```
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,
  `Credit Card` = 1
)

# Train the k-NN model with the optimal k value
final_knnmodel <- knn(train = as.matrix(predictor),
                      test = as.matrix(new__customer),
                      cl = class_lbl,
                      k = optimal_k)


new__customer_prediction <- as.factor(final_knnmodel)

#prediction
print(new__customer_prediction)
```

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

Question 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?

- Answer 5. This below R code splits the data into training, validation, and test sets, applies the k-NN method with the previously determined optimal k value, and compares the confusion matrices of the training, validation, and test sets. Differences in the confusion matrices indicate variations in the model's performance across the different datasets.

```
set.seed(123)

# percentage for each set
training_perc <- 0.5
validation_per <- 0.3
test_per <- 0.2
```

```r
train_indices <- createDataPartition(class_lbl, p = training_perc, list = FALSE)
validation_test_data <- data[-train_indices, ]


validation_per_adjusted <- validation_per / (validation_per + test_per)
test_per_adjusted <- test_per / (validation_per + test_per)


validation_indices <- createDataPartition(
  validation_test_data$"Personal Loan",
  p = validation_per_adjusted,
  list = FALSE)
validation_data <- validation_test_data[validation_indices, ]
test_data <- validation_test_data[-validation_indices, ]


final_knnmodel <- knn(train = predictor[train_indices, ],
                      test = predictor[train_indices, ],
                      cl = class_lbl[train_indices],
                      k = optimal_k)


train_prediction <- as.factor(final_knnmodel)


train_confusion_mat <- confusionMatrix(train_prediction, class_lbl[train_indices])


validation_pred <- knn(train = predictor[train_indices, ],
                           test = predictor[validation_indices, ],
                           cl = class_lbl[train_indices], k = optimal_k)


validation_confusion_mat <- confusionMatrix(
  validation_pred,
  class_lbl[validation_indices])


test_pred <- knn(train = predictor[train_indices, ],
                      test = predictor[-c(train_indices,
                                          validation_indices),],
                      cl = class_lbl[train_indices],
                      k = optimal_k)


test_confusion_mat <- confusionMatrix(
  test_pred,
  class_lbl[-c(train_indices, validation_indices)])


print("Training Confusion Matrix:")
```

```
## [1] "Training Confusion Matrix:"
```

```r
print(train_confusion_mat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2225  169
##          1   35   71
##
##                Accuracy : 0.9184
##                  95% CI : (0.907, 0.9288)
##     No Information Rate : 0.904
##     P-Value [Acc > NIR] : 0.006996
##
##                   Kappa : 0.3736
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9845
##             Specificity : 0.2958
##          Pos Pred Value : 0.9294
##          Neg Pred Value : 0.6698
##              Prevalence : 0.9040
##          Detection Rate : 0.8900
##    Detection Prevalence : 0.9576
##       Balanced Accuracy : 0.6402
##
##        'Positive' Class : 0
##
```

```r
print("Validation Confusion Matrix:")
```

```
## [1] "Validation Confusion Matrix:"
```

```r
print(validation_confusion_mat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1319  117
##          1   20   44
##
##                Accuracy : 0.9087
##                  95% CI : (0.8929, 0.9228)
##     No Information Rate : 0.8927
##     P-Value [Acc > NIR] : 0.02309
##
##                   Kappa : 0.3515
##
##  Mcnemar's Test P-Value : 2.367e-16
##
```

```
##             Sensitivity : 0.9851
##             Specificity : 0.2733
##          Pos Pred Value : 0.9185
##          Neg Pred Value : 0.6875
##              Prevalence : 0.8927
##          Detection Rate : 0.8793
##    Detection Prevalence : 0.9573
##       Balanced Accuracy : 0.6292
##
##        'Positive' Class : 0
##
```

```
print("Test Confusion Matrix:")
```

```
## [1] "Test Confusion Matrix:"
```

```
print(test_confusion_mat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1581  125
##          1   39   40
##
##                Accuracy : 0.9081
##                  95% CI : (0.8938, 0.9211)
##     No Information Rate : 0.9076
##     P-Value [Acc > NIR] : 0.4881
##
##                   Kappa : 0.2851
##
##  Mcnemar's Test P-Value : 3.193e-11
##
##             Sensitivity : 0.9759
##             Specificity : 0.2424
##          Pos Pred Value : 0.9267
##          Neg Pred Value : 0.5063
##              Prevalence : 0.9076
##          Detection Rate : 0.8857
##    Detection Prevalence : 0.9557
##       Balanced Accuracy : 0.6092
##
##        'Positive' Class : 0
##
```

Differences and their reasons.

- Training set:- shows the best performance as the model is trained on this data. It exhibits low error rates and high accuracy.

- Validation set:- Shows slightly lower performance compared to the training set as the model is evaluated on unseen data. It helps assessing the model's generalization ability.
- Test set:- Provides a final assessment of the model's generalization ability. Performance should ideally be similar to or slightly worse than the validation set. Significant discrepancies may indicate the issue like overfitting.

Differences in the performance across these sets helps to evaluate the model's ability to generalize the new data. Consistent performance indicates a robust model, while discrepancies may signal potential issues that need addressing.

Thank You !