Customer Churn Prediction

Asım Akçay

5/27/23

TASK1 The task is to solve the "Customer Loss Estimation" problem. To predict churn, we need to build a model using the features in the dataset. The goal is to accurately predict customers' churn status.

```
library(readr)
column_types <- cols(
  customer_id = col_double(),
  credit_score = col_double(),
  age = col_double(),
  tenure = col_double(),
  balance = col_double(),
  products_number = col_double(),
  credit_card = col_double(),
  active_member = col_double(),
  estimated_salary = col_double(),
  country = col_character(),
  gender = col_character(),
  churn = col_double()
)

veri <- read_csv("Bank Customer Churn Prediction.csv")
```

```
Rows: 10000 Columns: 12
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (2): country, gender
dbl (10): customer_id, credit_score, age, tenure, balance, products_number, ...

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.
```

```r
boyut <- dim(veri)
satir_sayisi <- boyut[1]
sutun_sayisi <- boyut[2]
print(paste("Satır Sayısı:", satir_sayisi))
```

```
[1] "Satır Sayısı: 10000"
```

```r
print(paste("Sütun Sayısı:", sutun_sayisi))
```

```
[1] "Sütun Sayısı: 12"
```

TASK2 I checked with the "dim" function, which specifies how many rows and columns the dataset consists of in size (dimension). I used the "str" function to determine the variable types. Additionally, I got some statistics (mean, min, max, etc.) of the variables in the dataset with the "summary" function.

```r
degerler <- sapply(veri, class)
print(degerler)
```

```
    customer_id     credit_score          country           gender
      "numeric"        "numeric"      "character"      "character"
            age           tenure          balance  products_number
      "numeric"        "numeric"        "numeric"        "numeric"
    credit_card    active_member estimated_salary            churn
      "numeric"        "numeric"        "numeric"        "numeric"
```

```r
str(veri)
```

```
spc_tbl_ [10,000 x 12] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ customer_id     : num [1:10000] 15634602 15647311 15619304 15701354 15737888 ...
 $ credit_score    : num [1:10000] 619 608 502 699 850 645 822 376 501 684 ...
 $ country         : chr [1:10000] "France" "Spain" "France" "France" ...
 $ gender          : chr [1:10000] "Female" "Female" "Female" "Female" ...
 $ age             : num [1:10000] 42 41 42 39 43 44 50 29 44 27 ...
 $ tenure          : num [1:10000] 2 1 8 1 2 8 7 4 4 2 ...
 $ balance         : num [1:10000] 0 83808 159661 0 125511 ...
```

```
 $ products_number : num [1:10000] 1 1 3 2 1 2 2 4 2 1 ...
 $ credit_card     : num [1:10000] 1 0 1 0 1 1 1 1 0 1 ...
 $ active_member   : num [1:10000] 1 1 0 0 1 0 1 0 1 1 ...
 $ estimated_salary: num [1:10000] 101349 112543 113932 93827 79084 ...
 $ churn           : num [1:10000] 1 0 1 0 0 1 0 1 0 0 ...
 - attr(*, "spec")=
  .. cols(
  ..   customer_id = col_double(),
  ..   credit_score = col_double(),
  ..   country = col_character(),
  ..   gender = col_character(),
  ..   age = col_double(),
  ..   tenure = col_double(),
  ..   balance = col_double(),
  ..   products_number = col_double(),
  ..   credit_card = col_double(),
  ..   active_member = col_double(),
  ..   estimated_salary = col_double(),
  ..   churn = col_double()
  .. )
 - attr(*, "problems")=<externalptr>
```

```
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.4.0      v dplyr   1.0.10
v tibble  3.1.8      v stringr 1.4.1
v tidyr   1.2.1      v forcats 0.5.2
v purrr   0.3.5
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

```
veri <- veri %>%
  mutate_if(is.numeric, ~if_else(is.na(.), mean(., na.rm = TRUE), .))
```

TASK3 I need to train logistic regression model, decision tree and random forest models. I trained these models using the "caret" package. I trained the logistic regression model with the "train" function, using the "glm" method. I trained the decision tree model with the "train"

function, using the “rpart” method. It can train the random forest model with the “train” function, using the “rf” method.

```
sütunlar <- colnames(veri)
print(sütunlar)
```

```
 [1] "customer_id"      "credit_score"     "country"          "gender"
 [5] "age"              "tenure"           "balance"          "products_number"
 [9] "credit_card"      "active_member"    "estimated_salary" "churn"
```

```
#install.packages("caret")
#install.packages("dplyr")
#install.packages("randomForest")
#install.packages("rpart")
library(caret)
```

```
Loading required package: lattice
```

```

Attaching package: 'caret'
```

```
The following object is masked from 'package:purrr':

    lift
```

```
library(randomForest)
```

```
randomForest 4.7-1.1
```

```
Type rfNews() to see new features/changes/bug fixes.
```

```

Attaching package: 'randomForest'
```

```
The following object is masked from 'package:dplyr':

    combine
```

```
The following object is masked from 'package:ggplot2':

    margin
```

```
library(rpart)
library(dplyr)
```

```
veri$churn <- as.factor(veri$churn)
```

```
set.seed(123)
trainIndex <- createDataPartition(veri$churn, p = 0.7, list = FALSE)
trainData <- veri[trainIndex, ]
testData <- veri[-trainIndex, ]
```

```
logistic_model <- train(churn ~ ., data = trainData, method = "glm", family = "binomial")
```

Train the Decision Tree model

```
decision_tree_model <- train(churn ~ ., data = trainData, method = "rpart")
```

Train the Random Forest model

```
random_forest_model <- train(churn ~ ., data = trainData, method = "rf")
```

Evaluate model performance

```
logistic_pred <- predict(logistic_model, newdata = testData)
decision_tree_pred <- predict(decision_tree_model, newdata = testData)
random_forest_pred <- predict(random_forest_model, newdata = testData)
```

Performans metriklerini hesaplama

```
logistic_acc <- confusionMatrix(logistic_pred, testData$churn)$overall["Accuracy"]
decision_tree_acc <- confusionMatrix(decision_tree_pred, testData$churn)$overall["Accuracy
random_forest_acc <- confusionMatrix(random_forest_pred, testData$churn)$overall["Accuracy
```

```
model_comparison <- data.frame(Model = c("Logistic Regression", "Decision Tree", "Random F
                               Accuracy = c(logistic_acc, decision_tree_acc, random_forest
```

```
print(model_comparison)
```

```
               Model  Accuracy
1 Logistic Regression 0.8212738
2       Decision Tree 0.8279426
3       Random Forest 0.8639547
```

TASK4 To compare the performance of models, you must use a benchmark independent of the cutoff value. For example, the measure of accuracy can be used. The accuracy of the model can be calculated with the "confusionMatrix" function. Evaluate the performance of the models by comparing the calculated accuracy values.

```
library(caret)
train_indices <- createDataPartition(veri$churn, p = 0.7, list = FALSE)

train_data <- veri[train_indices, ]

test_data <- veri[-train_indices, ]
```

Logistic Regression

```
lr_formula <- churn ~ .
lr_model <- glm(lr_formula, data = train_data, family = "binomial")
```

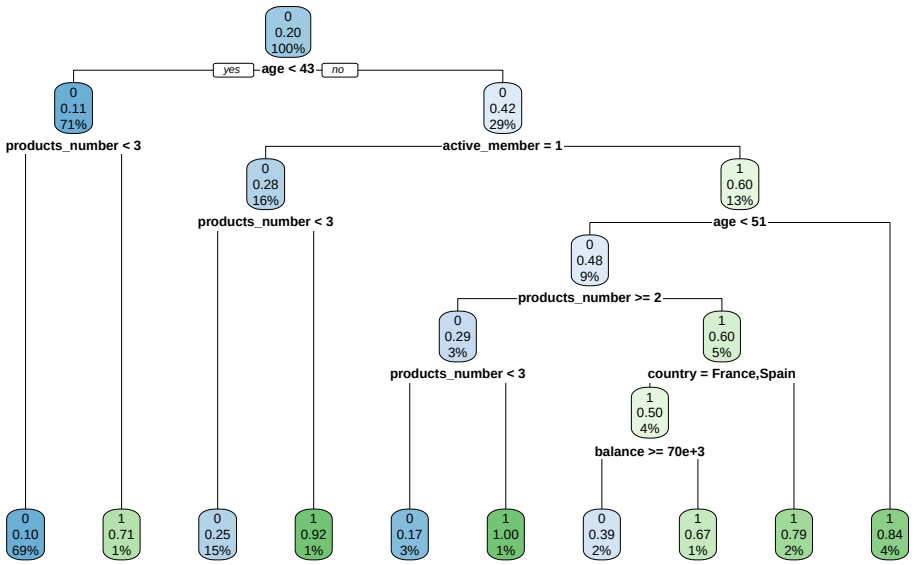
Decision Tree

```
dt_formula <- churn ~ .
dt_model <- rpart(dt_formula, data = train_data, method = "class")
```

```
# install.packages("rpart.plot")
library(rpart)
library(rpart.plot)

# Decision Tree
dt_formula <- churn ~ .
dt_model <- rpart(dt_formula, data = train_data, method = "class")

# Decision Tree Plot
rpart.plot(dt_model)
```



Random Forest

```
rf_formula <- churn ~ .
rf_model <- randomForest(rf_formula, data = train_data, ntree = 100)
```

```r
rf_predictions <- predict(rf_model, newdata = test_data)
```

```r
rf_predictions <- as.numeric(as.character(rf_predictions))
```

```r
# Confusion matrix
confusion_matrix <- table(Actual = test_data$churn, Predicted = rf_predictions)
print(confusion_matrix)
```

```
      Predicted
Actual    0    1
     0 2311   77
     1  349  262
```

```r
#Accuracy
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", accuracy))
```

```
[1] "Accuracy: 0.857952650883628"
```

```r
#Precision
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
print(paste("Precision:", precision))
```

```
[1] "Precision: 0.772861356932153"
```

```r
#Recall
recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
print(paste("Recall:", recall))
```

```
[1] "Recall: 0.428805237315876"
```

```r
f1_score <- 2 * precision * recall / (precision + recall)
print(paste("F1 Score:", f1_score))
```

```
[1] "F1 Score: 0.551578947368421"
```

```r
test_data$churn <- factor(test_data$churn, levels = c(0, 1))

#install.packages("pROC")
library(pROC)
```

```
Type 'citation("pROC")' for a citation.
```

```
Attaching package: 'pROC'
```

```
The following objects are masked from 'package:stats':

    cov, smooth, var
```

```r
roc <- roc(test_data$churn, rf_predictions)
```

```
Setting levels: control = 0, case = 1
```

```
Setting direction: controls < cases
```

```r
auc_roc <- auc(roc)
print(paste("AUC-ROC Score:", auc_roc))
```

```
[1] "AUC-ROC Score: 0.698280340600986"
```

```r
confusion_matrix <- table(test_data$churn, rf_predictions > 0.5)
print(confusion_matrix)
```

```
   FALSE TRUE
  0  2311   77
  1   349  262
```

```r
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
f1_score <- 2 * (precision * recall) / (precision + recall)

print(paste("Accuracy:", accuracy))
```

```
[1] "Accuracy: 0.857952650883628"
```

```r
print(paste("Precision:", precision))
```

```
[1] "Precision: 0.772861356932153"
```

```r
print(paste("Recall:", recall))
```

```
[1] "Recall: 0.428805237315876"
```

```r
print(paste("F1 Score:", f1_score))
```

```
[1] "F1 Score: 0.551578947368421"
```

TASK5

```r
country_levels <- unique(train_data$country)
train_data$country <- as.integer(factor(train_data$country, levels = country_levels))
test_data$country <- as.integer(factor(test_data$country, levels = country_levels))
```

```r
train_data$gender <- factor(train_data$gender)
test_data$gender <- factor(test_data$gender)
```

```r
library(gbm)
```

```
Loaded gbm 2.1.8.1
```

```
gbm_model <- gbm(churn ~ ., data = train_data, distribution = "bernoulli", n.trees = 100,
```

```
gbm_predictions <- predict(gbm_model, newdata = test_data, type = "response")
```

```
Using 100 trees...
```

```
confusion_matrix <- table(Actual = test_data$churn, Predicted = ifelse(gbm_predictions > 0
print(confusion_matrix)
```

```
< table of extent 2 x 0 >
```

```
print(table(Actual = test_data$churn, Predicted = rf_predictions))
```

```
      Predicted
Actual    0    1
     0 2311   77
     1  349  262
```

```
correct_predictions <- sum(test_data$churn == rf_predictions)
print(paste("Correct predictions:", correct_predictions))
```

```
[1] "Correct predictions: 2573"
```

```
total_samples <- length(test_data$churn)
print(paste("Total samples:", total_samples))
```

```
[1] "Total samples: 2999"
```