Obtaining Logistic Regression, Decision Tree and Random Forest Models and Making Appropriate Comparisons

Rümeysa Kurt

2023-05-22

Calling the data set.

The dataset is taken from Kaggle.

```
library(readr)
Bank_Customer_Churn_Prediction <- read_csv("Bank Customer Churn Prediction.csv")
```

1.Detail your task with the problem, features, and target.

Problem

This report includes customer data of account holders at ABC Multinational Bank and our purpose here is to estimate customer churn.

Features

1. customer_id: unused variable.
2. credit_score: used as input.
3. country: used as input.
4. gender: used as input.
5. age: used as input.
6. tenure: used as input.

7. balance: used as input.
8. products_number: used as input.
9. credit_card: used as input.
10. active_member: used as input.
11. estimated_salary: used as input.
12. churn: used as the target. 1 if the client has left the bank during some period or 0 if he/she has not.

Target

churn: used as the target. 1 if the client has left the bank during some period or 0 if he/she has not.

2. Describe the dataset in your task in terms of the dimension, variable type, and some other that you want to add.

In this data set we don't have any NA observations.

```
sum(is.na(Bank_Customer_Churn_Prediction))
```

```
[1] 0
```

In here, customer_id is omitted from the dataset because it is an unused feature. Therefore, our new data set contains 11 features.

```
Bank_Customer_Churn_Prediction <-Bank_Customer_Churn_Prediction[,-1]
```

The str() function is used to browse the dataset. In this way, the number of observations in the dataset, the number of features, the type of features, and the dimension of the dataset were reached.

The size of the data set is [10,000 $\times$ 11].

credit_score, age, tenure, balance, products_number, credit_card, active_member, estimated_salary and churn features are numeric.country and gender features are categorical.

```
str(Bank_Customer_Churn_Prediction)
```

```
tibble [10,000 x 11] (S3: tbl_df/tbl/data.frame)
 $ 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 ...
```

3.Train a logistic regression model, a decision tree, and a random forest model.

Splitting The Data Set

The sample() function is used to split the dataset into test and train sets. Using the index object, the observations are set to train and test.

```
set.seed(123) # for reproducibility
index <- sample(1 : nrow(Bank_Customer_Churn_Prediction),
round(nrow(Bank_Customer_Churn_Prediction) * 0.80))
train <- Bank_Customer_Churn_Prediction[index, ]
test <- Bank_Customer_Churn_Prediction[-index, ]
```

The dim() function is used to see how many rows and how many columns are in data sets separated into test and train.

```
dim(train)
```

```
[1] 8000   11
```

```
dim(test)
```

```
[1] 2000   11
```

While there are 2000 observations and 11 features in the test part, there are 8000 observations and 11 features in the train part.

Train a Logistic Regression Model

The "glm()" function is used to train a logistic regression model. "Y" is the target variable that deals with features to be predicted and "x"s are features that use the information to predict the target variable. family=binomial defines the target property. We have two levels(or outcome, possibility) here so it's coded as binomial.

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

```
Call:  glm(formula = churn ~ ., family = "binomial", data = train)

Coefficients:
     (Intercept)      credit_score    countryGermany      countrySpain
      -3.406e+00        -6.923e-04         7.884e-01         3.842e-02
      genderMale               age            tenure           balance
      -5.475e-01         7.399e-02        -1.979e-02         2.592e-06
 products_number       credit_card     active_member  estimated_salary
      -1.036e-01        -3.221e-02        -1.090e+00         5.595e-07

Degrees of Freedom: 7999 Total (i.e. Null);  7988 Residual
Null Deviance:      8100
Residual Deviance: 6834     AIC: 6858
```

If you want to know more details about the model, you can use the "Summary()" function.

```
summary(model)
```

```
Call:
glm(formula = churn ~ ., family = "binomial", data = train)

Coefficients:
                   Estimate Std. Error z value Pr(>|z|)
(Intercept)      -3.406e+00  2.750e-01 -12.385  < 2e-16 ***
credit_score     -6.923e-04  3.138e-04  -2.206   0.0274 *
countryGermany    7.884e-01  7.585e-02  10.395  < 2e-16 ***
countrySpain      3.842e-02  7.883e-02   0.487   0.6260
genderMale       -5.475e-01  6.098e-02  -8.978  < 2e-16 ***
age               7.399e-02  2.905e-03  25.468  < 2e-16 ***
```

```
tenure           -1.979e-02  1.049e-02  -1.887   0.0592 .
balance           2.592e-06  5.738e-07   4.517 6.27e-06 ***
products_number  -1.036e-01  5.289e-02  -1.959   0.0501 .
credit_card      -3.221e-02  6.648e-02  -0.485   0.6280
active_member    -1.090e+00  6.471e-02 -16.843  < 2e-16 ***
estimated_salary  5.595e-07  5.322e-07   1.051   0.2931
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 8099.8  on 7999  degrees of freedom
Residual deviance: 6834.2  on 7988  degrees of freedom
AIC: 6858.2

Number of Fisher Scoring iterations: 5
```

If we are asked to comment, we can look at the p-values to see if the variables are significant. Here, when interpreting the levels of categorical variables, we can say that if even one of them is significant, the others are also significant.

Train a decision tree.

```
install.packages("tidymodels") # training models
install.packages("rpart.plot") # visualizing decision tree
library(tidymodels)
library(rpart.plot)
```

Model specification:

```
bank_customer_churn_prediction_split <- initial_split(data = Bank_Customer_Churn_Predictio
                                       prop = 0.80)
dt_train <- bank_customer_churn_prediction_split |> training()
dt_test  <- bank_customer_churn_prediction_split |> testing()
```

Defining model specification:

- `type`: model type, e.g. regression, decision tree or etc.
- `engine`: different R packages have engines

- `mode`: learning task, e.g. regression or classification

```
dt_model <- decision_tree() |>
  set_engine("rpart") |>
  set_mode("regression")
```

Model training:

```
dt_bank_customer <- dt_model |>
  fit(dt_train$churn ~., data = dt_train)
dt_bank_customer
```

```
parsnip model object

n= 8000

node), split, n, deviance, yval
      * denotes terminal node

 1) root 8000 1299.664000 0.20412500
   2) age< 42.5 5660  583.805500 0.11678450
     4) products_number< 2.5 5546  516.177400 0.10385860
       8) products_number>=1.5 2784  114.827600 0.04310345 *
       9) products_number< 1.5 2762  380.715400 0.16509780 *
     5) products_number>=2.5 114   21.622810 0.74561400 *
   3) age>=42.5 2340  568.246200 0.41538460
     6) active_member>=0.5 1294  255.790600 0.27125190
      12) products_number< 2.5 1242  229.079700 0.24396140
        24) products_number>=1.5 541   57.190390 0.12014790 *
        25) products_number< 1.5 701  157.195400 0.33951500 *
      13) products_number>=2.5 52    3.692308 0.92307690 *
     7) active_member< 0.5 1046  252.318400 0.59369020
      14) age< 50.5 717  178.772700 0.47419800
        28) country=France,Spain 494  116.690300 0.38259110 *
        29) country=Germany 223   48.753360 0.67713000 *
      15) age>=50.5 329   40.996960 0.85410330 *
```

n shows us number of observation the data set.

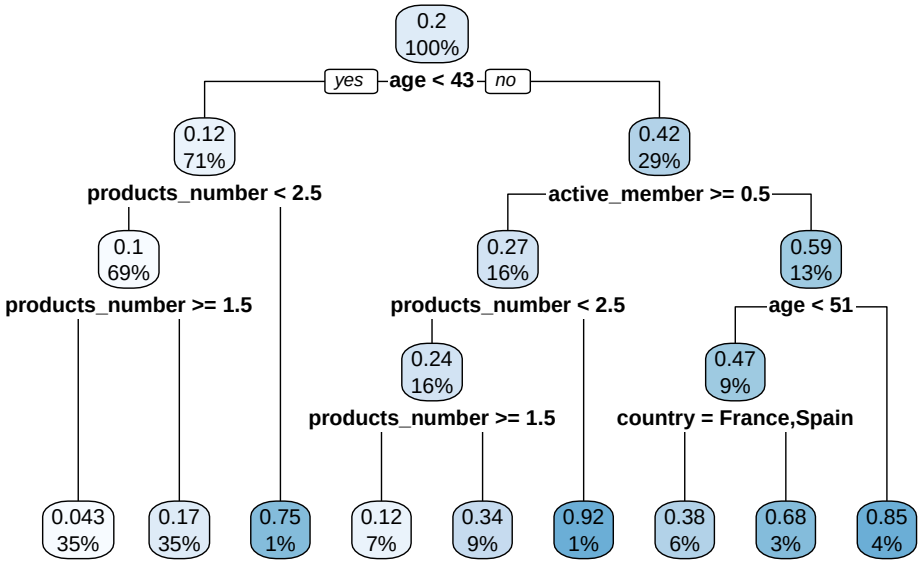
In here we can see whole splitting rules of the decision tree.

"1)" … 0.20412500 shows us root mean square error.

"2)", "3)" are the branch of root node.

Visualizing the decision tree:

```
rpart.plot(dt_bank_customer$fit)
```



age<43 is the root node, products_number<2.5 and active_member>=0.5 is internal/sub node,

"0.2" shows us the average of observations.

"100%" show us the percentage of observations.

This decision tree has 9 leaf nodes.

Make predictions by using the trained model:

```
bank_customer_churn_predictions <- dt_bank_customer |>
  predict(new_data = dt_test)
bank_customer_churn_predictions
```

```
# A tibble: 2,000 x 1
    .pred
    <dbl>
 1 0.746
 2 0.0431
 3 0.0431
 4 0.120
 5 0.0431
 6 0.165
 7 0.165
 8 0.165
 9 0.165
10 0.165
# i 1,990 more rows
```

Train a random forest.

Random forests model can be trained in same manner. Do not forget to use `set.seed()` in training phase also, because both of the models consists the random process!

```
library(randomForest)
library(ranger)
library(yardstick)
```

```
library(rsample)
set.seed(123)
bank_customer_churn_split <- initial_split(data = Bank_Customer_Churn_Prediction,
                                           prop = 0.80)
bcc_train <- bank_customer_churn_split |> training()
bcc_test  <- bank_customer_churn_split |> testing()
```

```
set.seed(123)
trained_rf <- ranger(churn ~ .,
                     data = bcc_train)
```

Let's see the ouput of the model object:

```
trained_rf
```

```
Ranger result

Call:
 ranger(churn ~ ., data = bcc_train)

Type:                             Regression
Number of trees:                  500
Sample size:                      8000
Number of independent variables:  10
Mtry:                             3
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.1054434
R squared (OOB):                  0.3513261
```

4.Compare the performance of the models with a criterion which is independent from the cutoff value.

Measuring Model Performance: Logistic Regression

In here we use additional argument.(type=response) When we set the type as a response we can calculate the probability of the target feature.

```r
predicted_probs <- predict(model, test[,-11], type = "response")
head(predicted_probs)
```

```
        1         2         3         4         5         6
0.24512258 0.24506937 0.25785568 0.11421165 0.04111743 0.04277063
```

When calculating the probability of the target variable, it is actual values were excluded from the test set.

```r
predicted_classes <- ifelse(predicted_probs > 0.5, 1, 0)
head(predicted_classes)
```

```
1 2 3 4 5 6
0 0 0 0 0 0
```

Here, the result is not a probability value because we convert the probabilities into classes or labels. If the values of the estimated probabilities are greater than 0.5, they will be equal to 1, and if they are smaller, they will be equal to 0. (predicted_probs $>$ 0.5 is a condition)

```
TP <- sum(predicted_classes[which(test$churn == "1")] == 1)
FP <- sum(predicted_classes[which(test$churn == "1")] == 0)
TN <- sum(predicted_classes[which(test$churn == "0")] == 0)
FN <- sum(predicted_classes[which(test$churn == "0")] == 1)
```

```
recall <- TP / (TP + FN)
specificity <- TN / (TN + FP)
precision <- TP / (TP + FP)
accuracy <- (TN + TP) / (TP + FP + TN + FN)
```

```
recall
```

```
[1] 0.5620915
```

```
specificity
```

```
[1] 0.8283703
```

```
precision
```

```
[1] 0.2133995
```

```
accuracy
```

```
[1] 0.808
```

According to the values of performance metrics, the model classifies the observations with 0.80 accuracy. Its precision is 0.21 means that the model classify only 21% of the positive class, which is canceled reservations in the problem, correctly.

```
table(train$churn) / dim(train)[1]
```

```
      0       1
0.79575 0.20425
```

In this step, we can say that 79% of the variables in the train set belong to the 0 class (customers who do not leave the bank) and 20% belong to the 1st class (customers who leave the bank). This indicates that there is an imbalance between classes.

There is a signal for imbalancedness problem here. This may cause the model learn less from the minority class and normally the model couldn't classify correctly the observation in majority class.

The following steps were followed to obtain the ROC curve in R.

With the explainer object created above, the model_performance() function from the {DALEX} package is used to plot ROC curves and others.

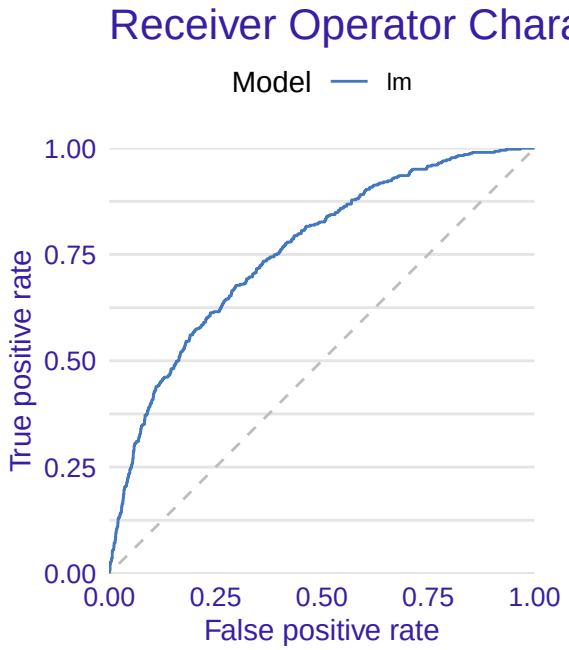
The ROC curve tells us how good the model is at predicting. It is one of the metrics commonly used to evaluate the performance of machine learning algorithms, especially in situations with imbalanced data sets.

If the ROC curve is close to the reference line it means that model is a random model and it does not perform well.

```r
library(dplyr)
library(DALEX)
```

```r
explain_lr <- explain(model = model,          # trained model
data = test[, -11],                           # test set without target
y = test$churn == "1",                        # observed values of target
                                              # with reference class
type = "classification",                      # type of task
verbose = FALSE)                              # remove some messages
```

```r
performance_lr <- model_performance(explain_lr)
plot(performance_lr, geom = "roc")
```



Here, the ROC curve is close to the left top, we can say that the model performance is excellent.

```
performance_lr
```

```
Measures for:  classification
recall     : 0.2133995
precision  : 0.5620915
f1         : 0.3093525
accuracy   : 0.808
auc        : 0.7577918

Residuals:
         0%         10%         20%         30%         40%         50%
-0.87267532 -0.33010251 -0.23453307 -0.17147023 -0.13220738 -0.09748351
        60%         70%         80%         90%        100%
-0.06758197 -0.04495876  0.12343130  0.70324206  0.97167062
```

Area under the curve (AUC) is used as a statistics to show the summarization of ROC curve. The performance_lr object is used to calculate the area under the curve (AUC). The higher AUC indicates higher performance. The AUC is between 0 and 1. Here the AUC value is 0.75. This shows us that the performance of the model is quiet good.

Model Performance: Decision Tree

tibble, is a kind of special data frame.

tibble is the newer version of data frame. It works bit faster.

The first column is predicted values of the target variable in test set. The second column is real values of the target variable in test set.

```
bcc_results <- tibble(predicted = bank_customer_churn_predictions$.pred,
                              actual    = dt_test$churn)
bcc_results
```

```
# A tibble: 2,000 x 2
   predicted actual
       <dbl>  <dbl>
 1    0.746       1
 2    0.0431      0
 3    0.0431      0
 4    0.120       0
 5    0.0431      0
 6    0.165       0
 7    0.165       0
 8    0.165       0
 9    0.165       0
10    0.165       0
# i 1,990 more rows
```

Then we can calculate the RMSE of the model by using `rmse()` function with obligatory arguments:

`truth` must be assigned with actual values,

`estimate` must be assigned with predicted values of target variable.

```
bcc_results |> rmse(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       0.337
```

```
  bcc_results |> rsq(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard       0.296
```

Streaming model fitting by `last_fit()` function. It takes a model specification, model formula, and data split object.

```
  bcc_last_fit <- dt_model |>
    last_fit(churn ~., split = bank_customer_churn_prediction_split)
  bcc_last_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits              id               .metrics .notes   .predictions .workflow
  <list>              <chr>            <list>   <list>   <list>       <list>
1 <split [8000/2000]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

Collecting metrics:

```
  bcc_last_fit |> collect_metrics()
```

```
# A tibble: 2 x 4
  .metric .estimator .estimate .config
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard       0.337 Preprocessor1_Model1
2 rsq     standard       0.296 Preprocessor1_Model1
```

Collecting predictions:

```
  bcc_last_fit |> collect_predictions()
```

```
# A tibble: 2,000 x 5
   id                .pred  .row churn .config
   <chr>             <dbl> <int> <dbl> <chr>
 1 train/test split 0.746      8     1 Preprocessor1_Model1
 2 train/test split 0.0431    12     0 Preprocessor1_Model1
 3 train/test split 0.0431    15     0 Preprocessor1_Model1
 4 train/test split 0.120     24     0 Preprocessor1_Model1
 5 train/test split 0.0431    35     0 Preprocessor1_Model1
 6 train/test split 0.165     37     0 Preprocessor1_Model1
 7 train/test split 0.165     39     0 Preprocessor1_Model1
 8 train/test split 0.165     58     0 Preprocessor1_Model1
 9 train/test split 0.165     65     0 Preprocessor1_Model1
10 train/test split 0.165     70     0 Preprocessor1_Model1
# i 1,990 more rows
```