Prediction of Credit Risk with Logistic Regression, Decision Tree and Random Forest

Sahranur İnce

2023-05-21

Calling The Dataset

```
library(readr)
customer_data <- read_csv("customer_data.csv")
```

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

Problem:

We are dealing with the credit risk prediction. We will look for an answer and prediction to our problem with the features in the dataset.

Features:

There are 12 features in this dataset. These are:`id`, `fea_1`, `fea_2`, `fea_3`, `fea_4`, `fea_5`, `fea_6`, `fea_7`, `fea_8`, `fea_9`, `fea_10`, `fea_11`.

Target:

label: Used as the target. 1 if the customer is in high credit risk or 0 if the customer is in low credit risk.

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

Here, id is omitted from the dataset because it is an unused feature. Therefore, our new dataset contains 11 features.

```r
customer_data <- customer_data[,-2]
```

Here, the `str()` function is used to determine dimensions and what the types of features are.

```r
str(customer_data)
```

```
tibble [1,125 x 12] (S3: tbl_df/tbl/data.frame)
 $ label : num [1:1125] 1 0 0 1 0 0 1 1 0 0 ...
 $ fea_1 : num [1:1125] 5 4 7 7 7 6 4 5 5 4 ...
 $ fea_2 : num [1:1125] 1246 1277 1298 1336 NA ...
 $ fea_3 : num [1:1125] 3 1 1 1 2 3 3 3 3 2 ...
 $ fea_4 : num [1:1125] 77000 113000 110000 151000 59000 56000 35000 78000 218000 35000 ...
 $ fea_5 : num [1:1125] 2 2 2 2 2 2 2 2 2 2 ...
 $ fea_6 : num [1:1125] 15 8 11 11 11 6 8 15 15 8 ...
 $ fea_7 : num [1:1125] 5 -1 -1 5 5 -1 9 -1 5 5 ...
 $ fea_8 : num [1:1125] 109 100 101 110 108 100 85 111 112 101 ...
 $ fea_9 : num [1:1125] 5 3 5 3 4 3 5 3 4 3 ...
 $ fea_10: num [1:1125] 151300 341759 72001 60084 450081 ...
 $ fea_11: num [1:1125] 245 207 1 1 197 ...
```

According to the results obtained from the output;

The dimension of this dataset is [1,125 $\times$ 12] matrix.

All features are numeric.

Here, the `sum(is.na())` function is used to check if there are missing observations in the dataset.

```r
sum(is.na(customer_data))
```

```
[1] 149
```

According to the results obtained, there are 149 missing values in the dataset. We need to remove these values from the dataset.

Here, the `na.exclude()` function is used to remove missing values from the dataset.

```
customer_data <- na.exclude(customer_data)
```

So our dataset now consists of 976 observations.

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

Splitting The Data Set

Here, the `sample()` function used to split the data set as test and train set. The `set.seed()` function is used for reproducibility.

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

As a result, there are 781 observation and 12 features in the train set. There are 195 observation and 12 features in the test set.

Here, the `dim()` function is used to show the row and column counts of the test and train sets.

```
dim(train)
```

```
[1] 781  12
```

```
dim(test)
```

```
[1] 195  12
```

Train a Logistic Regression Model

Here, the `glm()` function is used to train a logistic regression model. We used the train data and model formula that we split in the previous step. We used y ~. instead of defining all features you want to input to the model.

```r
logreg_model <- glm(label ~ ., data = train, family = "binomial")
logreg_model
```

```

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

Coefficients:
(Intercept)        fea_1        fea_2        fea_3        fea_4        fea_5
 -1.011e+00    1.680e-01   -1.011e-04    1.631e-01   -6.098e-06    3.397e-02
      fea_6        fea_7        fea_8        fea_9       fea_10       fea_11
  5.251e-04   -2.390e-02   -1.091e-02    2.515e-02    6.546e-07   -1.474e-04

Degrees of Freedom: 780 Total (i.e. Null);  769 Residual
Null Deviance:      764.1
Residual Deviance: 733.5    AIC: 757.5
```

```r
summary(logreg_model)
```

```

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

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.011e+00  2.828e+00  -0.358  0.72057
fea_1        1.680e-01  7.266e-02   2.312  0.02077 *
fea_2       -1.011e-04  2.015e-03  -0.050  0.96000
fea_3        1.631e-01  1.181e-01   1.381  0.16726
fea_4       -6.098e-06  1.906e-06  -3.200  0.00137 **
fea_5        3.397e-02  3.769e-01   0.090  0.92818
fea_6        5.251e-04  3.970e-02   0.013  0.98945
fea_7       -2.390e-02  3.552e-02  -0.673  0.50105
fea_8       -1.091e-02  7.727e-03  -1.412  0.15793
fea_9        2.515e-02  1.093e-01   0.230  0.81800
```

```
fea_10       6.546e-07  7.044e-07   0.929  0.35272
fea_11      -1.474e-04  9.165e-04  -0.161  0.87227
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 764.13  on 780  degrees of freedom
Residual deviance: 733.52  on 769  degrees of freedom
AIC: 757.52

Number of Fisher Scoring iterations: 5
```

In the output above; We see some significant statistics for the features. For example, we can look at the p-values for the significance of the variables.

Train a Decision Tree

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

Data splitting:

```r
set.seed(123)
customer_data_split <- initial_split(data = customer_data, # dataset to split
                                  prop = 0.80)      # proportion of train set
customer_train <- customer_data_split |> training()
customer_test  <- customer_data_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_customer <- dt_model |>
  fit(label ~., data = customer_train)
dt_customer
```

```
parsnip model object

n= 780

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

 1) root 780 120.537200 0.1910256
   2) fea_4>=75500 594  70.680130 0.1380471
     4) fea_1< 6 334  30.538920 0.1017964 *
     5) fea_1>=6 260  39.138460 0.1846154
      10) fea_2>=1205.75 252  35.662700 0.1706349 *
      11) fea_2< 1205.75 8   1.875000 0.6250000 *
   3) fea_4< 75500 186  42.865590 0.3602151
     6) fea_4>=50500 127  26.629920 0.2992126
      12) fea_2< 1240.25 72  11.875000 0.2083333 *
      13) fea_2>=1240.25 55  13.381820 0.4181818
        26) fea_2>=1243.25 35   7.142857 0.2857143 *
        27) fea_2< 1243.25 20   4.550000 0.6500000 *
     7) fea_4< 50500 59  14.745760 0.4915254 *
```

In the above output;

n shows us the number of observation the train set. So, we have 780 observation in the train set.

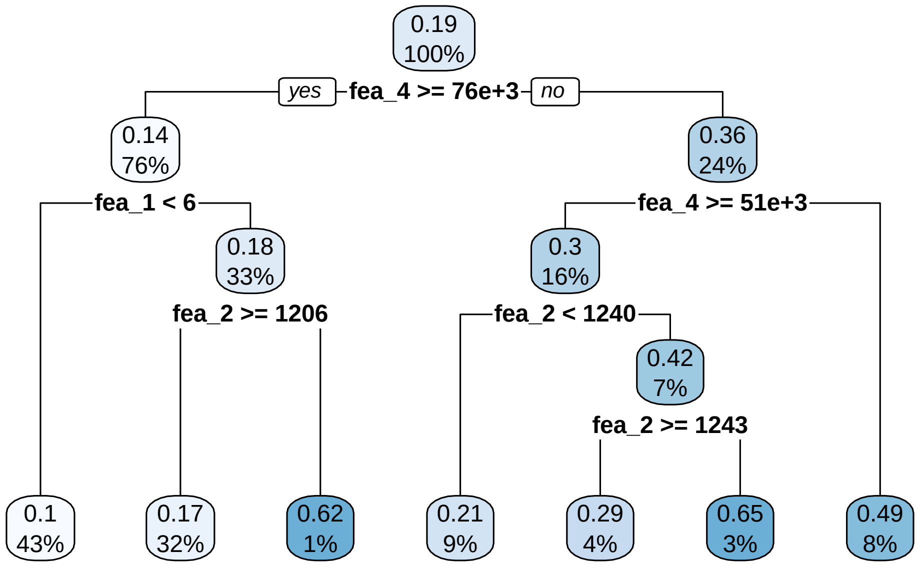
In here, we can see whole splitting rules of the decision tree.

The value(0.1910256) in line `1)` shows us root mean square error.

`2)` and `3)` are the branch of root node.

Visualizing the decision tree:

```r
rpart.plot(dt_customer$fit)
```



A value of 0.19 indicates the mean of the observations at the root node, 100% indicates the percentage of observations.

This decision tree has 5 internal nodes and 7 leaf nodes.

Make predictions by using the trained model:

```r
customer_predictions <- dt_customer |>
  predict(new_data = customer_test)
customer_predictions
```

```
# A tibble: 196 x 1
   .pred
   <dbl>
 1 0.102
 2 0.171
 3 0.102
 4 0.171
 5 0.171
 6 0.492
 7 0.171
 8 0.492
 9 0.286
10 0.171
# i 186 more rows
```

Train a Random Forest Model

```r
install.packages("ranger")
library(ranger)
```

Data Splitting:

```r
library(rsample)
set.seed(123)
customer_rf_split <- initial_split(data = customer_data,
                                   prop = 0.80)
customer_rf_train <- customer_rf_split |> training()
customer_rf_test  <- customer_rf_split |> testing()
```

Training random forests:

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.

```r
set.seed(123)
trained_rf <- ranger(label ~ .,
                     data = customer_rf_train)
```

```r
trained_rf
```

```
Ranger result

Call:
 ranger(label ~ ., data = customer_rf_train)
```

```
Type:                             Regression
Number of trees:                  500
Sample size:                      780
Number of independent variables:  11
Mtry:                             3
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.1537845
R squared (OOB):                  0.006131201
```

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

Measuring Model Performance of Logistic Regression Model

Here, `predicted()` function is used to predict values of the target variable on test set. We used test set for check the model performance because we are interested to train such a model has a good generalizability performance. We should exclude the true values of the target variable from the test set. Therefore, the target variable in the 1th column of the dataset has been exclude.

```r
predicted_probs <- predict(logreg_model, test[,-1], type = "response")
head(predicted_probs)
```

```
         1          2          3          4          5          6 
0.21051278 0.21316570 0.21729251 0.02060788 0.25715915 0.28389719 
```

Here, to measure the model performance, we transform the probabilities to classes. 1 indicates the customer is in high credit risk, while 0 indicates the customer is in low credit risk. In here we used ifelse() function. The first argument condition means that if the values of the predicted_probs vector is higher than the 0.5 assings it as 1, if it is not assigns it as 0.

```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 
```

Since all the values of the predicted__probs vector above are less than 0.5, we got the labels of 0 here.

Here, we calculated the metrics based on the confusion matrix.

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

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

```r
recall
```

```
[1] NaN
```

```r
specificity
```

```
[1] 0.8051282
```

```r
precision
```

```
[1] 0
```

```r
accuracy
```

```
[1] 0.8051282
```

According to the values of performance metrics, the model classifies the observations with 0.805 accuracy. Its precision is 0 means that the model is not correctly classified into the positive class. The fact that the precision value is not good and not close to the accuracy value signals us that there is an imbalance problem here.

```r
table(train$label) / dim(train)[1]
```

```
        0         1 
0.8079385 0.1920615 
```

According to the results 81% of the variables in the train set belong to the 0 class (the customer is in low credit risk.). 19% of the variables belong to class 1 (the customer is in high credit risk). So we can see that, the classes of the target feature is not balanced. In other words, the number of observations in these two classes are not equal or approximate.There is a signal for imbalancedness problem here. This situation, it may cause that the model can learn less from the minority class, so the model could not achieve a satisfying performance on classifying of the minority class.

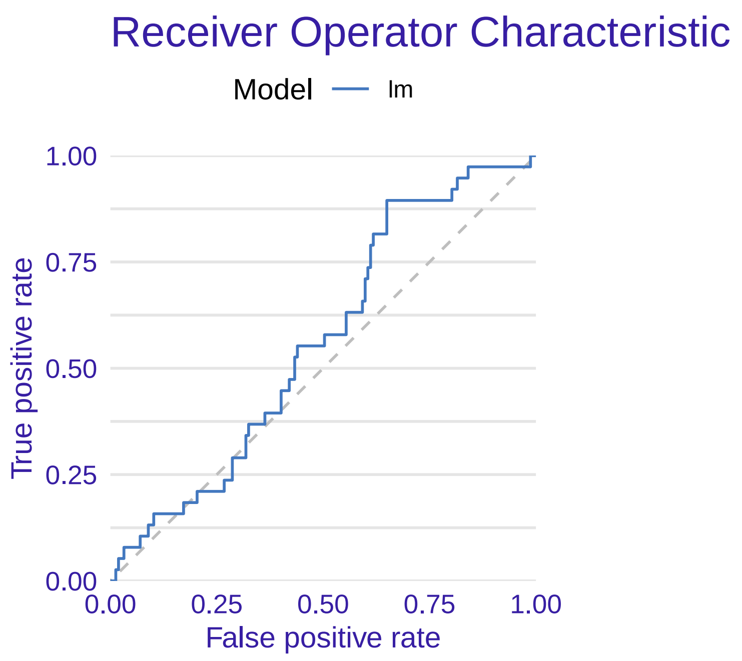
```r
install.packages("dplyr")
install.packages("DALEX")
library(dplyr)
library(DALEX)
```

Here, the `ROC curve` is calculated. The ROC curve is a graph showing the performance of a classification model.

```r
explain_lr <- explain(model = logreg_model,
data = test[, -1],
y = test$label == "1",
type = "classification",
verbose = FALSE)
```

In this graph, there is the FP(False Positive) rate on the x-axis and the TP(True Positive) rate on the y-axis.

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



```
performance_lr
```

```
Measures for:  classification
recall     : 0
precision  : NaN
f1         : NaN
accuracy   : 0.8051282
auc        : 0.5621857

Residuals:
         0%         10%         20%         30%         40%         50% 
-0.37149844 -0.28392989 -0.25613591 -0.22703651 -0.19879956 -0.16211104 
        60%         70%         80%         90%        100% 
-0.14294137 -0.09290538 -0.01989496  0.78599928  0.97142956 
```

AUC stands for area under the ROC Curve. The higher AUC indicates higher performance. AUC takes values between the 0 and 1. According to the results obtained above, we see that the AUC value is 0.56. So, we can say that the performance of the model is not quite good.

Measuring Model Performance of Decision Tree

Here, the `tibble()` function is a kind of special data frame. We used tibble function instead of data frame because it is the newer version of data frame and it works better.

```
customer_results <- tibble(predicted = customer_predictions$.pred,
                           actual    = customer_test$label)
customer_results
```

```
# A tibble: 196 x 2
   predicted actual
       <dbl>  <dbl>
 1     0.102      1
 2     0.171      0
 3     0.102      1
 4     0.171      0
 5     0.171      0
 6     0.492      0
 7     0.171      0
 8     0.492      0
 9     0.286      0
10     0.171      0
# i 186 more rows
```

In the above output;

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

The second column shows us the real values of the target variable in test set.

Here, 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.

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

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       0.421
```

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

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard    0.000462
```

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

```r
customer_last_fit <- dt_model |>
  last_fit(label~., split = customer_data_split)
customer_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 [780/196]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

Collecting metrics:

```r
customer_last_fit |> collect_metrics()
```

```
# A tibble: 2 x 4
  .metric .estimator .estimate .config
  <chr>   <chr>          <dbl> <chr>
1 rmse    standard    0.421    Preprocessor1_Model1
2 rsq     standard    0.000462 Preprocessor1_Model1
```

Collecting predictions:

```r
customer_last_fit |> collect_predictions()
```

```
# A tibble: 196 x 5
   id               .pred  .row label .config
   <chr>            <dbl> <int> <dbl> <chr>
 1 train/test split 0.102     1     1 Preprocessor1_Model1
 2 train/test split 0.171     3     0 Preprocessor1_Model1
 3 train/test split 0.102     7     1 Preprocessor1_Model1
 4 train/test split 0.171     9     0 Preprocessor1_Model1
 5 train/test split 0.171    12     0 Preprocessor1_Model1
 6 train/test split 0.492    22     0 Preprocessor1_Model1
 7 train/test split 0.171    25     0 Preprocessor1_Model1
 8 train/test split 0.492    27     0 Preprocessor1_Model1
 9 train/test split 0.286    28     0 Preprocessor1_Model1
10 train/test split 0.171    32     0 Preprocessor1_Model1
# i 186 more rows
```