IST438-W3-Applications

3/13/23

Supervised Learning: Logistic Regression Models

In this application, we will interest the regression problem under the following sections:

- Training model
- Measuring model performance
- Checking over and underfitting problem

Packages

We need to install `{DALEX}` package to use `titanic` data set in applications. Please use the two-step codes below: (1) install, (2) load the package.

```
# install.packages("DALEX")
# install.packages("caret")
# install.packages("ROCR")
library(DALEX)
library(caret)
library(ROCR)
```

The first line is `hashtag`ed to faster this process. Do not forget to un-`hashtag` it in your first run. Because any line, which is hashtaged, is not run in R. It is turned the command line!

Dataset

We use the `titanic` data set from `{DALEX}` package. The task is to predict the survival probability of the passangers in Titanic. The data set contains several informations about the passenger and their survive status.

```
# calling titanic from {DALEX} package
data(titanic)
```

You can use `str()` function to take a look data set. It returns a data frame consists information about the data set such as:

- number of observation
- number of features (variables)
- name of features
- type of features
- a few observations of features

```
# Take a look to dataset
str(titanic)
```

```
'data.frame':   2207 obs. of  9 variables:
 $ gender  : Factor w/ 2 levels "female","male": 2 2 2 1 1 2 2 1 2 2 ...
 $ age     : num  42 13 16 39 16 25 30 28 27 20 ...
 $ class   : Factor w/ 7 levels "1st","2nd","3rd",..: 3 3 3 3 3 3 2 2 3 3 ...
 $ embarked: Factor w/ 4 levels "Belfast","Cherbourg",..: 4 4 4 4 4 4 2 2 2 4 ...
 $ country : Factor w/ 48 levels "Argentina","Australia",..: 44 44 44 15 30 44 17 17 26 16 .
 $ fare    : num  7.11 20.05 20.05 20.05 7.13 ...
 $ sibsp   : num  0 0 1 1 0 0 1 1 0 0 ...
 $ parch   : num  0 2 1 1 0 0 0 0 0 0 ...
 $ survived: Factor w/ 2 levels "no","yes": 1 1 1 2 2 2 1 2 2 2 ...
```

Remove `country` feature from the data set. It is a problematic categorical feature in model training, because it has many classes.

```
titanic <- titanic[, -5]
```

Remove missing observations in the dataset or you can impute them but if the number of observations is enough, to remove missing observations is easier way to handle missing data. It is necessary because when you split data set, some of the classes may not be seen in the train set then the model can not learn anything about these classes. Thus, it is not possible

to predict the value of target feature with the predictors (features) that have these unseen classes.

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

Task: Classification

In this application, we try to train a logistic regression model on the `titanic` data set to predict the survive probability of the passangers.

Step 1 - Splitting the data set

We can use `sample()` function to split the data set as `test` and `train` set. Do not forget to set a seed to reproduce this process in future. It is important to get same observations in each run of these codes.

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

In the codes above, we get a sample from a vector has a length is equal to the number of observation in the data set. We must input a sequence as the first argument and the ratio of the train set as the second argument. Then we can save the randomly selected values of the sequence as `index` object. We can set the observations to `train` and `test` objects by using the `index` object.

Step 2 - Train a logistic regression model

We can use the `glm()` function to train a logistic regression model. It needs three obligatory arguments: (1) model formula, (2) data set that used to train the modeli and (3) the family distribution of the target feature. It must be setted as "binomial" in binary logistic regression models. To define the model formula like `y ~ x1 + x2 + ...` where `y` is the target variable which is interested features to predict and `x`s are the features that used the information to predict the target variable.

There is a tip about the model formula: you can use `y ~.` instead of defining all features you want to input to the model.

```
lr_model <- glm(survived ~ ., data = train, family = "binomial")
```

In above, we train a linear regression model by using the `train` data that we splited in previous step and the model formula. Then, we assigned the output of `glm()` function to the `lr_model` object. Do not forget that you can give another name to the model object whatever you want, because it is just an object!

Let see the output of the model:

```
lr_model
```

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

Coefficients:
          (Intercept)             gendermale                    age
            3.373307             -2.609006             -0.036670
              class2nd               class3rd         classdeck crew
           -1.027984             -2.114790              1.008424
classengineering crew  classrestaurant staff  classvictualling crew
           -1.040450             -3.588824             -1.054615
    embarkedCherbourg     embarkedQueenstown    embarkedSouthampton
            0.705627              0.097308              0.134285
                fare                 sibsp                 parch
            0.001821             -0.314243             -0.036776

Degrees of Freedom: 1742 Total (i.e. Null);  1728 Residual
Null Deviance:      2209
Residual Deviance: 1630     AIC: 1660
```

Model output returns the information about model formula, train data, and the estimated values of model parameters under the `Coefficients` title. You can see the numeric (continuous) features with its name, but the categorical features with `its name + name of the category` such as `districtOchota`.

If you want to see more detail about the model, use the `summary()` function.

```
summary(lr_model)
```

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

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5912  -0.7112  -0.5005   0.6184   2.5528

Coefficients:
                       Estimate Std. Error z value Pr(>|z|)
(Intercept)            3.373307   0.466759   7.227 4.93e-13 ***
gendermale            -2.609006   0.173764 -15.015  < 2e-16 ***
age                   -0.036670   0.005841  -6.278 3.44e-10 ***
class2nd              -1.027984   0.274634  -3.743 0.000182 ***
class3rd              -2.114790   0.270436  -7.820 5.29e-15 ***
classdeck crew         1.008424   0.384095   2.625 0.008653 **
classengineering crew -1.040450   0.290723  -3.579 0.000345 ***
classrestaurant staff -3.588824   0.785298  -4.570 4.88e-06 ***
classvictualling crew -1.054615   0.284424  -3.708 0.000209 ***
embarkedCherbourg      0.705627   0.320026   2.205 0.027461 *
embarkedQueenstown     0.097308   0.384207   0.253 0.800060
embarkedSouthampton    0.134285   0.242161   0.555 0.579217
fare                   0.001821   0.002195   0.830 0.406790
sibsp                 -0.314243   0.098853  -3.179 0.001478 **
parch                 -0.036776   0.100481  -0.366 0.714363
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2209.1  on 1742  degrees of freedom
Residual deviance: 1630.4  on 1728  degrees of freedom
AIC: 1660.4

Number of Fisher Scoring iterations: 5
```

It returns some statistics about the model significance such as null and residual deviances.

Step 3 - Measuring model performance

It is necessary to check the model performance of the model on test set. Because we are interested to train such a model has a good generalizability performance. To do this, we first

calculate the predicted values of the target variable on test set.

Do not forget to exclude the true values of the target variable from the test set!

```
predicted_probs <- predict(lr_model, test[,-8], type = "response")
head(predicted_probs)
```

```
         3          15          21          22          42          43
0.10741258 0.67094677 0.94424351 0.06712159 0.12602636 0.32462728
```

The `predicted_probs` vector has the predicted survive probability of the passengers. To measure the model performance, we transform the probabilities to classes.

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

```
 3 15 21 22 42 43
 0  1  1  0  0  0
```

Now, we can calculate the metrics based on the confusion matrix. Assume that 1 is the positive and 0 is the negative class.

```
TP <- sum(predicted_classes[which(test$survived == "yes")] == 1)
FP <- sum(predicted_classes[which(test$survived == "yes")] == 0)
TN <- sum(predicted_classes[which(test$survived == "no")] == 0)
FN <- sum(predicted_classes[which(test$survived == "no")] == 1)

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

recall
```

```
[1] 0.7788462
```

```
specificity
```

```
[1] 0.8373494
```

```
precision
```

```
[1] 0.6
```

```
accuracy
```

```
[1] 0.8233945
```

According to the values of performance metrics, the model classifies the observations with 0.82 accuracy. Its precision is 0.6 means that the model classify only 60% of the positive class, which is survived passengers in the problem, correctly. In practice this level of performance may not be satisfied.

In that case, we must check the imbalancedness of the classes in the target feature.

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

```
       no       yes
0.6706827 0.3293173
```

It is seen that, the classes of the target feature is not balanced. Thus, 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. In that case, some solution ways can be performed based on the balancing of imbalanced classes of the target variable.

We will learn these ways in next weeks!

The following steps to calculate the metrics are not user-friendly. Of course, there are many ready-to-use function to calcute these metrics in just a line of code. One of them is `confusionMatrix()` function in the `{caret}` package which is one of the most popular ML package in R.

This function requires an obligatory and an optional argument that you should check its value carefully. First one is a table that contains the observed and predicted class of the target variable, and the second is the positive class label. If you do not specify it, the function takes the first observations as reference. For example, you can try this while removing the second argument.

Let's calculate the confusion matrix and performance metrics.

```r
confusionMatrix(table(ifelse(test$survived == "yes", "1", "0"),
                      predicted_classes),
                positive = "1")
```

```
Confusion Matrix and Statistics

   predicted_classes
      0   1
  0 278  23
  1  54  81

               Accuracy : 0.8234
                 95% CI : (0.7843, 0.858)
    No Information Rate : 0.7615
    P-Value [Acc > NIR] : 0.0010805

                  Kappa : 0.559

 Mcnemar's Test P-Value : 0.0006289

            Sensitivity : 0.7788
            Specificity : 0.8373
         Pos Pred Value : 0.6000
         Neg Pred Value : 0.9236
             Prevalence : 0.2385
         Detection Rate : 0.1858
   Detection Prevalence : 0.3096
      Balanced Accuracy : 0.8081

       'Positive' Class : 1
```

In the output of the function, you can see some new statistics that you may not see before. The most commonly used ones are Accuracy, Sensitivity (Recall), Specificity, and Balanced Accuracy.

If you want to learn more about the outputs, you can check its manual: https://rdrr.io/cran/caret/man/confusion

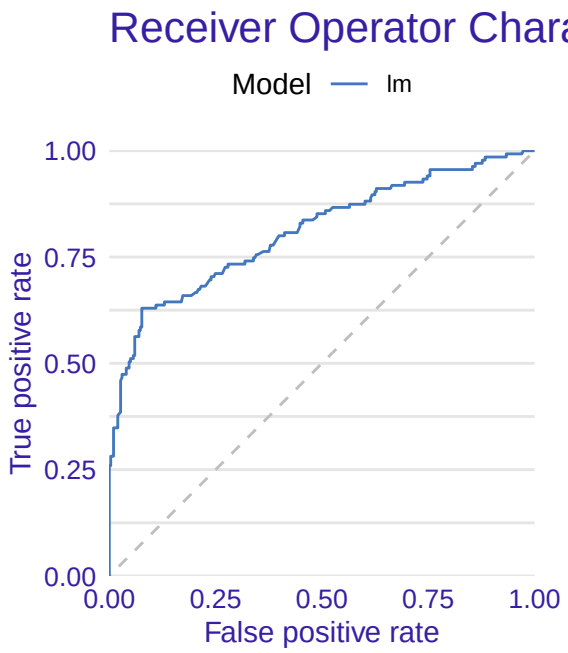
ROC Curve

To obtain a ROC curve is not an easy task in R. We can draw it by following a few of steps.

```r
# Create an explainer (an onject in DALEX universe)
explain_lr <- explain(model   = lr_model,               # trained model
                      data    = test[, -8],             # test set without target
                      y       = test$survived == "yes", # observed values of target
                                                        # with reference class
                      type    = "classification",       # type of task
                      verbose = FALSE)                  # remove some messages
```

Then we can use `model_performance()` function from `{DALEX}` package with explainer object that we created above to draw ROC curves and some others.

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



To calculate the area under the curve (AUC) value, just print the `performance_lr` object!

```r
performance_lr
```

```
Measures for:  classification
recall     : 0.6
precision  : 0.7788462
```

```
f1          : 0.6778243
accuracy    : 0.8233945
auc         : 0.8097945

Residuals:
         0%          10%          20%          30%          40%          50%
-0.80268512 -0.31681732 -0.25757224 -0.21537708 -0.17192974 -0.12602033
        60%          70%          80%          90%         100%
-0.08619287  0.03598069  0.27086289  0.71399158  0.97358362
```

It is about 0.81 means that the area under the curve is equal to 0.81 because the axes range between 0 and 1. Thus the maximum value of auc can be 1, and it means that the model performance is perfect (100% accuracy)!