IST438-W4-Applications

3/20/23

Model validation and pre-processing

In this application, we will interest to get more predictions of model performance and using some pre-processing steps:

- Model validation techniques
- Handling missing data
- Transformations

Packages

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

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

1. MODEL VALIDATION

In here, we can focus on the predicting survive status of titanic passengers.

You can use `trainControl()` function in `{caret}` package to configurate the validation way. Then `train()` function to train a logistic regression model. Let's follow the steps below:

- 1. Obtain the method and the parameters belonging to.
- 2. Train model considering the previous step.

```
set.seed(123)
control <- trainControl(method = "cv",
                        number = 10)
model <- train(as.factor(survived) ~., # model formula
               data = titanic_imputed, # all data (not train data!)
               trControl = control,    # validation setup
               method = "glm")         # method you used to train model
```

Let's see the model output:

```
model
```

```
Generalized Linear Model

2207 samples
   7 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1986, 1986, 1987, 1987, 1986, 1987, ...
Resampling results:

  Accuracy   Kappa
  0.7988554  0.5095597
```

It returns the model used, dimension of data set, and some info about the folds. If you want to see more details about the process:

```
model$resample
```

```
    Accuracy     Kappa Resample
1  0.8190045 0.5444238   Fold01
2  0.7782805 0.4616455   Fold02
3  0.7818182 0.4568460   Fold03
4  0.7954545 0.5160344   Fold04
```

```
5  0.7963801 0.5206536   Fold05
6  0.8181818 0.5509746   Fold06
7  0.8099548 0.5293103   Fold07
8  0.7432432 0.3806167   Fold08
9  0.8181818 0.5714425   Fold09
10 0.8280543 0.5636496   Fold10
```

```r
model$results
```

```
  parameter  Accuracy     Kappa AccuracySD   KappaSD
1      none 0.7988554 0.5095597 0.02571808 0.0595742
```

It is seen that the model performance looks stable because the accuracy values of the model changes between 0.74 and 0.82 in folds. The average accuracy is about 0.80.

We can also validate the model by using the LOOCV method. It may takes for a while because the LOOCV method is computationally expensive.

```r
set.seed(123)
control <- trainControl(method = "LOOCV",
                       savePredictions = TRUE)

model <- train(as.factor(survived) ~., # model formula
               data = titanic_imputed, # all data (not train data!)
               trControl = control,    # validation setup
               method = "glm")         # method you used to train model
```

```r
model
```

```
Generalized Linear Model

2207 samples
   7 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Leave-One-Out Cross-Validation
Summary of sample sizes: 2206, 2206, 2206, 2206, 2206, 2206, ...
Resampling results:
```

```
 Accuracy   Kappa 
0.7970095 0.5056671 
```

If you want to discover more about the package, visit here: https://topepo.github.io/caret/

2. MISSING DATA

The easiest way to check the missing values in data:

```r
anyNA(titanic)
```

```
[1] TRUE
```

2.1. Missing data summaries

We can summarizes the missing values in vector or data frame format. To summarize the missing values in a data set, `{naniar}` provides very useful functions as below:

- `n_miss()` returns number of missing values in data set
- `n_complete()` returns number of completed (aka not missing) values in data set
- `miss_var_summary()` returns number and percentage of missing values in data set for each variable
- `miss_case_summary()` returns number and percentage of missing values in data set for each observation

```r
n_miss(titanic)
```

```
[1] 129
```

```r
n_complete(titanic)
```

```
[1] 19734
```

```r
miss_var_summary(titanic)
```

```
# A tibble: 9 x 3
  variable n_miss pct_miss
  <chr>     <int>    <dbl>
1 country      81   3.67
2 fare         26   1.18
3 sibsp        10   0.453
4 parch        10   0.453
5 age           2   0.0906
6 gender        0   0
7 class         0   0
8 embarked      0   0
9 survived      0   0
```

```
miss_case_summary(titanic)
```

```
# A tibble: 2,207 x 3
    case n_miss pct_miss
   <int>  <int>    <dbl>
 1   577      4     44.4
 2   145      3     33.3
 3   151      3     33.3
 4   238      3     33.3
 5   517      3     33.3
 6   616      3     33.3
 7   681      3     33.3
 8  1095      3     33.3
 9  1190      3     33.3
10  1305      3     33.3
# ... with 2,197 more rows
```

- `miss_var_table()` returns a summary table consists number and percentage of missing values over variables.

```
miss_var_table(titanic)
```

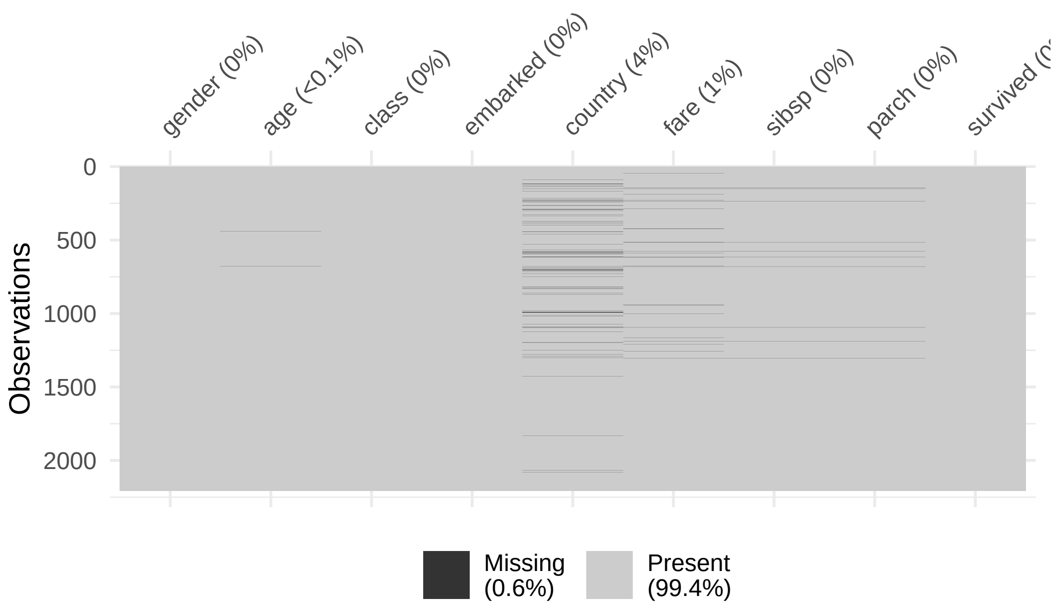
```
# A tibble: 5 x 3
  n_miss_in_var n_vars pct_vars
          <int>  <int>    <dbl>
1             0      4     44.4
2             2      1     11.1
```

3	10	2	22.2
4	26	1	11.1
5	81	1	11.1

The table shows that there are four variables do not have and missing values, and the other variables have different number of missing values.

We can visualize the missing values to see the big picture!

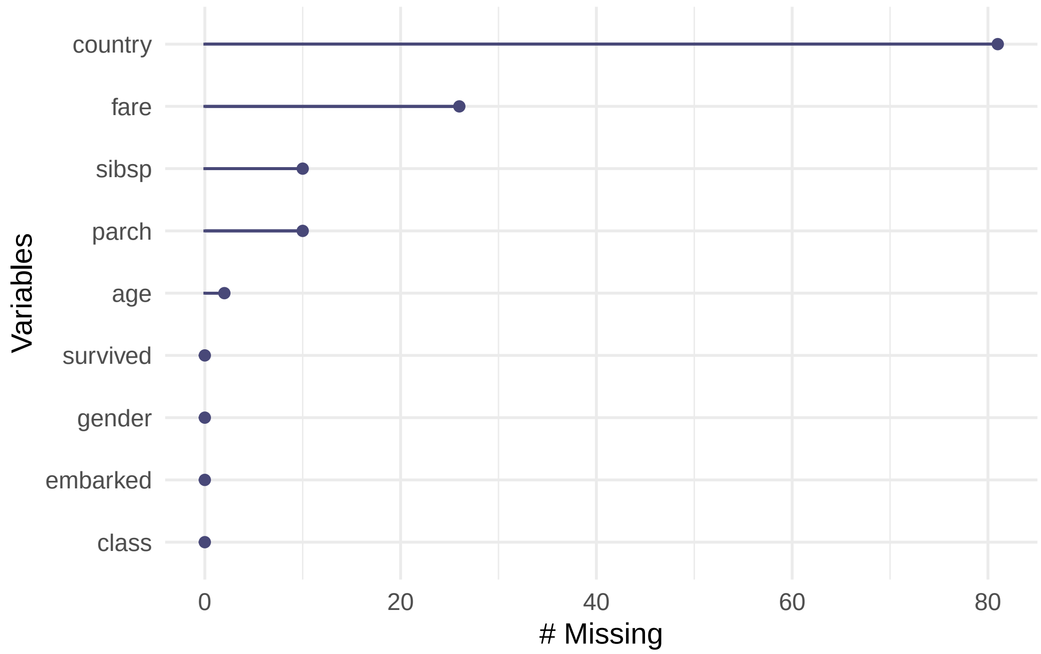
```
vis_miss(titanic)
```



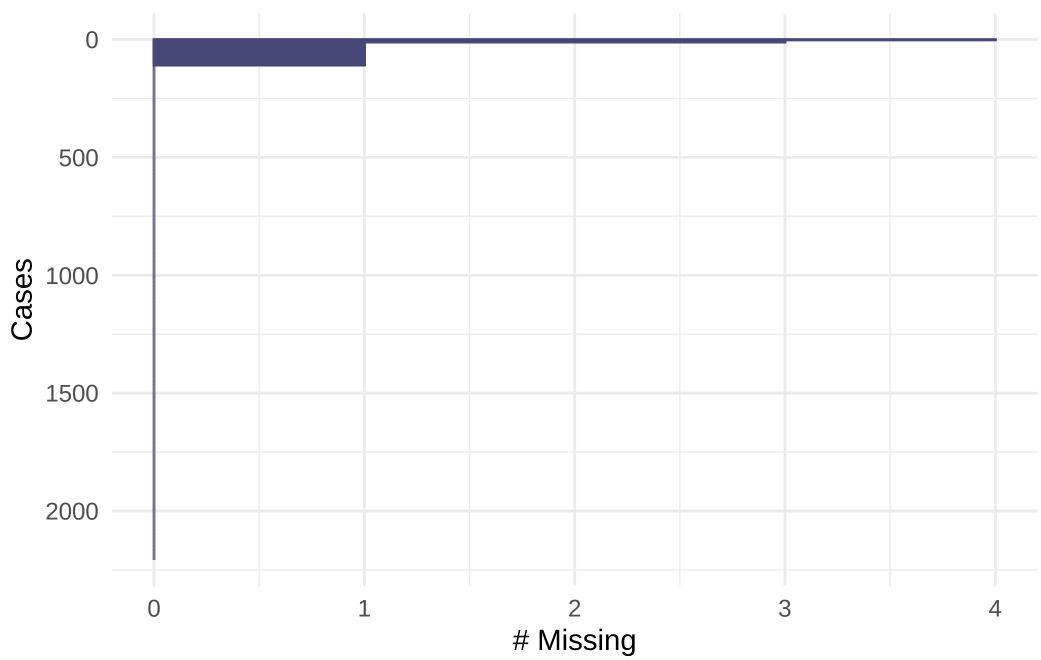
The graph above shows that the 0.6% of the observations is missing. Most of these missing values is in the `country` feature (aka variable). Also, the variables `fare`, `sibsp`, `parch`, and `age` have some missing values.

We can also visualize the missing values by variable and observation level.

```
gg_miss_var(titanic)
```




```
gg_miss_case(titanic)
```



2.2. Deletion (Removing)

```
titanic <- titanic[complete.cases(titanic), ]
```

2.3. Imputation

Imputation can be done for a single feature or all data observations. Let's try to impute a feature first:

```
data(titanic) # call the data again because we deleted the observation
              # in previous subsection

titanic$age[is.na(titanic$age)] <- median(titanic$age, na.rm = TRUE)
```

You can use `preProcess()` function from `{caret}` package to impute the missing values in all data:

```
data(titanic)

impute_mean <- preProcess(titanic, method = "medianImpute")
titanic_imp <- predict(impute_mean, titanic)
```

Let's check it is done:

```
anyNA(titanic_imp)
```

```
[1] TRUE
```

There is still some missing values in the data set, but they are in categorical features. Because the imputation methods based on mean, median, and etc. do not work with categorical features. In the categorical variables, you can impute missing values with the most frequently seen class in the complete part of the feature.

3. TRANSFORMATIONS

Let's check the scale of the features in `titanic_imputed` data set.

```
summary(titanic_imputed)
```

```
    gender          age                          class              embarked   
 female: 489   Min.   : 0.1667   1st              :324   Belfast    : 197  
 male  :1718   1st Qu.:22.0000   2nd              :284   Cherbourg  : 271  
               Median :29.0000   3rd              :709   Queenstown : 123  
               Mean   :30.4363   deck crew        : 66   Southampton:1616  
               3rd Qu.:38.0000   engineering crew:324                      
               Max.   :74.0000   restaurant staff: 69                      
                                 victualling crew:431                      
      fare             sibsp             parch            survived     
 Min.   :  0.000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
 1st Qu.:  0.000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000  
 Median :  7.151   Median :0.0000   Median :0.0000   Median :0.0000  
 Mean   : 19.992   Mean   :0.2959   Mean   :0.2284   Mean   :0.3222  
 3rd Qu.: 21.000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:1.0000  
 Max.   :512.061   Max.   :8.0000   Max.   :9.0000   Max.   :1.0000  
```

It is seen that the scale of `age` and `fare` is quite different. The scale of `age` is between 0 and 74, while the scale of `fare` is between 0 and 512. They are not in same/similar scale. This may be effective on the model performance.

You can try to transform the data set then compare the model performance with and without transformations.

3.1. Min-max transformation

Min-max transformation maps the values of feature to the interval of [0, 1]. `preProcess()` function from `{caret}` package can be used to transform the data set. It is needed to set `="range"` the `method` argument in the function to use the min-max transformation.

```
pp <- preProcess(titanic_imputed[, -8], method = "range")
scaled_titanic <- cbind(predict(pp, titanic_imputed[, -8]), survived = titanic_imputed[,8]

set.seed(123) # for reproducibility
index <- sample(1 : nrow(titanic_imputed), round(nrow(titanic_imputed) * 0.80))
```

```r
train_scaled <- scaled_titanic[index, ]
test_scaled  <- scaled_titanic[-index, ]
train <- titanic_imputed[index, ]
test  <- titanic_imputed[-index, ]


model_scaled <- glm(survived ~ ., data = train_scaled, family = "binomial")
predicted_probs_scaled <- predict(model_scaled, test_scaled[,-8], type = "response")
predicted_classes_scaled <- ifelse(predicted_probs_scaled > 0.5, 1, 0)

confusionMatrix(table(test$survived,
                      predicted_classes_scaled),
                positive = "1")
```

```
Confusion Matrix and Statistics

   predicted_classes_scaled
      0   1
  0 295  25
  1  44  77

               Accuracy : 0.8435
                 95% CI : (0.8062, 0.8762)
    No Information Rate : 0.7687
    P-Value [Acc > NIR] : 6.621e-05

                  Kappa : 0.5869

 Mcnemar's Test P-Value : 0.03024

            Sensitivity : 0.7549
            Specificity : 0.8702
         Pos Pred Value : 0.6364
         Neg Pred Value : 0.9219
             Prevalence : 0.2313
         Detection Rate : 0.1746
   Detection Prevalence : 0.2744
      Balanced Accuracy : 0.8126

       'Positive' Class : 1
```

Let's compare the model performance with the model trained on untransformed data.

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

predicted_probs <- predict(model, test[,-8], type = "response")

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

confusionMatrix(table(test$survived,
                      predicted_classes),
                positive = "1")
```

```
Confusion Matrix and Statistics

   predicted_classes
      0   1
  0 295  25
  1  44  77

               Accuracy : 0.8435
                 95% CI : (0.8062, 0.8762)
    No Information Rate : 0.7687
    P-Value [Acc > NIR] : 6.621e-05

                  Kappa : 0.5869

 Mcnemar's Test P-Value : 0.03024

            Sensitivity : 0.7549
            Specificity : 0.8702
         Pos Pred Value : 0.6364
         Neg Pred Value : 0.9219
             Prevalence : 0.2313
         Detection Rate : 0.1746
   Detection Prevalence : 0.2744
      Balanced Accuracy : 0.8126

       'Positive' Class : 1
```

It is seen that the performance of the models are totally same! This means that there is no change seen in the model performance after scaling.

3.2. Normalization

Normalization transformation maps the values of feature to normalize. `preProcess()` function from `{caret}` package can be used to transform the data set. It is needed to set `=c("center", "scale")` the `method` argument in the function to use the normalization transformation.

```
pp <- preProcess(train[, -8], method = c("center", "scale"))
centered_titanic <- cbind(predict(pp, titanic_imputed[, -8]), survived = titanic_imputed[,
train_centered <- centered_titanic[index, ]
test_centered  <- centered_titanic[-index, ]


model_centered <- glm(survived ~ ., data = train_centered, family = "binomial")
predicted_probs_centered <- predict(model_centered, test_centered[,-8], type = "response")
predicted_classes_centered <- ifelse(predicted_probs_centered > 0.5, 1, 0)

confusionMatrix(table(test$survived,
                      predicted_classes_centered),
                positive = "1")
```

```
Confusion Matrix and Statistics

   predicted_classes_centered
      0   1
  0 295  25
  1  44  77

               Accuracy : 0.8435
                 95% CI : (0.8062, 0.8762)
    No Information Rate : 0.7687
    P-Value [Acc > NIR] : 6.621e-05

                  Kappa : 0.5869

 Mcnemar's Test P-Value : 0.03024

            Sensitivity : 0.7549
            Specificity : 0.8702
         Pos Pred Value : 0.6364
         Neg Pred Value : 0.9219
             Prevalence : 0.2313
         Detection Rate : 0.1746
```

```
Detection Prevalence : 0.2744
   Balanced Accuracy : 0.8126

    'Positive' Class : 1
```

It is also seen that the performance of the models are totally same! This means that there is no change seen in the model performance after normalization.