Using Feature Engineering Methods to Improve Model Performance

Machine Learning and Applications

Tufan Bostan
08 Apr. 2021

In this work, logistic regression model will be trained using R. Data to be used here is "heart" from “catdata” package. The data is a retrospective sample of males in a heart-disease high-risk region of the Western Cape, South Africa. Based on the information’s of the males, an attempt will be made to decision whether they have heart-disease risk or not. First, some descriptive statistics will be obtained and interpreted. Then, the data will be split into two parts which are “Train” and “Test”. Then using the “Train” part of data the LRM methods will be applied and performance of the models on train and test sets will be interpreted. The performance of the model will be tried to be increased by organizing the data using Feature Engineering methods. Finally, obtained result will be interpreted and the model performances result of both are compared.

Data Structure

First of all, “catdata” package installed to access “heart” data (`install.packages("catdata")`). After that the “catdata” package must be called to access all of features of it. (`library(catdata)`) . Now “heart” data introduced to R. The “heart” data assigned to “data”. There were two variables that were not correctly classified. Before the model training process, these variables types changed numeric to factor. First 10 observations of the data are listed below. Then, structure of the data displayed. (`str(data)`) After that, data summary displayed to gather more information about variables. (`summary(data)`). Also, some of the packages were installed and called to use in this project, later.

```
#install.packages("tidymodels")
#install.packages("catdata")
#install.packages("dplyr")
library(tidymodels)
library(catdata)
library(dplyr)
data(heart)
data <- data.frame(heart)                #The data assigned to "data".
head(data,10)                            #First 10 row were displayed.
data$y <- as.factor(data$y)              #Variable type chanced to factor
data$famhist <- as.factor(data$famhist)  #Variable type chanced to factor

##    y sbp tobacco  ldl adiposity famhist typea obesity alcohol age
## 1  1 160   12.00 5.73     23.11       1    49   25.30   97.20  52
## 2  1 144    0.01 4.41     28.61       0    55   28.87    2.06  63
## 3  0 118    0.08 3.48     32.28       1    52   29.14    3.81  46
## 4  1 170    7.50 6.41     38.03       1    51   31.99   24.26  58
## 5  1 134   13.60 3.50     27.78       1    60   25.99   57.34  49
## 6  0 132    6.20 6.47     36.21       1    62   30.77   14.14  45
## 7  0 142    4.05 3.38     16.20       0    59   20.81    2.62  38
## 8  1 114    4.08 4.59     14.60       1    62   23.11    6.72  58
## 9  0 114    0.00 3.83     19.40       1    49   24.86    2.49  29
## 10 1 132    0.00 5.80     30.96       1    69   30.11    0.00  53

#Structure of The Data:
str(data)
```

```
## 'data.frame':    462 obs. of  10 variables:
##  $ y        : Factor w/ 2 levels "0","1": 2 2 1 2 2 1 1 2 1 2 ...
##  $ sbp      : num  160 144 118 170 134 132 142 114 114 132 ...
##  $ tobacco  : num  12 0.01 0.08 7.5 13.6 6.2 4.05 4.08 0 0 ...
##  $ ldl      : num  5.73 4.41 3.48 6.41 3.5 6.47 3.38 4.59 3.83 5.8 ...
##  $ adiposity: num  23.1 28.6 32.3 38 27.8 ...
##  $ famhist  : Factor w/ 2 levels "0","1": 2 1 2 2 2 1 2 2 2 ...
##  $ typea    : num  49 55 52 51 60 62 59 62 49 69 ...
##  $ obesity  : num  25.3 28.9 29.1 32 26 ...
##  $ alcohol  : num  97.2 2.06 3.81 24.26 57.34 ...
##  $ age      : num  52 63 46 58 49 45 38 58 29 53 ...

summary(data)             #To see levels of target variable and more
                          information of others.

##  y            sbp           tobacco             ldl           adiposity
##  0:302   Min.   :101.0   Min.   : 0.0000   Min.   : 0.980   Min.   : 6.74
##  1:160   1st Qu.:124.0   1st Qu.: 0.0525   1st Qu.: 3.283   1st Qu.:19.77
##          Median :134.0   Median : 2.0000   Median : 4.340   Median :26.11
##          Mean   :138.3   Mean   : 3.6356   Mean   : 4.740   Mean   :25.41
##          3rd Qu.:148.0   3rd Qu.: 5.5000   3rd Qu.: 5.790   3rd Qu.:31.23
##          Max.   :218.0   Max.   :31.2000   Max.   :15.330   Max.   :42.49
##  famhist     typea          obesity         alcohol            age
##  0:270   Min.   :13.0   Min.   :14.70   Min.   :  0.00   Min.   :15.00
##  1:192   1st Qu.:47.0   1st Qu.:22.98   1st Qu.:  0.51   1st Qu.:31.00
##          Median :53.0   Median :25.80   Median :  7.51   Median :45.00
##          Mean   :53.1   Mean   :26.04   Mean   : 17.04   Mean   :42.82
##          3rd Qu.:60.0   3rd Qu.:28.50   3rd Qu.: 23.89   3rd Qu.:55.00
##          Max.   :78.0   Max.   :46.58   Max.   :147.19   Max.   :64.00
```

The data has 10 variables and 462 observations. There are 2 factors and 8 numeric variables. Here, the target variable will be “y”. The other variables are the independent variables, to be used to predict the “y”. “sbp” is representing, systolic blood pressure and it has 138.3 mean and ranges 101 to 218. “tobacco” is representing cumulative tobacco, it has 3.6356 mean, and ranges 0 to 31.2. “ldl” is representing, low density lipoprotein cholesterol, it has 4.74 mean, and ranges 0.98 to 15.33. “adiposity” is representing, adiposity, it has 25.41 mean, and ranges 6.74 to 42.49. “femhist” is representing, family history of heart disease and it’s a factor, has 2 levels which are “0” and “1”. They were observed 270 and 192 times, respectively. “typea” is representing, type-A behavior, it has 53.1 mean and ranges 13 to 78. “obesity” is representing, obesity, and it has 26.04 mean and ranges 14.7 to 46.58. “alcohol” is representing, current alcohol consumption and it has 17.04 mean and ranges 0 to 147.19. “age” is representing, age at onset and it has 42.82 mean and ranges 15 to 64. Finally, “y” is representing, coronary heart disease. This will be the target variable for this model and it’s also a factor, has 2 levels which are “1”(yes) and “0”(no). They were observed 160 and 302 times, respectively. However, since "1” has fewer observations than the “0”, the possibility of encountering imbalance problem may occur.

Splitting Data

A sample was created using "heart" data. This sample has two parts one of them is going to be used for training to model and the other part for testing to the model. These are assigned to variables named `data_train` and `data_test`, respectively. Also, using `dim(data_train);dim(data_test)`, dimensions of each variable is shown. `data_train` has 371 row and 10 columns which are observations and variables, respectively. `data_test` has 91 row and 10 columns which are observations and variables, respectively. After this process, the LRM is ready to be generated.

```
set.seed(2380) # for reproducebility
# data splitting
data_split <- initial_split(data, prop = 0.80, strata = y)
# creating train set
data_train <- data_split %>%
training()
# creating test set
data_test <- data_split %>%
testing()
# dimension of train and test set
dim(data_train)

## [1] 371  10

dim(data_test)

## [1] 91 10

table(data_train$y);table(data_test$y)

##        0        1
##      242      129

##
##        0        1
##       60       31
```

As can be seen above, the data is divided into two parts. Still "1" has fewer observations than the "0". This has been mentioned before for main data. Although, the data is split, there is nothing changed. It may be still suspected that there is an imbalance problem.

Training LRM

Before training the LRM, the reference has been determined. Here the reference model determined as "glm" and "classification". `logistic_model <- logistic_reg() %>% set_engine('glm') %>% set_mode('classification')`. After all, the model is generated by and assigned to `logistic_fit` and the model summary were displayed.

```
logistic_model <- logistic_reg() %>%
set_engine('glm') %>% set_mode('classification')
# Fitting a linear regression model
logistic_fit <- logistic_model %>%
fit(y ~ ., data = data_train)
# Obtaining the estimated parameters
tidy(logistic_fit)

## # A tibble: 10 x 5
##    term        estimate std.error statistic  p.value
##    <chr>          <dbl>     <dbl>     <dbl>    <dbl>
##  1 (Intercept) -5.25      1.47       -3.57  0.000353
##  2 sbp          0.00302   0.00649     0.466 0.641
##  3 tobacco      0.0881    0.0289      3.05  0.00227
##  4 ldl          0.187     0.0680      2.76  0.00585
##  5 adiposity    0.0430    0.0325      1.32  0.186
##  6 famhist1     0.874     0.257       3.40  0.000670
##  7 typea        0.0413    0.0139      2.98  0.00287
##  8 obesity     -0.104     0.0506     -2.06  0.0398
##  9 alcohol     -0.00109   0.00502    -0.218 0.828
## 10 age          0.0425    0.0131      3.23  0.00124
```

The model summary output has a column of coefficients, standard errors, statistics and their probabilities. In theory, coefficients refer to how much a one-unit increase in the input variable will increase or decrease the target variable.

One-unit increase in the variable **sbp** is increases the target variable 0.003 times of that unit.

One-unit increase in the variable **tobacco** is increases the target variable 0.0881 times of that unit.

One-unit increase in the variable **obesity** is decreases the target variable 0.104 times of that unit.

A few examples are given above for different situations. With the same way, similar interpretations can be made for other variables.

The hypothesis regarding this result is $H_0: \beta_i = 0,\ H_1: \beta_i \neq 0,\ i = 0,1, \ldots$ If H_0 is not rejected, it means that the coefficient of that variable is not significant. In this case, it means that this variable does not make a significant difference in the model. This decision is made by looking at the p values in the output. If the significance of a categorical variable is discussed, this variable has a significant effect on the target variable if any of its levels are meaningful so in this model. So, "sbp", "adiposity" and "alcohol" variables are not significant at 0.05 significant level on this model. All p-values are greater than 0.05. H_0 could not rejected. Variables does not have a significant statistical effect on the target variable "y". However, rest of them are significant at 0.05 significant level on this model. All p-values are smaller than 0.05. H_0 rejected. Variables has a significant statistical effect on the target variable "y".

Predicting the Target Variable & Model Performance for Train

To predict the probabilities, `predict()` function used. First, their labels estimated and then their probabilities estimated. After that, results are displayed and confusion matrix were created.

```
# Estimated labels
data_labels_train <- logistic_fit %>%
predict(new_data = data_train, type = 'class')


# Estimated probabilities
data_preds_train <- logistic_fit %>%
predict(new_data = data_train, type = 'prob')


data_results_train <- data_train %>%
  dplyr::select(y) %>%
  bind_cols(data_preds_train, data_labels_train)
head(data_results_train,10)

##    y   .pred_0   .pred_1 .pred_class
## 2  1 0.6924379 0.3075621           0
## 3  0 0.7114239 0.2885761           0
## 5  1 0.2943719 0.7056281           1
## 6  0 0.3546433 0.6453567           1
## 7  0 0.7769086 0.2230914           0
## 8  1 0.3935561 0.6064439           1
## 9  0 0.8592741 0.1407259           0
## 10 1 0.3979106 0.6020894           1
## 11 1 0.3735014 0.6264986           1
## 12 1 0.2589838 0.7410162           1
```

Here, the prediction is determined according to the highest probability. Since "0" has higher probability value for the 2th observation in the first row, we will assume that this observation predicts as "0". This process will continue for other predicted observations.

```
# Confusion matrix
conf_mat(data_results_train, truth = y, estimate = .pred_class) %>%
summary()

## # A tibble: 13 x 3
##    .metric              .estimator .estimate
##    <chr>                <chr>          <dbl>
##  1 accuracy             binary         0.725
##  2 kap                  binary         0.369
##  3 sens                 binary         0.835
##  4 spec                 binary         0.519
##  5 ppv                  binary         0.765
##  6 npv                  binary         0.626
##  7 mcc                  binary         0.372
##  8 j_index              binary         0.354
##  9 bal_accuracy         binary         0.677
## 10 detection_prevalence binary         0.712
## 11 precision            binary         0.765
## 12 recall               binary         0.835
## 13 f_meas               binary         0.798
```

This ratios represents, model's performance on “train”. The result will be interpreted after the obtaining the test performance.

Predicting the Target Variable & Model Performance for Test

The procedures applied for the train are also applied for the test.

```
# Estimated labels
data_labels_test <- logistic_fit %>%
predict(new_data = data_test, type = 'class')


# Estimated probabilities
data_preds_test <- logistic_fit %>%
predict(new_data = data_test, type = 'prob')


data_results_test <- data_test %>%
  dplyr::select(y) %>%
  bind_cols(data_preds_test, data_labels_test)
head(data_results_test,10)

##    y   .pred_0    .pred_1 .pred_class
## 1  1 0.3251225 0.67487748           1
## 4  1 0.2983985 0.70160151           1
## 13 0 0.9604897 0.03951025           0
## 18 1 0.1069905 0.89300954           1
## 22 0 0.5934246 0.40657540           0
## 39 0 0.7446136 0.25538636           0
## 40 1 0.1051863 0.89481368           1
## 43 0 0.9763627 0.02363728           0
## 47 1 0.1349478 0.86505218           1
## 49 0 0.9642964 0.03570357           0

# Confusion matrix
conf_mat(data_results_test, truth = y, estimate = .pred_class) %>%
summary()
```

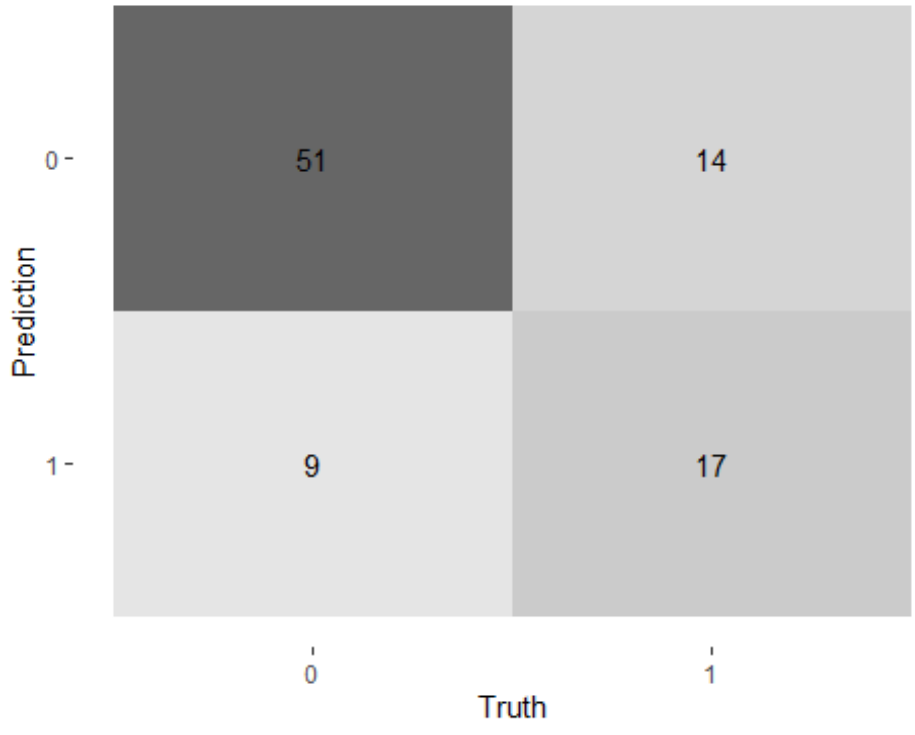
```
## # A tibble: 13 x 3
##    .metric              .estimator .estimate
##    <chr>                <chr>          <dbl>
##  1 accuracy             binary         0.747
##  2 kap                  binary         0.415
##  3 sens                 binary         0.85
##  4 spec                 binary         0.548
##  5 ppv                  binary         0.785
##  6 npv                  binary         0.654
##  7 mcc                  binary         0.418
##  8 j_index              binary         0.398
##  9 bal_accuracy         binary         0.699
## 10 detection_prevalence binary         0.714
## 11 precision            binary         0.785
## 12 recall               binary         0.85
## 13 f_meas               binary         0.816
```

This ratios represents, model's performance on “test”. According the accuracy of the model, model classifies the “test” approximately %75 correctly. This value obtained from confusion matrix that crated for test. We did the same process that we did with "train" before to testing data. The accuracy ratio that we obtained using the “train” was 0.725 which means model classifies the “train” approximately %73 correctly.” train” accuracy is smaller than “test”. We might suspect of underfitting problem. Getting more training data, increasing the size or number of parameters in the model or increasing the complexity of the model may decrease or fix this problem. The higher accuracy ratio means; the better estimation result we get.

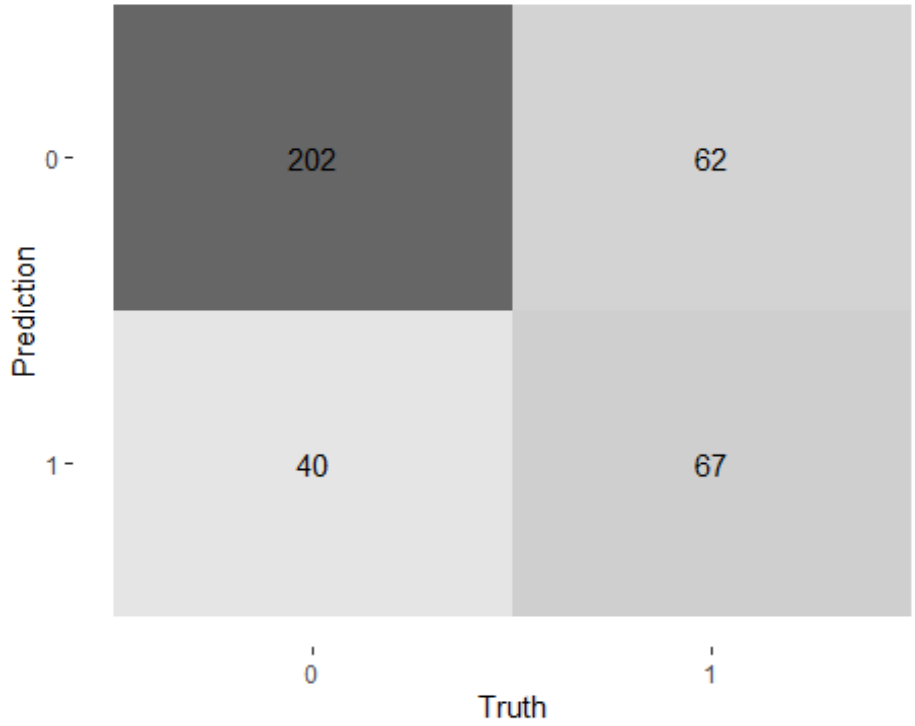
Visualizing Model Performance

In this section, some graphs were crated to see model performance.

```
# heatmap of the confusion matrix
conf_mat(data_results_test, truth = y, estimate = .pred_class) %>%
  autoplot(type = 'heatmap')
```

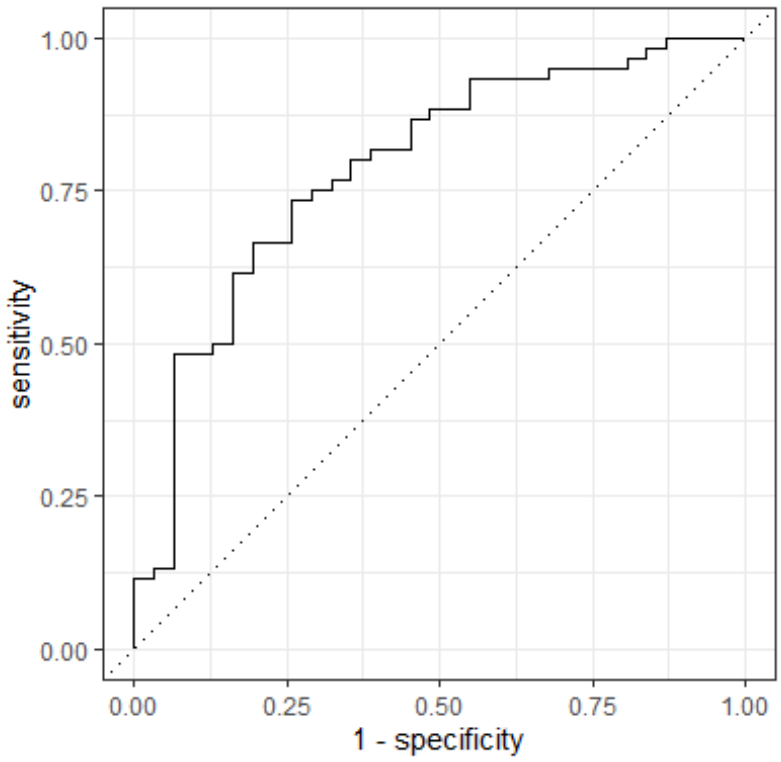


```r
conf_mat(data_results_train, truth = y, estimate = .pred_class) %>%
  autoplot(type = 'heatmap')
```

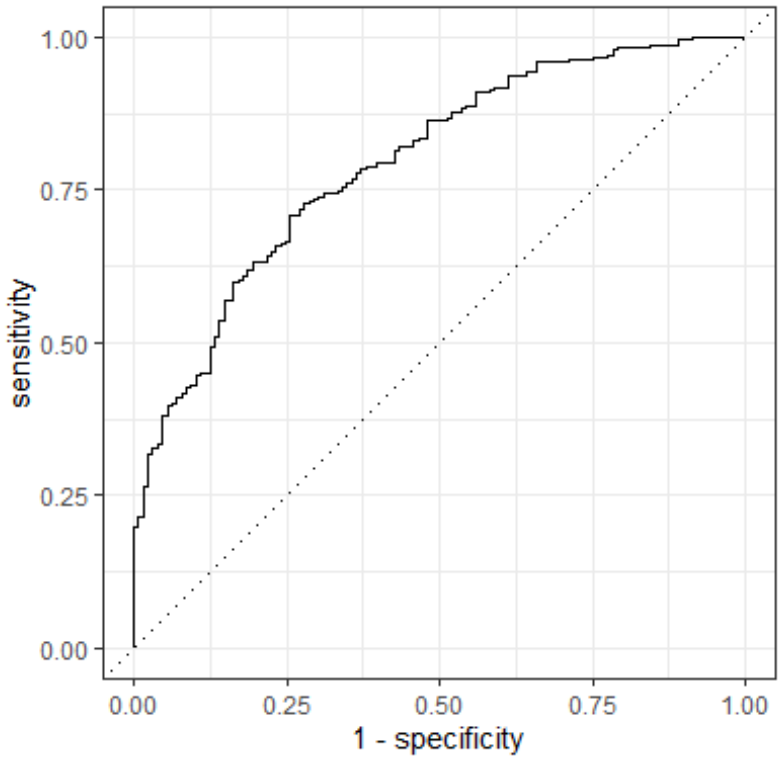


It can be said that these graphs are the visualization of the confusion matrix. Here, there are 2 graphs first one for “test” and second one for “train”. The same interpretation can be made by looking those graphs. Accuracy, specificity etc. can be calculated on this chart.

```r
# Visualizing performance across thresholds
data_results_test %>%
  roc_curve(truth = y, estimate = .pred_0) %>%
  autoplot()
```




```
data_results_train %>%
  roc_curve(truth = y, estimate = .pred_0) %>%
  autoplot()
```



Here is another graph that indicates the performance of the model. These graphs belong to "test" and "train" respectively. It is concluded that the closer the curve here is to the upper-left corner,

the better the model. We can also decide the proximity to that corner by calculating the area under the curve. The larger the area means that, the better model.

```
# Calculating ROC AUC
roc_auc(data_results_test, truth = y, .pred_0)

## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.789

roc_auc(data_results_train, truth = y, .pred_0)

## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.795
```

Now, areas under the curve were calculated. Here again, it is calculated for "test" and "train", respectively. The larger the area means that, the better model. The values obtained here will be compared with the values obtained after performing feature engineering methods.

Feature Engineering for Test

Here some of the feature engineering methods have been applied to the data. First, variables with high correlations were removed. Then all the numeric predictors were normalized and dummy variables were created. After that, these operations are applied to data_test and data_traine and the model was trained and the class predictions and estimate probabilities were obtained.

```
data_recipe <- recipe(y ~ ., data = data_train) %>%
  # Removed correlated predictors
  step_corr(all_numeric(), threshold = 0.8) %>%
  # Normalize numeric predictors
  step_normalize(all_numeric()) %>%
  # Create dummy variables
  step_dummy(all_nominal(), -all_outcomes())
# Train recipe
data_recipe_prep <- data_recipe %>%
  prep(training = data_train)
# Transform training data
data_train_prep <-data_recipe_prep %>%
  bake(new_data = data_train)
# Transform test data
data_test_prep <- data_recipe_prep %>%
  bake(new_data = data_test)

#############################################################################################

# Train logistic model
logistic_fit <- logistic_model %>%
  fit(y ~., data = data_train_prep)

# Obtain class predictions
class_preds <- predict(logistic_fit, new_data = data_test_prep,type = 'class')

# Obtain estimated probabilities
prob_preds <- predict(logistic_fit, new_data = data_test_prep,type = 'prob')

# Combine test set results
data_results_feature <- data_test_prep %>%
  dplyr::select(y) %>%
  bind_cols(class_preds, prob_preds)
```

```r
# calculating accuracy etc.
conf_mat(data_results_feature, truth = y, estimate = .pred_class) %>%
summary()
```

```
## # A tibble: 13 x 3
##    .metric              .estimator .estimate
##    <chr>                <chr>          <dbl>
##  1 accuracy             binary         0.747
##  2 kap                  binary         0.415
##  3 sens                 binary         0.85
##  4 spec                 binary         0.548
##  5 ppv                  binary         0.785
##  6 npv                  binary         0.654
##  7 mcc                  binary         0.418
##  8 j_index              binary         0.398
##  9 bal_accuracy         binary         0.699
## 10 detection_prevalence binary         0.714
## 11 precision            binary         0.785
## 12 recall               binary         0.85
## 13 f_meas               binary         0.816
```

Here, Values for the performance of the model were obtained. Some of these are accuracy=0.747, specificity=0.548 and sensitivity=0.85. Here we will focus on the accuracy value and again we will compare this value with the results we will get for the "train". Now let's calculate the train set performance;

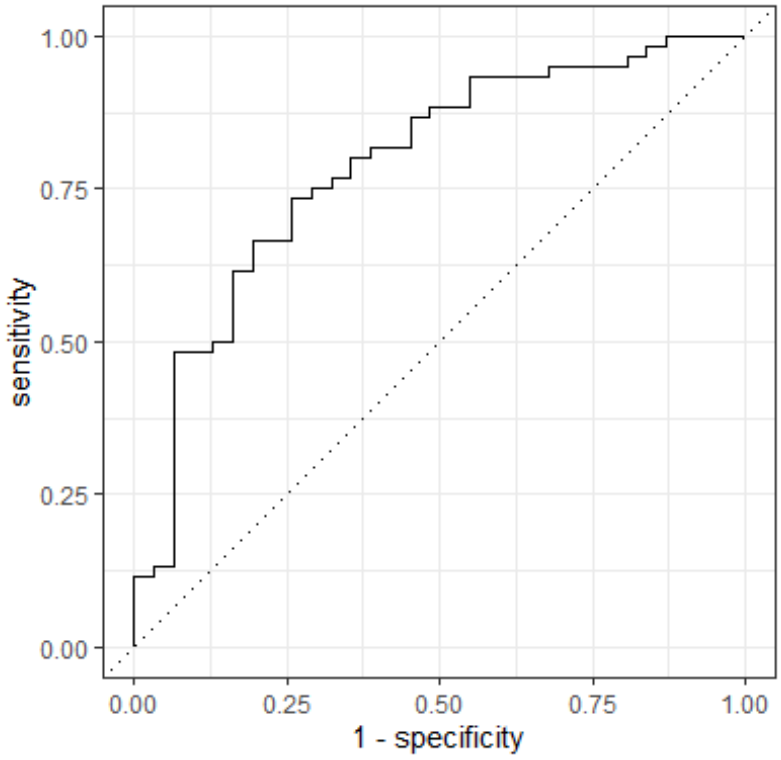
```r
# Obtain class predictions
class_preds_train <- predict(logistic_fit, new_data = data_train_prep,type = 'class')

# Obtain estimated probabilities
prob_preds_train <- predict(logistic_fit, new_data = data_train_prep,type = 'prob')

# Combine test set results
data_results_feature_train <- data_train_prep %>%
  dplyr::select(y) %>%
  bind_cols(class_preds_train, prob_preds_train)
# calculating accuracy
conf_mat(data_results_feature_train, truth = y, estimate = .pred_class) %>%
summary()
```

```
## # A tibble: 13 x 3
##    .metric              .estimator .estimate
##    <chr>                <chr>          <dbl>
##  1 accuracy             binary         0.725
##  2 kap                  binary         0.369
##  3 sens                 binary         0.835
##  4 spec                 binary         0.519
##  5 ppv                  binary         0.765
##  6 npv                  binary         0.626
##  7 mcc                  binary         0.372
##  8 j_index              binary         0.354
##  9 bal_accuracy         binary         0.677
## 10 detection_prevalence binary         0.712
## 11 precision            binary         0.765
## 12 recall               binary         0.835
## 13 f_meas               binary         0.798
```

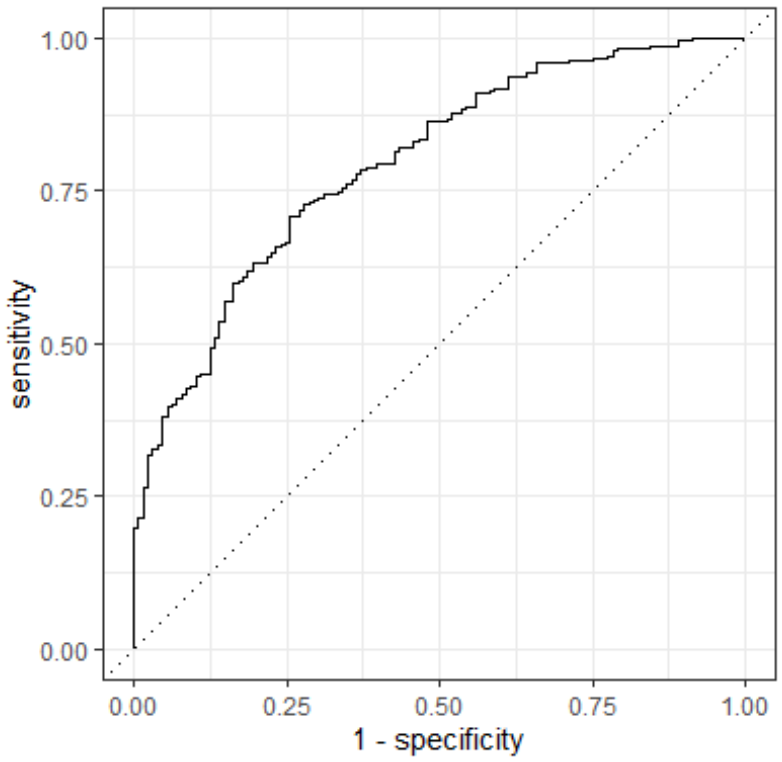
Accuracy values were obtained for test and train. This value is 0.747 for the test set and 0.725 for the train set. After applying feature engineering methods, the success of the model in predicting the test set is approximately 75%, while it is 73% for the train set. "train" accuracy is smaller than "test". We might suspect of underfitting problem like before applying feature engineering methods.

Getting more training data, increasing the size or number of parameters in the model or increasing the complexity of the model may decrease or fix this problem. We have seen the performance of the model after applying feature engineering methods. Now we will compare the performance of this model with its performance before applying feature engineering. It was mentioned that this comparison can be made with the ROC curve or the area under that curve (AUC). ROC curve for test and train is shown below, respectively.

```
# Visualizing performance across thresholds
data_results_feature %>%
  roc_curve(truth = y, estimate = .pred_0) %>%
  autoplot()
```




```
# Visualizing performance across thresholds
data_results_feature_train %>%
  roc_curve(truth = y, estimate = .pred_0) %>%
  autoplot()
```



As it was said before, the fact that this curve is close to the upper left corner indicates that it is a more successful model. The ROC curves of the models obtained before applying the feature engineering methods for test and train sets look the same to the ROC curves obtained after applying the feature engineering methods. But to be sure, it should be compared to the area under these two curves (AUC).

```
# Calculating ROC AUC test
roc_auc(data_results_feature, truth = y, .pred_0)

## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.789
```

For the test set, the AUC value we calculated before applying feature engineering methods was 0.789. It has been observed that this value does not change after applying feature engineering methods. In this case, it can be said that feature engineering methods failed to increase the performance of this model. A table has been created to compare all performance values.

```
acc_before_test <- accuracy(data_results_test,truth = y, estimate = .pred_class)
acc_before_train <- accuracy(data_results_train,truth = y, estimate = .pred_class)
spc_before_test <- spec(data_results_test,truth = y, estimate = .pred_class)
spc_before_train <- spec(data_results_train,truth = y, estimate = .pred_class)
sen_before_test <- sens(data_results_test,truth = y, estimate = .pred_class)
sen_before_train <- sens(data_results_train,truth = y, estimate = .pred_class)
auc_before_test <- roc_auc(data_results_test, truth = y, .pred_0)
auc_before_train <- roc_auc(data_results_train, truth = y, .pred_0)
acc_after_test <- accuracy(data_results_feature,truth = y, estimate = .pred_class)
acc_after_train <- accuracy(data_results_feature_train,truth = y, estimate = .pred_class)
spc_after_test <- spec(data_results_feature,truth = y, estimate = .pred_class)
spc_after_train <- spec(data_results_feature_train,truth = y, estimate = .pred_class)
sen_after_test <- sens(data_results_feature,truth = y, estimate = .pred_class)
sen_after_train <- sens(data_results_feature_train,truth = y, estimate = .pred_class)
auc_after_test <- roc_auc(data_results_feature, truth = y, .pred_0)
```

```
auc_after_train <- roc_auc(data_results_feature_train, truth = y, .pred_0)
comp <- data.frame( "Before"=c(acc_before_test$.estimate,acc_before_train$.estimate,spc_bef
ore_test$.estimate,spc_before_train$.estimate,sen_before_test$.estimate,sen_before_train$.e
stimate,auc_before_test$.estimate,auc_before_train$.estimate),
                    "After"=c(acc_after_test$.estimate,acc_after_train$.estimate,spc_after_
test$.estimate,spc_after_train$.estimate,sen_after_test$.estimate,sen_after_train$.estimate
,auc_after_test$.estimate,auc_after_train$.estimate),
                    row.names = c("Test Accuracy","Train Accuracy","Test Specificity", "Tra
in Specificity", "Test Sensitivity","Train Sensitivity", "Test Area Under Curve", "Train Ar
ea Under Curve")
)
comp

##                            Before     After
## Test Accuracy           0.7472527 0.7472527
## Train Accuracy          0.7250674 0.7250674
## Test Specificity        0.5483871 0.5483871
## Train Specificity       0.5193798 0.5193798
## Test Sensitivity        0.8500000 0.8500000
## Train Sensitivity       0.8347107 0.8347107
## Test Area Under Curve   0.7892473 0.7892473
## Train Area Under Curve 0.7946057 0.7946057
```

As can be seen above, As can be seen, all performance values are equal. As a result, no positive or negative changes were encountered in the performance of the model after the application of feature engineering methods.