Heart Attack Prediction

Gizem ALTUN

Supervised Learning: Logistic Regression Models

In this task, the following steps were followed and the logistic regression was dealt with.

1. Definition of the Problem, Target and Features
2. Describing the Data Set
3. Training a Logistic Regression Model
4. Training Decision Tree
5. Train a Random Forest

1. Definition of the Problem, Target and Features

In this task, to predict the probability of the which have patients heart attack.

Target variable and features were examined in this dataset. A dependent variable(target) is a variable that is observed to change in response to independent variables(features). Then the target variable in this data set is the "output" variable.(Y: output) Since we have a target variable in this dataset, this process as "supervised learning" process. The target variable is categorical variable. It means that the target variable or any categorical variable can only text the values of some classes. In this task, only two possible outcome of this variable so then we an call the status variable as categorical variable which takes just two values. The values are or 0= less chance of heart attack 1= more chance of heart attack. This is called the "Binary Classification Task" because the target variable(output) takes only two classes.(1 or 0)

Independent variables(features) are variables that cause a change in dependent variable(target). The features in this dataset are;X1:age= Age of the patient, X2:sex= Sex of the patient, X3:cp= Chest Pain type chest pain type, X4:trtbps= resting blood pressure (in mm Hg), X5:chol= cholestoral in mg/dl fetched via BMI sensor, X6:fbs= (fasting blood sugar $>$ 120 mg/dl) (1

= true; 0 = false), X7:restecg= esting electrocardiographic results, X8:thalachh= maximum heart rate achieved, X9:exng= exercise induced angina (1 = yes; 0 = no), X10:oldpeak= Previous peak, X11:slp=, X12:caa, X13:thall. There are a total of 13 features and 1 target variable(output) in this task.

Packages

Before proceeding to the steps written above about logistic regression, the packages required for these steps have been installed.As a first step, the packages were installed. As a second step, ıt was called from the library.

```
install.packages("DALEX")
#Comparing performance across multiple models convenient.
install.packages("readxl")
#This package allows us to read our dataset.
install.packages("caret")
#set of functions that attempt to streamline the process for creating predictive models.
#install.packages("tidymodels") # training models
#install.packages("rpart.plot")# visualizing decision tree
#install.packages("yardstick")
#install.packages("ROCR")
#install.packages("ranger")
#install.packages("dplyr")
library(dplyr)
library(DALEX)
library(readxl)
library(caret)
library(tidymodels)
library(rpart.plot)
library(yardstick)
library(ROCR)
library(ranger)
```

The first line is hashtaged to faster this process.

2. Describing the Data Set

There are 13 features in this data set that affect the likelihood of having a heart attack. Thanks to these features, we will try to predict the likelihood of having a heart attack. This is also classified in the data set as follows. The values are or 0= less

chance of heart attack 1= more chance of heart attack. This dataset is taken from Kaggle.(https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset) The readxl package was used because the data set's Excel file is of the xlsx file type. At the same time, import dataset from environment window to add the dataset.

```
heart <- read_excel("heart.xlsx")
```

The data set was looked at using the str() function. This function gives us the following information; number of observation, number of features, name of features, type of features, a few observations of features.

```
str(heart)
```

```
tibble [303 x 14] (S3: tbl_df/tbl/data.frame)
 $ age     : num [1:303] 63 37 41 56 57 57 56 44 52 57 ...
 $ sex     : num [1:303] 1 1 0 1 0 1 0 1 1 1 ...
 $ cp      : num [1:303] 3 2 1 1 0 0 1 1 2 2 ...
 $ trtbps  : num [1:303] 145 130 130 120 120 140 140 120 172 150 ...
 $ chol    : num [1:303] 233 250 204 236 354 192 294 263 199 168 ...
 $ fbs     : num [1:303] 1 0 0 0 0 0 0 0 1 0 ...
 $ restecg : num [1:303] 0 1 0 1 1 1 0 1 1 1 ...
 $ thalachh: num [1:303] 150 187 172 178 163 148 153 173 162 174 ...
 $ exng    : num [1:303] 0 0 0 0 1 0 0 0 0 0 ...
 $ oldpeak : num [1:303] 2.3 3.5 1.4 0.8 0.6 0.4 1.3 0 0.5 1.6 ...
 $ slp     : num [1:303] 0 0 2 2 2 1 1 2 2 2 ...
 $ caa     : num [1:303] 0 0 0 0 0 0 0 0 0 0 ...
 $ thall   : num [1:303] 1 2 2 2 2 1 2 3 3 2 ...
 $ output  : num [1:303] 1 1 1 1 1 1 1 1 1 1 ...
```

Describing the Dataset:

This dataset contains 303 observation of 14 variables. This also equates to 303 rows and 14 columns. Variables:

1.Age: Numeric and continuous variable. Continuous because, it takes the values between zero and positive infinity it takes any values.

2.Sex: Categoric variable.

3.cp: Categoric variable.

4.trtbps: Numeric and continuous variable.

5.chol: Numeric and continuous variable.

6.fbs: Categoric variable.

7.restecg: Categoric variable.

8.thalachh: Numeric and continuous variable.

9.exng: Categoric variable.

10.oldpeak: Numeric and continuous variable.

11.slp: Categoric variable.

12.caa: Categoric variable.

13.thall: Categoric variable.

14.output: Categoric variable.

From the first variable to the thirteenth variable, they are all features. The fourteenth variable(output) is the target variable. As can be seen, the dataset contains the target variable. This process as "Supervised Learning" process. According to type of the target variable we can conclude that type of task. The type of target variable is categoric this is called classification task and only two possible outcome of this variable so then we an call the status variable as categorical variable which takes just two values. The values are 0 or 1. This is called the "Binary Classification Task" because the target variable(output) takes only two classes.(The values are or 0= less chance of heart attack 1= more chance of heart attack) Since the type of target variable is categorical, logistic regression is used.

```
dim(heart)
```

```
[1] 303  14
```

The dim() function shows us to dimension of dataset. When there are how many observations in the first column, we see how many variables there are in the second column. This dataset contains 303 observation of 14 varibles.

Before training the model, let's look at the target variable type and missing value states.

Since our target variable remains a character type, we need to make it categorical. While doing this, we will use the as.factor function. The syntax is as.factor(input), where the input is a vector, column, or data frame and returns the requested column specified as a factor rather than a character one. The reason we do this is that we should attribute it to taking values as a probability in the following way; y is a binary target variable takes the values (0,1). p is the probability of y=1 (more chance of heart attack) , p =P(Y=1), p is the probability of y=0(less chance of heart attack) , p =P(Y=0).

```
heart$output <- as.factor(heart$output)
```

Later, used to the as.factor function, it was turned into a categorical data type. In order to check, the class was checked used to the class function.

```
class(heart$output)
```

```
[1] "factor"
```

We checked that the data type of the target variable is correct.

Firstly, before training the model, excluded the NA variables in the dataset.

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

It is necessary because when we split dataset, 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.

3. Splitting the Data Set(Train and Test Set)

Train/ Test split is used rather than just validating the model on train set, it gives also an estimate how well the model performances on new data.That we should train model a dataset then we check performance of model in a different dataset so far doing this. We can seperate our original dataset to train and test set data. We can do this steps by selecting randomly observation so in practice mostly the number of observation in train dataset much more than test data. The ratio like 80% train data, 20% test data. This dataset was also splitted at this ratio. The split the dataset is important because it allows us to "generalizability" the model.

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

We should split the data set to two different set named train and test set. We will use sample function to create a new object which is index. In this sample we need to use a sequence. When we create an sequence starting with 1 up to the number of rows of the heart data set. then we use the same dataset to split 80% of the dataset.

3. Training a Logistic Regression Model

The process to estimate the model coefficients is called model training. This task predict the alive probability of the which have patients breast cancer. So, the glm() function was used to the lr_model object to train the model. Model formula tips: (y ~ .) Here "y" refers to the target variable, dot refers to the features. Since the dot mark is set, this includes all the features. Finally, the family distribution of the target feature. It must be setted as "binomial" in binary logistic regression models.

```
lrm_model <- glm(output ~ ., data = train, family = "binomial")
lrm_model
```

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

Coefficients:
(Intercept)          age          sex           cp       trtbps         chol
   4.555347    -0.013283    -1.540661     0.769378    -0.017916    -0.003182
        fbs      restecg     thalachh         exng      oldpeak          slp
   0.132748     0.658516     0.019761    -1.077694    -0.678520     0.263729
        caa        thall
  -0.646435    -1.015510

Degrees of Freedom: 241 Total (i.e. Null);  228 Residual
Null Deviance:      333.5
Residual Deviance: 173.1    AIC: 201.1
```

This code output gives information about the model. The estimated values of the model parameters which are Betas and the trained data. From this code output, it shows which variables are numerical and which variables are categorical.(name + name of the category) summary() function was used to see more details about the model.

```
summary(lrm_model)
```

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

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
```

```
(Intercept)  4.555347   2.794683   1.630  0.10310
age         -0.013283   0.025289  -0.525  0.59941
sex         -1.540661   0.509861  -3.022  0.00251 **
cp           0.769378   0.195309   3.939 8.17e-05 ***
trtbps      -0.017916   0.011003  -1.628  0.10348
chol        -0.003182   0.004177  -0.762  0.44617
fbs          0.132748   0.566248   0.234  0.81465
restecg      0.658516   0.389009   1.693  0.09049 .
thalachh     0.019761   0.011439   1.727  0.08408 .
exng        -1.077694   0.448609  -2.402  0.01629 *
oldpeak     -0.678520   0.240961  -2.816  0.00486 **
slp          0.263729   0.411859   0.640  0.52195
caa         -0.646435   0.201346  -3.211  0.00132 **
thall       -1.015510   0.330194  -3.075  0.00210 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 333.48  on 241  degrees of freedom
Residual deviance: 173.07  on 228  degrees of freedom
AIC: 201.07

Number of Fisher Scoring iterations: 6
```

We can see some significant statistics for the features. If any class of a categorical variable is significant which means the p-value is lower than the significance level we can evaluate the whole level of categorical variables is significant. For example in here, two different classes of oldpeak categorical variable two of them significant but even if only one of them was significant again, in this case, too it means that oldpeak feature significant. It has an important contribution to the target variable. It is same for any other categorical variable in here. We can use the AIC value to compare the models.

Coefficients shows us the logs odds ratios of the coefficient. We can take the odds ratios of features for doing this. We can take the exponential of them by using the exp() function.

5. Performance of the Trained Logistic Regression Model

It is necessary to check the model performance of the model on test set. Because we are interested in train such a model has a good generalizability. To do this, we first calculated the predicted values of the target variable on test set. There are some additional steps different from the regression task. In here, to predict the value of target variable, we use additional argument. It is type when we set the type as response we can calculate the probability of

the target variable or predicted probabilities of target variable. We assign predicted_probs to predict underscore props object. Exclude the true values of target variable from test set.

```
predicted_probs <- predict(lrm_model, test[ , -14], type = "response")
head(predicted_probs)
```

```
        1         2         3         4         5         6
0.7079596 0.9397678 0.9823383 0.9748455 0.6193038 0.9260086
```

We can see that this is are whole probability values because it differs from 0 and 1.

Then we should convert this probability values to the values of classes. There are two classes in our target variable these are "1" or "0". Convert them one and zero. We can use ifelse() function for doing this. So, first argument is a condition. It means that if the values of predicted props object is higher than 0.5 let's assign it as one. If otherwise assign in as zero.

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

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

Measuring the Model Performance

There is a binary classification task and the aim of the model is to predict the value of or classes of target variable correctly possible. Confusion Matrix, two different part; observed class and predicted class. Classes of the model of target variable are positive and negative class. Model assign any positive values as positive this is a kind of correct decision and we call the number of observation this cell "True Positive" (TP), if the observed class of the target variable is negative then model may label this observation like positive so we call this cell "False Positive" (FP), ıf the observed class of the target variable is positive then the model may label them as negative so this is wrong decision so we called this cell as "False Negative" (FN) and last possibility is observed class of target variable may be negative and model may assign these observation as negative so this correct decision we call this cell is "True Negative" . The following assignments were made in this way. The metrics is calculated based on these four cell.

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

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

recall
```

```
[1] 0.7948718
```

```
specificity
```

```
[1] 0.9090909
```

```
precision
```

```
[1] 0.9393939
```

```
accuracy
```

```
[1] 0.8360656
```

Some information about these metrics. Recall addresses the question: "Among all the true positives, how many of them are indeed correctly captured by the model?" In here, 79 percent. Specificity, It tells us the proportion of correctly identified negative labels (TN) among all the negative labels (TN + FP). In here, 90 percent. Precision addresses the question: How many percent of the positive class is classified correctly? In here, we can say that 93 percent of the right patients in the test set is classified correctly. Accuracy is important metric. Because it means that 83 percent of the observation is classified correctly by the model. Looking at just one metric can sometimes be misleading, so it's also useful to examine most of them. Although these metric values look good.

The following steps to calculate the metrics are not user-friendly. So, we use confusionMatrix() function in the {caret} package. This function requires an obligatory an an optional argument that we should check its value carefully. First one is a table that contains the observed and predicted class of target variable, and the second is positive class label.

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

```
Confusion Matrix and Statistics

   predicted_classes
     0  1
  0 20  8
  1  2 31

               Accuracy : 0.8361
                 95% CI : (0.7191, 0.9185)
    No Information Rate : 0.6393
    P-Value [Acc > NIR] : 0.000614

                  Kappa : 0.6645

 Mcnemar's Test P-Value : 0.113846

            Sensitivity : 0.7949
            Specificity : 0.9091
         Pos Pred Value : 0.9394
         Neg Pred Value : 0.7143
             Prevalence : 0.6393
         Detection Rate : 0.5082
   Detection Prevalence : 0.5410
      Balanced Accuracy : 0.8520

       'Positive' Class : 1
```

In the output of the function, we can see some new statistics. The most commonly used one is Balanced Accuracy. Balanced Accuracy (average of accuracy), ıt's the arithmetic mean of sensitivity and specificity, its use case is when dealing with imbalanced data, when one of the target classes appears a lot more than the other. The reason why it is important is that if the Balanced Accuracy value is lower than the accuracy value, it allows us to make the comment that there is an imbalanced problem here. Then we not say that there is an imbalanced problem here for this task.

Any model performance metrics in classification is depend on the cutoff value (c) to assign a class to observation. In logistic regression models we predict the probability of intended class then we convert it to class label. For example, when we use logistic regression model

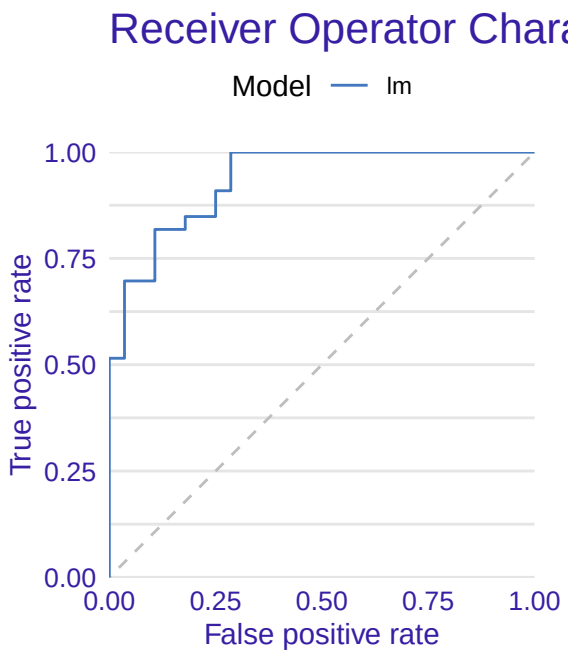
we predict the probability of an intended class. So, then we should convert this probability to class label. So, in this measuring step we use 0.5 because we must define a cutoff value in practice. So, the result of this metrics is heavily depend on the value of cutoff value. The model performance can be manipulated by fixing the values of c, if we use only this 4 metrics. So, we use some alternative metrics. ROC curve gives us more detailed information about the model performances which is "independently" from the value of c. Area Under the Curve metrics gives us the area under the ROC curve.

We use explain an object in DALEX package.

```
explain_lrm <- explain(model = lrm_model,
                       data = test[ , -14],
                       y = test$output == "1",
                       type ="classification",
                       verbose = FALSE)
```

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

```
performance_lrm <- model_performance(explain_lrm)
plot(performance_lrm, geom = "roc")
```



If ROC curve getting closer to the left top, we can say that model performance is going to perfect. When we look at the graph it does not perform well.

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

```
performance_lrm
```

```
Measures for:  classification
recall     : 0.9393939
precision  : 0.7948718
f1         : 0.8611111
accuracy   : 0.8360656
auc        : 0.9339827

Residuals:
         0%         10%         20%         30%         40%         50%
-0.92500426 -0.66440247 -0.23803316 -0.05420990 -0.00710291  0.01057154
        60%         70%         80%         90%        100%
 0.02908870  0.05916045  0.19665257  0.30022165  0.59491469
```

It is about 0.93 means that the AUC equals to 0.93 because the axes range between 0 and 1. Thus maximum value of AUC can be 1. In here, the model performance is looks good.

#4. Train a Decision Tree

A decision tree is a hierarchical model used in decision support that depicts decisions and their potential outcomes, incorporating chance events, resource expenses, and utility. This algorithmic model utilizes conditional control statements and is non-parametric, supervised learning, useful for both classification and regression tasks. The tree structure is comprised of a root node, branches, internal nodes, and leaf nodes, forming a hierarchical, tree-like structure.

important features for decision tree;

Non- parametric algorithm, work by partitioning the feature space into a number of smaller regions, using splitting rules so easy to interpret, produce simple rules that are easy to interpret, visualize with tree diagrams.

Most well-known method for constructing decision trees is the classification and regression trees proposed in Breiman.

A basic decision tree partitions the training data into homogeneous subgroups is to aim. The subgroups are formed recursively using binary partitions. This is done a number of times until a stopping criteria is satisfied. The model predicts the output based on: the average target values for all observation that fall in that subgroup. The class that has majority representation.

1. Data Splitting

We are separating our data set with a different method this time. We split the dataset with probability 0.80 using the initial_split() function. Then, we obtain this data set, which we have separated, by using the training() and testing() functions as train and test set. The results will be the same as the values which we obtained above. In order not to be confused with the above, we assign it to different names.

```
set.seed(123)
heart_split <- initial_split(data = heart, # dataset to split
                                    prop = 0.80)    # proportion of train set
```

```
heart_train <- heart_split |> training()
heart_test  <- heart_split |> testing()
```

2. 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

Defining Model Specification

```
heart_dt <- decision_tree() |>
  set_engine("rpart") |>
  set_mode("classification")
```

3. Model Training

It is very similar to what we do in logistic regression, since I described it in detail above, I will go through the next parts by explaining it more briefly. It is just that the fit() functionwas used instead of the glm() function, and we didn't specify that it should be binomial in addition.

```
heart_dtm <- heart_dt |>
  fit(output ~., data = heart_train)

heart_dtm
```

```
parsnip model object

n= 242
```

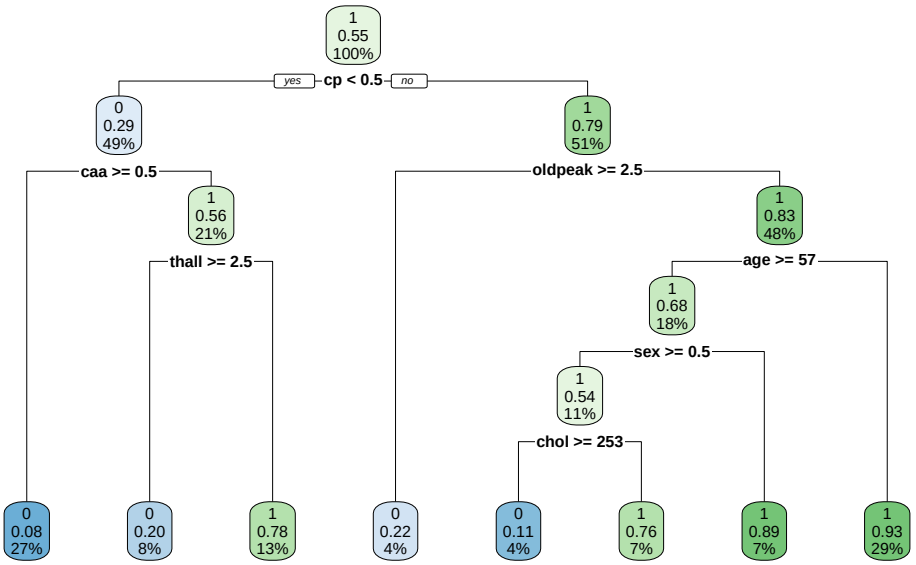
```
node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 242 110 1 (0.45454545 0.54545455)
   2) cp< 0.5 118  34 0 (0.71186441 0.28813559)
     4) caa>=0.5 66   5 0 (0.92424242 0.07575758) *
     5) caa< 0.5 52  23 1 (0.44230769 0.55769231)
      10) thall>=2.5 20   4 0 (0.80000000 0.20000000) *
      11) thall< 2.5 32   7 1 (0.21875000 0.78125000) *
   3) cp>=0.5 124  26 1 (0.20967742 0.79032258)
     6) oldpeak>=2.45 9   2 0 (0.77777778 0.22222222) *
     7) oldpeak< 2.45 115  19 1 (0.16521739 0.83478261)
      14) age>=56.5 44  14 1 (0.31818182 0.68181818)
        28) sex>=0.5 26  12 1 (0.46153846 0.53846154)
          56) chol>=253 9   1 0 (0.88888889 0.11111111) *
          57) chol< 253 17   4 1 (0.23529412 0.76470588) *
        29) sex< 0.5 18   2 1 (0.11111111 0.88888889) *
      15) age< 56.5 71   5 1 (0.07042254 0.92957746) *
```

After training the model, the output lists which variables the tree will take from the root to the branches with which values. The output needs to be visualized as it will be difficult to interpret.

4. Visualizing the Decision Tree

We can use the visualize rpart.plot() function.

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



We can compute the predicted values of the target variables in the test set using predict() function. The function we use is no different from the logistic regression model.

```
heart_dtm_predictions <- heart_dtm |>
  predict(new_data = heart_test)

heart_dtm_predictions
```

```
# A tibble: 61 x 1
   .pred_class
   <fct>
 1 0
 2 1
 3 1
 4 1
 5 1
 6 1
 7 1
 8 1
 9 1
10 1
# i 51 more rows
```

Make predictions by using the trained model. If you want to calculate the predicted probabilities, it is necessary to assign the `type` argument as `"prob"`.

```r
heart_dtm |>
  predict(new_data = heart_test,
          type     = "prob")
```

```
# A tibble: 61 x 2
   .pred_0 .pred_1
     <dbl>   <dbl>
 1  0.778    0.222
 2  0.0704   0.930
 3  0.0704   0.930
 4  0.111    0.889
 5  0.219    0.781
 6  0.0704   0.930
 7  0.0704   0.930
 8  0.219    0.781
 9  0.0704   0.930
10  0.0704   0.930
# i 51 more rows
```

5. Evaluating Model Performance

{yardstick} package is used to evaluate/measure the model performance. Its functions require a data.frame or tibble with model results.

```r
heart_results <- tibble(predicted = heart_dtm_predictions$.pred_class,
                        actual    = heart_test$output)
heart_results
```

```
# A tibble: 61 x 2
   predicted actual
   <fct>     <fct>
 1 0         1
 2 1         1
 3 1         1
 4 1         1
 5 1         1
 6 1         1
 7 1         1
 8 1         1
 9 1         1
10 1         1
```

```
# i 51 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.

```
heart_results |> conf_mat(truth    = actual,
                           estimate = predicted)
```

```
          Truth
Prediction  0  1
         0 20  2
         1  8 31
```

The `accuracy` metric can be calculated in similar manner by `accuracy()` function:

```
heart_results |> accuracy(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.836
```

We have a result 0.83. Their interpretation is the same as logistic regression.

The `sensitivity` metric can be calculated in similar manner by `sens()` function:

```
heart_results |> sens(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 sens    binary         0.714
```

We have a result 0.71. Their interpretation is the same as logistic regression.

```
heart_results |> spec(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 spec    binary         0.939
```

We have a result 0.93. Their interpretation is the same as logistic regression.

```
heart_results |> recall(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 recall  binary         0.714
```

We have a result 0.71. Their interpretation is the same as logistic regression.

```
heart_results |> precision(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric   .estimator .estimate
  <chr>     <chr>          <dbl>
1 precision binary         0.909
```

We have a result 0.90. Their interpretation is the same as logistic regression.

```
heart_results |> bal_accuracy(truth = actual, estimate = predicted)
```

```
# A tibble: 1 x 3
  .metric      .estimator .estimate
  <chr>        <chr>          <dbl>
1 bal_accuracy binary         0.827
```

We have a result 0.82. Their interpretation is the same as logistic regression.

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

```r
heart_dtm_last_fit <- heart_dt |>
  last_fit(output ~., split = heart_split)
heart_dtm_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 [242/61]> train/test split <tibble> <tibble> <tibble>     <workflow>
```

Collecting Metrics

After the model fitting, we can obtain the metrics and predictions using collect_metrics()and collect_predictions() functions

```r
heart_dtm_last_fit |> collect_metrics()
```

```
# A tibble: 2 x 4
  .metric  .estimator .estimate .config
  <chr>    <chr>          <dbl> <chr>
1 accuracy binary         0.836 Preprocessor1_Model1
2 roc_auc  binary         0.893 Preprocessor1_Model1
```

Collecting Predictions

```r
heart_dtm_last_fit |> collect_predictions()
```

```
# A tibble: 61 x 7
   id               .pred_0 .pred_1  .row .pred_class output .config
   <chr>              <dbl>   <dbl> <int> <fct>       <fct>  <chr>
 1 train/test split  0.778    0.222     2 0           1      Preprocessor1_Mode~
 2 train/test split  0.0704   0.930     3 1           1      Preprocessor1_Mode~
 3 train/test split  0.0704   0.930    12 1           1      Preprocessor1_Mode~
 4 train/test split  0.111    0.889    15 1           1      Preprocessor1_Mode~
 5 train/test split  0.219    0.781    19 1           1      Preprocessor1_Mode~
 6 train/test split  0.0704   0.930    28 1           1      Preprocessor1_Mode~
 7 train/test split  0.0704   0.930    38 1           1      Preprocessor1_Mode~
 8 train/test split  0.219    0.781    44 1           1      Preprocessor1_Mode~
```

```
 9 train/test split  0.0704   0.930    47 1           1      Preprocessor1_Mode~
10 train/test split  0.0704   0.930    49 1           1      Preprocessor1_Mode~
# i 51 more rows
```

#5. Train a Random Forest

Randomforest is often used the subset of features to train each tree. The main hyperparameters of randomforest are; -the number of trees - the number of features to consider at any given split: (mtry) - the complexity of each tree.

1. Train a Random Forest

we determined the number of features, namely mtry, as 8 according to the target variable in our train data and assigned it to heart_rfm.

```
set.seed(123)
heart_rfm <- ranger(output ~ .,
                    data = heart_train,
                    mtry = 8)
heart_rfm
```

```
Ranger result

Call:
 ranger(output ~ ., data = heart_train, mtry = 8)

Type:                             Classification
Number of trees:                  500
Sample size:                      242
Number of independent variables:  13
Mtry:                             8
Target node size:                 1
Variable importance mode:         none
Splitrule:                        gini
OOB prediction error:             21.49 %
```

heart_rfm includes the hyperparameters mentioned above. Since it would be better to obtain optimal value of the number of features at 10 times, the value that we have is 500. If we run the same code this time without specifying try, it will automatically specify try as 3.

```
set.seed(123)
heart_rfm_y <- ranger(output ~ .,
```

```
                      data = heart_train)
heart_rfm_y
```

```
Ranger result

Call:
 ranger(output ~ ., data = heart_train)

Type:                             Classification
Number of trees:                  500
Sample size:                      242
Number of independent variables:  13
Mtry:                             3
Target node size:                 1
Variable importance mode:         none
Splitrule:                        gini
OOB prediction error:             18.60 %
```

While we measure the performance of the model where we set mtry as 8 in the first code, we measure the performance of the model by using the automatically determined value in the second code. Accuracy increased slightly from 80% with mtry = 8 to 83% at 3. Likewise, their 19 balanced accuracy also led to a similar slight increase. In this case, mtry is a hyperparameter that affects the performance of the model.

```
preds_bt_heart <- predict(heart_rfm, heart_test)

confusionMatrix(preds_bt_heart$predictions,
                heart_test$output)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 19  3
         1  9 30

               Accuracy : 0.8033
                 95% CI : (0.6816, 0.894)
    No Information Rate : 0.541
    P-Value [Acc > NIR] : 1.767e-05
```

```
                 Kappa : 0.5974

 Mcnemar's Test P-Value : 0.1489

            Sensitivity : 0.6786
            Specificity : 0.9091
         Pos Pred Value : 0.8636
         Neg Pred Value : 0.7692
             Prevalence : 0.4590
         Detection Rate : 0.3115
   Detection Prevalence : 0.3607
      Balanced Accuracy : 0.7938

       'Positive' Class : 0
```

```
preds_rf_heart <- predict(heart_rfm_y, heart_test)

confusionMatrix(preds_rf_heart$predictions,
                heart_test$output)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 19  1
         1  9 32

               Accuracy : 0.8361
                 95% CI : (0.7191, 0.9185)
    No Information Rate : 0.541
    P-Value [Acc > NIR] : 1.184e-06

                  Kappa : 0.6626

 Mcnemar's Test P-Value : 0.02686

            Sensitivity : 0.6786
            Specificity : 0.9697
         Pos Pred Value : 0.9500
```

```
       Neg Pred Value : 0.7805
           Prevalence : 0.4590
       Detection Rate : 0.3115
 Detection Prevalence : 0.3279
    Balanced Accuracy : 0.8241

     'Positive' Class : 0
```

When we look at all three techniques, the accuracy of the model was 83% in the logistic regression, while it was 0.75 in the decision tree and 80% in the random forest technique.