**Direction of post**

In the next few posts, we will look at model specific post-hoc analysis which involves ranking the variables according to importance to the model. Though these interpretation can be applied on both white box and black box models, we will be explaining black box models in these posts as white box models can be explained by the models themselves. Specifically, we will explain random forest in this post and gradient boosting in future posts.

#library

library(tidyverse)

library(tidymodels)

#import

heart<-read\_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data", col\_names = F)

# Renaming var

colnames(heart)<- c("age", "sex", "rest\_cp", "rest\_bp",

"chol", "fast\_bloodsugar","rest\_ecg","ex\_maxHR","ex\_cp",

"ex\_STdepression\_dur", "ex\_STpeak","coloured\_vessels", "thalassemia","heart\_disease")

#elaborating cat var

##simple ifelse conversion

heart<-heart %>% mutate(sex= ifelse(sex=="1", "male", "female"),fast\_bloodsugar= ifelse(fast\_bloodsugar=="1", ">120", "<120"), ex\_cp=ifelse(ex\_cp=="1", "yes", "no"),

heart\_disease=ifelse(heart\_disease=="0", "no", "yes"))

heart<-heart %>% mutate(

rest\_cp=case\_when(rest\_cp== "1" ~ "typical",rest\_cp=="2" ~ "atypical", rest\_cp== "3" ~ "non-CP pain",rest\_cp== "4" ~ "asymptomatic"), rest\_ecg=case\_when(rest\_ecg=="0" ~ "normal",rest\_ecg=="1" ~ "ST-T abnorm",rest\_ecg=="2" ~ "LV hyperthrophy"), ex\_STpeak=case\_when(ex\_STpeak=="1" ~ "up/norm", ex\_STpeak== "2" ~ "flat",ex\_STpeak== "3" ~ "down"), thalassemia=case\_when(thalassemia=="3.0" ~ "norm",

thalassemia== "6.0" ~ "fixed", thalassemia== "7.0" ~ "reversable"))

# convert missing value "?" into NA

heart<-heart%>% mutate\_if(is.character, funs(replace(., .=="?", NA)))

# convert char into factors

heart<-heart %>% mutate\_if(is.character, as.factor)

#train/test set

set.seed(4595)

data\_split <- initial\_split(heart, prop=0.75, strata = "heart\_disease")

heart\_train <- training(data\_split)

heart\_test <- testing(data\_split)

# create recipe object

heart\_recipe<-recipe(heart\_disease ~., data= heart\_train) %>%

step\_knnimpute(all\_predictors()) %>%

step\_dummy(all\_nominal(), -heart\_disease)# need dummy variables to be created for some `randomForestexplainer` functions though random forest model itself doesnt need explicit one hot encoding

# process the traing set/ prepare recipe(non-cv)

heart\_prep <-heart\_recipe %>% prep(training = heart\_train, retain = TRUE)

**Random forest**

**Ensemble Models (tree-based)**

Random forests and gradient boosting are also examples of tree based ensemble models. In tree based ensemble models, multiple variants of a simple tree are combined to develop a more complex model with higher prediction power. The trade off of the ensemble is that the model becomes a black box and cannot be interpreted as simply as a decision tree. Thus, post-hoc analysis is required to explain the ensemble model.

**Random forest mechanism**

Random forest is a step up of bootstrap aggregating/ bagging and bootstrap aggregating is a step up of decision tree.  
In bagging, multiple bootstrap samples are of the training data are used to train multiple single regression tree. As bootstrap samples are used, the same observation from the training set can be used more than once to train a regression tree. For regression problems, the ensemble occurs when the predictions of each tree are averaged to provide the predicted value for the bagging model. For classification problems, the ensemble occurs when the class probabilities of each tree are averaged to provide the predicted class. Alternatively, the majority predicted class of the ensemble of trees will be the predicted class for the model.  
Random forest is similar to bootstrap aggregating except that a random subset of variables is used for each split. The number of variables in this random subset of variables is known as mtry. The default mtry in regression problems is a third of the total number of available variables in the dataset. The default mtry in classification problems is the square root of the total number of available variables.

**Training random forest models**

We will train two random forest where each model adopts a different ranking approach for feature importance. The two ranking measurements are:

1. Permutation based.  
   Permuting values in a variable decouples any relationship between the predictor and the outcome which renders the variable pseudo present in the model. When the out of bag sample is passed down the model with the permutated variable, the increased in error/ decreased in accuracy of the model can be calculated. This accuracy/error is compared against the baseline accuracy/error *(for regression problems, the measurement will be R2)* to calculate the importance score. This computation is repeated for all variables. The variable with the largest decrease in accuracy/largest increase in error is considered the most important variable because not considering the variable has the largest penalty on prediction. In the case of the randomForest package, the score measures the decrease in accuracy and this score is refer as the MeanDecreaseAccuracy.
2. Impurity based. Impurity refers to gini impurity/ gini index. The concept of impurity for random forest is the same as regression tree. Features which are more important have a lower impurity score/ higher purity score/ higher decrease in impurity score. The randomForest package, adopts the latter score which known as MeanDecreaseGini.

set.seed(69)

rf\_model<-rand\_forest(trees = 2000, mtry = 4, mode = "classification") %>% set\_engine("randomForest",

# importance = T to have permutation score calculated

importance=T,

# localImp=T for randomForestExplainer(next post)

localImp = T, ) %>% fit(heart\_disease ~ ., data = juice(heart\_prep))

**Feature Importance**

We can extract permutation based importance as well as gini based importance directly from the model trained by the randomForest package.

**Permutation-based importance**

Using the tidyverse approach to the extract results, remember to convert MeanDecreaseAccuracy from character to numeric form for arrange to sort the variables correctly. Otherwise, R will recognise the value based on the first digit while ignoring log/exp values. For instance, if MeanDecreaseAccuracy was in character format, rest\_ecg\_ST.T.abnorm will have the highest value as R recognises the first digit in 4.1293934444743e-05 and fails to recognise that e-05 renders that value close to 0.

cbind(rownames(rf\_model$fit$importance), rf\_model$fit$importance) %>% as\_tibble() %>% select(predictor= V1, MeanDecreaseAccuracy) %>%

mutate(MeanDecreaseAccuracy=as.numeric(MeanDecreaseAccuracy)) %>% arrange(desc((MeanDecreaseAccuracy))) %>% head()

## # A tibble: 6 x 2

## predictor MeanDecreaseAccuracy

##

## 1 ex\_STdepression\_dur 0.0365

## 2 thalassemia\_norm 0.0332

## 3 thalassemia\_reversable 0.0323

## 4 ex\_maxHR 0.0226

## 5 sex\_male 0.0111

## 6 coloured\_vessels\_X2.0 0.0109

As the model is trained using the randomForest package, we can use randomForest::importance to extract the results too. Use argument type=1 to extract permutation-based importance and set scale=F to prevent normalization.

cbind(rownames(randomForest::importance(rf\_model$fit, type=1, scale=F)) %>% as\_tibble(), randomForest::importance(rf\_model$fit, type=1, scale=F) %>% as\_tibble())%>% arrange(desc(MeanDecreaseAccuracy)) %>% head()

## value MeanDecreaseAccuracy

## 1 ex\_STdepression\_dur 0.03651391

## 2 thalassemia\_norm 0.03321383

## 3 thalassemia\_reversable 0.03231399

## 4 ex\_maxHR 0.02256749

## 5 sex\_male 0.01106189

## 6 coloured\_vessels\_X2.0 0.01087665

**Impurity-based importance**

Likewise, we can use both approaches to calculate impurity-based importance. Using the tidyverse way:

cbind(rownames(rf\_model$fit$importance), rf\_model$fit$importance) %>% as\_tibble() %>% select(predictor= V1, MeanDecreaseGini) %>%

mutate(MeanDecreaseGini=as.numeric(MeanDecreaseGini)) %>% arrange(desc((MeanDecreaseGini))) %>% head(10)

## # A tibble: 10 x 2

## predictor MeanDecreaseGini

##

## 1 ex\_maxHR 14.9

## 2 ex\_STdepression\_dur 14.5

## 3 thalassemia\_norm 11.1

## 4 thalassemia\_reversable 9.90

## 5 rest\_bp 8.84

## 6 age 8.69

## 7 chol 8.00

## 8 ex\_cp\_yes 5.56

## 9 coloured\_vessels\_X2.0 3.77

## 10 rest\_cp\_typical 3.34

For the randomForest::importance function, we will change the argument type to 2. The scale argument does not work for impurity-based scores.

cbind(rownames(randomForest::importance(rf\_model$fit, type=2)) %>% as\_tibble(), randomForest::importance(rf\_model$fit, type=2) %>% as\_tibble())%>% arrange(desc(MeanDecreaseGini)) %>% head(10)

## value MeanDecreaseGini

## 1 ex\_maxHR 14.850652

## 2 ex\_STdepression\_dur 14.537241

## 3 thalassemia\_norm 11.128131

## 4 thalassemia\_reversable 9.902711

## 5 rest\_bp 8.838706

## 6 age 8.686010

## 7 chol 7.997923

## 8 ex\_cp\_yes 5.557652

## 9 coloured\_vessels\_X2.0 3.769974

## 10 rest\_cp\_typical 3.342365

**Comments**

The length of the script for the randomForest::importance is shorter but users need to be familiar with the details of the package and the functions. On the other hand, the tidyverse approach is more verbose but it is user friendly to everyday R users.