**Recap**

We’ve covered various approaches in explaining model predictions globally. Today we will learn about another model specific post hoc analysis. We will learn to understand the workings of gradient boosting predictions.

Like past posts, the Clevaland heart dataset as well as tidymodels principle will be used. Refer to the [first post of this series for more details](https://notast.netlify.com/post/explaining-predictions-interpretable-models-logistic-regression/).

**Gradient Boosting**

Besides [random forest introduced in a past post](https://notast.netlify.com/post/explaining-predictions-random-forest-post-hoc-analysis-permutation-impurity-variable-importance/), another tree-based ensemble model is gradient boosting. In gradient boosting, a shallow and weak tree is first trained and then the next tree is trained based on the errors of the first tree. The process continues with **a** new tree being sequentially added to the ensemble and the new successive tree improves on the errors of the ensemble of preceding trees. On the hand, random forest is an ensemble of deep independent trees.

#library

library(tidyverse)

library(tidymodels)

theme\_set(theme\_minimal())

#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")) #remember to leave it as numeric for DALEX

## complex ifelse conversion using `case\_when`

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)

The gradient boosting package which we’ll use is xgboost. xgboost only accepts numeric values thus one-hot encoding is required for categorical variables. However, I was still able to train a xgboost model without one-hot encoding when I used the parsnip interface.

# create recipe object

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

step\_knnimpute(all\_predictors())

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

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

No tunning was done, the hyperparameters are default settings which were made explicit.

# boosted tree model

bt\_model<-boost\_tree(learn\_rate=0.3, trees = 100, tree\_depth= 6, min\_n=1, sample\_size=1, mode="classification") %>% set\_engine("xgboost", verbose=2) %>% fit(heart\_disease ~ ., data = juice(heart\_prep))

**Feature Importance (global level)**

The resulting gradient boosting model bt\_model$fit represented as a parsnip object does not inherently contain feature importance unlike a [random forest model represented as a parsnip object](https://notast.netlify.com/post/explaining-predictions-random-forest-post-hoc-analysis-permutation-impurity-variable-importance/).

summary(bt\_model$fit)

## Length Class Mode

## handle 1 xgb.Booster.handle externalptr

## raw 66756 -none- raw

## niter 1 -none- numeric

## call 7 -none- call

## params 9 -none- list

## callbacks 1 -none- list

## feature\_names 20 -none- character

## nfeatures 1 -none- numeric

We can extract the important features from the boosted tree model with xgboost::xgb.importance. Although we did the pre-processing and modelling using tidymodels, we ended up using the original Xgboost package to explain the model. Perhaps, tidymodels could consider integrating prediction explanation for more models that they support in the future.

library(xgboost)

xgb.importance(model=bt\_model$fit) %>% head()

## Feature Gain Cover Frequency

## 1: thalassemianorm 0.24124439 0.05772889 0.01966717

## 2: ex\_STdepression\_dur 0.17320374 0.15985018 0.15279879

## 3: ex\_maxHR 0.10147873 0.12927719 0.13615734

## 4: age 0.07165646 0.09136876 0.12859304

## 5: chol 0.06522754 0.10151576 0.15431165

## 6: rest\_bp 0.06149660 0.09178222 0.11497731

**Variable importance score**

Feature importance are computed using three different importance scores.

1. Gain: Gain is the relative contribution of the corresponding feature to the model calculated by taking each feature’s contribution for each tree in the model. A higher score suggests the feature is more important in the boosted tree’s prediction.
2. Cover: Cover is the relative observations associated with a predictor. For example, feature X is used to determine the terminal node for 10 observations in tree A and 20 observations in tree B. The absolute observations associated with feature X is 30 and the relative observation is 30/sum of all absolute observation for all features.
3. Frequency: Frequency refers to the relative frequency a variable occurs in the ensembled of trees. For instance, feature X occurs in 1 split in tree A and 2 splits in tree B. The absolute occurrence of feature X is 3 and the (relative) frequency is 3/sum of all absolute occurrence for all features.

[Category variables especially those with minimal cardinality will have low frequency score as these variables are seldom used in each tree. Compared to continuous variables or to some extend category variables with high cardinality as they have are likely to have a larger range of values which increases the odds of occurring the model](https://towardsdatascience.com/be-careful-when-interpreting-your-features-importance-in-xgboost-6e16132588e7). Thus, [the developers of xgboost discourage using frequency score unless you’re clear about your rationale for selecting frequency as the feature importance score](https://xgboost.readthedocs.io/en/latest/R-package/discoverYourData.html). Rather, [gain score is the most valuable score to determine variable importance](https://towardsdatascience.com/be-careful-when-interpreting-your-features-importance-in-xgboost-6e16132588e7). xgb.importance selects gain score as the fault measurement and arranges features according to the descending value of gain score resulting in the most important feature to be displayed at the top.

**Plotting variable importance**

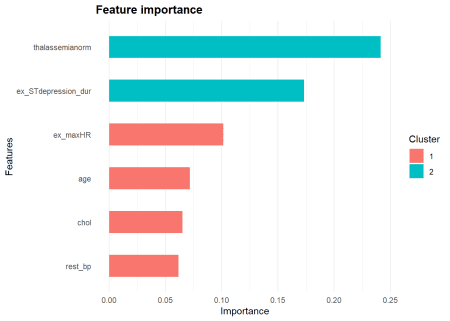
xgbosst provides two options to plot variable importance.

1. Using basic R graphics via xgb.plot.importance
2. Using ggplot interface via xgb.ggplot.importance. I’ll be using the latter.

The xgb.ggplot.importance uses the gain variable importance measurement by default to calculate variable importance. The default argument measure=NULL can be changed to use other variable importance measurements. However, based on the previous section, it will be wiser to leave the argument as the default.  
The xgb.ggplot.importance graph also displays the cluster of variables that have similar variable importance scores. The xgb.ggplot.importance graph displays each variable’s gain score as a relative contribution to the overall model importance by default. The sum of all the gain scores will equal to 1.

xgb.importance(model=bt\_model$fit) %>% xgb.ggplot.importance(

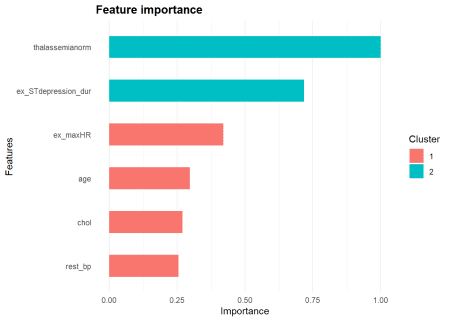
top\_n=6, measure=NULL, rel\_to\_first = F)



Alternatively, the gain scores can be presented as relative scores to the most important feature. In this case, the most importance feature will have a score of 1 and the gain scores of the other variables will be scaled to the gain score of the most important feature. This alternate demonstration of gain score can be achieved by changing the default argument rel\_to\_first=F to rel\_to\_first=T.

xgb.importance(model=bt\_model$fit) %>% xgb.ggplot.importance(

top\_n=6, measure=NULL, rel\_to\_first = T)



**Sum up**

This is the last post of this series looking at explaining model predictions at a global level. We first started this series explaining predictions using white box models such as [logistic regression](https://notast.netlify.com/post/explaining-predictions-interpretable-models-logistic-regression/) and [decision tree](https://notast.netlify.com/post/explaining-predictions-interpretable-models-decision-tree/). Next, we did model specific post hoc evaluation on black box models. Specifically, for [random forest](https://notast.netlify.com/post/explaining-predictions-random-forest-post-hoc-analysis-randomforestexplainer-package/) and Xgboost.