**What is SHAP?**

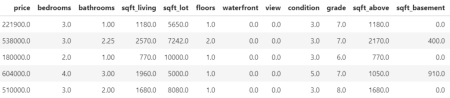
A couple of years ago, the concept of Shapely values from game theory from the 1950ies was discovered e.g. by Scott Lundberg as an interesting approach to explain predictions of ML models.

The basic idea is to decompose a prediction in a fair way into additive contributions of features. Repeating the process for many predictions provides a brilliant way to investigate the model as a whole.

**Illustration**

As an example, we will try to model log house prices of 20’000 sold houses in Kings County. The dataset is available e.g. on [OpenML.org](http://openml.org/) under ID 42092.

Some rows and columns from the Kings County house dataset.



**Fetch and prepare data**

We start by downloading the data and preparing it for modelling.

R

library(farff)

library(OpenML)

library(dplyr)

library(xgboost)

library(ggplot2)

library(SHAPforxgboost)

# Load King Country house prices dataset on OpenML

# ID 42092, <https://www.openml.org/d/42092>

df <- getOMLDataSet([data.id](http://data.id) = 42092)$data

head(df)

# Prepare

df <- df %>%

mutate(

log\_price = log(price),

log\_sqft\_lot = log(sqft\_lot),

year = as.numeric(substr(date, 1, 4)),

building\_age = year - yr\_built,

zipcode = as.integer(as.character(zipcode))

)

# Define response and features

y <- "log\_price"

x <- c("grade", "year", "building\_age", "sqft\_living",

"log\_sqft\_lot", "bedrooms", "bathrooms", "floors", "zipcode",

"lat", "long", "condition", "waterfront")

# random split

set.seed(83454)

ix <- sample(nrow(df), 0.8 \* nrow(df))

**Fit XGBoost model**

Next, we fit a manually tuned XGBoost model to the data.

R

dtrain <- xgb.DMatrix(data.matrix(df[ix, x]),

label = df[ix, y])

dvalid <- xgb.DMatrix(data.matrix(df[-ix, x]),

label = df[-ix, y])

params <- list(

objective = "reg:squarederror",

learning\_rate = 0.05,

subsample = 0.9,

colsample\_bynode = 1,

reg\_lambda = 2,

max\_depth = 5

)

fit\_xgb <- xgb.train(

params,

data = dtrain,

watchlist = list(valid = dvalid),

early\_stopping\_rounds = 20,

print\_every\_n = 100,

nrounds = 10000 # early stopping

)

The resulting model consists of about 600 trees and reaches a validation RMSE of 0.16. This means that about 2/3 of the predictions are within 16% of the observed price, using the [empirical rule](https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule).

**Compact SHAP analysis**

ML models are rarely of any use without interpreting its results, so let's use SHAP to peak into the model.

The analysis includes a first plot with SHAP importances. Then, with decreasing importance, dependence plots are shown to get an impression on the effects of each feature.

R

# Step 1: Select some observations

X <- data.matrix(df[sample(nrow(df), 1000), x])

# Step 2: Crunch SHAP values

shap <- shap.prep(fit\_xgb, X\_train = X)

# Step 3: SHAP importance

shap.plot.summary(shap)

# Step 4: Loop over dependence plots in decreasing importance

for (v in shap.importance(shap, names\_only = TRUE)) {

p <- shap.plot.dependence(shap, v, color\_feature = "auto",

alpha = 0.5, jitter\_width = 0.1) +

ggtitle(v)

print(p)

}

Some of the plots are shown below. The code actually produces all plots,

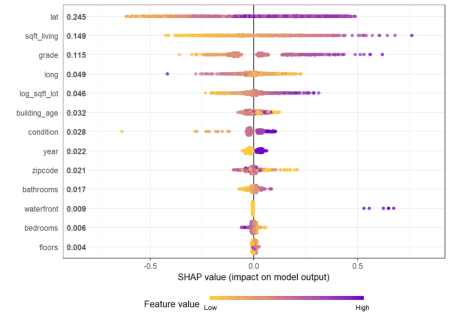
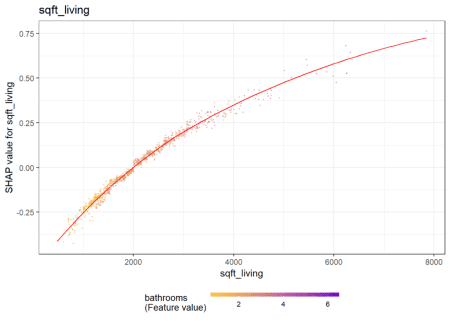


Figure 1: SHAP importance for XGBoost model.



The results make intuitive sense. Location and size are among the strongest predictors.

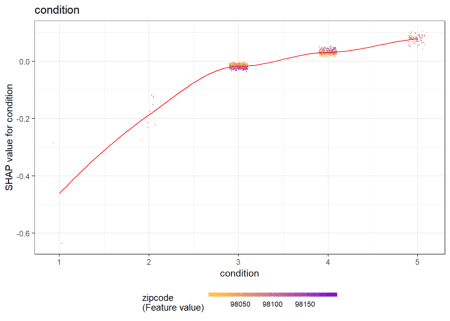
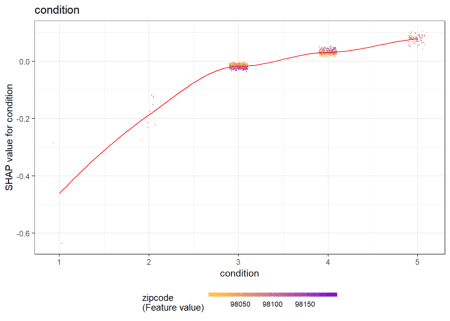
Figure 2: SHAP dependence for the second strongest predictor.   


Figure 3: SHAP dependence for a less important predictor.



The larger the living area, the higher the log price. There is not much vertical scatter, indicating that living area acts quite additively on the predictions on the log scale. The effect of "condition" 4 vs 3 seems to depend on the zipcode (see the color). For some zipcodes, the condition does not have a big effect on the price, while for other zipcodes, the effect is clearly higher.

**Same workflow for LightGBM**

Let's try out the SHAPforxgboost package with LightGBM.

*Note: LightGBM Version 3.2.1 on CRAN is not working properly under Windows. This will be fixed in the next release of LightGBM. As a temporary solution, you need to build it from the current* [*master branch*](https://github.com/microsoft/LightGBM).

R

library(lightgbm)

dtrain <- lgb.Dataset(data.matrix(df[ix, x]),

label = df[ix, y])

dvalid <- lgb.Dataset(data.matrix(df[-ix, x]),

label = df[-ix, y])

params <- list(

objective = "regression",

learning\_rate = 0.05,

subsample = 0.9,

reg\_lambda = 2,

num\_leaves = 15

)

fit\_lgb <- lgb.train(

params,

data = dtrain,

valids = list(valid = dvalid),

early\_stopping\_rounds = 20,

eval\_freq = 100,

eval = "rmse",

nrounds = 10000

)

Early stopping on the validation data selects about 900 trees as being optimal and results in a validation RMSE of also 0.16.

**SHAP analysis**

We use exactly the same short snippet to analyze the model by SHAP.

R

X <- data.matrix(df[sample(nrow(df), 1000), x])

shap <- shap.prep(fit\_lgb, X\_train = X)

shap.plot.summary(shap)

for (v in shap.importance(shap, names\_only = TRUE)) {

p <- shap.plot.dependence(shap, v, color\_feature = "auto",

alpha = 0.5, jitter\_width = 0.1) +

ggtitle(v)

print(p)

}

Again, we only show some of the output and refer to the [html](https://htmlpreview.github.io/?https://github.com/lorentzenchr/notebooks/blob/master/blogposts/2021-06-23-shapforxgboost.html) of the corresponding rmarkdown. Overall, the model seems to be very similar to the one obtained by XGBoost.

Figure 4: SHAP importance for LightGBM. By chance, the order of importance is the same as for XGBoost.

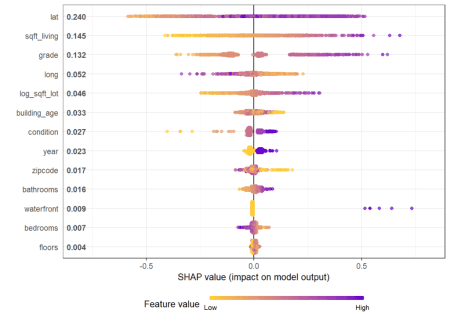
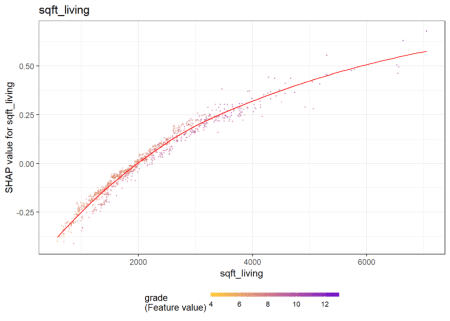


Figure 5: The dependence plot for the living area also looks identical in shape than for the XGBoost model.



**How does the dependence plot selects the color variable?**

By default, Scott's [shap](https://github.com/slundberg/shap) package for Python uses a statistical heuristic to colorize the points in the dependence plot by the variable with possibly strongest interaction. The heuristic used by SHAPforxgboost is slightly different and directly uses conditional variances. More specifically, the variable X on the x-axis as well as each other feature Z\_k is binned into categories. Then, for each Z\_k, the conditional variance across binned X and Z\_k is calculated. The Z\_k with the highest conditional variance is selected as the color variable.

Note that the heuristic does not depend on "shap interaction values" in order to save time (and because these would not be available for LightGBM).

The following simple example shows how/that it is working. First, a dataset is created and a model with three features and strong interaction between x1 and x2 is being fitted. Then, we look at the dependence plots to see if it is consistent with the model/data situation.

n <- 1000

set.seed(334)

df <- data.frame(

x1 = runif(n),

x2 = runif(n),

x3 = runif(n)

) %>%

mutate(

y = x1 \* x2 + x3 + runif(n)

)

x <- c("x1", "x2", "x3")

dtrain <- lgb.Dataset(data.matrix(df[, x]),

label = df[, "y"])

params <- list(

objective = "regression",

learning\_rate = 0.05,

subsample = 0.9,

reg\_lambda = 2,

num\_leaves = 15

)

fit\_lgb <- lgb.train(

params,

data = dtrain,

eval = "rmse",

nrounds = 100

)

shap <- shap.prep(fit\_lgb, X\_train = data.matrix(df[, x]))

shap.plot.summary(shap)

shap.plot.dependence(shap, "x1", color\_feature = "auto")

shap.plot.dependence(shap, "x2", color\_feature = "auto")

shap.plot.dependence(shap, "x3", color\_feature = "auto")

Here the dependence plots for x1 and x3.

Figure 6: The dependence plots for x1 shows a clear interaction effect with the color variable x2. This is as simulated in the data.

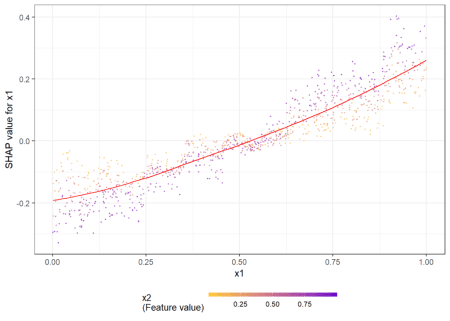


Figure 7: The dependence plots for x3 does not show clear interaction effects, consistent with the data situation.

