DDMO Project: Tuning XGBoost with SPOT on the Census Data Set

Larissa Melo (11145746) Mirco Friedrichs (11094675)
Andrey Domnyshev (11146992) Anantharaman Iyer (11147113)

Automation & IT (Master's program)

Abstract

The project is dedicated to finding upper, lower bounds and transformations for XGBoost hyperparameters which are tuned with SPOT. XGBoost is used to build a model for classification task using the "Census Income KDD" data set. We described our approach to determine bounds experimentally and using theoretical information about hyperparameters and optimization algorithm XGBoost. We proved the stability of the result using different seeds and input data. The visualisation of SPOT tuning and files with settings and results are provided.

1. Introduction

The current project work is dedicated to finding parameters of the `SPOT` optimizer for tuning the `XGBoost` algorithm's hyperparameters. `XGBoost` is used to build a model for classification task using the "Census Income KDD" data set.

1.1. Data

For our project we used the "Census Income KDD" data set, that can be downloaded to the Rstudio environment from the OpenML library:

```
library("OpenML")
dirName= "oml.cache"
if (! (dir.exists(dirName))) dir.create(dirName)
setOMLConfig(apikey = "c1994bdb7ecb3c6f3c8f3b35f4b47f1f", cachedir = dirName)
dataOML <- getOMLDataSet(4535)$data
```

The data represent the results from the population surveys for 1994 and 1995, conducted by the U.S. Census Bureau. It has 41 features related to people's personal (like age, gender, country of origin, etc.) and social (education, citizenship, marital status, etc.) attributes. The last 42nd feature is the level of income: upper $50 000 or lower. This feature that has only 2 unique values is chosen as the target of prediction. The data set contains 299 285 observations.

1.2. The Algorithm

`XGBoost` (Extreme Gradient Boosting) is used for supervised learning problems. The same as the Random forest technique Gradient Boosting creates decision trees to solve regression or classification problems. The difference is how the weights on these models are trained. On Random forest, we build a large number of trees

and then create a model, that chooses trees, that leads to a smaller value of an objective function. Gradient Boosting creates trees sequantionaly: after each step, the algorithm creates a tree that tries to decrease the residuals of the observations target value (difference between real and predicted values for regression and difference between 1 or 0 and found probability for classification).

Introduction into Boosted Trees for classification problems can be found in [1] and `XGBoost` application tutorials in [2].

In the current project, we optimize 9 hyperparameters of the `XGBoost` algorithm. The short description, range, practical importance, interconnection with other parameters is given in the project description [3].

- *nrounds* specifies the number of boosting steps (integer);
- *eta* is learning rate (double, $eta \in [0, 1]$);
- *lambda* is used for the regularization of the model (double, $lambda \in [0, +\infty]$);
- *alpha* is used for the regularization of the model (double, $alpha \in [0, +\infty]$);
- *subsample* specifies the portion of the data that is randomly selected in each iteration (double, $subsample \in [0, 1]$);
- *colsample_bytree* is a number of features is chosen for the splits of a tree (double, $colsample\_bytree \in [0, 1]$);
- *gamma* controls the number of splits of a tree by assuming a minimal improvement for each split (double, $gamma \in [0, +\infty]$);
- *max_depth* is the maximum depth (number of levels) of single decision tree (integer);
- *min_child_weight* restricts the number of splits of each tree (double, $min\_child\_weight \in [0, +\infty]$).

1.3. The Optimizer

On the project, we were suggested to use Sequential Parameter Optimization Toolbox (`SPOT`) to tune the `XGBoost` algorithm (fig. 1). `SPOT` is a set of tools for model-based optimization (we used the previous semester) and tuning of algorithms (we currently use) [4].

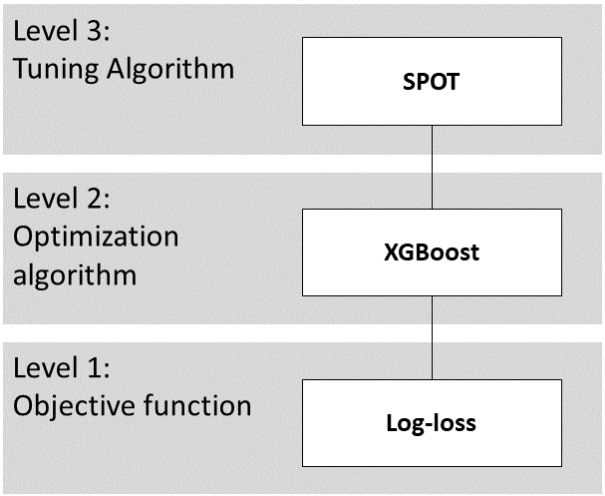


Figure 1: Structure of the project

The Log-loss function is used as an objective function and it represents how close the predicted probability of each entry of data to be classified into exact class to the corresponding true value. Our task is to provide a binary classification problem, so actual (true) values can be only 0 or 1. The more the predicted probability diverges from the actual value, the higher is the log-loss value. [5]

We have specified model performance measures into the experiment configuration section:

```
if(model=="xgboost"){
   tunepars <- c("nrounds","eta","lambda","alpha","subsample","colsample_bytree","gamma",
                 "max_depth","min_child_weight")
   lower <- c( 1,  0.4,  -4, -4,  0.8,   0.2,  -4,   3,  -4)
   upper <- c(200,   1,   9,  7,    1,   0.8,   4,  15,   7)
   type <-  c("integer","numeric","numeric","numeric","numeric","numeric","numeric",
              "integer","numeric")
   if(task.type=="classif") {fixpars <- list(eval_metric="logloss", nthread=1)}
   else {fixpars <- list(eval_metric="rmse", nthread=1)}
   factorlevels <- list()
   transformations <- c(trans_id, trans_id, trans_2pow, trans_2pow, trans_id, trans_id,
                        trans_2pow, trans_id, trans_2pow)
   dummy=TRUE
   relpars <- list()
   }
```

On the chunk above we also specified the lower and upper bounds of `XGBoost` hyperparameters and their transformations - scaling of the hyperparameters.

2. Materials and Methods

2.1. Project restrictions specification

The task of tuning hyperparameters of the optimization algorithm could include a lot of experiments and investigation, but we were proposed to work inside some restrictions.

- We use only data set, objective function, optimization algorithm, and tuning algorithm as specified in the introduction.
- We are allowed to use the full data set or its sample. We can specify the way of data sampling, but for all experiments, we kept a 60% - 40% split:

```
rsmpl <- makeResampleDesc("Holdout", split=0.6)
```

- We are allowed to use any time budget for tuning that is less than 15 hours.

```
timebudget <- 15*3600
timeout <- timebudget/20
```

- We are allowed to assign any bounds and transformations for tuned hyperparameters. The only restriction is the mathematical bounds specified in the project description [3].

```
tunepars <- c("nrounds","eta","lambda","alpha","subsample","colsample_bytree","gamma",
              "max_depth","min_child_weight")
lower <- c( 1,  0.4,  -4, -4,  0.8,   0.2,  -4,   3,  -4)
upper <- c(200,   1,   9,  7,    1,   0.8,   4,  15,   7)
transformations <- c(trans_id, trans_id, trans_2pow, trans_2pow, trans_id, trans_id,
                     trans_2pow, trans_id, trans_2pow)
```

- Changes in the `SPOT` configuration are strongly not recommended. We change only $seedFun$ and $seedSPOT$ parameters.

2.2. Experiments organisation

All our trials history can be divided into 3 phases:

- **Initial experiments.** We tried to understand which settings parameters affect the value of the objective function, how hyperparameters of `XGBoost` interconnected, and which hyperparameters' range leads to best algorithm performance.
- **Finding the best settings parameters.** The hyperparameters' bounds are the targets of experiments. We used recommendations from project description [3] as a guideline and we used examples from literature about `XGBoost` tuning as a starting point. The general approach can be described as follow: *The hyperparameter value that leads to better results of objective function should not lie down on the bounds, but somewhere in the middle. The range of bounds should be as smaller as possible.*
- **Prove the stability of the result.** During the previous steps we noticed how huge impact to the result of objective function create seed or input data changing. In this step, we estimate not only the single result value of the objective function but also which dispersion it has. We introduced the loop inside the given project code to be able to provide several experiments with different values of *set.seed*, $seedFun$ and $seedSPOT$ parameters:

```
# Number of loops
maxi=5

#initialize result vectors
ybest_list<- NULL
xbest_list<- list(NULL)
result_list<- list(NULL)
for (index in 1:maxi) {
   ybest_list[index] <- NULL
   xbest_list[index] <- list(NULL)
   result_list[index]<- list(NULL)
}

# Our Tuning Run with SPOT
for (index in 1:maxi) {
   set.seed(2*index) #in each loop we have different seed
   result <- spot(fun = objf,
                  lower=cfg$lower,
                  upper=cfg$upper,
                  control = list(types=cfg$type,
                                 maxTime = timebudget/60, #convert to minutes
                                 plots=TRUE,
                                 progress = TRUE,
                                 model=buildKriging,
                                 optimizer=optimDE,
                                 noise=TRUE,
                                 seedFun=index, #in each loop we have different seed
                                 seedSPOT=tuner.seed+index, #in each loop we have different seed
                                 designControl=list(size=5*length(cfg$lower)),
                                 funEvals=Inf,
                                 modelControl=list(target="y",
                                                   useLambda=TRUE,
                                                   reinterpolate=TRUE),
                                 optimizerControl=list(funEvals=100*length(cfg$lower))
```

```
                    )
    )
    ybest_list[index]<-result$ybest
    xbest_list[[index]]<-result$xbest
    result_list[[index]]<-result
}
```

- **Improve the speed of search with specific transformation.** In case of limited time budget, the improvement of `SPOT` searching speed improves the result of the objective function. We implemented transformation *trans_2pow* for the not integer hyperparameters, which are distributed near zero. With the usage of power transformation of hyperparameters, `SPOT` spends less attention for funding optimum in the high ranges of hyperparameters.

3. Results

3.1. Results of experiments with time budget for tuning

In the current section, we present the result of the investigation of how different settings affect to result of the objective function. One of the first experiments was conducted with different values of the *timebudget* parameter, which represents the overall budget for tuning in seconds (table 1, fig. 2 and 3).

Table 1. Settings for first initial experiments

Parameter	*Values*
time budget	300, 3000
data.seed	1
tuner.seed	1
dataOML	dataOML
lower	c(1, 2^(-10), 2^(-10), 2^(-10), 0.1, 0, 2^(-7), 3, 1)
upper	c(10, 1, 2^(10), 2^(10), 1, 1, 2^(6), 200, 2^(7))
transformations	c(trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id)
seedFun	123
seedSPOT	1

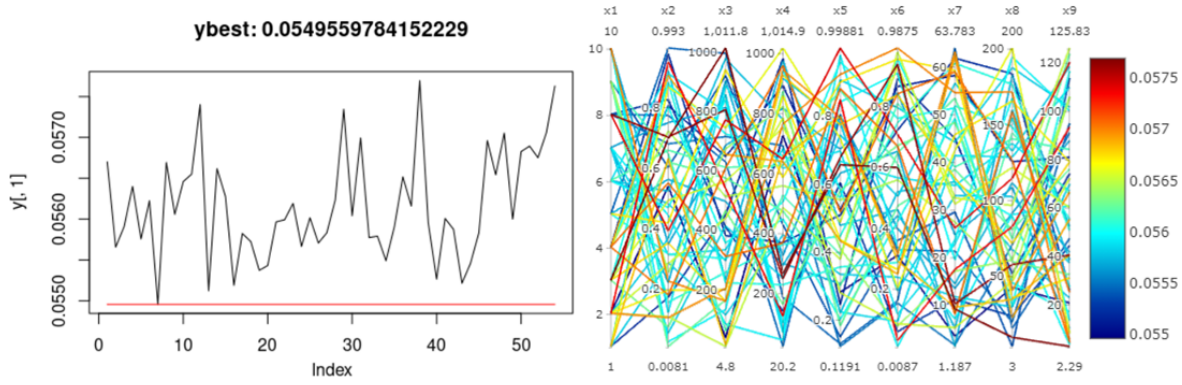


Figure 2: Time budget for tuning: 300 sec

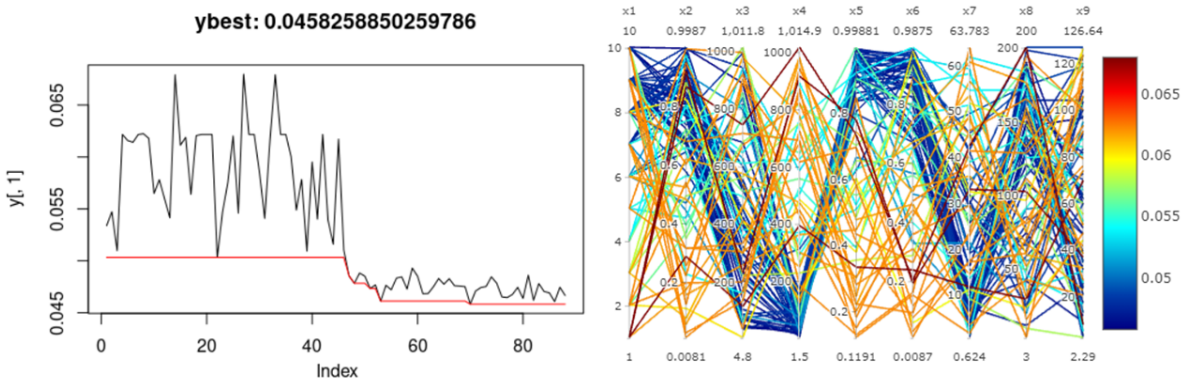


Figure 3: Time budget for tuning: 3000 sec

We notice, that productive tuning has started after approximately 1500 seconds, and obtained results could be used to decide on how bounds of hyperparameters could be modified.

We continue to increase the time budget, the information about the next 4 experiments (fig. 4, 5, 6, and 7) is described in table 2. The ranges of hyperparameters' bound are large.

Table 2. Settings for experiments with time budget

Parameter	*Values*
time budget	$5 * 3600$, $10 * 3600$, $13 * 3600$, $15 * 3600$
data.seed	1
tuner.seed	1
dataOML	dataOML
lower	c(1, 2^(-10), 2^(-10), 2^(-10), 0.1, 0, 2^(-7), 3, 1)
upper	c(200, 1, 2^(10), 2^(10), 1, 1, 2^(6), 20, 2^(7))
transformations	c(trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id)
seedFun	1
seedSPOT	2

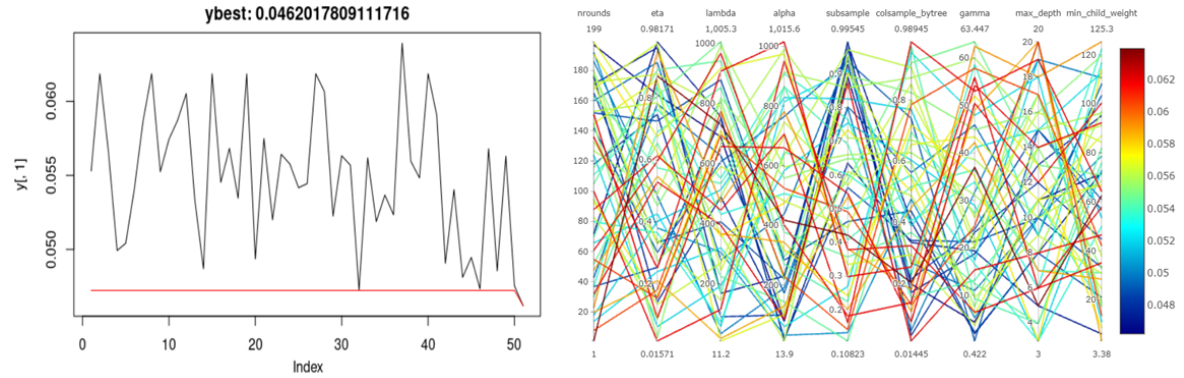


Figure 4: Time budget for tuning: 5*3600 sec

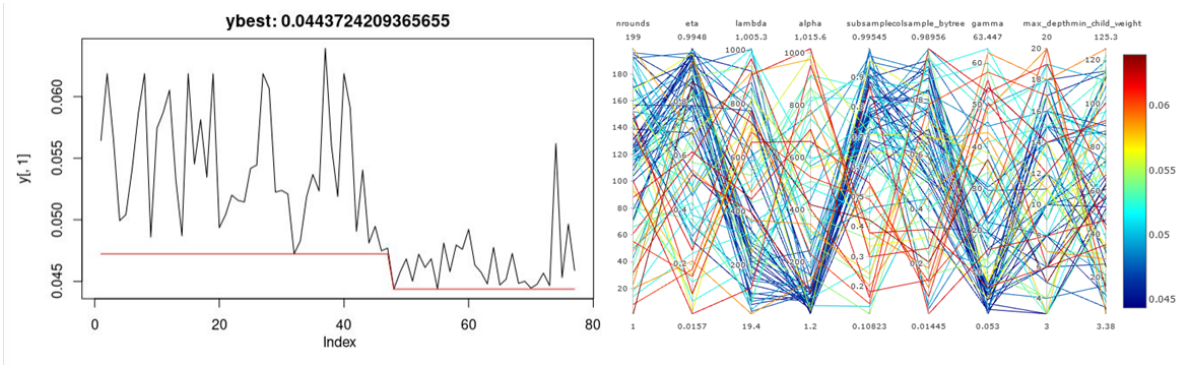


Figure 5: Time budget for tuning: 10*3600 sec

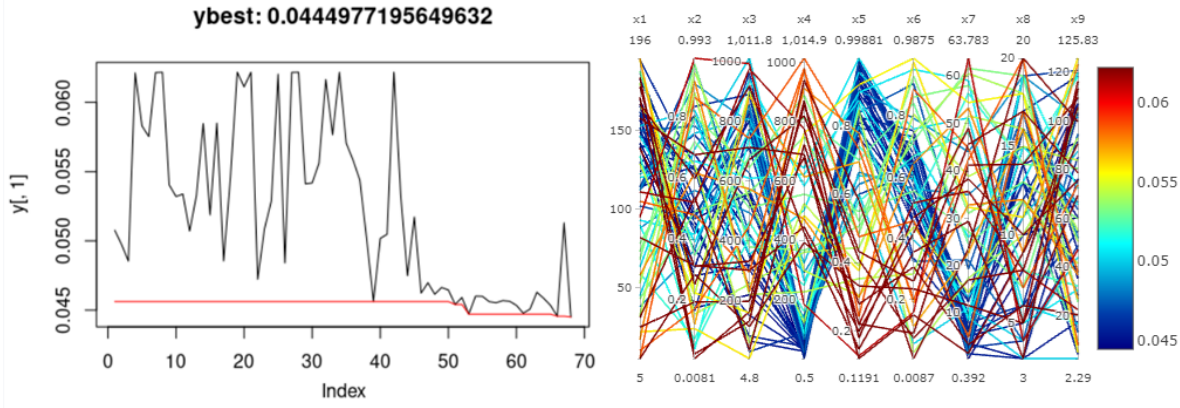


Figure 6: Time budget for tuning: 13*3600 sec

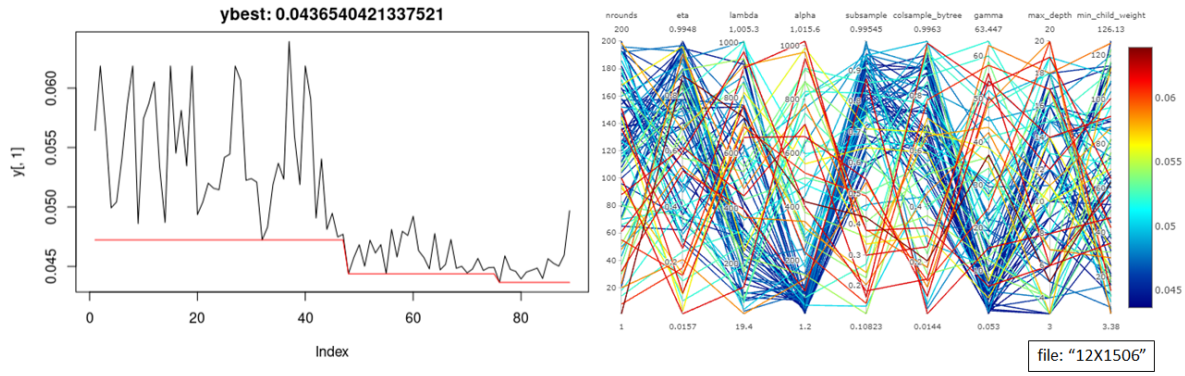


Figure 7: Time budget for tuning: 15*3600 sec

3.2. Large bounds range

The first idea that we have is to assign a very large bounds range for hyperparameters and a big time budget for tuning. The next experiment (table 3, fig. 8) shows, that this approach is not useful.

Table 3. Settings for the experiment with time large bounds range and big time budget for tuning

Parameter	*Values*
time budget	$15 * 3600$
data.seed	1
tuner.seed	1
dataOML	dataOML
lower	c(1, 2^(-10), 2^(-10), 2^(-10), 0,2^(-10), 2^(-10), 3, 2^(-10))
upper	c(10^6, 1, 2^(10), 2^(10), 1, 1, 2^(10), 20, 2^(10))
transformations	c(trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id)
seedFun	1
seedSPOT	2

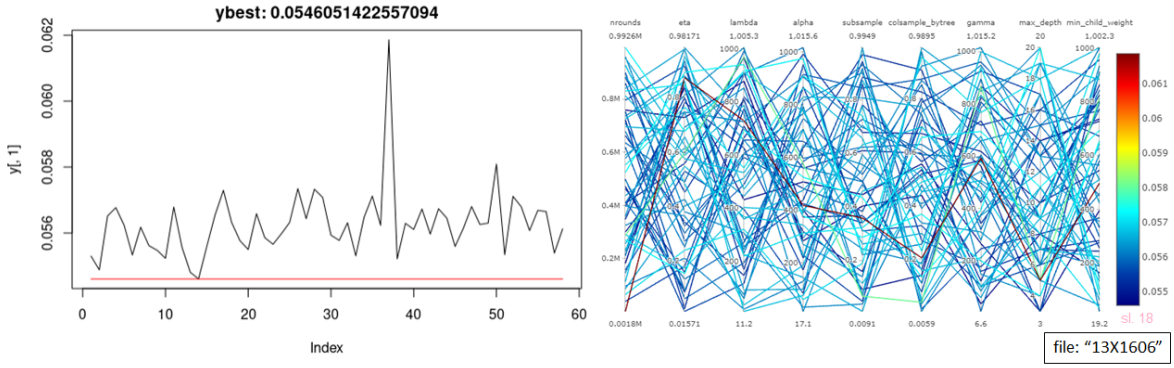


Figure 8: Time budget for tuning: 15*3600 sec

Even though we assign a very large time budget for tuning, `SPOT` has not even started "productive" optimization (compare with a plateau on fig. 7 after $index > 50$), because it spent all time trying different hyperparameters' values in different regions of a large range of bounds.

3.3. Find best bounds of hyperparameters

In the experiments organization section we explained the general approach that we follow: *The hyperparameter value that leads to better results of objective function should not lie down on the bounds, but somewhere in the middle. The range of bounds should be as smaller as possible.* Let us illustrate how we applied it with several experiments.

3.3.1. An example of a possible expansion of boundaries

In these experiments (table 4), we have exactly the same settings, but different upper bound for *nrounds*. In the first experiment, most *nrounds* values that lead to the best results are located on the upper bound (fig. 9).

Table 4. Settings for experiments with varying nrounds

Parameter	*Values*
time budget	3600
data.seed	1
tuner.seed	1
dataOML	dataOML
lower	c(1, 0.4, 1, 1, 0.8, 0.2, 1, 3, 2)
upper	c(**VARY**, 1, 300, 70, 1, 0.8, 10, 15, 110)
transformations	c(trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id)
seedFun	1
seedSPOT	2

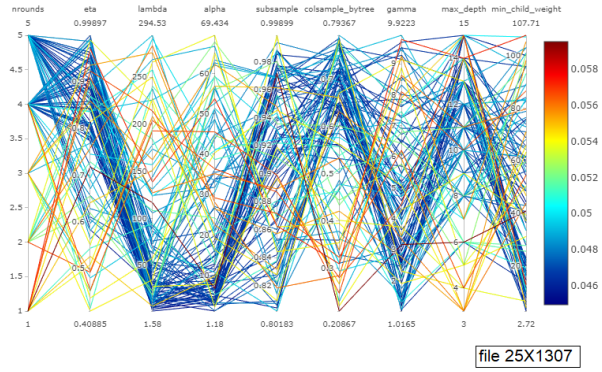


Figure 9: Experiments with varying nrounds: upper bound of $nrounds = 5$

```
load("results/25X1307.Rdata")
result_list[[1]]$ybest
```

```
##              [,1]
## [1,] 0.04494044
```

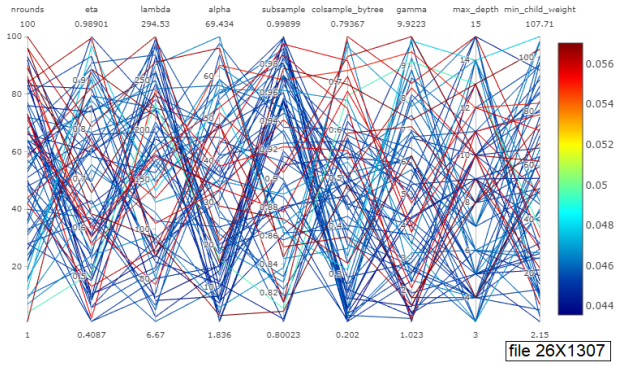


Figure 10: Experiments with varying nrounds: upper bound of $nrounds = 100$

```
load("results/26X1307.Rdata")
result_list[[1]]$ybest
```

```
##             [,1]
## [1,] 0.0435538
```

We notice, how the variation of a single bound can significantly affect the value of the objective function. As a matter of fact, sensitivity for hyperparameter *nround* is described in the project description as following: "*nrounds* also has a higher sensitivity, especially with low values".

3.3.2. An example of a possible narrowing of boundaries

Table 5. Settings for experiments with varying nrounds

Parameter	*Values*
time budget	3000
data.seed	1
tuner.seed	1
dataOML	dataOML
lower	c(1, 2^(-10), 2^(-10), 2^(-10), 0.1, 0, 2^(-7), 3, 1)
upper	c(10, 1, 2^(10), 2^(10), 1, 1, 2^(6), 20, 2^(7))
transformations	c(trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id)
seedFun	123
seedSPOT	1

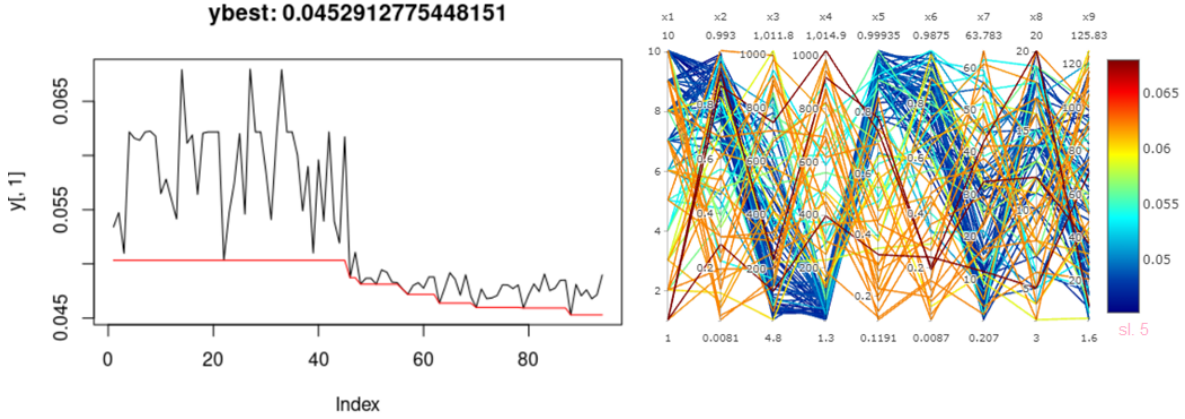


Figure 11: Experiments with limited bounds for *eta* (x2), *subsample* (x5), *colsample_bytree* (x6), *lambda* (x3), and *alpha* (x4)

Fig. 11 also shows that "best" values of *eta* (x2), *subsample* (x5), and *colsample_bytree* (x6) are located on their upper bounds and logically we also would like to extend the upper bound. However, all these 3 parameters have domain $\in [0, 1]$, and increasing of upper bounds is not possible. In order to save time budget for tuning, we could increase the lower bound to make the bound range smaller.

A similar situation for hyperparameters *lambda* (x3 on fig. 11) and *alpha* (x4) - the "best" values are located on their lower bounds, but the domain of these parameters are $\in [0, +\infty]$. So, we can not decrease the lower bounds but could decrease the upper bounds.

3.3.3. Example when changing the border has no effect

The modification of bounds not for all hyperparameters causes the changes for objective function value. Consider the following example (fig. 12) of *max.depth* upper bound modification. Although "best" *max_depth* values are located near its upper bound, changing of upper bound does not affect the result.

Table 6. Settings for experiments with varying max.depth

Parameter	*Values*
time budget	3000
data.seed	1
tuner.seed	1
dataOML	dataOML
lower	c(1, 2^(-10), 2^(-10), 2^(-10), 0.1, 0, 2^(-7), 3, 1)
upper	c(10, 1, 2^(10), 2^(10), 1, 1, 2^(6), **VARY**, 2^(7))
transformations	c(trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id)
seedFun	123
seedSPOT	1

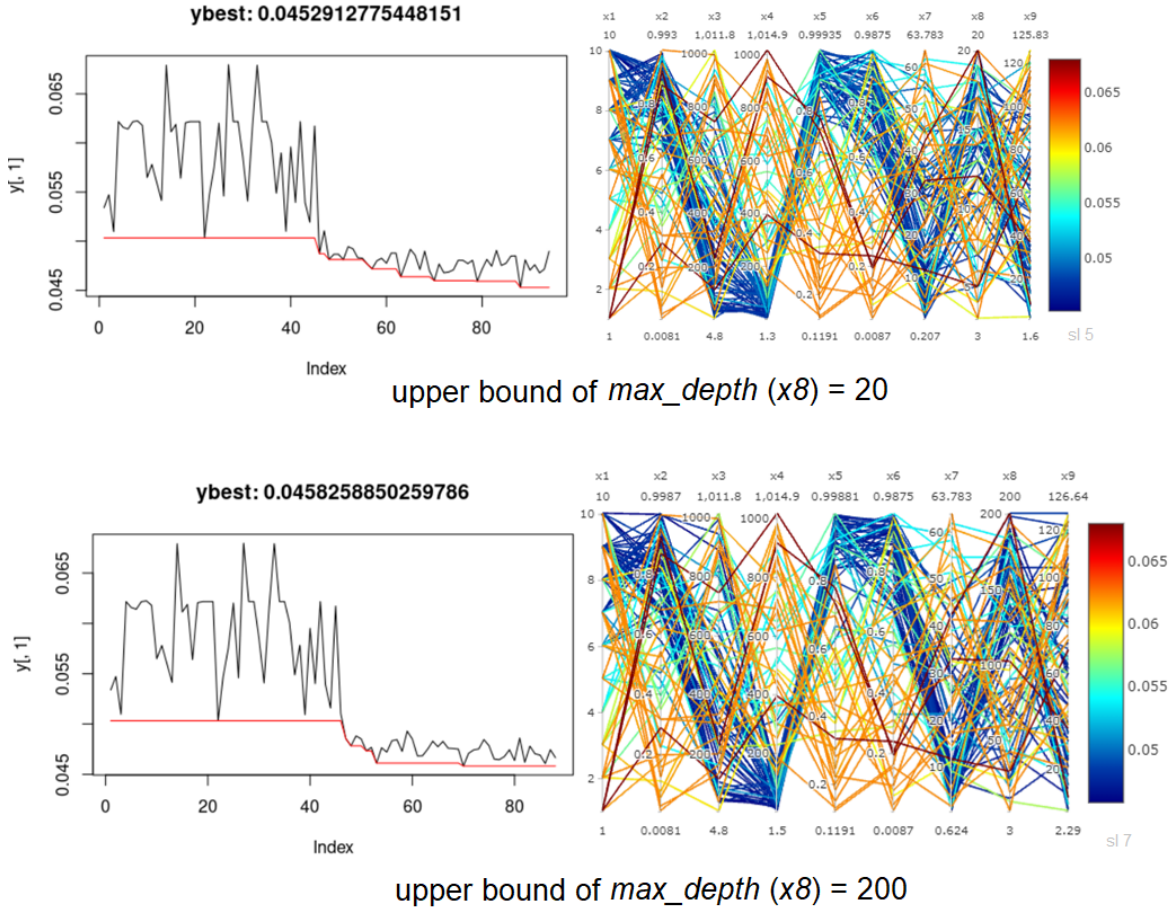


Figure 12: Experiments with varying *max_depth*

3.4. Example of hyperparameter interdependencies

Consider the following example, when 2 experiments (table 7, fig. 13) have a difference on "best" values of *nrounds* and *eta* hyperparameters.

Table 7. Settings for experiments to show interdependencies between nrounds and eta

Experiment 1

Parameter	*Values*
time budget	3000
data.seed	1
tuner.seed	1
dataOML	dataOML
lower	c(1, 2^(-10), 2^(-10), 2^(-10), 0.1, 0, 2^(-7), 3, 1)
upper	c(250, 1, 2^(5), 2^(5), 1, 1, 2^(6), 20, 2^(7))
transformations	c(trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id)
seedFun	123
seedSPOT	1

Experiment 2

Parameter	*Values*
time budget	3000
data.seed	1
tuner.seed	1
dataOML	dataOML
lower	c(1, 2^(-10), 2^(-10), 2^(-10), 0.1, 0, 2^(-7), 3, 1)
upper	c(200, 1, 2^(10), 2^(10), 1, 1, 2^(6), 20, 2^(7))
transformations	c(trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id, trans_id)
seedFun	1
seedSPOT	2

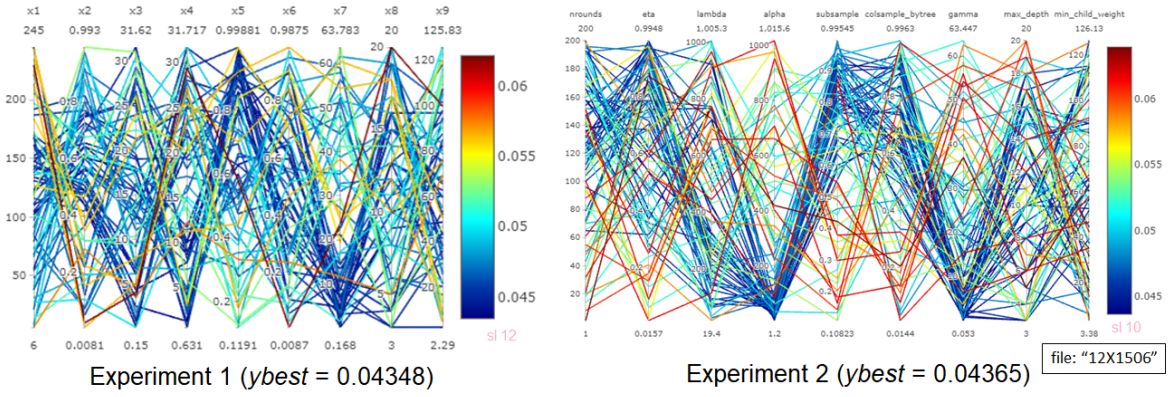


Figure 13: Experiments to show interdependencies between *nrounds* and *eta*

Project description [3] refers to the interconnection between *nrounds* and *eta*: "There is a connection between *eta* and *nrounds*: If one of the two parameters increases, the other should be decreased if the error remains the same..." This connection we can see in fig. 13:

- When "best" *nrounds* are located in range $[140; 160]$, "best" *eta* values are located in range $[0.4; 0.6]$.
- When "best" *nrounds* are located in range $[120; 140]$, "best" *eta* values are located in range $[0.8; 1.0]$.

3.5. Stability of results

When we conducted several experiments with different seeds, we received different results. We can notice how huge impact to the result of objective function create seed. In this section, we will show that input data changing influence as well. We conducted mentioned above experiments in loops to show how results are affected by different data sampling (first 10 000 entities, second and third) an different seeds (conduct 5 trials using loop). For the current section "3.5. Stability of results" bounds and transformations are the same (table 8).

Table 8. Stability experiments: settings for tuning

Hyperparameter	*Upper*	*Lower*	*Transformation*
nrounds	200	1	trans_id
eta	1	0.4	trans_id
lambda	300	1	trans_id
alpha	70	1	trans_id
subsample	1	0.8	trans_id
colsample.bytree	0.8	0.2	trans_id
gamma	10	1	trans_id
max_depth	15	3	trans_id
min.child.weight	110	2	trans_id

Apply the hyperparameters bounds and transformation to build the model, using whole data set.

Table 9. Stability experiments with the whole data

Parameter	*Experiment 1*	*Experiment 2*
time budget	$3 * 3600$ and 5 loops	$15 * 3600$
data.seed	2	2
tuner.seed	2	2
dataOML	dataOML	dataOML
seedFun	index	1
seedSPOT	tuner.seed+index	2

Experiment 1 (time budget $3 * 3600$ and 5 loops).

```
## [1] "ybest_list (file: 18X2306.Rdata)"
```

```
## [1] 0.04358722 0.04351204 0.04384617 0.04367075 0.04419700
```

Experiment 2 (time budget $15 * 3600$).

```
## [1] "ybest (file: 26X1307.Rdata)"
```

```
## [1] 0.0435538
```

Now we conduct experiments, but not for the whole data set, but its part.

Table 10. Stability experiments with 1st part of data

Parameter	*Experiment 1*	*Experiment 2*
time budget	$3 * 3600$ and 5 loops	$15 * 3600$
data.seed	2	2
tuner.seed	2	2
dataOML	dataOML[10000,]	dataOML[10000,]
seedFun	index	1
seedSPOT	tuner.seed+index	3

Experiment 1 (time budget $3 * 3600$ and 5 loops).

```
## [1] "ybest_list (file: 19X2306.Rdata)"
```

```
## [1] 0.04675 0.04225 0.04800 0.04325 0.04700
```

Experiment 2 (time budget $15 * 3600$).

```
## [1] "ybest (file: 20X2406.Rdata)"
```

```
## [1] 0.0435
```

Table 11. Stability experiments with 2nd part of data

Parameter	*Experiment 1*	*Experiment 2*
time budget	$3 * 3600$ and 5 loops	$15 * 3600$
data.seed	1	1
tuner.seed	1	1
dataOML	dataOML[10001:20000,]	dataOML[10001:20000,]
seedFun	index	1
seedSPOT	tuner.seed+index	2

Experiment 1 (time budget $3 * 3600$ and 5 loops).

```
## [1] "ybest_list (file: 21X2506.Rdata)"
```

```
## [1] 0.04950 0.04825 0.04575 0.05025 0.04650
```

Experiment 2 (time budget $15 * 3600$).

```
## [1] "ybest (file: 22X2606.Rdata)"
```

```
## [1] 0.0495
```

Table 12. Stability experiments with 3rd part of data

Parameter	*Experiment 1*	*Experiment 2*
time budget	$3 * 3600$ and 5 loops	$15 * 3600$
data.seed	1	1
tuner.seed	1	1
dataOML	dataOML[20001:299285,]	dataOML[20001:299285,]
seedFun	index	1
seedSPOT	tuner.seed+index	2

Experiment 1 (time budget $3 * 3600$ and 5 loops).

```
## [1] "ybest_list (file: 23X1307.Rdata)"
```

```
## [1] 0.04360241 0.04311904 0.04246558 0.04224180 0.04333387
```

Experiment 2 (time budget $15 * 3600$).

```
## [1] "ybest (file: 24X1307.Rdata)"
```

```
## [1] 0.04360241 0.04311904 0.04246558 0.04224180 0.04333387
```

3.6. Transformations of hyperparameters

Consider the following initial example (table 13). We apply transformation *trans_10pow* for hyperparameters *lambda* and *min_child_weight*. These parameters are numeric (we can not apply transformations on integer type parameters) and we have assigned a very large bounds range.

Table 13. Transformation experiment: initial experiment

Parameter	*Values*
time budget	$2 * 3600$
data.seed	1
tuner.seed	1
dataOML	dataOML
lower	c(1, 0.95, 1, 1, 0.8, 0.2, 1, 3, 2)
upper	c(200, 1, 300, 70, 1, 0.8, 10, 15, 110)
transformations	c(trans_id, trans_id, **VARY**, trans_id, trans_id, trans_id, trans_id, trans_id, **VARY**)
seedFun	index
seedSPOT	tuner.seed+index

The optimizer was launched several times and we consider the best value of the objective function on each loop.

All transformations were *trans_id*:

```
## [1] "ybest_list (file: 15X2006.Rdata)"
```

```
## [1] 0.04389629 0.04499056 0.04413853 0.04479844 0.04524951
```

For *lambda* and *min_child_weight* transformation *trans_10pow* was used.

```
## [1] "ybest_list (file: 17X2206.Rdata)"
```

```
## [1] 0.05488080 0.05459679 0.05502280
```

The experiment above shows that rash implementation of transformation could worsen the results. Now we conduct the experiment with taken into account how transformation works: we apply transformations only on hyperparameters, that are distributed near zero: *lambda*, *alpha*, *gamma*, *min_child_weight*. The real bounds are the same: e.g. upper bound for *min_child_weight* was 110 in the previous experiment, applying *trans_2pow*, we assign an upper bound equal to 7: $2^7 = 128 \sim 110$ (table 14).

Table 14. Transformation experiment: applying only on hyperparameters, that are distributed near zero

Hyperparameter	*Upper*	*Lower*	*Transformation*
nrounds	200	1	trans_id
eta	1	0.4	trans_id
lambda	9	-4	trans_2pow
alpha	7	-4	trans_2pow
subsample	1	0.8	trans_id
colsample.bytree	0.8	0.2	trans_id
gamma	4	-4	trans_2pow
max_depth	15	3	trans_id
min.child.weight	7	-4	trans_2pow

And again we conduct 2 experiments: 5 trials with a medium time budget and 1 trial with a maximum time budget (the same idea as in the previous section to prove the stability of the result).

Table 15. Transformation experiment with the whole data

Parameter	*Experiment 1*	*Experiment 2*
time budget	$3 * 3600$ and 5 loops	$15 * 3600$
data.seed	1	1
tuner.seed	1	1
dataOML	dataOML	dataOML
seedFun	index	1
seedSPOT	tuner.seed+index	2

Experiment 1 (time budget $3 * 3600$ and 5 loops).

```
## [1] "ybest_list (file: 30X1507)"
```

```
## [1] 0.04212540 0.04237600 0.04165762 0.04185810 0.04210034
```

Experiment 2 (time budget $15 * 3600$).

4. Discussion

4.1. Time budget for tuning

The experiments discussed in section "3.1. Results of experiments with time budget for tuning" (fig. 2, 3, 4, 5, 6, and 7) show that increasing of time budget for `SPOT` tuning has a positive influence on the result of the objective function. The final solution will use the maximum possible time budget for tuning.

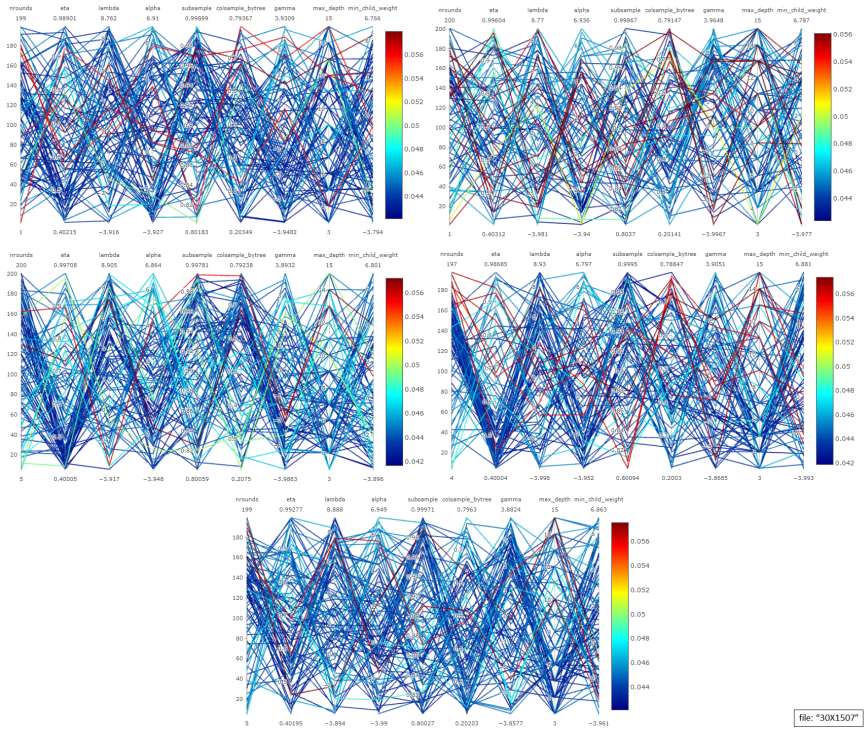


Figure 14: Transformation experiment: time budget $3 * 3600$, *trans_2pow* for *lambda*, *alpha*, *gamma* and *min_child_weight*

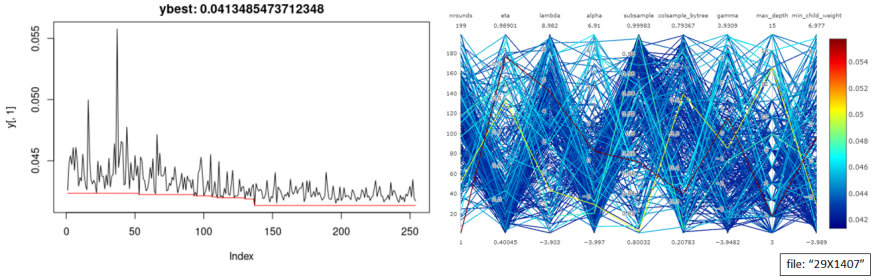


Figure 15: Transformation experiment: time budget $15 * 3600$, *trans_2pow* for *lambda*, *alpha*, *gamma* and *min_child_weight*

4.2. Hyperparameters bounds

For the experiments shown in fig. 2, 3, 4, 5, 6, and 7 we used large bounds and the results were satisfied. Then we increased the bounds range as much as possible (section "3.2. Large bounds range", fig. 8) and the result of objective function became worse.

The section "3.3. Find best bounds of hyperparameters" shows our approach to find the best bounds for hyperparameters: fig. 9 and fig. 10 shows that we increase the upper bound (or decrease lower) if the value of hyperparameter that leads to the best result of the objective function lies on the bound. But sometimes it is not possible due to hyperparameter domain restriction (we discussed hyperparameters' domain in the section "1.2. The Algorithm")

The results in fig. 9 and fig. 11 also show that we could reduce the range of bounds if the "best" values of the hyperparameter are not distributed in the whole range.

We noticed that model performance has a different sensitivity to different hyperparameters bounds. The experiment in fig. 12 (section "3.3.3. Example when changing the border has no effect") illustrates the low influence of *max_depth* changing for values more than 20 to the result of the objective function.

Experiment described in the section "3.4. Example of hyperparameter interdependencies" showed the example of interconnection between *nrounds* and *eta*. The project description [3] refers to interconnections between following hyperparameters:

- *nrounds*, *eta* and *subsumple*;
- *lambda* and *alpha*;
- *min_child_weight*, *gamma* and *max_depth*.

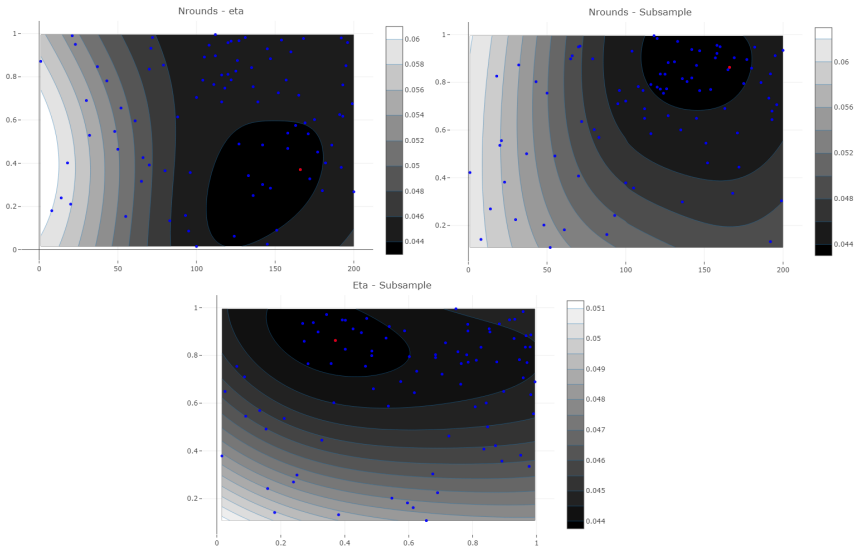


Figure 16: Hyperparameter interdependencies: nrounds, eta, subsumple

4.3. Stability of results

When we introduced the loops to our project code and compared the results of object function for the same settings but with different seeds and input data, we notice a significant standard deviation. For example, for experiment 1 described in table 10:

```
## [1] "ybest_list (file: 19X2306.Rdata)"
```

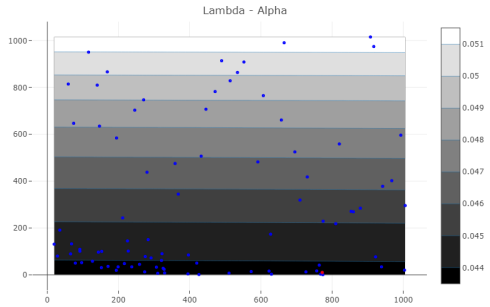


Figure 17: Hyperparameter interdependencies: lambda and alpha

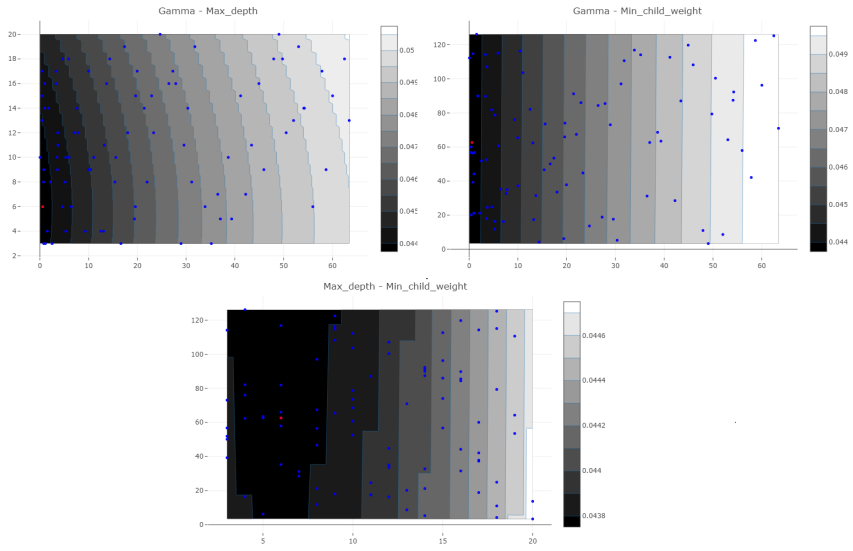


Figure 18: Hyperparameter interdependencies: min.child.weight, gamma and max.depth

```
## [1] 0.04675 0.04225 0.04800 0.04325 0.04700
```

```
## [1] "Range"
```

```
## [1] 0.04225 0.04800
```

```
## [1] "Standart deviation"
```

```
## [1] 0.002533525
```

The result conducted in one of the loops with a time budget $3 * 3600$ is better than with a time budget $15 * 3600$.

```
## [1] "ybest (file: 20X2406.Rdata)"
```

```
## [1] 0.0435
```

This observation gives us the idea of another way how we can use the time budget for tuning.

4.4. Time budget for tuning usage

In the previous section, we discussed that random factor and input data changing has a large impact on objective function result. Instead of using the whole time budget for tuning in a single experiment, we could provide several with different parameters of seed (fig. 19).

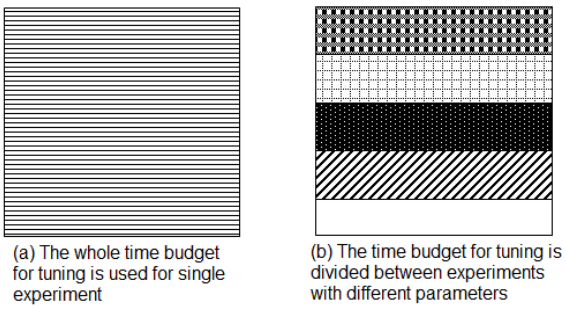


Figure 19: Usage of time budget for tuning

4.5. Transformations of hyperparameters

The transformation provides scaling of the values of the hyperparameters. It can be applied to non-integer hyperparameters only. Transformation is useful only for hyperparameters, that are distributed near zero: *lambda*, *alpha*, *gamma*, *min_child_weight* (table 14). Implementation of transformation on other hyperparameters is useless because their "best" values are evenly distributed in the range between bounds.

5. Conclusion

Our task is to find upper and lower bounds for `XGBoost` hyperparameters which are tuned using `SPOT`. We described our approach to determine them experimentally and using theoretical information about these hyperparameters and optimization algorithm `XGBoost`. We proved the stability of the result using different seeds and input data. Tuning algorithm `SPOT` allows to apply transformations for tuned parameters and we used power 2 transformations to increase the search speed of `SPOT`. The final solution is shown in table 16 (maximum possible time budget for tuning should be applied).

Table 16. Solution for the given task

Hyperparameter	*Upper*	*Lower*	*Transformation*
nrounds	200	1	trans_id
eta	1	0.4	trans_id
lambda	9	-4	trans_2pow
alpha	7	-4	trans_2pow
subsample	1	0.8	trans_id
colsample.bytree	0.8	0.2	trans_id
gamma	4	-4	trans_2pow
max.depth	15	3	trans_id
min.child.weight	7	-4	trans_2pow

6. Bibliography

[1] StatQuest with Josh Starmer. Gradient Boost Part 3 (of 4): Classification. Link: https://www.youtube.com/watch?v=jxuNLH5dXCs

[2] Xgboost developers. XGBoost Documentation. Link: https://xgboost.readthedocs.io/en/latest/index.html

[3] Thomas Bartz-Beielstein. Data Driven Modeling and Optimization. Internal Lecture Notes. Chapter 3 "DDMO Project".

[4] CRAN repository. SPOT: Sequential Parameter Optimization Toolbox. Link: https://CRAN.R-project.org/package=SPOT

[5] Gaurav Dembla. Intuition behind Log-loss score. Towerds data science. Link: https://towardsdatascience.com/intuition-behind-log-loss-score-4e0c9979680a

7. Appendix

7.1. The file history

- 12X1506 - max time budget, okay bounds, trans_id, best result
- 13X1606 - max time budget, large bounds, trans id, poor result
- 14X2006 - small time budget, small bounds, trans id
- 15X2006 - medium time budget, small bounds, trans id, 5 experiments 0.04389629 0.04499056 0.04413853 0.04479844 0.04524951
- 16X2006 - medium time budget, small bounds, trans id, 5 experiments, second seed set 0.04423877 0.04510751 0.04370416 0.04472326 0.04366240
- 17X2206 - medium time budget, small bounds, trans id+trans_10pow, 3 experiments, data,tuner.seed<-2 0.05488080 0.05459679 0.05502280
- 18X2306 - medium time budget, small bounds, modified eta, trans id, 5 experiments, data,tuner.seed<-2

- 31X1307 - max time budget, small bounds, modified eta, trans id, data,tuner.seed<-2 0.04157408
- 19X2306 - medium time budget, small bounds, modified eta, sampling (first 10k), trans id, 5 experiments, data,tuner.seed<-2 0.04675 0.04225 0.04800 0.04325 0.04700
- 20X2406 - max time budget, small bounds, modified eta, sampling (first 10k), trans id, data,tuner.seed<-2 0.0435
- 21X2506 - medium time budget, small bounds, modified eta, sampling (second 10k), trans id, 5 experiments, 0.04950 0.04825 0.04575 0.05025 0.04650
- 22X2606 - max time budget, small bounds, modified eta, sampling (second 10k), trans id 0.0495
- 23X1307 - medium time budget, small bounds, modified eta, sampling (third 10k), trans id, 5 experiments 0.04360241 0.04311904 0.04246558 0.04224180 0.04333387
- 24X1307 - max time budget, small bounds, modified eta, sampling (third 10k), trans id 0.04360241 0.04311904 0.04246558 0.04224180 0.04333387
- 25X1307 "Settings for experiments with varying nrounds",maxi=1, budget 1*3600, small bounds, modified eta, trans id, max nrounds=5 0.04494044
- 26X1307 "Settings for experiments with varying nrounds",maxi=1, budget 1*3600, small bounds, modified eta, trans id, max nrounds=100 0.0435538
- 29X1407 max time budget, small bounds, trans_2pow for for lambda, alpha, gamma, min_child_weight 0.04134855
- 30X1507 medium time budget, 5 experiments, small bounds, trans_2pow for for lambda, alpha, gamma, min_child_weight 0.04212540 0.04237600 0.04165762 0.04185810 0.04210034
- 32X1707 max time budget, small bounds (big for subsample,colsample_bytree), trans_2pow for for lambda, alpha, gamma, min_child_weight 0.04181633
- 33X1907 medium time budget, 5 experiments, small bounds (big for subsample,colsample_bytree), trans_2pow for for lambda, alpha, gamma, min_child_weight 0.04200010 0.04260989 0.04256812 0.04224234 0.04189151

7.2. Script

Important settings:

```
Sys.setenv("CUDA_VISIBLE_DEVICES" = -1)
# Number of loops
maxi=1

# File name where to save and load the results
filename<-'results/24X1307.Rdata'

# Budget for tuning in seconds (max 15*3600)
timebudget <- 15*3600
```

Now, you can proceed Part 1 to conduct experiments, or start from Part 2 to proceed Rdata file.

Part 1. Conduct the experiment

Data Set: the Census-Income (KDD) Data Set

```
## devtools::install_gitlab("bartz/spot",host="http://owos.gm.fh-koeln.de:8055")

library("OpenML")
library("mlr")
library("plotly")
library("SPOT")
```

```
library("farff")
library("SPOTMisc") # For PC only

if (packageVersion("SPOT") < "2.4.8") message("Please update 'SPOT'")

dirName= "oml.cache"
if (! (dir.exists(dirName))) dir.create(dirName)
setOMLConfig(apikey = "c1994bdb7ecb3c6f3c8f3b35f4b47f1f",
             cachedir = dirName # default cachedir is only temporary.
 )
 dataOML <- getOMLDataSet(4535)$data

 # Different input data
 #newdataOML<-dataOML[20001:299285,]
 #dataOML<-newdataOML

 task <- makeClassifTask(data=dataOML,target="V42")

 rsmpl <- makeResampleDesc("Holdout", split=0.6)
```

Project Setup

```
task.type <- "classif"
data.seed <- 1 #2
tuner.seed <- 1 #2
timeout <- timebudget/20 # use 1/20 times the budget before tuning is stopped
```

Learner (Algorithm) Definition

```
model <- "xgboost"
learner <- paste(task.type, model, sep=".")
```

Experiment Configuration

```
if(model=="xgboost"){
   tunepars <- c("nrounds","eta","lambda","alpha","subsample","colsample_bytree",
                 "gamma","max_depth","min_child_weight")
   lower <- c( 1,  0.4,  -4, -4,  0.8,   0.2,  -4,   3,  -4)
   upper <- c(200,   1,   9,  7,    1,   0.8,   4,  15,   7)
   type <-  c("integer","numeric","numeric","numeric","numeric","numeric",
              "numeric","integer","numeric")
   if(task.type=="classif"){
      fixpars <- list(eval_metric="logloss",# suppress warning given when default metric is used.
                      nthread=1) #one thread, not parallel
      }else{
         fixpars <- list(eval_metric="rmse",# suppress warning given when default metric is used.
                         nthread=1) #one thread, not parallel
         }
   factorlevels <- list()
   transformations <- c(trans_id, trans_id, trans_2pow, trans_2pow, trans_id, trans_id,
                        trans_2pow, trans_id, trans_2pow)
   dummy=TRUE
```

```
   relpars <- list()
   }
```

Input missing values

```
task <- impute(
  task,
  classes = list(
    factor = imputeMode(),
    integer = imputeMedian(),
    numeric = imputeMean()
    )
  )$task
```

Replace all factor features with their dummy variables. Internally model.matrix is used. Non factor features will be left untouched and passed to the result.

```
if(dummy){
  task <- createDummyFeatures(task)
}
# str(task)
```

Set Seed

```
set.seed(data.seed)
```

Compile the information as a list.

```
cfg <- list(
 learner=learner,
   tunepars=tunepars,
   lower=lower,
   upper=upper,
   type=type,
   fixpars=fixpars,
   factorlevels=factorlevels,
   transformations=transformations,
   dummy=dummy,
   relpars=relpars,
   task=task,
   resample = rsmpl
)
```

Get objective function

```
objf <- get_objf(config=cfg, timeout=timeout)
```

Launch SPOT in loops *maxi* represents number of loops (is assigned at the beginning)

```
#initialize result vectors
ybest_list<- NULL
xbest_list<- list(NULL)
result_list<- list(NULL)

for (index in 1:maxi) {
   ybest_list[index] <- NULL
   xbest_list[index] <- list(NULL)
   result_list[index]<- list(NULL)
}

for (index in 1:maxi) {
   # Your Tuning Run with SPOT
   set.seed(2*index) #3*index
   result <- spot(fun = objf,
                  lower=cfg$lower,
                  upper=cfg$upper,
                  control = list(types=cfg$type,
                                 maxTime = timebudget/60,
                                 #convert to minutes
                                 plots=TRUE,
                                 progress = TRUE,
                                 model=buildKriging,
                                 optimizer=optimDE,
                                 noise=TRUE,
                                 seedFun=index,
                                 # seed is changed  seedFun=2*index
                                 seedSPOT=tuner.seed+index,
                                 # seed is changed seedSPOT=tuner.seed+2*index
                                 designControl=list(size=5*length(cfg$lower)),
                                 funEvals=Inf,
                                 modelControl=list(target="y",
                                                   useLambda=TRUE,
                                                   reinterpolate=TRUE),
                                 optimizerControl=list(funEvals=100*length(cfg$lower))
                  )
   )
   ybest_list[index]<-result$ybest
   xbest_list[[index]]<-result$xbest
   result_list[[index]]<-result
}
```

Save result of experiment

```
save(result_list, file=filename)
```

Save the input of experiment (all configurations). It will be saved in the same folder as specified at the beginning.

```
input<-list(task.type, data.seed, tuner.seed, timebudget, timeout, model, learner,
            lower, upper, type, fixpars, factorlevels, transformations, dummy, relpars)

names(input)<-c('task.type','data.seed','tuner.seed','timebudget','timeout',
```

```
                'model','learner','lower','upper','type','fixpars','factorlevels',
                'transformations','dummy','relpars')

save(input, file=paste(dirname(filename),'/input_',basename(filename),sep=''))
```

Part 2. Data analyzation

Load result

```
load(filename)
```

It is possible to check some experiment configuration in input_file

```
load(paste(dirname(filename),'/input_',basename(filename),sep=''))

# For example, time budget
input$timebudget
```

```
## [1] 54000
```

If the experiment have not conducted in current session (you have directly loaded the R.data file), please, proceed the next chunk. The chunk reorganizes lists with results of objective function and found values of hyperparameters.

```
# Load libraries
library("plotly")
library("SPOT")

maxi<-length(result_list)

# Initialize empty lists
ybest_list<- NULL
xbest_list<- list(NULL)

for (index in 1:maxi) {
   ybest_list[index] <- NULL
   xbest_list[index] <- list(NULL)
}

for (index in 1:maxi) {
   ybest_list[index]<-result_list[[index]]$ybest
   xbest_list[[index]]<-result_list[[index]]$xbest
}
```

Object function results were found

```
ybest_list
```

```
## [1] 0.04360241 0.04311904 0.04246558 0.04224180 0.04333387
```

Hyperparameters that were found

```
xbest_list
```

```
## [[1]]
##      [,1]      [,2]     [,3]     [,4]      [,5]     [,6]     [,7] [,8]     [,9]
## [1,]  118 0.8484587 27.77729 7.807825 0.9989881 0.219643 4.280499    9 39.24313
##
## [[2]]
##      [,1]      [,2]     [,3]     [,4]      [,5]      [,6]     [,7] [,8]
## [1,]  177 0.6080316 280.1381 1.165253 0.9871054 0.6807267 1.628387    7
##          [,9]
## [1,] 4.635133
##
## [[3]]
##      [,1]      [,2]     [,3]    [,4]      [,5]     [,6]     [,7] [,8]     [,9]
## [1,]  179 0.4775335 113.6119 1.00528 0.9559936 0.298056 2.329204   13 11.47544
##
## [[4]]
##      [,1]      [,2]     [,3]     [,4]      [,5]     [,6]     [,7] [,8]     [,9]
## [1,]  168 0.5138975 67.92825 2.473379 0.9300484 0.490031 1.445863   10 22.86275
##
## [[5]]
##      [,1]      [,2]     [,3]    [,4]      [,5]      [,6]     [,7] [,8]     [,9]
## [1,]  113 0.7972994 9.265985 3.59863 0.9127902 0.5149967 2.994998    8 16.40824
```

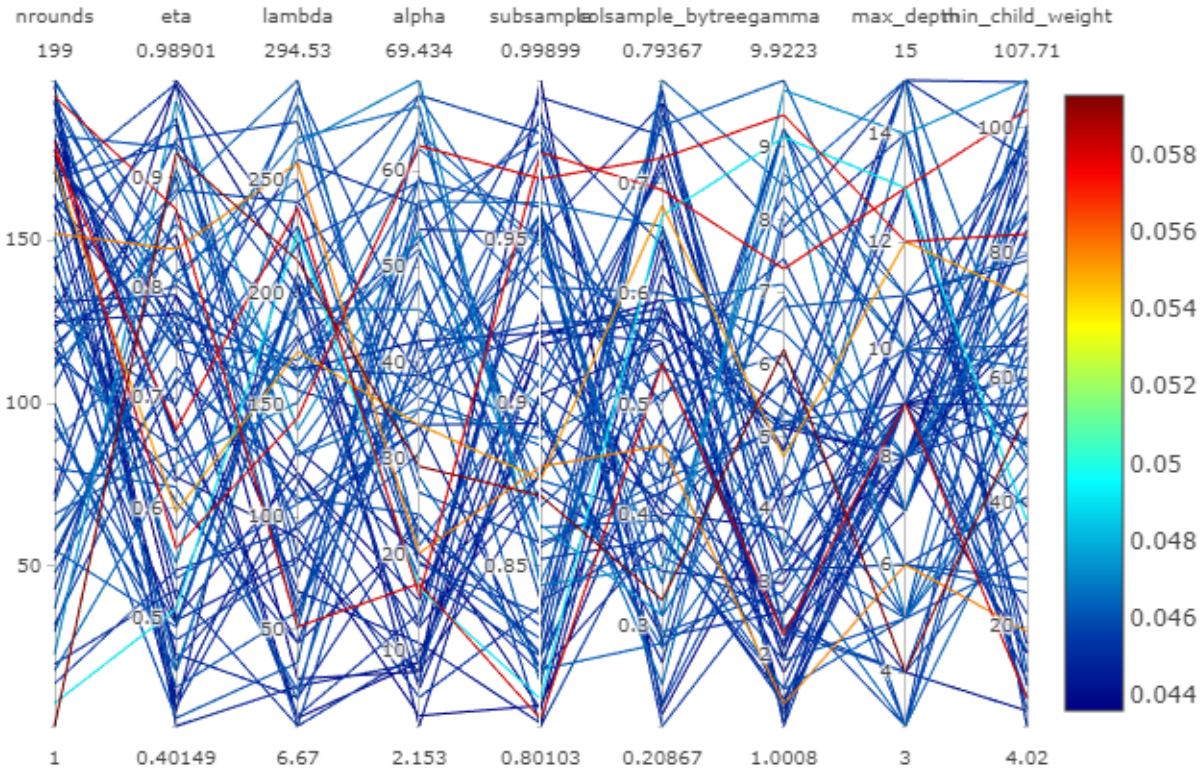
Make a list for each hyperparameter (separate list for each hyperparameter)

```
nrounds_list<- NULL
eta_list<- NULL
lambda_list<- NULL
alpha_list<- NULL
subsample_list<- NULL
colsample_bytree_list<- NULL
gamma_list<- NULL
max_depth_list<- NULL
min_child_weight_list<- NULL

for (index in 1:maxi) {
   nrounds_list[index] <- xbest_list[[index]][1]
   eta_list[index] <- xbest_list[[index]][2]
   lambda_list[index] <- xbest_list[[index]][3]
   alpha_list[index] <- xbest_list[[index]][4]
   subsample_list[index] <- xbest_list[[index]][5]
   colsample_bytree_list[index] <- xbest_list[[index]][6]
   gamma_list[index] <- xbest_list[[index]][7]
   max_depth_list[index] <- xbest_list[[index]][8]
   min_child_weight_list[index] <- xbest_list[[index]][9]
}

# Now we have a list, that contains results of hyperparameter from all experiments
nrounds_list
```

```
## [1] 118 177 179 168 113
```

Parallel plot for single experiment

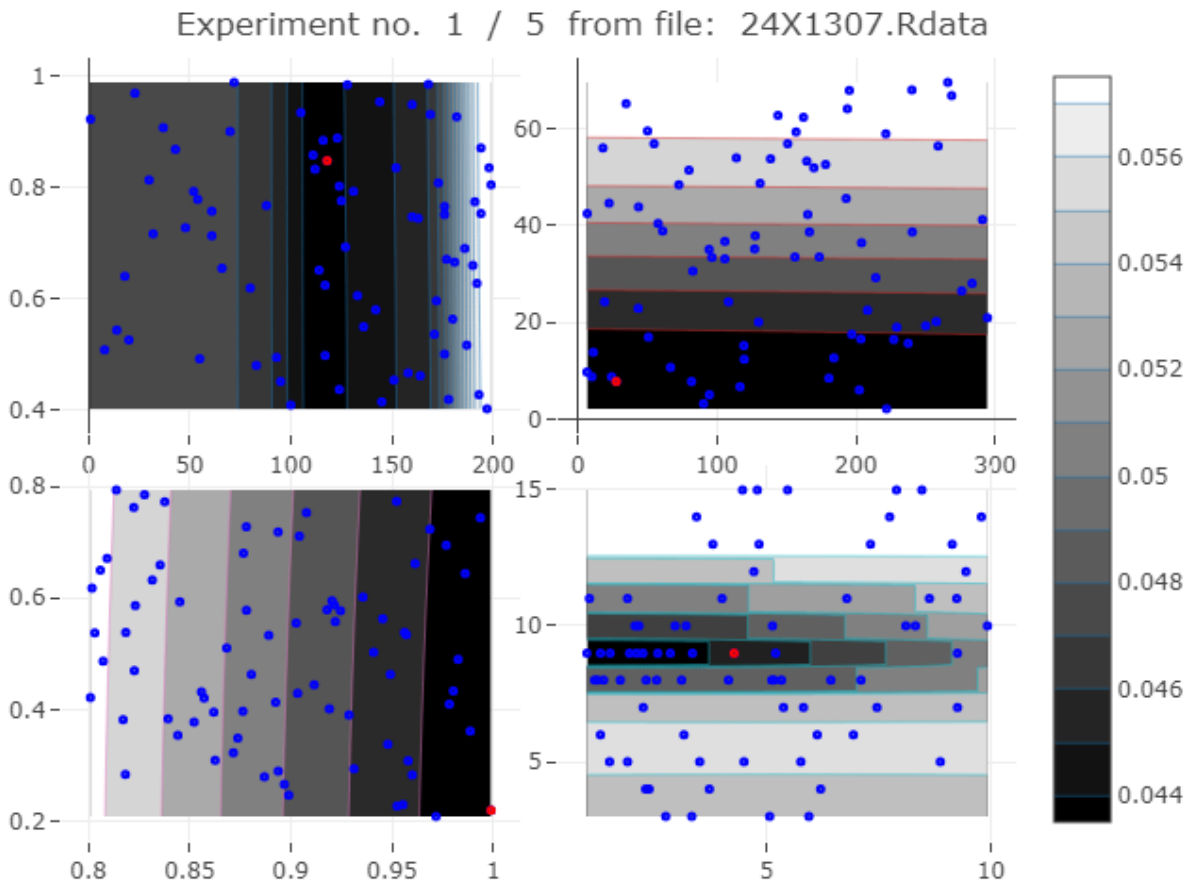
```
# Choose the number of experiment: from 1 to maxi
experiment_num=1
plot_parallel(result_list[[experiment_num]], yvar = 1,
              xlab = c("nrounds","eta","lambda","alpha","subsample",
                       "colsample_bytree","gamma","max_depth","min_child_weight"))
```



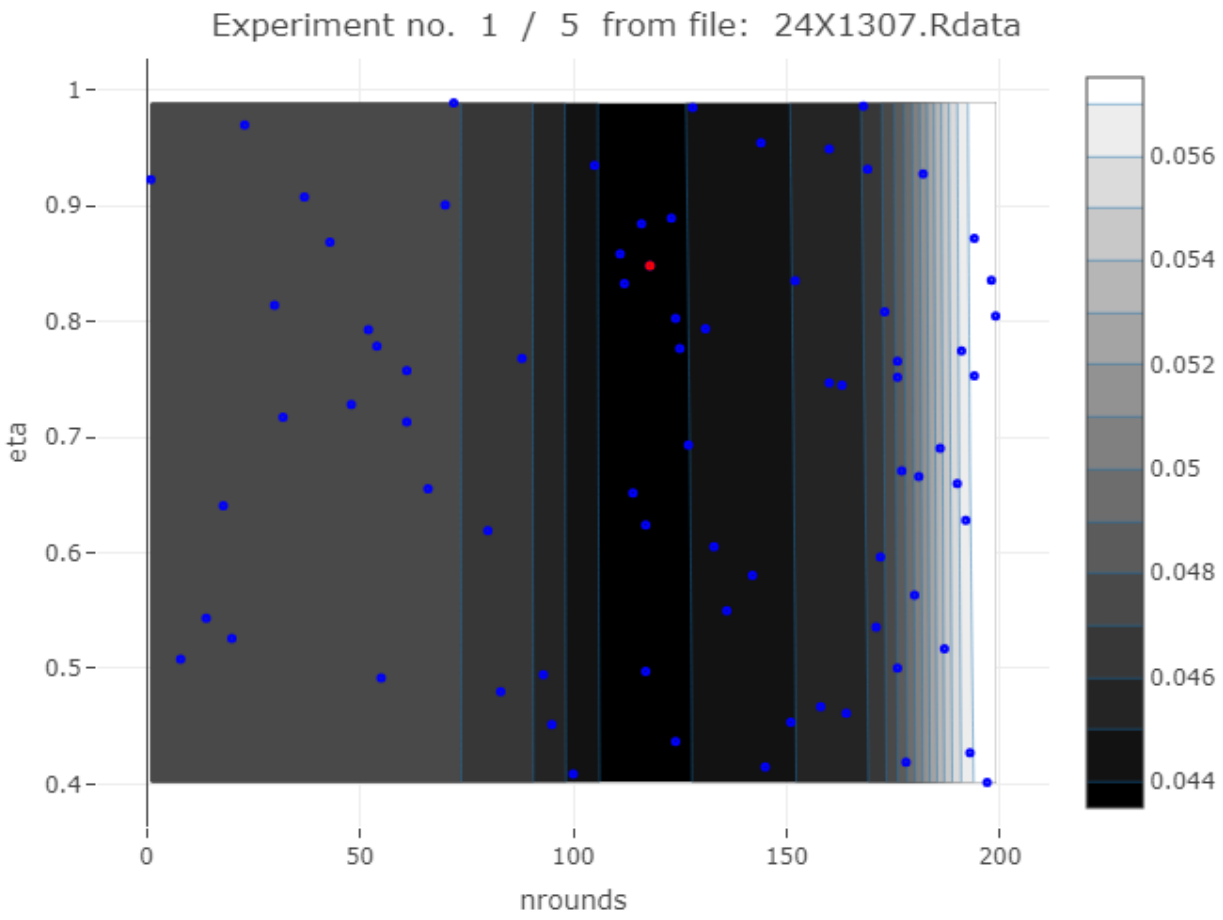
Plot surface -> Experiment number can be changed -> Hyperparameters combination can be changed

```
# Change the number of experiment: from 1 to maxi
experiment_num=1

# Make plots
p1<-plot_surface(result_list[[experiment_num]],
                 which=c(1,2), yvar=1, ylab = "nrounds - eta")
p2<-plot_surface(result_list[[experiment_num]],
                 which=c(3,4), yvar=1, ylab = "lambda - alpha")
p3<-plot_surface(result_list[[experiment_num]],
                 which=c(5,6), yvar=1, ylab ="subsample - colsample_bytree")
p4<-plot_surface(result_list[[experiment_num]],
                 which=c(7,8), yvar=1, ylab ="gamma - max_depth")
p5<-plot_surface(result_list[[experiment_num]],
                 which=c(8,9), yvar=1,  ylab = "max_depth - min_child_weight")

# Title string
plot_title<-paste('Experiment no. ',toString(experiment_num),' / ',
                  toString(maxi),' from file: ',basename(filename))
```

```
# Merge several plots
fig <- subplot(p1, p2,p3,p4, nrows=2)
fig <- fig %>% layout(
    title = plot_title)
fig
```

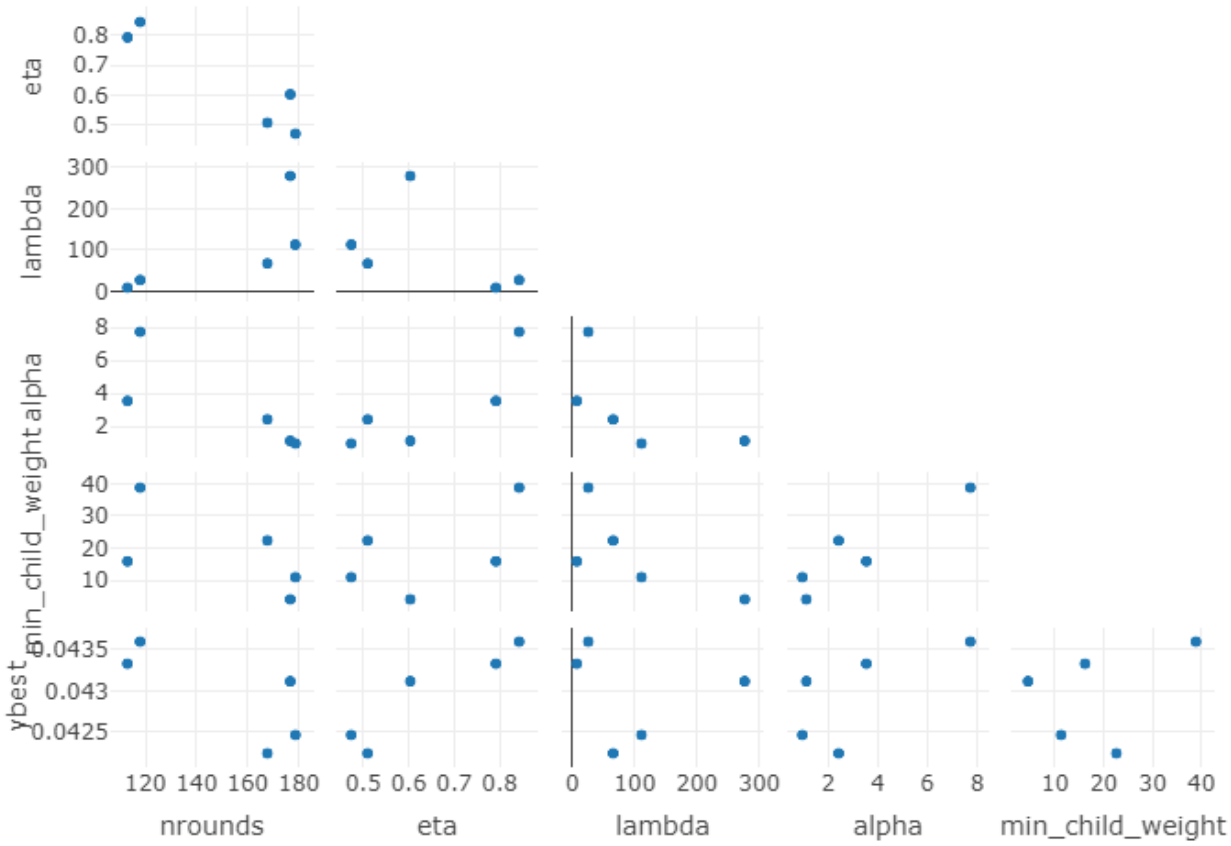



```
# Make single plot
plot_surface(result_list[[experiment_num]], which=c(1,2), yvar=1,
             ylab = plot_title, xlab=c('nrounds','eta'))
```



Create Scatterplot matrices. It is like 'pairs', but interactive. We can use options "Box select" or "Lasso select" to highlight specific points. *https://plotly-r.com/arranging-views.html#arranging-plotly-objects*

```
# Create data frame
df<-do.call(rbind, Map(data.frame,
                       nrounds=nrounds_list,
                       eta=eta_list,
                       lambda=lambda_list,
                       alpha=alpha_list,
                       # subsample=subsample_list,
                       # colsample_bytree=colsample_bytree_list,
                       # gamma=gamma_list,
                       # max_depth=max_depth_list,
                       min_child_weight=min_child_weight_list,
                       ybest=ybest_list))

# Crate interactive plots

dims <- dplyr::select_if(df, is.numeric)
dims <- purrr::map2(dims, names(dims), ~list(values=.x, label=.y))
plot_ly(
  type = "splom", dimensions = setNames(dims, NULL),
  showupperhalf = FALSE, diagonal = list(visible = FALSE)
)
```


eta
lambda
alpha
min_child_weight
ybest
nrounds
eta
lambda
alpha
min_child_weight