Chapter 9
Case Study II: Tuning of Gradient Boosting (xgboost)

Thomas Bartz-Beielstein, Sowmya Chandrasekaran, and Frederik Rehbach

Abstract This case study gives a hands-on description of Hyperparameter Tuning (HPT) methods discussed in this book. The Extreme Gradient Boosting (XGBoost) method and its implementation `xgboost` was chosen, because it is one of the most powerful methods in many Machine Learning (ML) tasks, especially when standard tabular data should be analyzed. This case study follows the same HPT pipeline as the first and third studies: after the data set is provided and pre-processed, the experimental design is set up. Next, the HPT experiments are performed. The R package `SPOT` is used as a "datascope" to analyze the results from the HPT runs from several perspectives: in addition to Classification and Regression Trees (CART), the analysis combines results from the surface, sensitivity, and parallel plots with a classical regression analysis. Severity is used to discuss the practical relevance of the results from an error-statistical point-of-view. The well-proven R package `mlr` is used as a uniform interface from the methods of the packages `SPOT` and `SPOTMisc` to the ML methods. The corresponding source code is explained in a comprehensible manner.

9.1 Introduction

This chapter considers the XGBoost algorithm which was detailed in Sect. 3.6. How to find suitable parameter values and bounds, and how to perform experiments w.r.t. the following nine XGBoost hyperparameters will be discussed: `nrounds`,

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-981-19-5170-1_9.

T. Bartz-Beielstein (✉) · S. Chandrasekaran · F. Rehbach
Institute for Data Science, Engineering and Analytics, TH Köln, Cologne, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

S. Chandrasekaran
e-mail: sowmya.chandrasekaran@th-koeln.de

F. Rehbach
e-mail: frederik.rehbach@th-koeln.de

eta, lambda, alpha, subsample, colsample, gamma, maxdepthx, and minchild.

9.2 Data Description

The first step is identical to the step in the ranger example in Chap. 8, because the Census-Income (KDD) Data Set (CID) will be used.[1] So, the function getDataCensus is called with the parameters from Table 8.3 to get the CID data from Table 8.2. The complete data set, $(\mathcal{X}, \mathcal{Y})$ contains $n = 299,285$ observations with 41 features on demography and employment.

9.3 getMlConfig: Experimental Setup and Configuration of the Gradient Boosting Model

Again, a subset with $n = 1e4$ samples that defines the subset $(X, Y) \in (\mathcal{X}, \mathcal{Y})$ is provided. The project setup is also similar to the setup described in Sect. 8.1. Therefore, only the differences will be shown. The full script is available in Sect. 9.10.

The function getMlConfig is called with the same arguments as in Chap. 8, with one exception: model is set to "xgboost". The function getMlConfig defines the ML task, the model configuration, and the data split (generation of the training and test data sets, i.e., $(X, Y)^{(\text{train})}$and $(X, Y)^{(\text{test})}$.) To achieve this goal, the functions getMlTask, getModelConf, and getMlrResample are executed. As a result, the list cfg with 13 elements is available, see Table 9.1.

```
model <- "xgboost"
cfg <- getMlConfig(
  target = target,
  model = model,
  data = dfCensus,
  task.type = task.type,
  nobs = nobs,
  nfactors = nfactors,
  nnumericals = nnumericals,
  cardinality = cardinality,
  data.seed = data.seed,
  prop = 2 / 3
)
```

[1] The data from CID is historical. It includes wording or categories regarding people which do not represent or reflect any views of the authors and editors.

Table 9.1 Result from the function `getMlConfig`: the `cfg` list

Parameter	Type	Value
`learner`	chr	`"classif.xgboost"`
`tunepars`	chr [1:9]	`"nrounds" "eta" "lambda" "alpha" ...`, see Table 9.2
`defaults`	num [1, 1:9]	`0 -1.74 0 -10 1 ...`
`lower`	num [1:9]	`0 -10 -10 -10 0.1 ...`
`upper`	num [1:9]	`11 0 10 10 1 1 10 15 7`
`type`	chr [1:9]	`"numeric" "numeric" "numeric" "numeric" ...`
`fixpars`	list	`eval_metric: chr "logloss"` and number of threads
`factorlevels`	list()	
`transformations:`	List of 9	Transformation functions, see Table 9.2
`dummy`	logi	`TRUE`
`relpars`	list()	Empty list (no parameters are relative to each other)
`task`	List of 6	`mlr` task object
`resample`	List of 5	Resample information

9.3.1 `getMlrTask`: *Problem Design and Definition of the Machine Learning Task*

The problem design describes the target and task type, the number of observations, as well as the number of factorial, numerical, and cardinal variables. It was described in Sect. 8.3.2.1.

9.3.2 `getModelConf` *Algorithm Design—Hyperparameters of the Models*

The function `getModelConf`, which is called from `getMlConf`, computes an adequate XGBoost hyperparameter setting. Examples from literature shown in Table 3.6 in Sect. 3.6 will be used as a guideline. These values were modified as follows:

`nrounds`:	An upper value (2^5), which is similar to the Random Forest (RF) configuration, was chosen. This value is smaller than the value used by Probst et al. (2019a), who used 5 000.
`colsample_bytree`:	The lower value was chosen as 1/`getTaskNFeats(task)`. This is a minor deviation from the settings used in Probst et al. (2019a). The reason for this modification

Table 9.2 XGBoost hyperparameter. N_{Feats} denotes the output from `getTaskNFeats(task)`

Name	Type	Default	Lower (mlr)	Upper (mlr)	Upper (SPOT)	Lower (SPOT)	Trans
nrounds	integer	–	1	Inf	0	5	2pow_round
eta	numeric	0.3	0	1	–10	0	2pow
lambda	numeric	1	0	Inf	–10	10	2pow
alpha	numeric	0	0	Inf	–10	10	2pow
subsample	numeric	1	0	1	0.1	1	id
colsample_bytree	numeric	1	0	1	$1/N_{\text{Feats}}$	1	id
gamma	numeric	0	0	Inf	–10	10	2pow
max_depth	integer	6	0	Inf	1	15	id
min_child_weight	numeric	1	0	Inf	0	7	2pow

is simple: a lower value of zero makes no sense, because at least one feature should be chosen via `colsample`.

`gamma`: A lower value of -10 was chosen. This value is smaller than the value chosen by Thomas et al. (2018). Accordingly, a larger upper value (10) than by Thomas et al. (2018) was selected.

Hyperparameter transformations are shown in the column `trans` in Table 9.2. These transformations are similar to the transformations used by Probst et al. (2019a) and Thomas et al. (2018) with one minor change: `trans_2pow_round` was applied to the hyperparameter `nrounds`.

The ML configuration list `cfg` contains information about the hyperparameters of the XGBoost model, see Table 9.2.

Background: XGBoost Hyperparameters

The complete list of XGBoost hyperparameters can also be shown using the function `getModelConf`. Note: the hyperparameter `colsample_bytree` is a relative hyperparameter, i.e., it depends on the number of features (`nFeatures`), see the discussion in Sect. 3.6. Hence, the value `nFeatures` must be determined before the hyperparameter bounds can be computed.

```
nFeatures <- sum(task$task.desc$n.feat)
modelCfg <- getModelConf(
  task.type = task.type,
  model = model,
  nFeatures = nFeatures
)
```

The list of hyperparameters is stored as the list element `tunepars`, see Table 9.2.

Furthermore, all factor features will be replaced with their dummy variables. Dummy variables are recommended for XGBoost: internally, a `model.matrix` is used and non-factor features will be left untouched and passed to the result. The seed can be set to improve reproducibility. Finally, these settings are compiled to the list `cfg`.

9.3.3 *`getMlrResample`: Training and Test Data*

The partition of the full data set is done as described in Sect. 8.3.2.3. `rsample` specifies a training data set, which contains 2/3 of the data and a testing data set with the remaining 1/3 of the data.

9.4 Objective Function (Model Performance)

Because the XGBoost method is more complex than RF, an increased computational budget is recommended, e.g., by choosing a budget for tuning of $6 \times 3{,}600$ s or six hours. The increased budget is used in the global study (Chap. 12). For the experiments performed in the current chapter, the budget was not increased.

Before the hyperparameter tuner is called, the objective function is defined: this function receives a configuration for a tuning experiment and returns an objective function to be tuned via `spot`. A detailed description of the objective function can be found in Sect. 8.4.4.

9.5 `spot`: Experimental Setup for the Hyperparameter Tuner

The R package `SPOT` is used to perform the actual tuning (optimization). Because the generic Sequential Parameter Optimization Toolbox (SPOT) setup was introduced in Sect. 4.5, this section highlights the modifications of the generic setup that were made for the `xgboost` hyperparameter tuning experiments.

The third step of the hyperparameter tuning pipeline as shown in Fig. 8.5 starts the SPOT hyperparameter tuner.

```
result <- spot(
  x = NULL,
  fun = objf,
  lower = cfg$lower,
  upper = cfg$upper,
  control = list(
    types = cfg$type,
    time = list(maxTime = timebudget / 60),
    noise = TRUE,
    OCBA = TRUE,
    OCBABudget = 3,
    seedFun = 123,
    designControl = list(
      replicates = Rinit,
      size = initSizeFactor * length(cfg$lower)
    ),
    replicates = 2,
    funEvals = Inf,
    modelControl = list(
      target = "ei",
      useLambda = TRUE,
      reinterpolate = FALSE
    ),
    optimizerControl = list(funEvals = 200 * length(cfg$lower)),
    multiStart = 2,
    parNames = cfg$tunepars,
    yImputation = list(
      handleNAsMethod = handleNAsMean,
      imputeCriteriaFuns = list(is.infinite, is.na, is.nan),
      penaltyImputation = 3
    )
  )
)
```

The result is written to a file and can be accessed via

```
load("supplementary/ch09-CaseStudyII/xgboost00001.RData")
```

The full R code for running this case study is shown in the Appendix (Sect. 9.10).

9.6 Tunability

9.6.1 Progress

The function `prepareProgressPlot` generates a data frame that can be used to visualize the hyperparameter tuning progress. The data frame can be passed to `ggplot`. Figure 9.1 visualizes the progress during the `ranger` hyperparameter tuning process during the `spot` tuning procedure.

After 60 min, 157 `xgboost` models were evaluated. Comparing the worst configuration that was observed during the HPT with the best, a 66.3743% reduction was obtained. After the initial phase, which includes 18 evaluations, the smallest Mean Mis-Classification Error (MMCE) reads 0.1793641. The dotted red line in Fig. 8.6 illustrates this result. The final best value reads 0.1724655, i.e., a reduction

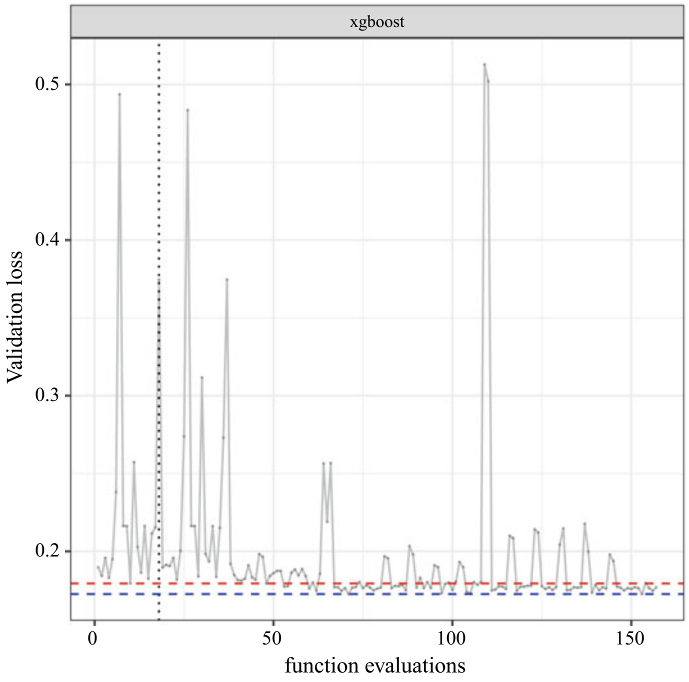

Fig. 9.1 XGB: Hyperparameter tuning progress. Validation loss plotted against the number of function evaluations, i.e., the number of evaluated XGBoost models. The red dashed line denotes the best value found by the initial designs to show the hyperparameter tuning progress. The blue dashed line represents the best value from the whole run

of the MMCE of 3.8462%. These values, in combination with results shown in the progress plot (Fig. 8.6), indicate that a quick HPT run is able to improve the quality of the `xgboost` method. It also indicates that increased run times do not result in a significant improvement of the MMCE.

! Attention

These results do not replace a sound statistical comparison, they are only indicators, not final conclusions.

Table 9.3 Comparison of the default and tuned hyperparameters of the XGBoost method. `colsample` denotes `colsample_bytree`. Table shows transformed values. Note: the `alpha` and `gamma` values are identical. They are computed as 2^{-10}, which is the lower bound value, because the theoretical default value 0 is infeasible. See also Table 3.8

Method	nrounds	eta	lambda	alpha	subsample	colsample	gamma	max_depth	min_child_weight
Default	1	0.3	1	0.001	1	1	0.001	6	1
Tuned	1873	0.058	155.3	2.46	0.992	0.408	0.004	13	1.83

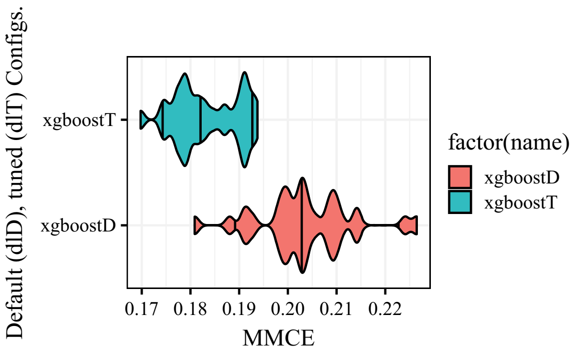

Fig. 9.2 Comparison of XGBoost methods with default (D) and tuned (T) hyperparameters. Classification error (MMCE) plotted on the horizontal axis. Vertical lines in the violin figures mark quantiles (0.25, 0.5, 0.75) of the corresponding distribution. Numerical values are shown in Table 9.3

9.6.2 `evalParamCensus`: Comparing Default and Tuned Parameters on Test Data

As a baseline for comparison, XGBoost was run with default hyperparameter values. The corresponding R code for replicating the experiment is available in the `code` folder. The best (minimum MMCE) result from thirty repeats is reported. The corresponding values are shown in Table 9.3. The function `evalParamCensus` was used to perform this comparison. By specifying the ML model, e.g., `"xgboost"` and the `runNr`, the function `evalParamCensus` was called.

The result files can be loaded and the violin plot of the obtained MMCE can be visualized (Fig. 9.2). It can be seen that the tuned solutions provide a better MMCE. Default and tuned results for the `ranger` model are available as `rangerDefaultEvaluation.RData` and `xgboost00001Evaluation.RData`, respectively.

The scores are stored as a `matrix`. Attributes are used to label the measures. The following measures are calculated for each hyperparameter setting: `accuracy`, `ce`, `f1`, `logLoss`, `mae`, `precision`, `recall`, and `rmse`. The comparison is based on the MMCE that was defined in Eq. (2.2). Hyperparameters of the default and the tuned configurations are shown in Table 9.3. The full procedure, i.e., starting from scratch, to obtain the default `xgboost` hyperparameters is shown in Sect. 9.10.

Next, the hyperparameters of the tuned `xgboost` methods are shown.

9.7 Analyzing the Gradient Boosting Tuning Process

The analysis and the visualizations are based on the transformed values.

To analyze effects and interactions between hyperparameters of the `xgboost` Model, a simple regression tree as shown in Fig. 9.3 and Fig. 9.4 can be used.

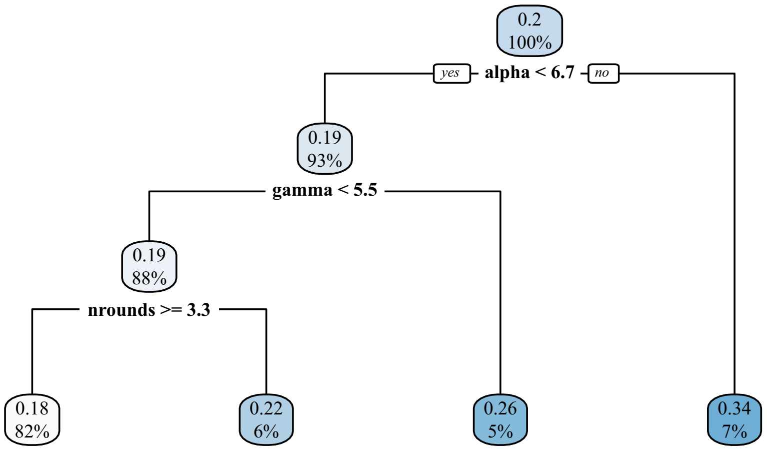

Fig. 9.3 Regression tree. Case study II. XGBoost

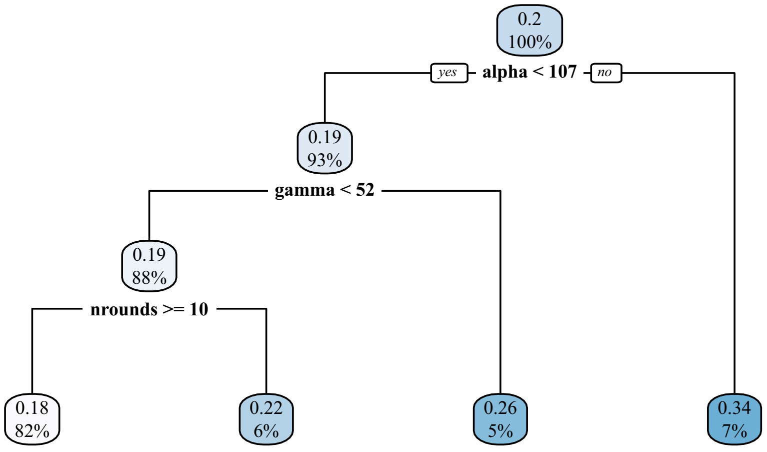

Fig. 9.4 Regression tree. Case study II. XGBoost. Hyperparameters are transformed values

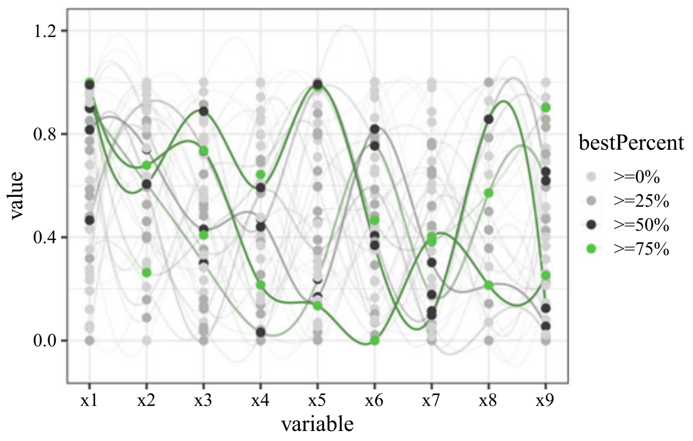

Fig. 9.5 Best configurations in green

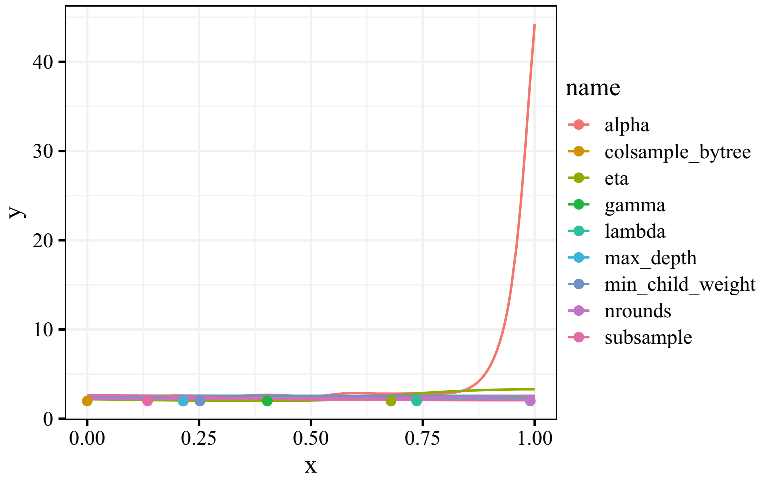

Fig. 9.6 Sensitivity plot (best). Too large alpha values result in poor results

Same tree with the transformed values:

The regression tree supports the observations that hyperparameter values for alpha, lambda, gamma, and nrounds have the largest effect on the MMCE.

	alpha	lambda	gamma	nrounds	subsample	eta	colsample_bytree
1	0.23112227	0.04431784	0.04039483	0.014028719	0.012203015	0.009397272	0.0028057437

alpha is the most relevant hyperparameter.

To perform a sensitivity analysis, parallel and sensitivity plots can be used (Figs. 9.5 and 9.6).

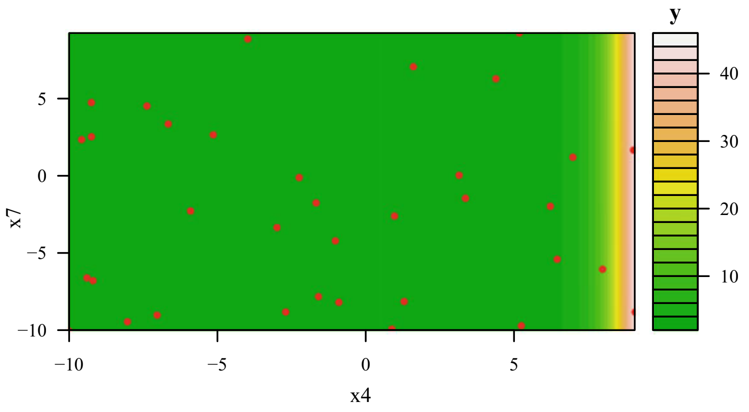

Fig. 9.7 Surface plot: x_3 plotted against x_1. This surface plot indicates that `alpha` has a large effect. Too large alpha values result in poor results

Results from the `spot` run can be passed to the function `plotSenstivity`, which generates a sensitivity plot as shown in Fig. 8.10. Sensitivity plots were introduced in Sect. 8.6. Contour plots are shown in Fig. 9.7.

Finally, a simple linear regression model can be fitted to the data. Based on the data from SPOT's `result` list, the summary is shown below.

```
##
## Call:
## lm(formula = y ~ ., data = df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.073397 -0.015307 -0.008367  0.001629  0.223535
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        0.2561669  0.0184896  13.855  < 2e-16 ***
## nrounds           -0.0032016  0.0012827  -2.496  0.01366 *
## eta               -0.0003031  0.0016994  -0.178  0.85870
## lambda             0.0021936  0.0007939   2.763  0.00646 **
## alpha              0.0072423  0.0007848   9.228 2.77e-16 ***
## subsample         -0.1033194  0.0137081  -7.537 4.55e-12 ***
## colsample_bytree   0.0050479  0.0132658   0.381  0.70411
## gamma              0.0010887  0.0009034   1.205  0.23007
## max_depth          0.0023106  0.0010527   2.195  0.02974 *
## min_child_weight   0.0082803  0.0025515   3.245  0.00145 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04047 on 147 degrees of freedom
## Multiple R-squared:  0.5222, Adjusted R-squared:  0.493
## F-statistic: 17.85 on 9 and 147 DF,  p-value: < 2.2e-16
```

Although this linear model requires a detailed investigation (a misspecification analysis is necessary) it also is in accordance with previous observations that hyper-parameters `alpha`, `lambda`, `gamma`, `nrounds` have significant effects on the loss function.

9.8 Severity: Validating the Results

Now, we utilize hypothesis testing and severity to analyze the statistical significance of the achieved performance improvement. Considering the results from the pre-experimental runs, the difference is $\bar{x} = 0.0199$. Since this value is positive, for the moment, let us assume that the tuned solution is superior. The corresponding standard deviation is $s_d = 0.0081$. Based on Eq. 5.14, and with $\alpha = 0.05$, $\beta = 0.2$, and $\Delta = 0.01$, let us identify the required number of runs for the full experiment using the `getSampleSize()` function.

Table 9.4 Case study II: result analysis

p-value	Decision	power	Cohen's d	Hedge's g	Severity
0	H0 rejected	0.9999999	2.3978067	2.3666664	$\Delta \leq 0.015$ are well supported

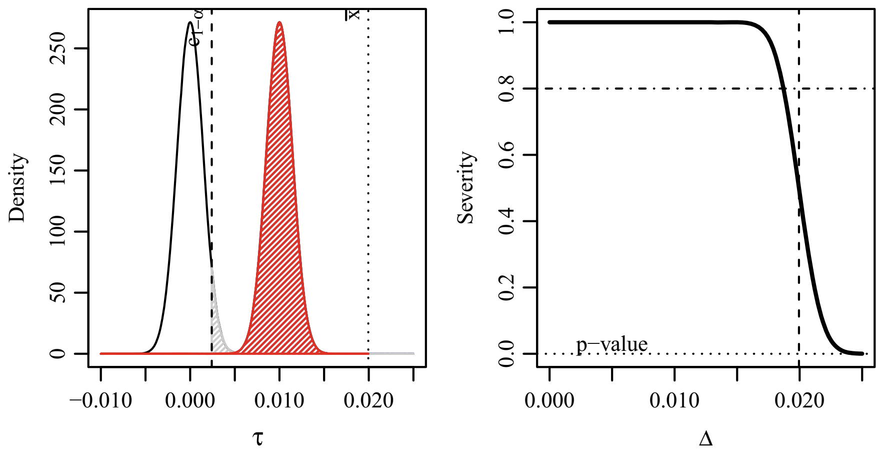

Fig. 9.8 Tuning XGB. Severity of rejecting H0 (red), power (blue), and error (gray). Left: The observed mean $\bar{x}$ =0.0199 is larger than the cut-off point $c_{1-\alpha}$ =0.0024 Right: The claim that the true difference is as large or larger than 0.01 is well supported by severity. However, any difference larger than 0.015 is not supported by severity

For a relevant difference of 0.01, approximately 8 runs per algorithm are required. Hence, we can proceed to evaluate the severity and analyze the performance improvement achieved through tuning the parameters of the xgboost.

The summary result statistics is presented in Table 9.4. The decision based on p-value is to reject the null hypothesis, i.e., the claim that the tuned parameter setup provides a significant performance improvement in terms of MMCE is supported. The effect size suggests that the difference is of a larger magnitude. For the chosen Δ =0.01, the severity value is at 1 and thus it strongly supports the decision of rejecting the H_0. The severity plot is shown in Fig. 9.8. Severity shows that performance differences smaller than 0.015 are well supported.

9.9 Summary and Discussion

The analysis indicates that hyperparameter alpha has the greatest effect on the algorithm's performance. The recommended value of alpha is 7.2791, which is much larger than the default value.

This case study demonstrates how functions from the R packages `mlr` and SPOT can be combined to perform a well-structured hyperparameter tuning and analysis. By specifying the time budget via `maxTime`, the user can systematically improve hyperparameter settings. Before applying ML algorithms such as XGBoost to complex classification or regression problems, HPT is recommended. Wrong hyperparameter settings can be avoided. Insight into the behavior of ML algorithms can be obtained.

9.10 Program Code

Program Code

```
target <- "age"
task.type <- "classif"
nobs <- 1e4
nfactors <- "high"
nnumericals <- "high"
cardinality <- "high"
data.seed <- 1
cachedir <- "oml.cache"

dfCensus <- getDataCensus(
  task.type = task.type,
  nobs = nobs,
  nfactors = nfactors,
  nnumericals = nnumericals,
  cardinality = cardinality,
  data.seed = data.seed,
  cachedir = cachedir,
  target = target
)
task <- getMlrTask(
  dataset = dfCensus,
  task.type = "classif",
  data.seed = 1
)

model <- "xgboost"
cfg <- getMlConfig(
  target = target,
  model = model,
  data = dfCensus,
  task.type = task.type,
  nobs = nobs,
  nfactors = nfactors,
  nnumericals = nnumericals,
  cardinality = cardinality,
  data.seed = data.seed,
  prop = 2 / 3
)
transformX(cfg$defaults, cfg$transformations)
```

```
##      [,1] [,2] [,3]         [,4] [,5] [,6]         [,7] [,8] [,9]
## [1,]    1  0.3    1 0.0009765625    1    1 0.0009765625    6    1
```