Buczak, Philip; Groll, Andreas; Pauly, Markus; Rehof, Jakob; Horn, Daniel

Article — Published Version

Using sequential statistical tests for efficient hyperparameter tuning

AStA Advances in Statistical Analysis

Suggested Citation: Buczak, Philip; Groll, Andreas; Pauly, Markus; Rehof, Jakob; Horn, Daniel (2024) : Using sequential statistical tests for efficient hyperparameter tuning, AStA Advances in Statistical Analysis, ISSN 1863-818X, Springer Berlin Heidelberg, Berlin/Heidelberg, Vol. 108, Iss. 2, pp. 441-460, https://doi.org/10.1007/s10182-024-00495-1

This Version is available at:
https://hdl.handle.net/10419/315188

AStA Advances in Statistical Analysis (2024) 108:441–460
https://doi.org/10.1007/s10182-024-00495-1

ORIGINAL PAPER

Using sequential statistical tests for efficient hyperparameter tuning

Philip Buczak[1] · Andreas Groll[1] · Markus Pauly[1,2] · Jakob Rehof[3] · Daniel Horn[1,2]

Received: 29 November 2022 / Accepted: 9 February 2024 / Published online: 14 March 2024

Abstract

Hyperparameter tuning is one of the most time-consuming parts in machine learning. Despite the existence of modern optimization algorithms that minimize the number of evaluations needed, evaluations of a single setting may still be expensive. Usually a resampling technique is used, where the machine learning method has to be fitted a fixed number of k times on different training datasets. The respective mean performance of the k fits is then used as performance estimator. Many hyperparameter settings could be discarded after less than k resampling iterations if they are clearly inferior to high-performing settings. However, resampling is often performed until the very end, wasting a lot of computational effort. To this end, we propose the sequential random search (SQRS) which extends the regular random search algorithm by a sequential testing procedure aimed at detecting and eliminating inferior parameter configurations early. We compared our SQRS with regular random search using multiple publicly available regression and classification datasets. Our simulation study showed that the SQRS is able to find similarly well-performing parameter settings while requiring noticeably fewer evaluations. Our results underscore the potential for integrating sequential tests into hyperparameter tuning.

✉ Philip Buczak
buczak@statistik.tu-dortmund.de

Andreas Groll
groll@statistik.tu-dortmund.de

Markus Pauly
pauly@statistik.tu-dortmund.de

Jakob Rehof
jakob.rehof@tu-dortmund.de

Daniel Horn
dhorn@statistik.tu-dortmund.de

[1] Department of Statistics, TU Dortmund University, 44227 Dortmund, Germany

[2] Research Center Trustworthy Data Science and Security, UA Ruhr, 44227 Dortmund, Germany

[3] Department of Computer Science, TU Dortmund University, 44227 Dortmund, Germany

Keywords Machine learning · Hyperparameter tuning · Sequential testing

1 Introduction

Whether for sales prediction, predictive maintenance, sports forecasting, treatment recommendation, neuroimaging analysis or creativity research, machine learning (ML) models are widely used (Adewumi and Akinyelu 2017; Bohanec et al. 2017; Huang et al. 2020; Groll et al. 2019; Susto et al. 2014; Hahn et al. 2022; Buczak et al. 2022). Just as there is a plethora of application problems, there is an ever growing variety of ML algorithms aiming to provide best possible solutions. Thus, the main issue of applying ML often is identifying the algorithm which performs best at the task at hand. The fact that most ML methods have a set of meta parameters (also called *hyperparameters*) whose optimal choice is problem-specific aggravates the problem of algorithm selection.

Usually, problem-optimal choices for hyperparameters are derived from a hyperparameter tuning process aimed at finding parameter settings that minimize the generalization error, i.e., the expected loss on unknown data from the same data generating process. However, the true generalization error is unknown and can only be estimated, e.g., using resampling methods such as k-fold cross-validation or bootstrapping. Thus, minimizing the generalization error is restricted to minimizing an (unbiased) estimate of it. In theory, this poses a stochastic optimization problem, which in practice is commonly approached through heuristically comparing a set of candidate settings from a pre-specified parameter search space. These settings can either be generated by *grid* or *random search* (Bergstra and Bengio 2012). The candidate configuration which minimizes the resampling error then results as the optimal setting. Within the context of stochastic optimization, the resampling error is equivalent to the stochastic target function and the resampling strategy ultimately describes a repeated evaluation in the same parameter.

A key advantage of the random search algorithm is the possibility of parallelizing model evaluations. However, single evaluations within the resampling can still lead to high computational efforts depending on the learner and the dimensionality of the dataset. As such, it would be desirable to stop the evaluation process for parameter settings whose inferior quality is already apparent after a few evaluation steps. An early stopping could prevent redundant computations and potentially save a lot of run time.

The idea of early stopping is also at the core of statistical sequential test theory. Contrary to regular statistical testing with a fixed sample size n, sequential tests dictate a process in which the decision to reject or accept the null hypothesis, or to continue sampling is determined at each sampling step anew. Therefore, sequential tests are especially useful when sampling is costly and it desirable to form a decision based on as few observations as possible. In addition, sequential tests control both, type I and II error, thus allowing for equal treatment of H_0 and H_1, whereas regular statistical tests only allow for rejecting H_0 (Siegmund 1985).

The aim of our work is to investigate the feasibility of employing sequential testing during hyperparameter tuning to save evaluations and computational time. In particular, we aim to answer the following two research questions:

1. Can a proper sequential test be constructed for use in the context of hyperparameter tuning?
2. How does such an approach perform in comparison to a regular random search?

We see studying these two questions as a crucial first step that may open many further avenues of making the hyperparameter tuning process more efficient.

In the next section, we will give a brief introduction to the general ML process and how hyperparameter tuning is incorporated. We introduce the datasets and ML algorithms used in our work in Sect. 3. In Sect. 4, we determine a suitable sequential test and use it to extend the regular random search in Sect. 5. We compare our algorithm and the regular random search in a simulation study on multiple datasets in Sect. 6. Finally, we review and discuss our findings in Sect. 7.

2 Machine learning and hyperparameter tuning

A basic object of ML is modeling a functional mapping $f : \mathcal{X} \to \mathcal{Y}$ between a vector of p features $\boldsymbol{X} = (X_1, \ldots, X_p)^T$ and a target variable Y. Since the true f is usually unknown, a ML method is used to determine an approximation $\hat{f}$ that describes f as well as possible. In the case of supervised learning this is done based on an annotated dataset consisting of n pairs of observations of the form $\left(\boldsymbol{x}_i, y_i\right)_{i=1,\ldots,n}$, where $\boldsymbol{x}_i$ are the feature values of the ith observation and y_i is the corresponding value of the (true) target variable. The goodness of $\hat{f}$ is assessed using a loss function $L(y, \hat{f}(\boldsymbol{x}))$. In the regression case, the squared (also called Gaussian) loss function is a common choice while the 0–1 loss function is commonly used in the classification context (Hastie et al. 2009).

The principal goal is to determine $\hat{f}$ such that it minimizes the generalization error (i.e., the expected loss over all possible data samples). However, since the distributions of $\boldsymbol{X}$ and Y are usually unknown and only finitely many data points are available, $\hat{f}$ is instead obtained by optimizing the estimated generalization error. Because one is generally interested in predicting unknown data as best as possible, the original dataset is usually split into a training and test set. The training set is then used for fitting the model, while the test set is used for evaluating the performance of the model, i.e. for estimating generalization error. Theoretically, it would be possible to use the same data to train and test the model. However, such an approach is problematic because the model is determined to explain the training data as well as possible. Thus, learning and validating on the same data leads to biased, too optimistic error rates (overfitting; Hastie et al. 2009). There exists a variety of different resampling techniques to generate training and test sets from an original dataset. Commonly used techniques include k-fold cross-validation (Hastie et al. 2009) and bootstrapping (Efron and Tibshirani 1993).

When using the squared and 0–1 loss functions, the generalization error is estimated on the test set consisting of n^{test} observation pairs $\left(\boldsymbol{x}_i^{\text{test}}, y_i^{\text{test}}\right)_{i=1,\ldots,n^{\text{test}}}$ via the *mean squared error* (MSE) and *mean misclassification error* (MMCE) performance measures, respectively, where

$$\text{MSE} = \frac{1}{n^{\text{test}}} \sum_{i=1}^{n^{\text{test}}} \left(y_i^{\text{test}} - \hat{f}(\boldsymbol{x}_i^{\text{test}})\right)^2 \text{ and } \text{MMCE} = \frac{1}{n^{\text{test}}} \sum_{i=1}^{n^{\text{test}}} \mathbb{1}_{y_i^{\text{test}} \neq \hat{f}(\boldsymbol{x}_i^{\text{test}})}.$$

An additional challenge in machine learning is that many learning methods have additional hyperparameters $\boldsymbol{\lambda} \in \Lambda$ besides the internally determined model parameters, which must be specified a priori and whose optimality is problem-specific. Thus, for a fixed $\hat{f}_{\boldsymbol{\lambda}}$ the original optimization problem is expanded by an outer optimization problem, which as before can only be approximated by

$$\boldsymbol{\lambda}^\star = \arg\min_{\boldsymbol{\lambda} \in \Lambda} \frac{1}{n^{\text{test}}} \sum_{i=1}^{n^{\text{test}}} L\left(y_i^{\text{test}}, \hat{f}_{\boldsymbol{\lambda}}(\boldsymbol{x}_i^{\text{test}})\right).$$

This optimization problem is also called *hyperparameter tuning problem*. Many algorithms have been developed to solve this problem. All such tuning algorithms work in a similar way: a set $\tilde{\Lambda} \subset \Lambda$ of multiple hyperparameter settings is proposed, the respective values of the loss function are estimated using a resampling procedure and finally, the value $\boldsymbol{\lambda} \in \tilde{\Lambda}$ leading to the smallest loss is used. The strategy of choosing the set $\tilde{\Lambda}$, however, differs for most optimization algorithms. While random and grid search operate on a single large batch, population-based methods such as evolutionary algorithms (EA) keep a population of parameter configurations that is continually optimized. This is achieved by recombining or mutating (i.e. locally transforming) already existing configurations into new candidates or sampling from them from distributions updated via the current population members. A popular EA commonly used for general black-box optimization problems is the *covariance matrix adaptation evolution strategy* (CMA-ES; Hansen and Ostermeier 1996) which has also been adapted specifically for the hyperparameter tuning of SVMs (Friedrichs and Igel 2005) and neural networks (Loshchilov and Hutter 2016). For further uses of EAs in hyperparameter tuning, see, e.g., Bochinski et al. (2017) and Young et al. (2015).

The class of *sequential model-based optimization* (SMBO, also known as Bayesian optimization) uses two components for optimizing parameter configurations: a probabilistic surrogate model and an acquisition function that is usually cheap to evaluate. The surrogate model is updated iteratively based on previous evaluations, while the acquisition function determines suitable new candidates for evaluation. A popular choice are Gaussian processes for the surrogate model combined with the *expected improvement* as acquisition function (Feurer et al. 2019). As alternative to Gaussian processes, neural networks (Snoek et al. 2015), random forests (Hutter et al. 2011) and tree parzen estimators (Bergstra et al. 2011) are used as well. Similar to SMBO, Monte-Carlo tree search (MCTS; Kocsis and Szepesvári 2006) has also been applied to hyperparameter tuning (Rakotoarison et al. 2019). MCTS

combines the classic tree search with ideas from Reinforcement Learning, exploring its search space and iteratively focusing on the most promising regions.

Another commonly employed approach in hyperparameter tuning is the reduction of evaluations, e.g., by eliminating suboptimal parameter configurations. An early example of this are the *Hoeffding Races* (Maron and Moore 1993), in which bounds (resembling a confidence interval) are placed around the error estimates and iteratively updated. If the lower error bound of one model is greater than the upper error bound of at least another model, the former model is discarded. The concept of racing has been modified and extended, for example in Domingos and Hulten (2001) or Mnih et al. (2008). Another popular variation is the *F-Race* algorithm (Birattari et al. 2002), which eliminates bad parameter configurations via the Friedman test. The *Iterated F-Race* algorithm (Birattari et al. 2010) and its extension, *Iterated Racing* (López-Ibáñez et al. 2016), added a population component to the original F-Race algorithm. Similar to us, Krueger et al. (2015) also employ a sequential test for detecting and eliminating weak parameter configurations early. However, their sequential test procedure does not operate directly on the error estimates but on indicators derived from them. Further details on existing hyperparameter optimization algorithms can be found in the review papers by Bergstra et al. (2011), and Yu and Zhu (2020).

3 Datasets and machine learning algorithms

Throughout this work, we benchmarked the performance of multiple ML methods on five regression and five binary classification datasets, see Table 1. The dataset *Insurance* was taken from Stednick (2020), *Diamond* from Wickham (2016), and *Wage* from James et al. (2017). The remaining datasets were obtained from the OpenML platform (Vanschoren et al. 2013). The original *Diamond* dataset contains 54,000 observations. In our analysis, we used a random sample of 5% of the original data.

As ML algorithms, we used decision trees from the `R`-package `rpart` (Therneau and Atkinson 2019), random forest from the `ranger` package (Wright and Ziegler 2017), XGBoost from the `xgboost` package (Chen et al. 2020), as well as elastic net linear regression from the `glmnet` package (Friedman et al. 2010). Table 2 lists the considered hyperparameters and corresponding search spaces for each learning method. Here, for decision trees (`rpart`), `cp` denotes the complexity parameter which specifies by how much a split must contribute to the improvement of the fit so that the corresponding sub-tree is not pruned. Moreover, the parameter `maxdepth` denotes the maximum tree depth. For the random forest, `mtry` describes the number of variables randomly drawn as split candidates, and `sample.fraction` and `replace` describe the fraction of observations used for each tree model and whether or not they are drawn with replacement. For XGBoost, `nrounds` denotes the maximum number of iterations, `eta` the learning rate, and `max_depth` the maximum depth of the trees used. For the elastic net linear regression, `alpha` regulates the mixture parameter of the elastic net regularization and `lambda` the degree

Table 1 Datasets used for simulation studies

Type	Dataset	Obs	Feat	Description
Regression	Boston	506	13	Median housing prices in the Boston area based on neighborhood characteristics
	Insurance	1338	6	Medical insurance charges based on patients
	Diamond	2700	9	Diamond prices based on cut characteristics
	Wage	3000	8	Wages based on socio-economic information
	Concrete	1030	8	Concrete compressive strength based on ingredients
Classification	German credit	1000	20	Credit risk based on customer attributes
	Phoneme	5404	5	Distinction of nasal and oral sounds based on frequency characteristics
	Pima Indians	768	8	Diabetes status in indigenous population based on diagnostic characteristics
	Cancer	569	30	Cancer recognition based on characteristics of a fine needle aspirate of a breast mass
	Ionosphere	351	34	Distinction of ’bad’ and ’good’ radar returns in the ionosphere w.r.t. free electrons

Table 2 Hyperparameters and search spaces used for tuning experiments

Method	R package	Hyperparameter	Support	Search space
Decision Tree	`rpart`	`cp`	[0, 1]	[0, 0.5]
		`maxdepth`	$\{1, \dots, 30\}$	$\{1, \dots, 30\}$
		`mtry`	$\{1, \dots, \#\text{Features}\}$	$\{1, \dots, \#\text{Features}\}$
Random Forest	`ranger`	`replace`	{`TRUE`, `FALSE`}	{`TRUE`, `FALSE`}
		`sample.fraction`	[0, 1]	[0.5, 1]
		`nrounds`	$\{1, 2, \dots\}$	$\{2, \dots, 100\}$
XGBoost	`xgboost`	`eta`	[0, 1]	[0.01, 1]
		`max_depth`	$\{0, 1, \dots\}$	$\{1, \dots, 15\}$
Elastic Net	`glmnet`	`lambda`	$[0, \infty)$	2^x with $x \in [-15, 15]$
		`alpha`	[0, 1]	[0, 1]

of penalization. For the remaining hyperparameters of the individual methods, which are not subject of the optimization, the respective default settings were used.

4 Determining an appropriate sequential test

The main goal of our modified random search is to reduce the number of required evaluation steps while obtaining high performing solutions. As computing resampling errors can be viewed as a sequential process, we regard sequential statistical tests as a natural fit for this kind of situation. For a general overview on sequential statistical tests see, e.g., Ghosh (1970) and Siegmund (1985). An essential class of sequential tests are *sequential probability ratio tests* (SPRT; Wald 1945). For real parameters $\theta_0 < \theta_1$, the classic form of a SPRT for $H_0 : \theta = \theta_0$ vs. $H_1 : \theta = \theta_1$ with $\theta_0 < \theta_1$ can be described as follows: having sampled observation u_n in step $n = 1, 2, \dots$, calculate the test statistic

$$Z_n := \ln \frac{f_n\left(u_1, \dots, u_n; \theta_1\right)}{f_n\left(u_1, \dots, u_n; \theta_0\right)},$$

where $f_n\left(\cdot\,; \theta_i\right)$ denotes the density function corresponding to $\theta_i, i = 0, 1$. Then,

(i) if $Z_n < b$, terminate and accept H_0,
(ii) if $Z_n > a$, terminate and accept H_1,
(iii) if $b < Z_n < a$, continue and sample a new observation u_{n+1},

where $b, a \in \mathbb{R},\ b < a$ define the continuation region $[b, a]$ of the test and are chosen such that the type I and II errors are controlled for pre-specified α and β values.

Many sequential tests are based on parametric assumptions. However, in the context of hyperparameter tuning it is not evident what kind of parametric properties can be assumed for the error estimates. As such, this necessitated an analysis of resampling error distributions beforehand to derive at least approximate parametric

properties. Thus, we first performed a small simulation study in which we fit several common distribution families via standard maximum-likelihood estimation to empirical resampling error distributions. We obtained the latter by benchmarking the ML methods from Section 3 to 1 000 bootstrap samples of the five regression and classification datasets, respectively. The goodness of the respective fits were determined with the *Cramér-von-Mises* (CvM) criterion (see e.g., Stephens 1974) using the `fitdistrplus` `R`-package (Delignette-Muller and Dutang 2015). The purpose of this small simulation study was not to obtain definitive conclusions about the distribution of error estimates, but rather to gain an indication which distribution families are suitable to be used for our heuristic method. The distribution families we considered here are the normal, gamma and Weibull distributions, as well as variations of these in the form of the log-normal, log-gamma, inverse gamma and inverse Weibull distributions.

One problem that arises when using many of these distributions is that, as in the case of the logarithmic and inverse distribution families, the respective support does not contain the value 0 at all or, as in the case of the gamma distribution, has a corresponding density value of 0. This is particularly problematic in the classification context, since errors of 0 are not uncommon for certain combinations of a (sub-) dataset and a learner. An obvious solution is to shift the data by an additive constant $c > 0$. However, it must be noted that the distributions in question are generally not invariant to such shifts, i.e., the distribution family is usually not preserved. Therefore, the goodness of fit can sometimes depend strongly on the choice of c, especially when the observed errors tend to be small, as is the case in classification, where the errors range between 0 and 1. To account for this, the fit of the distribution families was calculated for different shift sizes of $c \in \{0.001, 0.01, 0.1, 0.15, 0.25, 0.5, 1, 1.5\}$, and only the results for the best c are reported for each distribution family. For the regression context, this problem is less relevant, since on one hand, MSE values of 0 only occur in pathological examples and on the other hand, MSE values usually have a higher magnitude compared to MMCE values and are thus less affected by small additive shifts.

Figure 1 shows the CvM values achieved by the different distribution families in the regression case. Smaller CvM values indicate a better fit. Apart from the two Weibull families and the normal distribution, a relatively homogeneous picture emerged.

Similar results were achieved in the classification context as shown in Fig. 2. These findings are supported when looking at other criteria such as the Kolmogorov–Smirnov and Anderson–Darling criteria (results not shown). Thus, regarding the distributional fit, the choice of the distribution family is not crucial as long as one chooses from the set of generally well-performing distribution families.

However, out of these distribution families, the log-normal family offers two notable advantages. First, it allows performing a location test based on only one parameter since the median of a log-normally distributed variable only depends on μ. Second, it allows for applying a sequential test based on a normality assumption after a logarithmic transformation of the original data. Therefore, we decided to assume a log-normal distribution for designing our sequential random search. However, due to the presence of a nuisance parameter (σ^2), regular SPRTs could not be

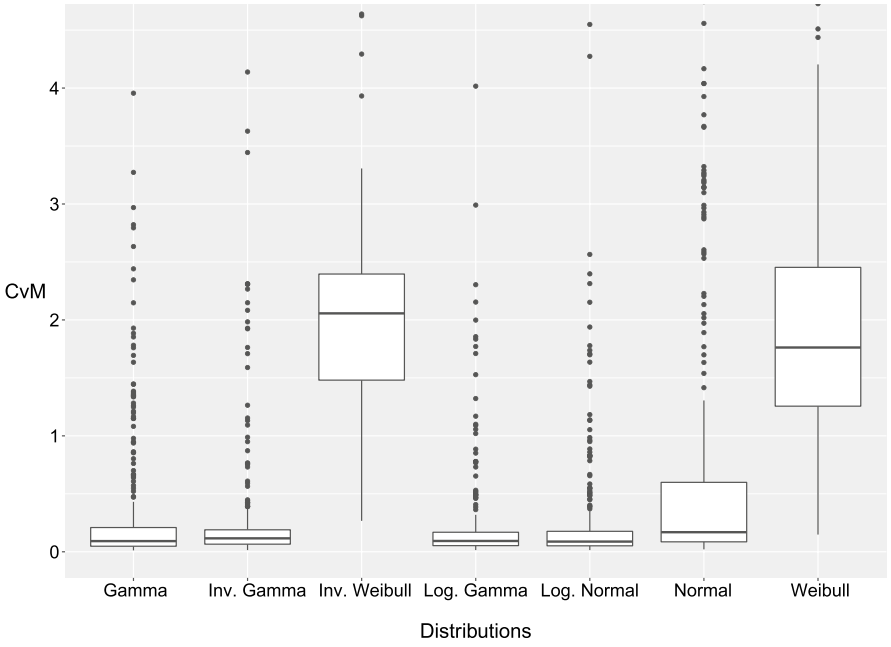

Fig. 1 Cramér–von-Mises criterion values for different distribution classes in the regression case

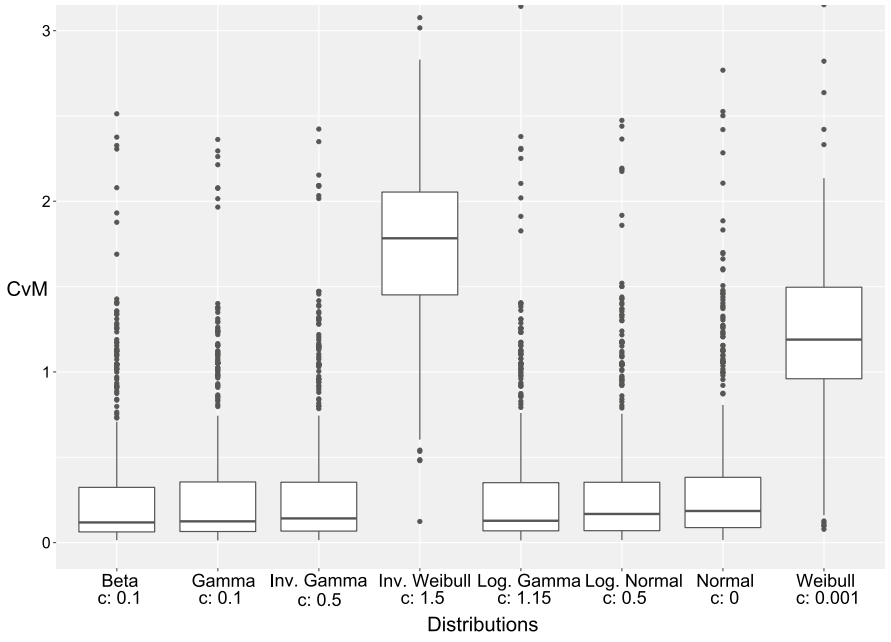

Fig. 2 Cramér–von-Mises criterion values for different distribution classes with individual additive shifts c in the classification case

applied. Thus, we used a *sequential likelihood ratio test* (SLRT) proposed by Ghosh (1970) for the sequential Behrens-Fisher-problem. Here, one considers two normally distributed i.i.d. random variables $U \sim \mathcal{N}(\mu_U, \sigma_U^2)$ and $W \sim \mathcal{N}(\mu_W, \sigma_W^2)$, where all parameters are unknown and one wants to test

$$\mathrm{H}_0 : \gamma := \mu_U - \mu_W = \gamma_0 \text{ vs. } \mathrm{H}_1 : \gamma = \gamma_1 \text{ with } \gamma_0, \gamma_1 \in \mathbb{R},\ \gamma_0 < \gamma_1,$$

with respective type I error rate α and type II error rate β. The continuation region of the test is given by

$$-\frac{s_{u(n)}^2 + s_{w(n)}^2}{\gamma_1 - \gamma_0} \ln \frac{1-\alpha}{\beta} < n\left(\bar{u}_{(n)} - \bar{w}_{(n)} - \frac{\gamma_0 + \gamma_1}{2}\right) < \frac{s_{u(n)}^2 + s_{w(n)}^2}{\gamma_1 - \gamma_0} \ln \frac{1-\alpha}{\beta},$$

where $\bar{u}_{(n)}$ and $\bar{w}_{(n)}$ denote the means and $s_{u(n)}^2$ and $s_{w(n)}^2$ the empirical variances of the respective sample including all observations up to n. Since it holds that

$$\mu_U - \mu_W = \gamma_0 \Leftrightarrow \frac{\exp(\mu_U)}{\exp(\mu_W)} = \exp(\gamma_0) \Leftrightarrow \frac{\mathrm{med}(\tilde{U})}{\mathrm{med}(\tilde{W})} = \exp(\gamma_0),$$

for $\ln(\tilde{U}) \sim \mathcal{N}(\mu_U, \sigma_U^2)$ and $\ln(\tilde{W}) \sim \mathcal{N}(\mu_W, \sigma_W^2)$, the test above is practically a test for the ratio of the medians of the two loss distributions (assuming log-normality) in the context of regression where no additive shift of the resampling errors is needed. In theory, one could also use a two-stage procedure instead of the SLRT, where the variances are estimated in a first step followed by the sequential test in the second step. However, this approach would require a pre-specified number of evaluations for the variance estimation alone. To keep the number of evaluation steps as small as possible, we thus opt for the SLRT.

5 Integrating the sequential test into random search

The general idea now is to combine the regular random search algorithm with the benefits of early stopping from the SLRT. In each iteration, a single new random hyperparameter setting is proposed and used to estimate the corresponding value of the loss function using a resampling strategy. However, instead of using a fixed number of resampling iterations, the resampling is continued until a statistically sound decision can be made. The new setting is compared to the current best setting until a significant difference between these two settings is found. The winner is kept as the new best setting. To save computational effort, evaluation results for a specific candidate are reused from previous comparisons if available. To prevent infinite run times, a maximum number of evaluations is defined. If the test cannot make a decision until this point, the setting with the smallest estimated loss function value is used, or in case of a tie, a winner is chosen at random. Pseudocode describing this sequential random search (SQRS) procedure in detail can be found in Algorithm 1. Here, the function `generateConfig()` can be any function that generates a new hyperparameter setting. In the most simple case, uniform sampling in the search space can be used, resulting in a random search

algorithm. However, it would also be possible to use more advanced optimization algorithms instead. The function `evaluateConfig()` evaluates the performance of a single hyperparameter setting using an arbitrary resampling procedure, e.g., a single bootstrap iteration. Performance values already obtained for `opt.config()` are reused to save additional evaluations. As termination criterion, one can for example use a maximum number of settings to be generated or a maximum computation time.

Algorithm 1 Sequential Random Search (SQRS)

procedure SQRS($max.iter$, γ, α)
  `opt.config` = `generateConfig()`
  while termination condition not fulfilled **do**
    `cand.config` = `generateConfig()`
    for $n = 1$ to $max.iter$ **do**
      $err_{opt,n}$ = `evaluateConfig(opt.config, n)`
      $err_{cand,n}$ = `evaluateConfig(cand.config, n)`
      if $n \geq 2$ **then**
        `upper` = $\frac{\sigma^2(\boldsymbol{err}_{opt,1:n})+\sigma^2(\boldsymbol{err}_{cand,1:n})}{2\gamma} \ln \frac{1-\alpha}{\alpha}$
        `lower` = $-$`upper`
        `statistic` = $n \cdot (\overline{err}_{opt,1:n} - \overline{err}_{cand,1:n})$
        if `statistic` $>$ `upper` **then**
          `opt.config` = `cand.config`
          break
        end if
        if `statistic` $<$ `lower` **then**
          break
        end if
        if $n = max.iter$ **then**
          `opt.config` = $\arg\min_{i \in \{opt,cand\}} \overline{err}_{i,1:n}$
        end if
      end if
    end for
  end while
  return `opt.config`
end procedure

Apart from the maximum number of evaluations *max.iter*, the SQRS procedure possesses two hyperparameters, α and γ, that are inherited from the sequential test and must be specified in advance. Both modulate how conservative SQRS is w.r.t. discarding inferior configurations and thus, both parameters balance the trade-off between preciseness and computational gain. Originally, the sequential test contains two significance levels, α and β, as well as the mean differences γ_1 and γ_0 that are tested for. Though one could specify all four parameters separately for SQRS, it appears most natural to test symmetrically, i.e., $\alpha = \beta$ and $\gamma_1 = -\gamma_0$.

As such, the SQRS requires the input of α and γ, and internally sets $\beta = \alpha$, $\gamma_1 = \gamma$, and $\gamma_0 = -\gamma$ when using the sequential test.

6 Simulation study

We will now compare SQRS with a regular random search in a simulation study designed to ensure maximum comparability between the two algorithms. Using the same learners, parameter search spaces and datasets as in Tables 1–2, we generated 1000 random configurations of hyperparameters for each combination of learner and dataset, and validated their performance using a bootstrap resampling with ten iterations. To make the resampling results comparable, we used the same resampling instance for both algorithms. We performed 100 replications. For the random search, we simply selected the parameter setting leading to the lowest mean resampling error (i.e., MMCE for classification or MSE for regression). For SQRS, we did not generate new random hyperparameter settings, but instead operated on the same set of parameter configurations as for the regular random search. As a consequence, SQRS could not find a better setting than the regular random search. Thus, for this simulation study, we were interested what the performance loss is when using SQRS and what computational gain in terms of saved evaluations could be achieved. The performance loss was measured by computing the relative percentage difference

$$RPD = \frac{err_{sqrs} - err_{rs}}{err_{rs}} \cdot 100\%,$$

where err_{sqrs} is the resampling error obtained by the SQRS candidate and err_{rs} denotes the resampling error achieved by the random search solution. Using the RPD as the performance loss measure helps comparing the performance loss over different datasets as it takes the scale of the error values into account. As for its interpretation, an RPD of, e.g., 1% would imply that the SQRS mean resampling error is 1% larger than the random search mean resampling error.

In our simulation, we considered four SQRS settings *A-D* for the regression case which varied in the choice of the SQRS parameters $\alpha \in \{0.01, 0.05\}$ and $\gamma \in \{0.1, 0.2\}$. In the classification case, we considered SQRS settings *E-H* where $\alpha \in \{0.01, 0.05\}$ and $\gamma \in \{0.01, 0.02\}$. The choices for γ were derived heuristically from a small simulation study that analyzed the power of the sequential test using the datasets and learners described above (results omitted here). In the regression context, the choices for γ roughly correspond to testing whether the ratio of the two resampling error distribution medians differ by 10% or 20%, respectively. Note that this is not an exact relation due to the exponentiation (e.g., $e^{-0.1} = 0.904$ and $e^{0.1} = 1.105$), but it can be thought of as an intuition for the meaning of γ. Due to the additive shift needed for classification errors, γ is not as easily interpretable as in the regression context. As specified by the resampling instance, the maximum number of evaluations was $max.iter = 10$.

Table 3 displays the mean RPD and its standard deviation (sd) w.r.t. the different parameter settings when aggregated over datasets and learners. It can be seen that

Table 3 Mean RPD and percentage of saved evaluations (standard deviations in parentheses) obtained by SQRS for different specifications of γ and α aggregated over datasets and learners

Type	Setting	γ	α	RPD (sd)	% Evaluations saved (sd)
Regression	A	0.2	0.05	0.26 (1.32)	64.01 (19.67)
	B	0.2	0.01	0.17 (0.76)	60.87 (20.89)
	C	0.1	0.05	0.13 (0.68)	58.83 (21.61)
	D	0.1	0.01	0.10 (0.61)	54.74 (22.81)
Classification	E	0.02	0.05	1.30 (3.47)	66.47 (14.30)
	F	0.02	0.01	0.86 (2.56)	59.73 (18.07)
	G	0.01	0.05	0.69 (2.24)	55.27 (20.13)
	H	0.01	0.01	0.50 (1.86)	46.54 (23.09)

the performance loss suffered from using SQRS was generally low. In the regression case, mean RPDs ranged between 0.10% (sd: 0.61%) and 0.26% (sd: 1.32%), i.e., the MSE reached by SQRS was at worst 0.26% larger (on average) compared to regular random search. As expected, the mean RPD decreased the smaller γ and α were chosen. This also held for the standard deviation of the RPD. In the classification case, the RPDs were slightly larger ranging from 0.50% (sd: 1.86%) to 1.30% (sd: 3.47%), and also displayed more variability when compared to the regression results. At the median, the RPD was 0 in all scenarios. When looking at the percentage of saved evaluations, SQRS on average saved between 54.74% (sd: 22.81%) and 64.01% (sd: 19.67%) of evaluations in the regression case and between 46.54% (sd: 23.09%) and 66.47% (sd: 14.30%) in the classification case. Thus, SQRS could on average cut the evaluations needed in all but one scenario by at least a half. Stricter choices for γ and α led to a decrease in mean evaluations saved and to an increased variability. A possible explanation for the latter could be that for parameter configurations with similar performance, SQRS takes more evaluations before coming to a decision (due to the lower tolerance for differences and/or errors) while for parameter configurations with distinctly different performances, SQRS can still discard the inferior candidate quickly. As such, this would lead to larger variability regarding the evaluation steps required.

When stratifying the performance loss by dataset and learner, Table 4 shows that the results were somewhat heterogeneous, especially in the classification case. For space reasons, we only show the most liberal (i.e., A for regression and E for classification) and most conservative settings (i.e., D for regression and H for classification) here. We refer to the Supplement for result tables for all SQRS settings. For regression, SQRS performed best on the *Insurance* and *Wage* datasets incurring almost no performance loss in most scenarios. In the classification context, the performance loss was slightly higher but still low overall. Here, SQRS performed best on the *German Credit*, *Phoneme* and *Pima Indians* datasets. For the *Ionosphere* and *Cancer* datasets, the mean RPDs and respective standard deviations were notably higher than for the other datasets. Regarding the learners, the lowest mean RPD was generally achieved for decision trees followed by elastic net and random forest. While for these three learners the mean RPD was seldomly larger than 1%, the mean

Table 4 Mean RPD (standard deviations in parentheses) w.r.t. datasets, learners and SQRS settings for regression (A, D) and classification (E, H)

Dataset	Setting	RPD (sd)			
		Decision tree	Random forest	XGBoost	Elastic Net
Boston	A	0.09 (0.30)	0.46 (1.41)	1.08 (2.39)	0.05 (0.21)
	D	0.08 (0.27)	0.10 (0.34)	0.38 (0.86)	0.02 (0.15)
Insurance	A	0.19 (0.56)	0.03 (0.14)	0.11 (0.45)	0.01 (0.04)
	D	0.07 (0.42)	0.01 (0.10)	0.03 (0.15)	0.00 (0.01)
Diamond	A	0.01 (0.07)	0.22 (0.54)	0.76 (1.33)	0.66 (4.22)
	D	0.01 (0.07)	0.07 (0.18)	0.44 (1.35)	0.20 (1.29)
Wage	A	0.18 (0.47)	0.01 (0.03)	0.04 (0.10)	0.00 (0.01)
	D	0.11 (0.36)	0.00 (0.02)	0.02 (0.08)	0.00 (0.01)
Concrete	A	0.05 (0.30)	0.19 (0.44)	0.99 (1.99)	0.04 (0.20)
	D	0.00 (0.00)	0.12 (0.37)	0.36 (1.52)	0.01 (0.05)
German credit	E	0.58 (1.34)	0.54 (0.85)	1.11 (1.51)	0.39 (0.78)
	H	0.16 (0.68)	0.16 (0.47)	0.38 (0.80)	0.09 (0.24)
Phoneme	E	0.29 (0.70)	0.66 (0.65)	1.24 (1.02)	0.03 (0.07)
	H	0.09 (0.34)	0.32 (0.49)	0.51 (0.78)	0.01 (0.06)
Pima Indians	E	0.77 (1.71)	0.49 (0.79)	0.86 (1.47)	0.40 (1.06)
	H	0.26 (1.02)	0.14 (0.35)	0.28 (0.85)	0.17 (0.78)
Cancer	E	0.71 (2.23)	3.36 (4.68)	4.16 (5.96)	4.71 (9.46)
	H	0.11 (0.41)	1.25 (2.60)	1.47 (3.67)	2.34 (5.10)
Ionosphere	E	0.58 (1.82)	1.53 (4.55)	2.53 (3.85)	1.06 (2.17)
	H	0.02 (0.14)	0.62 (2.08)	0.98 (2.28)	0.53 (1.41)

RPDs for XGBoost were around 1–2% more often and reached up to 4.16% on the *Cancer* dataset in the liberal setting. Overall, when regarding the median RPD, values of 0 were achieved in 72 out of the 80 combinations of dataset, SQRS setting and learner.

Table 5 shows the mean percentage of evaluations saved by SQRS stratified by dataset and learner. Overall, the results here were quite heterogeneous. Regarding the learners, SQRS could save evaluations the most when using decision trees (between 66.31% and 79.11% in the strict regression setting, and between 39.21% and 78.65% in the strict classification setting) and XGBoost (between 66.31% and 79.11% in the strict regression setting, and between 39.21% and 78.65% in the strict classification setting). For the liberal settings, the mean percentages of saved evaluations were higher, respectively. The least evaluations could be saved for elastic net in the regression case where for the most strict setting between 17.87% and 46.48% of the possible evaluations were saved. Interestingly, for classification, the results for elastic net were more in line with the results for the other learners. In the strict classification setting, elastic net could on average even save the most evaluations as the other three learners reacted more sensitively to the change from the liberal to the strict setting. Overall, we find the results from the simulation quite promising as the

Table 5 Mean percentage of evaluations (standard deviations in parentheses) saved by SQRS w.r.t. datasets, learners and SQRS settings for regression (A, D) and classification (E, H)

Dataset	Setting	% Evaluations saved (sd)			
		Decision tree	Random forest	XGBoost	Elastic Net
Boston	A	76.06 (3.90)	47.38 (24.94)	64.70 (12.77)	46.21 (10.50)
	D	66.31 (10.77)	23.69 (24.04)	42.13 (17.96)	39.22 (7.39)
Insurance	A	78.32 (1.34)	68.28 (10.45)	77.92 (2.23)	24.39 (21.11)
	D	75.52 (3.69)	50.25 (19.84)	73.30 (6.71)	17.87 (15.59)
Diamond	A	79.43 (0.38)	61.19 (15.62)	76.06 (4.55)	37.41 (12.79)
	D	78.56 (0.81)	42.90 (20.93)	66.13 (11.46)	29.79 (10.58)
Wage	A	78.89 (0.88)	71.76 (5.19)	76.13 (3.12)	39.61 (10.69)
	D	76.52 (3.74)	63.62 (10.40)	71.95 (5.23)	35.21 (7.61)
Concrete	A	79.65 (0.25)	69.20 (11.84)	76.19 (3.62)	51.45 (11.42)
	D	79.11 (0.75)	51.03 (20.88)	65.17 (9.85)	46.48 (10.80)
German Credit	E	67.91 (12.17)	58.90 (17.07)	67.92 (10.07)	68.38 (10.31)
	H	42.43 (25.60)	29.53 (19.38)	38.45 (18.21)	52.17 (14.04)
Phoneme	E	79.49 (0.32)	77.50 (3.81)	78.65 (1.71)	66.97 (6.78)
	H	78.65 (0.81)	67.14 (11.70)	72.61 (6.15)	61.87 (7.73)
Pima Indians	E	65.96 (9.61)	55.27 (19.39)	69.47 (9.30)	62.55 (12.28)
	H	39.63 (14.93)	23.18 (19.38)	44.64 (18.20)	51.25 (11.05)
Cancer	E	71.34 (7.91)	64.55 (15.95)	67.55 (13.83)	75.92 (3.81)
	H	54.10 (20.58)	33.95 (23.16)	38.31 (21.39)	67.03 (7.24)
Ionosphere	E	57.16 (8.69)	54.02 (21.41)	53.38 (19.80)	66.49 (8.20)
	H	39.21 (12.00)	24.82 (22.66)	19.61 (16.82)	52.13 (8.93)

performance loss seemed negligible in many scenarios, while at least 20% and up to 80% of evaluations could be saved by SQRS.

7 Discussion

In this work, we analyzed the feasibility of employing sequential statistical tests during the hyperparameter tuning process to save computational effort. We aimed at answering two main research questions. The first one pertained to the construction of a suitable sequential test for hyperparameter tuning. To study what kind of approximate parametric assumption could be made for the resampling error distributions, we performed a small simulation study in which we fitted multiple different distribution families to empirical resampling error distributions. Overall, typical flexible distribution families achieved comparably good fits: the gamma, the inverse gamma, the log-gamma and the log-normal distribution. We recognize that this approach was purely empirical, and no definitive conclusions should be derived from it. We decided for this approach to limit the scope of this work which serves more as a proof of concept for our heuristic method.

Although multiple distribution families appeared suitable, we decided for the log-normal family for practical reasons. Using a sequential test by Ghosh (1970), we implemented a sequential variant of the random search (abbreviated SQRS). The main difference between SQRS and regular random search is the early stopping possibility of the former. After each evaluation step, the error values of the two configurations up to this point are compared using the sequential test procedure. When a terminating decision (i.e., discarding one of the configurations early) cannot be made before reaching the maximum number of permitted evaluation steps, the procedure selects the parameter configuration leading to the smaller resampling error. Due to the sequential test, SQRS requires the input of two problem-specific hyperparameters α and γ. Even though we were able to derive settings for both classification and regression that worked well for our sets of data, definite recommendations would require more research on more datasets. Ultimately, however, it also up to the users to decide how they aim to navigate the balance between preciseness and computational gain, and what matters more in their specific use-case. While in principle both parameters could be tuned, their nature as parameters originating from a statistical test would lead us to advise proceeding with caution in this regard. As for choosing the maximum number of evaluations, we propose that when using SQRS as a counterpart to, e.g., k-fold cross-validation, a canonical choice is given by k. This choice could be adjusted depending on the specific situation. Choosing the value too low may harm the method's ability to discern differences between hyperparameter settings. On the other hand, finer differences can be identified by allowing for a high number of maximum evaluations. However, this negatively affects the runtime. For future work, it would be interesting to study the effect of this parameter in more detail.

Having implemented SQRS, our focus then turned to the second research question of how a hyperparameter tuning approach using a sequential statistical test would perform compared to a regular random search. We performed a simulation experiment in which SQRS and regular random search tackled the same regression and classification problems under identical conditions. We found that by using the sequential testing procedure instead of a full resampling, the number of evaluations could be greatly reduced without considerable performance loss. For the regression problems we studied, we could save between 55% and 64% of evaluations while incurring a MSE increase between 0.10% and 0.26% on average. For classification problems, we could save between 47% and 67% of evaluations while suffering from a MMCE increase between 0.50% and 1.30% on average. However, a closer look at the findings also revealed severe heterogeneity between the different learners. A possible explanation may be that some learners react more sensitive to changes in parameters while other learners might be more robust and display only small performance changes. The latter would then prolong the sequential testing process as more observations would be needed to discern small performance differences. In any case, the discovered heterogeneity indicates that the differences between learners should be analyzed more closely in the future.

We recognize our experiments are just a first proof of concept. Our comparisons were aimed at maximal comparability since both algorithms were forced to operate under the same laboratory conditions using an identical set of candidate

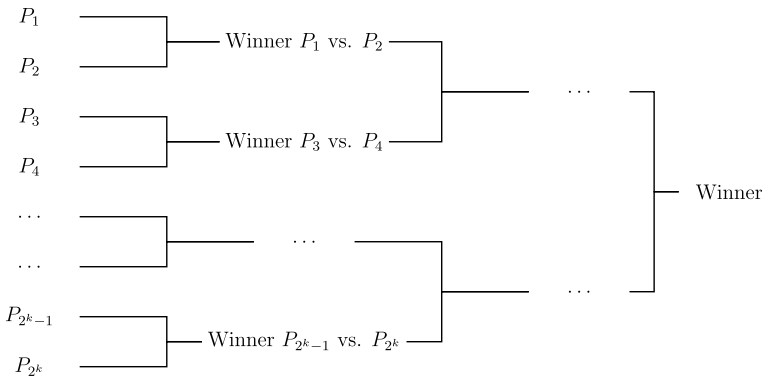

Fig. 3 Exemplary parallel variant of the SQRS for parameter configurations $P_1, \ldots, P_{2^k}$

settings. We are fully aware that our SQRS algorithm loses the most important advantage of random search: the ability to use (nearly) unlimited parallel computation power. However, we believe the SQRS procedure could also be enhanced by parallelization. One possibility could be to implement a procedure resembling a bracket from knockout tournaments in sports competitions as depicted in Fig. 3. For 2^k different parameter configurations $P_1, \ldots, P_{2^k}$ to be tested, the individual duels could be executed in a parallel fashion. Of course, the level of parallelization is less than for a default random search. Nonetheless, we also think it is important to consider the reduced computational effort from the perspective of sustainability. Solving a hyperparameter tuning problem using a highly parallelized random search may be a simple and efficient approach. However, it is also an approach that consumes a lot of resources (i.e., electrical power). More economical approaches are reasonable, and we hope that our approach can contribute here.

Ultimately, we do not view SQRS as a finished solution to the hyperparameter optimization problem, but rather as a proof of concept that shows the promising potential of integrating sequential statistical testing in hyperparameter tuning. At their core, most tuning algorithms can be distilled into two main components: a search strategy for generating new parameter settings and means of comparing their performance. We believe that SQRS represents a novel approach for improving the latter and that it could be used for enhancing already established tuning algorithms that include more sophisticated search strategies, e.g., MBO or EAs. To us, this points to the next logical step for future work. Such work could then also include an encompassing comparison study between already existing tuning algorithms and their enhanced counterparts. As such extensions are non-trivial, they were out of scope for this work. Overall, we believe that there remains a lot of untapped potential in integrating sequential test procedures into hyperparameter tuning that warrants further investigation in future work.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10182-024-00495-1.

Acknowledgements The authors gratefully acknowledge the computing time provided on the Linux HPC cluster at Technical University Dortmund (LiDO3), partially funded in the course of the Large-Scale Equipment Initiative by the German Research Foundation (DFG) as project 271512359.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest None of the authors have any conflict of interest to disclose.

References

Adewumi, A.O., Akinyelu, A.A.: A survey of machine-learning and nature-inspired based credit card fraud detection techniques. Int. J. Syst. Assur. Eng. Manag. **8**(2), 937–953 (2017)

Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)

Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K.Q. (eds.) Proceedings of the 24th International Conference on Neural Information Processing Systems. NIPS'11, pp. 2546–2554, Granada (2011)

Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. GECCO'02, pp. 11–18, New York (2002)

Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) F-Race and Iterated F-Race: An Overview, pp. 311–336. Springer, Berlin (2010)

Bochinski, E., Senst, T., Sikora, T.: Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In: 2017 IEEE International Conference on Image Processing (ICIP), pp. 3924–3928 (2017)

Bohanec, M., Borštnar, M.K., Robnik-Šikonja, M.: Explaining machine learning models in sales predictions. Expert Syst. Appl. **71**, 416–428 (2017)

Buczak, P., Huang, H., Forthmann, B., Doebler, P.: The machines take over: a comparison of various supervised learning approaches for automated scoring of divergent thinking tasks. J. Creat. Behav. **57**, 17–36 (2022)

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., Mitchell, R., Cano, I., Zhou, T., Li, M., Xie, J., Lin, M., Geng, Y., Li, Y.: Xgboost: Extreme gradient boosting. (2020). R package version 1.0.0.2. https://CRAN.R-project.org/package=xgboost

Delignette-Muller, M.L., Dutang, C.: fitdistrplus: an R package for fitting distributions. J. Stat. Softw. **64**(4), 1–34 (2015)

Domingos, P., Hulten, G.: A general method for scaling up machine learning algorithms and its application to clustering. In: Proceedings of the 18th International Conference on Machine Learning. ICML '01, pp. 106–113, Williamstown (2001)

Efron, B., Tibshirani, R.: An Introduction to the Bootstrap. Chapman & Hall/CRC, Boca Raton (1993)

Feurer, M., Hutter, F.: In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Hyperparameter Optimization, pp. 3–33. Springer, Cham (2019)

Friedman, J., Hastie, T., Tibshirani, R.: Regularization paths for generalized linear models via coordinate descent. J. Stat. Softw. **33**(1), 1–22 (2010)

Friedrichs, F., Igel, C.: Evolutionary tuning of multiple SVM parameters. Neurocomputing 64, 107–117 (2005). Trends in Neurocomputing: 12th European Symposium on Artificial Neural Networks 2004

Ghosh, B.K.: Sequential Tests of Statistical Hypotheses. Addison-Wesley Publishing Company, London (1970)

Groll, A., Ley, C., Schauberger, G., Eetvelde, H.V.: A hybrid random forest to predict soccer matches in international tournaments. J. Quant. Anal. Sports **15**(4), 271–287 (2019)

Hahn, T., Ernsting, J., Winter, N.R., Holstein, V., Leenings, R., Beisemann, M., Fisch, L., Sarink, K., Emden, D., Opel, N., Redlich, R., Repple, J., Grotegerd, D., Meinert, S., Hirsch, J.G., Niendorf, T., Endemann, B., Bamberg, F., Kröncke, T., Bülow, R., Völzke, H., von Stackelberg, O., Sowade, R.F., Umutlu, L., Schmidt, B., Caspers, S., Kugel, H., Kircher, T., Risse, B., Gaser, C., Cole, J.H., Dannlowski, U., Berger, K.: An uncertainty-aware, shareable, and transparent neural network architecture for brain-age modeling. Sci. Adv. **8**(1), 9471 (2022)

Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp. 312–317 (1996)

Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning, 2nd edn. Springer, New York (2009)

Huang, H., Pouls, M., Meyer, A., Pauly, M.: Travel time prediction using tree-based ensembles. In: International Conference on Computational Logistics, pp. 412–427 (2020). Springer

Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) Learning and Intelligent Optimization. Springer, Berlin (2011)

James, G., Witten, D., Hastie, T., Tibshirani, R.: ISLR: Data for an Introduction to Statistical Learning with Applications in R. (2017). R package version 1.2. https://CRAN.R-project.org/package=ISLR

Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) Machine Learning: ECML 2006, pp. 282–293. Springer, Berlin (2006)

Krueger, T., Panknin, D., Braun, M.: Fast cross-validation via sequential testing. J. Mach. Learn. Res. **16**(33), 1103–1155 (2015)

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)

Loshchilov, I., Hutter, F.: CMA-ES for hyperparameter optimization of deep neural networks. arXiv (2016). https://arxiv.org/abs/1604.07269

Maron, O., Moore, A.W.: Hoeffding races: Accelerating model selection search for classification and function approximation. In: Cowan, J.D., Tesauro, G., Alspector, J. (eds.) Advances in Neural Information Processing Systems, vol. 6, pp. 59–66 (1993). Morgan-Kaufmann

Mnih, V., Szepesvári, C., Audibert, J.-Y.: Empirical bernstein stopping. In: Proceedings of the 25th International Conference on Machine Learning. ICML '08, pp. 672–679, Helsinki (2008)

Rakotoarison, H., Schoenauer, M., Sebag, M.: Automated Machine Learning with Monte-Carlo Tree Search. arXiv (2019). https://arxiv.org/abs/1906.00170

Siegmund, D.: Sequential Analysis: Tests and Confidence Intervals. Springer, New York (1985)

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M.M.A., Prabhat, P., Adams, R.P.: Scalable Bayesian optimization using deep neural networks. In: Proceedings of the 32nd International Conference on International Conference on Machine Learning—Volume 37. ICML'15, pp. 2171–2180, Lille (2015)

Stednick, Z.: Machine learning with R datasets. GitHub (2020). https://github.com/stedy/Machine-Learning-with-R-datasets

Stephens, M.A.: EDF statistics for goodness of fit and some comparisons. J. Am. Stat. Assoc. **69**(347), 730–737 (1974)

Susto, G.A., Wan, J., Pampuri, S., Zanon, M., Johnston, A.B., O'Hara, P.G., McLoone, S.: An adaptive machine learning decision system for flexible predictive maintenance. In: 2014 IEEE International Conference on Automation Science and Engineering (CASE), pp. 806–811 (2014). IEEE

Therneau, T., Atkinson, B.: Rpart: Recursive Partitioning and Regression Trees. (2019). R package version 4.1-15. https://CRAN.R-project.org/package=rpart

Vanschoren, J., N. van Rijn, J., Bischl, B., Torgo, L.: OpenML: Networked science in machine learning. SIGKDD Explorations 15, 49–60 (2013)

Wald, A.: Sequential tests of statistical hypotheses. Ann. Math. Stat. **16**(2), 117–186 (1945)

Wickham, H.: Ggplot2: Elegant Graphics for Data Analysis. Springer, New York (2016)

Wright, M.N., Ziegler, A.: ranger: A fast implementation of random forests for high dimensional data in C++ and R. J. Stat. Softw. **77**, 1–17 (2017)

Young, S.R., Rose, D.C., Karnowski, T.P., Lim, S.-H., Patton, R.M.: Optimizing deep learning hyper-parameters through an evolutionary algorithm. In: Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments, Austin (2015)

Yu, T., Zhu, H.: Hyper-parameter optimization: a review of algorithms and applications. arXiv (2020). https://arxiv.org/abs/2003.05689