Using Sequential Statistical Tests to Improve the Performance of Random Search in hyperparameter Tuning

Philip Buczak  
Department of Statistics  
TU Dortmund University  
Vogelpothsweg 87, 44227 Dortmund, Germany  

Daniel Horn  
Department of Statistics  
TU Dortmund University  
Vogelpothsweg 87, 44227 Dortmund, Germany  

Editor:  

Abstract  
Hyperparameter tuning is one of the the most time-consuming parts in machine learning: The performance of a large number of different hyperparameter settings has to be evaluated to find the best one. Although modern optimization algorithms exist that minimize the number of evaluations needed, the evaluation of a single setting is still expensive: Using a resampling technique, the machine learning method has to be fitted a fixed number of $K$ times on different training data sets. As an estimator for the performance of the setting the respective mean value of the $K$ fits is used. Many hyperparameter settings could be discarded after less than $K$ resampling iterations, because they already are clearly inferior to high performing settings. However, in practice, the resampling is often performed until the very end, wasting a lot of computational effort.  

We propose to use a sequential testing procedure to minimize the number of resampling iterations to detect inferior parameter setting. To do so, we first analyze the distribution of resampling errors, we will find out, that a log-normal distribution is promising. Afterwards, we build a sequential testing procedure assuming this distribution. This sequential test procedure is utilized within a random search algorithm.  

We compare a standard random search with our enhanced sequential random search in some realistic data situation. It can be shown that the sequential random search is able to find comparably good hyperparameter settings, however, the computational time needed to find those settings is roughly halved.  

Keywords: Machine Learning, Hyperparameter Tuning, Sequential Testing

1. Introduction  
Whether for sales prediction, fraud detection or treatment recommendation, machine learning (ML) models are widely used in diverse application contexts. Just as there is a plethora of application problems, there is an evergrowing variety of ML algorithms aiming to provide best possible solutions. Thus, the main issue of applying ML is identifying the algorithm which performs best at the task at hand. However, as the famous no free lunch theorem (Wolpert and Macready, 1997) predicates, no single algorithm performs best at all tasks.
The problem of algorithm selection is aggrevated by the fact that most ML methods have a set of meta parameters (also called hyperparameters) whose optimal choice is problem-specific.

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 (DGP). 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. The candidate configuration which minimizes the resampling error (e.g., measured by the misclassification rate or MSE) 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 subpar quality becomes already apparent after a few evaluation steps. An early stopping could prevent redundant computations and save potentially a lot of run time.

The idea of early stopping is also at the core of statistics 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 hypothesis, or to continue sampling is determined at each sampling step anew. Therefore, sequential tests are especially useful when sampling is costly and being able to form a decision on as few observations as possible is desirable. Furthermore, since both, Type I and II error are controlled for, sequential tests allow for treating $H_0$ and $H_1$ equally important whereas regular statistical tests only allow for rejecting $H_0$, but not deciding for it.

The aim of our work is, thus, to study the feasibility of employing sequential tests during hyperparameter tuning to save evaluations and computation time. One main requirement for sequential tests is the existence of a potentially endless DGP to generate new samples. This seems to be met in hyperparameter tuning, since the DGP is represented by the resampling which, using (pseudo) random numbers, can in principle, generate arbitrarily many training and test samples from the original dataset. Many sequential tests are also based on some 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 necessitates an analysis of resampling error distributions beforehand to potentially derive at least approximate parametric properties.

If this succeeds, the question then becomes whether a tuning algorithm based on such a sequential test would be able to compete with the regular random search and other efficient tuning algorithms. This leads to the following three research questions in our work:

1. Which distribution family can be assumed for resampling errors?
2. How can a sequential statistical test for hyperparameter tuning problems be constructed?

3. How does a hyperparameter tuning approach using a sequential statistical test perform compared to a normal random search?

In the next section, we will give a brief introduction to the general machine learning scheme and how hyperparameter tuning is incorporated, especially to fix some notation. In section three, we will look at the first research question and try to find a sound distribution family for resampling errors. Section four again is a theoretical chapter, in which a sequential test procedure is introduced, and a sequential random search using this test procedure is constructed. In section five we conduct some experiments in which we show, that our sequential random search achieves comparable results to a usual random search, however, using significantly less resampling iterations. The paper is wrapped up in section six.

2. Hyperparameter Optimization

The basic object of machine learning is modeling a functional context $f: \mathcal{X} \rightarrow \mathcal{Y}$ between a vector of $p$ variables $\mathbf{X} = (X_1, \ldots, X_p)^T$ and a target variable $Y$. Since the true $f$ is mostly unknown, a (machine) learning method is used to determine an approximation $\hat{f}$ that describes $f$ as "good" as possible.

This is done in the case of supervised learning based on an annotated learning dataset $D^{\text{learn}}$ consisting of $n^{\text{learn}}$ pairs of observations of type $(x_i^{\text{learn}}, y_i^{\text{learn}})_{i=1,\ldots,n^{\text{learn}}}$ where $x_i^{\text{learn}}$ are the influence variable values of the $i$th observation from the learning dataset and $y_i^{\text{learn}}$ is the corresponding (true) target variable value. The approximation $\hat{f}$ is determined on the so-called training data $D^{\text{train}}$ consisting of $n^{\text{train}}$ pairs of observations of type $(x_i^{\text{train}}, y_i^{\text{train}})_{i=1,\ldots,n^{\text{train}}}$. The goodness of $\hat{f}$ is assessed using a loss function $L(y, \hat{f}(x))$. A common choice of loss function in the regression case would be the quadratic (also called Gaussian) loss function (Hastie et al., 2009, p. 18)

$$L_{sq}(y, \hat{f}(x)) = (y - \hat{f}(x))^2$$

and in the classification case the 0-1 loss function (Hastie et al., 2009, p. 221) with

$$L_{0-1}(y, \hat{f}(x)) = 1_{y \neq \hat{f}(x)}.$$

The principal goal is to determine an approximation $\hat{f}^*$ which minimizes the generalization error (the expected loss over all possible data), i.e. (cf. Bischi et al., 2012, p. 251f.)

$$\hat{f}^* = \arg\min_{\hat{f}} E_{\mathbf{X},Y} \left( L \left( Y, \hat{f}(\mathbf{X}) \right) \right).$$

However, since the distributions of $\mathbf{X}$ and $Y$ are usually unknown in practice and only finitely many data points are available, $\hat{f}^*$ is instead obtained by optimizing the estimated generalization error on a test data set $D^{\text{test}}$ consisting of pairs of observations $(x_i^{\text{test}}, y_i^{\text{test}})_{i=1,\ldots,n^{\text{test}}}$, i.e.

$$\hat{f}^* = \arg\min_{\hat{f}} \frac{1}{n^{\text{test}}} \sum_{i=1}^{n^{\text{test}}} L \left( y_i^{\text{test}}, \hat{f}(x_i^{\text{test}}) \right).$$
Using the above loss functions, the estimation of the generalization error then yields the two performance measures \( \text{MSE} \) (Mean Squared Error) and \( \text{MMCE} \) (Mean Misclassification Error) with (cf. James et al., 2017, pp. 29 u. 37):

\[
\text{MSE} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \left( y_{i_{\text{test}}} - \hat{f}(x_{i_{\text{test}}}) \right)^2 \quad \text{and} \quad \text{MMCE} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \mathbb{1}_{y_{i_{\text{test}}} \neq \hat{f}(x_{i_{\text{test}}})}.
\]

Theoretically, it would be possible to use the same data to learn and to validate the model, i.e. \( \hat{f} \), i.e. \( D_{\text{learn}} = D_{\text{train}} = D_{\text{test}} \). However, such an approach is problematic in that the model is determined such that it explains the training data as well as possible. Thus, learning and validating on the same data leads to biased, optimistic error rates (see Hastie et al., 2009, pp. 228ff.).

In general, it is desirable to predict unknown observations as well as possible with the model. Therefore, only observations that were not already part of the training data should be used for the validation of the model: \( D_{\text{test}} \cap D_{\text{train}} = \emptyset \). Usually, resampling methods are used to repeatedly generate training and test datasets \( D_{\text{test}}^k \) and \( D_{\text{train}}^k \), \( k = 1, \ldots, K \) from an original dataset \( D_{\text{learn}} \) with \( D_{\text{test}}^k \cup D_{\text{train}}^k = D_{\text{learn}} \). One common example is Bootstrapping, where \( D_{\text{train}}^k \) is obtained by drawing \( n_{\text{train}} = n_{\text{learn}} \) observations with replacement from \( D_{\text{learn}} \) (i.e., some observations will be drawn more than once), and \( D_{\text{test}}^k = D_{\text{learn}} \setminus D_{\text{train}}^k \). For an overview of other resampling methods as well as application recommendations, see Bischl et al. (2012). All resampling methods have in common that they calculate random partitions \( D_{\text{train}}^k \) and \( D_{\text{test}}^k \). Hence, values of the loss functions \( L \) whose calculation is based on such a partition are random numbers.

An additional complexity that often arises in determining an optimal model \( \hat{f}^* \) is that many learning methods have additional meta-parameters \( \lambda \in \Lambda \), so-called hyperparameters, in addition to the internally determined model parameters, which must be determined a priori and whose optimality is problem-specific. Thus, for a fixed \( \hat{f}_\lambda \) the original optimization problem is expanded by the outer optimization problem (cf. Bergstra and Bengio, 2012, p. 282)

\[
\lambda^* = \arg \min_{\lambda \in \Lambda} E_{X,Y} \left( L \left( Y, \hat{f}_\lambda (X) \right) \right).
\]

As before, this is a stochastic optimization problem whose solution, due to lack of knowledge of the distributions of \( X \) and \( Y \) and finitely many data, can only be approximated by

\[
\lambda^* = \arg \min_{\lambda \in \Lambda} \bar{L}_{(D_{\text{train}},D_{\text{test}})}(\lambda) := \arg \min_{\lambda \in \Lambda} \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} L \left( y_{i_{\text{test}}} , \hat{f}_\lambda (x_{i_{\text{test}}}) \right).
\]

This optimization problem is also called the hyperparameter tuning problem. All tuning algorithms solve this problem function in a similar way: Multiple values \( \lambda \in \hat{\Lambda} \subset \Lambda \) are proposed, the belonging value of the loss function is estimated using a resampling procedure and finally, the \( \lambda \) with the best estimated value of \( L \) is used:

\[
\lambda^* = \arg \min_{\lambda \in \hat{\Lambda}} \frac{1}{K} \sum_{k=1}^{K} \bar{L}_{(D_{\text{train}}^k,D_{\text{test}}^k)}(\lambda).
\]
The optimization algorithms differ in how the $\tilde{\Lambda}$ is selected: Some algorithms propose only a single large batch (random search, grid search), some other algorithms perform complicated sequential procedures in order to propose a sequential list of $\lambda$ that are as good as possible (racing, evolutionary and model-based procedures).

However, in the end, all these algorithms come down to the same basic operation: The performance of two settings $\lambda_1, \lambda_2 \in \Lambda$ has to be compared, and the better setting has to be determined. Usually, this is done in a rather simple way: The entire resampling procedure is performed for both $\lambda_1$ and $\lambda_2$ and the setting with the better value of $\frac{1}{K} \sum_{k=1}^{K} \tilde{L}(D_{k}^{\text{train}}, D_{k}^{\text{test}}) (\lambda)$ is declared the winner. Hereby, the loss function is treated as if it is a deterministic function. This is achieved by using the same fixed partition $D_{k}^{\text{test}}$ and $D_{k}^{\text{train}}$, $k = 1, \ldots, K$ for each setting $\lambda$.

Although this approach works pretty well in practice, from a theoretical view it is not 100% correct. Both $\tilde{L}(D_{k}^{\text{train}}, D_{k}^{\text{test}}) (\lambda_1)$ and $\tilde{L}(D_{k}^{\text{train}}, D_{k}^{\text{test}}) (\lambda_2)$ are random variables, and we want to find out which of these two random variable is smaller (or larger, depending on the loss function). In statistics, this is usually reduced to the question, which of the two random variables has the smaller expected value. Since the mean value $\frac{1}{K} \sum_{k=1}^{K} \tilde{L}(D_{k}^{\text{train}}, D_{k}^{\text{test}}) (\lambda)$ is an estimator for the expected value, the usual approach is not entirely wrong. From a statistical point of view, however, a statistical test should be used with the null hypothesis

$$H_0 : \tilde{L}(D_{k}^{\text{train}}, D_{k}^{\text{test}}) (\lambda_1) < \tilde{L}(D_{k}^{\text{train}}, D_{k}^{\text{test}}) (\lambda_2).$$

A classical statistical test procedure has two major disadvantages: First, it handles the two hypothesis a-symmetrically: It is only possible to reject $H_0$ and decide for $H_1$. It is not possible to decide for $H_0$. This disadvantage could be neglected by performing a two-sided test to find a difference, and the sign of the difference to decide for one of the hypothesis. In this case, however, there would be the possible outcome that two settings are indistinguishable with respect to $\tilde{L}(D_{k}^{\text{train}}, D_{k}^{\text{test}})$, which is again not desireable. Actually, all of this is the reason why in usual tuning approaches, only the mean values are compared.

The other disadvantage is the fixed number of observations needed. However, in at least some situations it may be possible after seeing just two observations to come to a decision. For example, consider $x = \{1, 2, \ldots\}$ and $y = \{y = 10001, 10002, \ldots\}$. Here, after seeing the first two observations, a significant difference between the vectors is obvious, and it is not needed to consider the remaining ones. (Of course, as always in statistics, there is a chance of being wrong.) Especially in the context of hyperparameter tuning, when every evaluation of $L(D_{k}^{\text{train}}, D_{k}^{\text{test}})$ is time consuming, since an entire model fit has to be performed, it is desired to use as less evaluations as possible. On the other hand, for two settings with pretty similar values of $L(D_{k}^{\text{train}}, D_{k}^{\text{test}})$, it might be desirable to perform some extra evaluations, in order to be more sure about the made decision.

Both of these disadvantages are neglected by a sequential test: It can decide for both $H_0$ and $H_1$, it will always make a decision in the end, and it will do so in a minimal amount of evaluations of $\tilde{L}(D_{k}^{\text{train}}, D_{k}^{\text{test}})$. As always with statistical tests, it is both possible to construct a parametric and a non-parametric one, with the usual advantages and disadvantages. Here, we decided to construct a parametric test, hence, at first, we need to find a fitting parametric
distribution family in the next chapter, before we can define a sequential test procedure in the chapter after.

3. Distribution of the loss function

In this paper, five classification and five regression problems are considered, as well as four different machine learning methods each. Table 1 gives a more detailed overview of the classification data sets and Table 2 of the regression data sets respectively. Most of the data sets are taken from Vanschoren et al. (2013), insurance from Stednick (2020), diamond from the R-package ggplot2 (Wickham, 2016) and wage from R-package ISLR (James et al., 2017).

Table 1: Overview classification datasets

| dataset          | Observations | Features |
|------------------|--------------|----------|
|                  | class 0 | class 1 | discrete | numeric |
| german credit    | 300     | 700     | 13       | 7       |
| phoneme          | 3818    | 1586    | 0        | 5       |
| pima indians     | 500     | 268     | 0        | 8       |
| cancer           | 357     | 212     | 0        | 30      |
| ionosphere       | 126     | 225     | 0        | 34      |

Table 2: Overview regression datasets

| dataset     | observations | features | target variable |
|-------------|--------------|----------|-----------------|
|             | discrete | numeric | \( y_{\text{min}} \) | \( \bar{y} \) | \( y_{\text{max}} \) |
| boston      | 506     | 2 11    | 5                | 22.53          | 50               |
| insurance   | 1338    | 3 3     | 1122             | 13270          | 63770            |
| diamond     | 2700    | 3 6     | 326              | 4038.40        | 18788            |
| wage        | 3000    | 6 2     | 20.09            | 111.70         | 318.34           |
| concrete    | 1030    | 0 8     | 2.33             | 35.82          | 82.60            |

The learning methods used are classification and regression trees (rpart: Therneau and Atkinson (2019)), random forest (ranger: Wright and Ziegler (2017)), XGBoost (xgboost: Chen et al. (2020)), and logistic and linear regression with “elastic net” regularization (glmnet: Friedman et al. (2010)). Table 3 lists the considered hyperparameters and corresponding search spaces for each learning method. Here, for rpart, the parameter maxdepth denotes the maximum allowed depth of the tree and cp denotes the complexity parameter, which specifies by how much a split must contribute to the improvement of the fit so that the corresponding subtree is not pruned. For the Random Forest (ranger), 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
Table 3: Overview of the used machine learning methods and their tuning parameters

| method   | hyperparameter | search space          |
|----------|----------------|-----------------------|
| rpart    | cp             | [0, 0.5]              |
|          | maxdepth       | \{1, \ldots, 30\}    |
| ranger   | mtry           | \{1, \ldots, \#Features\} |
|          | replace         | \{TRUE, FALSE\}      |
|          | sample.fraction | [0.5, 1]             |
| xgboost  | nrounds        | \{2, \ldots, 100\}  |
|          | eta            | [0.01, 1]            |
|          | max_depth      | \{1, \ldots, 15\}   |
| glmnet   | lambda         | \(2^x\) with \(x \in [-15, 15]\) |
|          | alpha          | [0, 1]               |

of iterations, \(\text{eta}\) the learning rate, and \(\text{max\_depth}\) the maximum depth of the trees used. For \text{glmnet}, \(\text{alpha}\) regulates the mixture parameter of the Elastic Net regularization and \(\text{lambda}\) the degree of penalization. For the remaining hyperparameters of the individual methods, which are not subject of the optimization, the respective default settings are used.

All calculations and simulations in this work are performed in \texttt{R} (version 3.6.3; \texttt{R} Core Team (2020)). For any aspects of the machine learning process, i.e. model fitting, validation, tuning, etc., the \texttt{R} package \texttt{mlr} (Bischl et al., 2016) is used, which provides a unified interface for running machine learning experiments in \texttt{R}.

Simulation of the error measures is performed as follows: For each combination of data set and machine learning method, ten random parameter settings are generated and validated using 1000 resampling iterations, where Bootstraping is used as resampling strategy. The same data set partitions \((D_{\text{train}}^k, D_{\text{test}}^k)\) are used for each data set and learning procedure. The error measures considered are Mean Misclassification Error \((\text{MMCE})\) in the classification case and MSE in the regression case, hence, 1000 observations of the value of the loss function are calculated for each setting.

In order to investigate the distribution family of the loss function, multiple usual distributions are fitted to the observations via Maximum-Likelihood estimation. Afterwards the \textit{Cramér-von-Mises} criterion (CvM) is calculated in order to determine the goodness of the fit (see, e.g., Stephens, 1974). This is done using the \texttt{R} package \texttt{fitdistrplus} (Delignette-Muller and Dutang, 2015). The CvM-criterion is defined as the test statistics of the \textit{Cramér-von-Mises-test}. It sorts the observation values in ascending order \(x_1 < x_2 < \ldots < x_n\), evaluates them with distribution function \(F\) and compares the values with values that are to be expected if the observations would actually follow that distribution:

\[
T = \frac{1}{12n} + \sum_{i=1}^{n} \left( \frac{2i - 1}{2n} - F(x_i) \right)^2.
\]

The smaller \(T\) is, the better the data is fitted by the distribution. Figure 1 shows the distribution of the CvM-criterion in some artificial situations: a normal distribution is
fitted to some data that is actually normal, uniform or exponential distributed. For the normal distribution (i.e. the assumed distribution corresponds to the true distribution), the CvM is always smaller than 0.5, for the uniform distribution (which is kind of similar to the normal distribution) the values are between 1 and 2, for the exponential distribution (a quite different distribution), the values are between 6 and 8. Hence, for a good fitting distribution we would expect values smaller than 1, values up to 2 may be okay as well.

![Figure 1: Distribution of the CvM criterion when fitting a normal distribution to data points, that are actually \{normal, uniform, exponential\} distributed.](image)

It cannot be expected to find a single distribution family that is always the best, contrary, it is to be expected that the distribution differs quite a lot for different settings. However, finding a distribution family that has a good fit in many situations seems sufficient in order to build a sequential test around it. Distribution families considered are the normal distribution, the gamma distribution, the beta distribution (for classification only), and the Weibull distribution, as well as variations of these in the form of the log normal distribution, the log gamma distribution, the inverse gamma distribution, and the inverse Weibull distribution.

A problem that arises when using many of these distributions is that, as in the case of the logarithmic and inverse distribution families, the respective carrier 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 (sub)data set and learner. An obvious solution is to shift the data with 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 is calculated for different values 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 the one hand MSE values of 0 only found 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 2 shows the result for the classification, Figure 3 for the regression case. It turns out that especially the distributions of the Weibull type seem to be comparatively unsuitable for modeling both MMCE and MSE distributions. For all other distribution families, the CvM criterion takes values that are mostly smaller than 1, and in more than 75% even smaller than 0.5. This fit is by far not as good as fitting a normal distribution to actually normal distributed data, but it is far better than fitting a normal distribution to actually uniform distributed data (compare Figure 1). The one exception, not as bad as the Weibull distribution, but notably worse than the other distribution families is the normal distribution in the regression case. For the other distribution families, a relatively homogeneous picture emerges. All have a quite good (not perfect, but quite good) fit to the data and achieve comparable results. This result is confirmed when looking at other criteria than the CvM criterion (respective result plots are omitted here). Hence, from the view of the distribution fit, the choice of the distribution family is not crucial, as long as one does choose along the well performing distribution families.

Figure 2: Distribution of the CvM criterion in the classification case for different distribution classes with individual additive shifts $c$. 
4. Designing a Sequential Random Search

The main goal of our modified random search is reducing the evaluation steps needed while maintaining 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). An essential class of sequential tests are Sequential Probability Ratio Tests by Wald (1945). In its classic form, the test for $H_0 : \theta = \theta_0$ vs. $\theta = \theta_1$ with $\theta_0 < \theta_1$ is described by the following procedure:

Having sampled observation $u_n$ in step $n = 1, 2, \ldots$, calculate the test statistic

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

where $f_n(\cdot ; \theta_i)$ 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 $u_{n+1}$,
Sequential Random Search

where \( b, a \in \mathbb{R} \), \( b < a \) define the so-called **continuity region** \([b, a]\) of the test and are chosen such that the type I and type II errors are controlled for pre-specified \( \alpha \) and \( \beta \) values. Our first simulation study indicated that mostly flexible multi-parameter distribution families are suitable for modelling the loss. We decide to use a log-normal distribution, since in comparison to the beta and gamma distribution families, the log-normal distribution offers the advantage of performing a location test based on only one parameter since the median of a log-normally distributed variable only depends on \( \mu \). A further advantage is the possibility to use test methods based on normality assumptions after a logarithmic transformation of the original data.

However, due to the presence of a nuisance parameter \( (\sigma^2) \), regular SPRTs can not be applied. Thus, we use a **Sequential Likelihood Ratio Test** (SLRT) proposed by Ghosh (1970) for the sequential Behrens-Fisher-problem of two normally distributed i.i.d. random variables \( \{U_i\} \sim \mathcal{N}(\mu_U, \sigma_U^2) \) and \( \{W_i\} \sim \mathcal{N}(\mu_W, \sigma_W^2) \) where all parameters are unknown and one wants to test

\[
H_0 : \gamma := \mu_U - \mu_W = \gamma_0 \text{ vs. } H_1 : \gamma = \gamma_1 \text{ with } \gamma_0, \gamma_1 \in \mathbb{R}, \gamma_0 < \gamma_1
\]

with type I error rate \( \alpha \) and type II error rate \( \beta \). For this test, Ghosh (1970, S. 335) proposes the following continuity region:

\[
-\frac{s_u^2(n) + s_w^2(n)}{\gamma_1 - \gamma_0} \ln \frac{1 - \alpha}{\beta} < n \left( \bar{u}(n) - \bar{w}(n) - \frac{\gamma_0 + \gamma_1}{2} \right) < \frac{s_u^2(n) + s_w^2(n)}{\gamma_1 - \gamma_0} \ln \frac{1 - \alpha}{\beta},
\]

where \( \bar{u}(n) \) and \( \bar{w}(n) \) denote the mean and \( s_u^2(n) \) and \( s_w^2(n) \) the empirical variance of the respective sample until the \( n \)-th observation. Since it holds that

\[
H_0 : \mu_U - \mu_W = \gamma_0 \Leftrightarrow \frac{\exp(\mu_U)}{\exp(\mu_W)} = \exp(\gamma_0) \Leftrightarrow \frac{\text{med}(\bar{U})}{\text{med}(\bar{W})} = \exp(\gamma_0)
\]

for \( \ln(\bar{U}) \sim \mathcal{N}(\mu_U, \sigma_U^2) \) and \( \ln(\bar{W}) \sim \mathcal{N}(\mu_W, \sigma_W^2) \), the test above is practically a test of the relative difference between the medians of the two loss distributions (assuming log-normality). However, this is only true in the regression case, since an additive shift of the data is required for classification tasks. In theory, one could also use a two-stage procedure instead of the SLRT, but this 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 therefore opt for the SLRT.

The general idea now is to combine the advantages of 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. However, instead of using a fixed number of, let us say, ten resampling iteration, 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 those two settings is found, and the better setting is kept as the new best setting. Hereby, always the same amount of evaluations of the loss function are used. For the new setting, a new observation of the loss function has to be calculated every time using a resampling, for the current best setting prior evaluations are reused if available.
To prevent infinite run times, a maximum number of evaluations per setting should be defined and the test is forced to make a decision between the two settings if this maximum amount is reached, in case of doubt, the setting with the lower estimated loss function value is used. Pseudo-Code describing this Sequential Random Search 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. The function `generateConfig()` evaluates the performance of a single hyperparameter setting, using an arbitrary resampling procedure, for example a single iteration of Bootstrapping As the termination criterion, usually a maximum number of iterations or computation time should be used.

**Algorithm 1** Sequential Random Search (SQRS)

1: procedure SQRS($n_{max}, \gamma_0, \gamma_1, \alpha, \beta, c$)
2:     opt.config = generateConfig()
3: while termination condition not fulfilled do
4:     cand.config = generateConfig()
5:     for $n = 1$ to $n_{max}$ do
6:         $p_{opt,n} = \text{evaluateConfigs}(\text{opt.config}) + c$
7:         $p_{cand,n} = \text{evaluateConfigs}(\text{cand.config}) + c$
8:         if $n \geq 2$ then
9:             upper = $\sigma^2(p_{opt,1:n}) + \sigma^2(p_{cand,1:n}) \ln \frac{1-\beta}{\alpha}$
10:                lower = $-\text{upper}$
11:                statistic = $n \cdot (\bar{p}_{opt,1:n} - \bar{p}_{cand,1:n}) - \gamma_0 - \gamma_1$
12:         if statistic > upper then
13:             opt.config = cand.config
14:             break
15:     if statistic < lower then
16:         break
17:     if $n = n_{max}$ then
18:         opt.config = arg min$_{i \in \{\text{opt, cand}\}}$ $\bar{p}_{i,1:n}$
19: return opt.config

For the implementation of `generateConfig()` two generally different approaches can be considered: One possibility is to create a resampling instance with $n_{max}$ different data set partition samples ($D_{i_{test}}^{\text{test}}, D_{i_{train}}^{\text{train}}$), $i = 1, ..., n_{max}$ at the beginning of the optimization, on which all configurations are evaluated from then on, i.e. in the $n$-th evaluation step all configurations are validated on the same bootstrap sample. This allows immediate comparability of configurations (on the single sample) and represents the usual approach to regular random search. However, this approach does not do justice to the stochastic nature of the optimization problem and therefore the second approach is to always draw a new bootstrap sample for each evaluation of a configuration, i.e., two configurations are validated on different bootstrap samples in the $n$-th step. In both cases, performance values already obtained for `opt.config()` should be reused to save additional evaluations.
5. Simulation Study

We will now compare the SQRS with a regular random search in an 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 and 3, we generate 50 random configurations of hyperparameters for each combination of learner and dataset and validate their performance using a bootstrap resampling with ten iterations. To make the resampling results comparable, we use the same resampling instance for both algorithms. We perform 100 replications where the tuning process is carried out in a sequential manner, i.e. without parallelizing computations.

For the random search, we now simply look for the parameter setting with the minimal MMCE and MSE respectively and use this setting as the random search result. The sequential random search will now operate on exactly the same data: `generateConfig()` will not generate new random hyperparameter setting, but it will return the same 50 random configuration as used by the random search, but in a random order. When comparing the current best and the new configuration, SQRS uses exactly the same dataset partitions as random search, and, hence, the values of the loss function calculated during the random search are reused. The only real difference to the random search is, that the SQRS will not always use all ten iterations of the resampling, but it is allowed to discard a candidate setting as soon as the sequential tests shows that it is worse than the current best setting.

For the parameters of the SQRS, we use an additive shift of $c = 0.5$ in classification and $c = 0$ in regression tasks. As specified by the resampling instance, the maximum number of evaluations is $n_{\text{max}} = 10$. We vary the choice for parameters $\alpha$ and $\beta$ between 0.01 and 0.05, and for $|\gamma_0| = \gamma_1$ between 0.01 and 0.02.

We aim to answer two questions: First, can the SQRS algorithm achieve a comparable accuracy, i.e., does it find the same optimal hyperparameter setting? And second, how much less function evaluation does it need to do so?

Table 4: Parameter settings for the Sequential Random Search

| setting | $\gamma_0$ | $\gamma_1$ | $\alpha, \beta$ |
|---------|------------|------------|-----------------|
| A       | -0.02      | 0.02       | 0.05            |
| B       | -0.02      | 0.02       | 0.01            |
| C       | -0.01      | 0.01       | 0.05            |
| D       | -0.01      | 0.01       | 0.01            |

First, we investigate how often the sequential random search leads to the same result as the random search (RS). Table 5 presents the proportion of identical solutions for the different considered values of $\gamma_0$, $\gamma_1$, $\alpha$ and $\beta$. In the classification case, the SQRS chooses the same parameter configuration approximately between 79% and 90% of the time. In the regression case, it does so 87% to 90% of the time. The more conservative the test becomes in terms of smaller (absolute) values for each test parameter, the more often the SQRS makes the same decision. It seems that the procedure tends to react more sensitively to changes in the test parameters chosen here in the classification case compared to the regression case.
Table 5: Proportion of identical solutions for SQRS and random search as a function of $\gamma_0$, $\gamma_1$, $\alpha$ and $\beta$ for classification and regression

| setting | classification | regression |
|---------|----------------|------------|
| A       | 0.789          | 0.871      |
| B       | 0.842          | 0.889      |
| C       | 0.864          | 0.897      |
| D       | 0.895          | 0.905      |

If we consider the cases in which the SQRS does not find the optimal solutions, the solutions found by the SQRS do not perform considerably worse. In Figure 4 the ratios of the performance of random search and the performance of SQRS are shown. A ratio of 1 corresponds to identical performances, ratios larger than 1 indicate a better performance of random search. Per definition of our experiment, SQRS cannot find a better setting than random search, hence, ratios smaller than 1 are not possible here. For a better clarity, we omit all datapoints with a ratio of 1 in the plot. It can be seen that both in classification and regression in most experiments the solution found by SQRS is only slightly worse than the solution of random search.

Figure 4: performance ratio of SQRS/RS in the case of different solutions (i.e., performance ration $\text{SQRS/RS} \neq 1$) for the four different settings.

Figure 5 (classification) and Figure 6 (regression) show the number of evaluations SQRS needed to reach its top performance compared to the total number of random search iterations. For classification in setting A, the quotient of the required evaluations for SQRS and RS is 0.476 at the median, 0.558 in Setting B, 0.608 in Setting C, and 0.712 in Setting D. In the most liberal case (A), sequential random search required fewer evaluations than random search 99.9% of the time and in 55.6% of the cases it requires at most half as many evaluations. Remember, in this case, 80% of the runs reached the same performance as with random search was reached, and and most other runs were only 10% worse at most. In the most conservative setting (D) the SQRS still requires fewer evaluations than the
maximum possible in 90.8% of the cases, and in 24.5% of the cases, the maximum number of evaluations possible is half of the possible evaluations are performed.

In the regression settings, the results are even more promising. The quotient of the required evaluations for the sequential and the ordinary median is 0.324 for Setting A, 0.360 for Setting B, 0.388 for Setting C, and 0.456 for Setting D. In all scenarios, the SQRS never exhausted the maximum number of the maximum evaluation number. In the liberal setting (E) in 71.1% and in the conservative setting (H) in 55.3% of the cases, at most half as many evaluations were than for the random search. Remember that here in all setting in around 90% of the runs the same performance as for random search was reached.

![Figure 5](image_url)

Figure 5: Percentage of SQRS number of resampling iterations compared to random search, depending on the four parameter settings in the classification setting.

6. Discussion

In this work, we analysed the feasibility of employing sequential statistical tests during the hyperparameter tuning process to save computational effort and time. We aimed at answering three main research questions. The first research question asked which distribution family could best be used to approximate the distribution of resampling errors.
We answered this question with a small simulation study: We picked multiple different datasets as well as different machine learning methods, both from the fields of classification and regression. We empirically calculated the distribution of the resampling errors in these situations and fitted multiple different distribution families. 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 is purely empirical and requires a more thorough analysis but it provided us with a first indication to continue our work upon.

![Graphs showing evaluation ratio versus density for different settings](image)

Figure 6: Percentage of SQRS number of resampling iterations compared to random search, depending on the four parameter settings in the regression setting.

Our second research question pertained to the construction of a tuning algorithm using a sequential test. The results to the first research question indicated that multiple distribution families seemed fitting. For practical reasons, we decided for the log-normal family. Using a sequential test by Ghosh (1970), we implemented a sequential variant of the random search (abbreviated SQRS). The main difference between the SQRS and regular random search was the early stopping possibility of the former. After each evaluation step, the resampling error samples of two configurations were compared using the sequential test procedure. When a terminating decision could not be made before reaching the maximum number
of permitted evaluation steps, the procedure selected the parameter setting leading to the lower resampling error.

Having implemented the SQRS, our focus then turned to our third and final research question of how a hyperparameter tuning approach using a sequential statistical test would perform compared to a normal random search. To study this, we performed a simple experiment in which we used a regular random search for some hyperparameter optimization problems. Afterwards, the SQRS had to tackle the same problems under identical conditions. We found that by using the sequential testing procedure instead of doing a full resampling for each comparison, the SQRS was able to speed up the procedure quite a lot without leading to a considerable performance loss.

This indicates to us that the inclusion of a sequential statistical test procedure within a hyperparameter tuning algorithm is a promising approach: The reached best performance of our SQRS was comparable to a normal random search in nearly all cases, but the number of resampling iterations to reach this performance was halved. However, our comparison was targeted at maximum comparability and both algorithms were forced to operate under the same laboratory conditions using an identical set of candidate settings. What is missing are real application experiments in which the actual run time advantage of SQRS is analyzed. Since such experiments are quite time consuming, they have been omitted in this first iteration.

Moreover, even if random search is still used for many hyperparameter optimization problems, at least some more advanced and better tuning algorithms do exist. All of them rely on their ability to compare two hyperparameter setting using a resampling, so our sequential testing procedure could be incorporated into most of those algorithms. There are also other algorithms that try to reduce the number of resampling iterations needed by eliminating bad settings early, FRacing has to be mentioned in the first place here. These algorithms follow a different (statistically less sound) way of eliminating bad settings, and it still has to be analyzed whether the additional theoretical work of SQRS is worth it.

References

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

B. Bischl, O. Mersmann, H. Trautmann, and C. Weihs. Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary Computation*, 20(2):249–275, 2012.

Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5, 2016.

Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, Mu Li, Junyuan Xie, Min Lin, Yifeng Geng, and Yutian Li. xgboost: Extreme Gradient Boosting, 2020. URL https://CRAN.R-project.org/package=xgboost. R package version 1.0.0.2.
Marie Laure Delignette-Muller and Christophe Dutang. fitdistrplus: An R package for fitting distributions. *Journal of Statistical Software*, 64(4):1–34, 2015.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

Bhaskar Kumar Ghosh. *Sequential Tests of Statistical Hypotheses*. Addison-Wesley Publishing Company, 1970.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2 edition, 2009.

Gareth James, Daniela Witten, Trevor Hastie, and Rob Tibshirani. *ISLR: Data for an Introduction to Statistical Learning with Applications in R*, 2017. URL https://CRAN.R-project.org/package=ISLR. R package version 1.2.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL https://www.R-project.org/.

Zach Stednick. Machine learning with r datasets. GitHub Repository, 2020. URL https://github.com/stedy/Machine-Learning-with-R-datasets.

Michael A. Stephens. Edf statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, 1974.

Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2019. URL https://CRAN.R-project.org/package=rpart. R package version 4.1-15.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15:49–60, 2013.

A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945.

Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Marvin N. Wright and Andreas Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77:1–17, 2017.