Chapter 2
Tuning: Methodology

Thomas Bartz-Beielstein, Martin Zaefferer, and Olaf Mersmann

Abstract This chapter lays the groundwork and presents an introduction to the process of tuning Machine Learning (ML) and Deep Learning (DL) hyperparameters and the respective methodology used in this book. The key elements such as the hyperparameter tuning process and measures of tunability and performance are defined. Practical considerations are presented and all the ingredients needed for successful hyperparameter tuning are explained. A special focus lies on how to prepare the data. This might be the most thorough overview presented yet.

2.1 Introduction to Hyperparameter Tuning

This book deals with the tuning of the hyperparameters of methods from the field ML and DL, focusing on supervised learning (both regression and classification). In the following, the scope of this work is explained, including terminology of important terms (Tables 2.1 and 2.2).

The *data points* x come from an input data space $\mathcal{X}$ ($x \in \mathcal{X}$) and can have different scale levels (nominal: no order; ordinal: order, no distance; cardinal: order, distances). Nominal and ordinal data are mostly discrete, and cardinal data are continuous. Usually, the data points are k-dimensional vectors. The vector elements are also called *features* or independent variables. The number of data points in a data set is n.

T. Bartz-Beielstein (✉) · O. Mersmann
Institute for Data Science, Engineering and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

O. Mersmann
e-mail: olaf.mersmann@th-koeln.de

M. Zaefferer
Bartz & Bartz GmbH and with Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: zaefferer@dhbw-ravensburg.de

Duale Hochschule Baden-Württemberg Ravensburg, Ravensburg, Germany

E. Bartz et al. (eds.), *Hyperparameter Tuning for Machine and Deep Learning with R*,
https://doi.org/10.1007/978-981-19-5170-1_2

Table 2.1 Symbols used in this book

Symbol	Name and description
$\mathcal{A}$	Algorithm, model, methods (see Definition 2.20)
$A_{\lambda(t)}$	Model with hyperparameter configuration λ at time step t
$A_{\lambda(*)}$	Model with best hyperparameter configuration λ
c	Number of classes/categories
$c_{1-\alpha}$	Threshold value or the cut-off point (used in hypothesis testing)
$d(\cdot)$	Test statistic
d_j	Difference between the j-th samples
$E(\dot{)}$	Expectation
Err_{test}	Test error, generalization error
f	Function that describes relationship between input and output
$f_{\text{acc}}^{(\text{train})}$	Training accuracy
$f_{\text{acc}}^{(\text{test})}$	Test accuracy
$f_{\text{acc}}^{(\text{val})}$	Validation accuracy
H_0	Null hypothesis
H_1	Alternative hypothesis
$\mathbf{I}$	Indicator function
k	(Problem) dimension
$k(x, x')$	Kernel function
$\mathcal{L}$	Loss function
n	Sample size or number of observations
n_{init}	Initial design size
N	Total number of samples
N_{Feats}	Number of features
$\mathbb{N}$	Natural numbers
$\mathcal{O}$	Optimizer
p	Norm, e.g., used in Eq. (2.1) or number of model coefficients
$p(\mathcal{A})$	Performance of algorithm $\mathcal{A}$
$P_H(x)$	Probability of x under the hypothesis H
$\mathbb{R}$	Set of real numbers
S_d	Sample standard deviation of differences
S_{nr}	Severity (non-rejection)
S_r	Severity (rejection)
$\mathcal{S}$	Surrogate (model)
T_i	i-th tree
$\lvert T \rvert$	Number of splits for a tree
$\mathcal{T}$	Tuner
$u_{1-\alpha}$	Upper $1-\alpha$ quantile of the normal distribution
$\mathcal{X}$	Input data space or complete data set

(continued)

Table 2.1 (continued)

Symbol	Name and description
$((\mathcal{X}, \mathcal{Y})_{\text{CID}}$	The full Census Income Data (CID) data set ($n = 299\,285$ observations)
$(X, Y)^{(\text{test})}$	Test data set
$(X, Y)^{(\text{train})}$	Training data set
$(X, Y)^{(\text{train} \cup \text{val})}$	Training and validation data set
$(X, Y)^{(\text{valtrain})}$	Validation set, subset of $(X, Y)^{(\text{train})}$, internally used by the model $\mathcal{A}$ in the inner optimization loop
$(X, Y)^{(\text{val})}$	Validation data set
t	Iteration counter. I.g., the t-th SPOT surrogate will be denoted as $\mathcal{S}(t)$
T	Transpose of a matrix or vector
x	Data point, feature, predictor, covariates, explanatory variable
$\overline{x}$	Observed difference
x^*	New/unknown data point
X	Data, usually partitioned into training, validation, and test data
y	Actually observed value, outcome, response variable, label
$y_{\text{val}}^{(*)}$	Best function value on the validation data $(X, Y)^{(\text{val})}$
$y_{\text{val}}^{(\text{OCBA}*)}$	Best function value on the validation data $(X, Y)^{(\text{val})}$, evaluated with OCBA
$\hat{y}_i$	Predicted value for the i-th observed value y_i
y^*	Output value for a new/unknown data point
$\bar{y}$	Mean difference between two observations, usually for paired samples
$\mathcal{Y}$	Output data space (dependent variables)

In addition, we consider output data (dependent variables) $y \in \mathcal{Y}$. These can also have different scale levels (nominal, ordinal, cardinal). Output data are usually scalar. Variable identifiers like x and y represent scalar or vector quantities, the meaning can be deduced from the context. In many practical applications, e.g., in the case studies in this book, a subset, (X, Y), of the full data set $(\mathcal{X}, \mathcal{Y})$ is used.

Definition 2.1 (*Supervised learning*) *Supervised learning* is a branch of ML where models are trained on *labeled* data sets to make predictions.

For each observation, both the input $x_i \in \mathcal{X}$ and the outcome $y_i \in \mathcal{Y}$ $(i = 1, 2, \ldots, n)$ is known during training. A supervised learning algorithm $\mathcal{A}$ learns the relationship between the inputs and output. That is, given a training data set (x_i, y_i), $i = 1, \ldots, n$, it returns a *model* $f : \mathcal{X} \mapsto \mathcal{Y}$ with which we can predict the expected value of y^* given x^*:

$$\hat{y^*} = f(x^*)$$

Table 2.2 Greek Symbols used in this book

Symbol	Name	Comment, Example
α	Significance level (in hypothesis testing)	Probability of a type I error, given that the null hypothesis is true
β	Regression coefficient; the probability of a type II error, given that the alternative hypothesis is true	
Δ	Relevant difference	
μ	Mean of a random variable	
λ	Hyperparameter configuration	Also: nugget in Kriging
λ_i	i-th hyperparameter configuration	Used in SMBO
λ_0	Default hyperparameter configuration	Used in SMBO
$\lambda^\star$	Best hyperparameter configuration	Best configuration in theory
$\hat{\lambda}$	Best hyperparameter configuration obtained by evaluating a finite set of samples	Best configuration "in practice"
Λ	Hyperparameter space	
π	Problem instance	
Φ	Cumulative distribution function of the standard normal distribution	
Ψ	Hyperparameter response space	
ψ_i	Hyperparameter response surface function evaluated for the i-th hyperparameter configuration λ_i	
$\psi^{(\text{test})}$	Hyperparameter response surface function (on test data)	As defined in Eq. (2.11)
$\psi^{(\text{train})}$	Hyperparameter response surface function (on train data)	
$\psi^{(\text{val})}$	Hyperparameter response surface function (on validation data)	As defined in Eq. (2.9)
σ^2	Variance	
τ	Possible values (in hypothesis testing)	
θ	Set of parameters	Also: Kriging hyperparameters

Definition 2.2 (*Regression*) If a supervised learning problem has an infinite number of labels or outcomes, i.e., $\mathcal{Y} \subseteq \mathbb{R}$, then it is called a *regression problem* and the task of finding a model that captures the relationship between the input space and output space is called *regression*.

Definition 2.3 (*Classification*) If a supervised learning problem has a finite number of labels or outcomes, i.e., $\mathcal{Y} = a_1, a_2, \ldots, a_c$ with $c \in \mathbb{N}$, it is called a *classification*

problem and the task of finding a model that captures the relationship between the input space and output space is called *classification*.

Example: Regression and Classification

A typical question in regression is "how does relative humidity depend on temperature?", whereas a typical question in classification is "how does the default on a loan (yes, no) depend on the income of the borrower?"

We study ML and DL *algorithms*, also referred to as *models* or *methods*.

Note

The terms "algorithm", "model", and "method" will be used interchangeably in this book. Their specific meanings can be derived from the context.

The learning algorithm itself has parameters called *hyperparameters*. Hyperparameters are distinct from model parameters.

Definition 2.4 (*Hyperparameter*) *Hyperparameters* are settings or configurations of the methods (models), which are freely selectable within a certain range and influence model performance (quality). One specific set of hyperparameters is denoted as λ, where Λ is the hyperparameter space.

Definition 2.5 (*Model parameters*) Model parameters are chosen during the learning process by the model itself.

Example: Hyperparameters and Model Parameters

The weights of the connections in an Neural Network (NN) are an example of model parameters, whereas the number of units or layers of a NN is a hyperparameter.

2.2 Performance Measures for Hyperparameter Tuning

2.2.1 Metrics

Metrics are used to measure the distance between points and then define the similarity between them.

Definition 2.6 (*Metric*) Let X denote a set. A *metric* is a function

$$\mathrm{d} : X \times X \to \mathbb{R}_0^+$$

that returns the distance between any two values from the set X. Metrics are symmetric, positive definite, and fulfill the triangle equality.

The *Minkowski* distance, which will be used in this book, is defined as follows.

Definition 2.7 (*Minkowski Distance*)

$$\mathrm{d}_{\mathtt{p}}(x, x') = \left\| x - x' \right\|_{\mathtt{p}} = \sqrt[\mathtt{p}]{\sum_{i=1}^{n} \left| x_i - x_i' \right|^{\mathtt{p}}}. \tag{2.1}$$

Some well-known metrics are special cases of the Minkowski distance:

- for $\mathtt{p} = 1$ we get the *Manhattan distance*,
- for $\mathtt{p} = 2$ we get the *Euclidean distance*,
- and for $\mathtt{p} = \infty$ we get the *Chebyshev distance*.

2.2.2 Performance Measures

Several measures are used to evaluate the performance of ML and DL methods, because performance can be expressed in many different ways, e.g., as a measure of the fit of the model to the observed data values. The performance measure is evaluated after a single ML or DL training step and returns a value to assess the quality of the model or the prediction.

Tips: Measures in R

Basic metrics are implemented in the package `SPOTMisc`. The `mlr` tutorial "Implemented Performance Measures"[1] presents a comprehensible overview. The R package `Metrics` is also a valuable tool.

Before we can define performance measures for classification or regression problems, we need some tools from which we will build these up.

Definition 2.8 (*Loss function*) A function

$$\mathcal{L}\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_0^+$$

[1] Available on https://mlr.mlr-org.com/articles/tutorial/measures.html.

is called a *loss function* or short a *loss*. $\mathcal{L}(y, \hat{y})$ is a measure of how "bad" it is to predict $\hat{y}$ given that the true label is y.

Concrete examples of loss functions are

Definition 2.9 (*Quadratic loss*) The loss function

$$\mathcal{L}_2(y, \hat{y}) = (y - \hat{y})^2$$

is called the *quadratic loss*.

Definition 2.10 (*Absolute value or L1 loss function*) The $L1$ or *absolute value loss* is defined as

$$\mathcal{L}_1(y, \hat{y}) = \left| y - \hat{y} \right|.$$

Definition 2.11 (*0-1 loss*) The loss function

$$\mathcal{L}_{01}(y, \hat{y}) = \mathbf{I}(y \neq \hat{y}) = \begin{cases} 0 & \text{when } y = \hat{y} \\ 1 & \text{else} \end{cases}$$

is called the *0-1 loss*.

Definition 2.12 (*Cross-entropy loss*) The *cross-entropy* loss function is defined as

$$\mathcal{L}_{CE}(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) = \begin{cases} \log(\hat{y}) & \text{when } y = 1 \\ \log(1 - \hat{y}) & \text{when } y = 0 \end{cases}.$$

Note that it is only defined for binary outcomes y, while the predicted label $\hat{y}$ can be any value and is usually assumed to be a class probability.

With these loss functions, we can define *performance measures*. A performance measure evaluates the loss on a data set and returns an aggregate loss. Depending on the ML task, e.g., classification or regression, different categories of measures are useful. In the following, $\hat{y}_i = f(x_i)$ is the predicted value of the corresponding learning model $\mathcal{A}$ for the i-th observation, x_i is the i-th data point, and y_i is the actually observed value.

2.2.3 Measures for Classification

Definition 2.13 (*Mean Mis-Classification Error*) Mean Mis-Classification Error (MMCE) is defined as

$$\text{MMCE} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{01}(y_i, \hat{y}_i) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{I}(y_i \neq \hat{y}_i). \tag{2.2}$$

MMCE is used to evaluate the performance of the ML methods in the case studies (Chaps. 8–10) and in the global study (Chap. 12).

Definition 2.14 (*Binary cross-entropy loss*) Binary Cross Entropy (BCE) loss or *log-loss* is defined as

$$\text{BCE} = -\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_{CE}(y_i, \hat{y}_i) \tag{2.3}$$

The DL methods in Chap. 10 are tuned based on the $\psi^{(\text{val})}$ (`validationLoss`), which computes the BCE loss.

Definition 2.15 (*Accuracy*) The *accuracy* is defined as

$$\text{ACC} = \frac{1}{n}\sum_{i=1}^{n} 1 - \mathcal{L}_{01}(y_i, \hat{y}_i) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(y_i = \hat{y}_i).$$

Note that, in contrast to the previous measures, higher accuracies are better than lower ones.

2.2.4 Measures for Regression

Typical regression measures are as follows.

Definition 2.16 (*Mean Squared Error*) The Mean Squared Error (MSE) is defined as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_2(y_i, \hat{y}_i).$$

When used for classification, the MSE is sometimes called the *Brier score*.

Definition 2.17 (*Root Mean Squared Error*) The Root Mean Squared Error (RMSE) is defined as the square root of the MSE:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_2(y_i, \hat{y}_i)}. \tag{2.4}$$

RMSE is used to evaluate the performance of the ML methods in the global study (Chap. 12).

Definition 2.18 (*Mean Absolute Error*) The Mean Absolute Error (MAE) is defined as

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_1(y_i, \hat{y}_i).$$

2.3 Hyperparameter Tuning

After specifying the data, methods for supervised learning with their hyperparameters and performance measures, the hyperparameter tuning problem can be defined.

Definition 2.19 (*Hyperparameter tuning*) The determination of the best possible hyperparameters is called *tuning* (Hyperparameter Tuning (HPT)). HPT develops tools to explore the space of possible hyperparameter configurations *systematically*, in a structured way, i.e., HPT is an optimization problem.

The terms HPT and Hyperparameter Optimization (HPO) are often used synonymously. In the context of the analyses presented in this book, these terms have different meanings:

HPO develops and applies methods to determine the best hyperparameters in an effective and efficient manner.

HPT develops and applies methods that try to analyze the effects and interactions of hyperparameters to enable *learning and understanding*.

HPT can be seen as an extension of HPO, because it provides additional tools and keeps experimenters and applicants in the loop. The relationship between HPT and HPO can be formulated as follows:

$$\text{HPO} \subset \text{HPT}.$$

It simplifies the notation in the book: whenever HPT is mentioned, HPO is covered as well.

In a data-rich situation, the best HPT approach is to randomly partition the data set (X, Y) into three parts as illustrated in Fig. 2.1.

The following definitions are based on Hastie (2009) and Bergstra and Bengio (2012). The objective of a learning algorithm $\mathcal{A}$ is to find a function f that minimizes some expected loss $\mathcal{L}(y, f(x))$ over samples $(x, y) \in (X, Y)$.

Definition 2.20 (*Learning algorithm*) A *learning algorithm* $\mathcal{A}$ is a functional that maps a data set $(X, Y)^{(\text{train})}$ (a finite set of samples) to a function $f : \mathcal{X} \to \mathcal{Y}$, i.e., $\mathcal{A}((X, Y)^{(\text{train})}) \mapsto f$.

Fig. 2.1 Dataset split into three parts: (i) a training set $(X, Y)^{(\text{train})}$ used to fit the models, (ii) a validation set $(X, Y)^{(\text{val})}$ to estimate prediction error for model selection, and (iii) a test set $(X, Y)^{(\text{test})}$ used for assessment of the generalization error

A learning algorithm $\mathcal{A}$ can estimate f through the optimization of a training criterion with respect to a set of parameters, $\lambda \in \Lambda$. Because the true relationship f is unknown in real-world settings, $\mathcal{A}$ will return an estimation of f, which will be denoted as $\hat{f}$.

The learning algorithm itself often has hyperparameters $\lambda \in \Lambda$, and the actual learning algorithm is the one obtained after choosing λ, which can be denoted $\mathcal{A}_\lambda$.

The computation performed by $\mathcal{A}$ itself often involves an "inner optimization" problem, e.g., optimizing the weights of a NN. The hyperparameter optimization can be considered as an "outer-loop" optimization problem. It can be formulated as follows:

$$\lambda^{(*)} = \arg\min_{\lambda \in \Lambda} E_{(x,y)\in(\mathcal{X},\mathcal{Y})} \left[\mathcal{L}\left(y, \mathcal{A}_\lambda((X, Y)^{(\text{train})})\right)\right]. \tag{2.5}$$

In practice, the underlying space $(\mathcal{X}, \mathcal{Y})$ is too large or the true relation between $\mathcal{X}$ and $\mathcal{Y}$ is unknown. Therefore, the validation set is used and Eq. (2.5) is replaced by

$$\lambda^{(*)} \approx \arg\min_{\lambda \in \Lambda} \frac{1}{\left|(X, Y)^{(\text{val})}\right|} \sum_{x \in (X,Y)^{(\text{val})}} \mathcal{L}\left(y, \mathcal{A}_\lambda((X, Y)^{(\text{train})})(x)\right) \tag{2.6}$$

Practitioners are interested in a way to choose λ so as to minimize generalization or test error, which is based on unknown data to avoid overfitting. The generalization error can be defined as follows.

Definition 2.21 (*Generalization error (Test error)*) *Generalization error*, also referred to as *test error*, is the estimated loss over an independent (test) sample $(x, y) \in (X, Y)^{(\text{test})}$:

$$\text{Err}_{\text{test}} = E_{(x,y)\in(X,Y)^{(\text{test})}} \left[\mathcal{L}(y, \mathcal{A}_\lambda((X, Y)^{(\text{train})}))\right].$$

Definition 2.22 (*Hyperparameter optimization problem*) The *hyperparameter optimization problem* can be stated in terms of a hyperparameter response function, $\psi \in \Psi$, as follows:

$$\lambda^{(*)} \approx \arg\min_{\lambda \in \Lambda} \psi(\lambda) \approx \arg\min_{\{\lambda^{(i)}\}_{i=1,2,\ldots,n}} \psi(\lambda) = \hat{\lambda}, \tag{2.7}$$

where

$$\psi(\lambda)^{(\text{test})} = \frac{1}{|(X, Y)^{(\text{test})}|} \sum_{x \in (X,Y)^{(\text{test})}} \mathcal{L}\left(y, \mathcal{A}_\lambda((X, Y)^{(\text{train})})\right). \tag{2.8}$$

! Attention: Validation and Test Data

The validation set $(X, Y)^{(\text{val})}$ is used during optimization to estimate the prediction error for model selection,

$$\psi(\lambda)^{(\text{val})} = \frac{1}{|(X, Y)^{(\text{val})}|} \sum_{x \in (X,Y)^{(\text{val})}} \mathcal{L}\left(y, \mathcal{A}_\lambda((X, Y)^{(\text{train})})\right), \tag{2.9}$$

whereas the test set in Eq. (2.8) is used for the assessment of the generalization error of the selected model.

Summarizing, we can define HPO in ML and DL as a minimization problem

Definition 2.23 (*Hyperparameter optimization*) Hyperparameter optimization is the minimization of

$$\psi(\lambda) \text{ over } \lambda \in \Lambda.$$

Definition 2.24 (*Hyperparameter surface*) Similar to the definition in Design of Experiments (DOE), the function $\psi \in \Psi$ is referred to as the hyperparameter *response surface*.

Different data sets, tasks (classification or regression), and methods define different sets Λ and functions Ψ.

A natural strategy for finding an adequate λ is described in Eq. (2.7): a set of candidate solutions, $\{\lambda^{(i)}\}_{i=1,2,\ldots,n}$, is chosen. Then $\psi(\lambda)$ is computed for each one, and the best hyperparameter configuration is returned as $\tilde{\lambda}$.

Whereas λ denotes an arbitrarily chosen hyperparameter configuration, important hyperparameter configurations will be labeled as follows: λ_i is the i-th hyperparameter configuration, λ_0 is the initial hyperparameter configuration, $\lambda^{(*)}(t)$ is the best hyperparameter configuration at iteration t, and $\lambda^{(*)}$ is the final best hyperparameter configuration.

Definition 2.25 (*Low effective dimension*) If a function f of two variables could be approximated by another function of one variable ($f(x_1, x_2) \approx g(x_1)$), we could say that f has a *low effective dimension*.

Several approaches exist for the tuning procedure. A model-based search is presented in this book. The corresponding model is called a surrogate model, or *surrogate*, $\mathcal{S}$, for short.

In Sect. 5.8, different experimental designs for benchmarking optimization methods are discussed: the most simple design evaluates *one* single algorithm on *one* problem, whereas the most complex design is used for comparing multiple algorithms on multiple problems. HPT can be seen as a variant of the simple design, because the experimenter is interested in the improved performance of one method on one problem. To obtain this goal, the best hyperparameter configuration is determined. However, this simple setting can be extended, because the performance of the tuned method can be compared to the performance of some default method or to a competitive state-of-the art method. In the latter case, the hyperparameters of the state-of-the-art method should also be tuned to enable a fair comparison. Although HPT is not a benchmarking method on its own, it can be seen as a prerequisite for

a fair and sound benchmark study. Note that the complex design requires adequate statistical methods for the comparison. An approach based on consensus ranking is presented in Chap. 12.

Tips: How to Select a Performance Measure

Kedziora et al. (2020) state that in classification "unsurprisingly", *accuracy*[2] is considered the most important performance measure. Accuracy might be an adequate performance measure for classification of balanced data. For unbalanced data, other measures are better. In general, there are many other ways to measure model quality, e.g., metrics based on time complexity and robustness or the model complexity (interpretability, see also Definition 2.27) Bartz-Beielstein et al. (2020a).

In contrast to classical optimization, where the same optimization function can be used for tuning and final evaluation, training of MLs and DL methods faces a different situation: Training and validation are usually based on the loss function whereas the final evaluation is based on a different measure, e.g., accuracy.

It is important to distinguish between estimates of performance (minimization of the generalization error) based on validation and test sets. The loss function, which has desirable mathematical properties, e.g., differentiability, acts as a surrogate for the performance measure the user is finally interested in. Several performance measures are used at different stages of the HPT procedures:

1. training loss, i.e., $\psi^{(\text{train})}$,
2. training accuracy, i.e., $f_{\text{acc}}^{(\text{train})}$,
3. validation loss, i.e., $\psi^{(\text{val})}$,
4. validation accuracy, i.e., $f_{\text{acc}}^{(\text{val})}$,
5. test loss, i.e., $\psi^{(\text{test})}$, and
6. test accuracy, i.e., $f_{\text{acc}}^{(\text{test})}$.

This complexity gives reason for the following question:

Question: Which performance measure should be used during the HPT (HPO) procedure?

Most authors recommend using test accuracy or test loss as the measure for hyperparameter tuning Schneider et al. (2019). In order to understand the correct usage of these performance measures, it is important to look at the goals, i.e., selection or assessment, of a tuning study.

To keep the discussion focus, accuracy was used in the previous considerations. Instead of accuracy, other measures, e.g., MMCE, can be considered.

[2] Accuracy in binary classification is the proportion of correct predictions among the total number of observations Metz (1978).

2.4 Model Selection and Assessment

Hastie et al. (2017) state that selection and assessment are two separate goals:

Selection: Estimating the performance of different models in order to choose the best one. Model selection is important *during* the tuning procedure.

Assessment: Model assessment is used for the *final* report (evaluation of the results). Having chosen a final hyperparameter configuration, $\lambda^{(*)}$, the assessment estimates the model's prediction error (generalization error) on new data based on Eq. (2.8). This determines whether predicted values from the model are likely to accurately predict responses on future observations or samples from the hold-out set $(X, Y)^{(\text{test})}$. This process may help to prevent problems such as overfitting.

In principle, there are two ways of model assessment and selection Hastie et al. (2017):

1. External assessment/selection uses different sets of data. The first p data samples are for model training and $n - p$ for validation. An explicit hold-out data set is used. Problem: holding back data from model fitting results in lower precision and power.
2. Internal assessment/selection uses data splitting and resampling methods. The true error might be *underestimated*, because the same data samples that were used for fitting the model are used for prediction. The so-called in-sample (also apparent, or resubstitution) error is smaller than the true error.

The test set $(X, Y)^{(\text{test})}$ should be used only at the end of the HPT procedure. It should not be used during the training and validation phase, because if the test set is used repeatedly, e.g., for choosing the model with smallest test-set error, "the test set error of the final chosen model will underestimate the true test error, sometimes substantially." Hastie et al. (2017).

The following example shows that there is no general agreement on how to use training, validation, and test sets as well as the associated performance measures.

Example: Basic Comparisons in Manual Search

Wilson et al. (2017) describe a manual search. They allocated a pre-specified budget on the number of epochs used for training each model.

- When a test set was available, it was used to chose the settings that achieved the best peak performance on the test set by the end of the fixed epoch budget.
- If no explicit test set was available, e.g., for Canadian Institute for Advanced Research, 10 classes (CIFAR-10), they chose the settings that achieved the lowest training loss at the end of the fixed epoch budget.

Theoretically, the results from the internal assessment are not of interest because new data values are not likely to coincide with their training set values. Bergstra and Bengio (2012) stated that "because of finite data sets, test error is not monotone in validation error, and depending on the set of particular hyperparameter values λ evaluated, the test error of the best-validation error configuration may vary", i.e.,

$$\psi_i^{(\text{val})} < \psi_j^{(\text{val})} \not\Longrightarrow \psi_i^{(\text{test})} < \psi_j^{(\text{test})},$$

where $\psi_i^{(\cdot)}$ denotes the value of the hyperparameter response surface for the i-th hyperparameter configuration λ_i.

Furthermore, the estimator, e.g., for loss, obtained by using a single hold-out test set usually has high variance. Therefore, Cross Validation (CV) methods were proposed. Hastie et al. (2017) concluded

> that estimation of test error for a particular training set is not easy in general, given just the data from that same training set. Instead, cross-validation and related methods may provide reasonable estimates of the expected error.

The standard practice for evaluating a model found by CV is to report the hyperparameter configuration that minimizes the loss on the validation data, i.e., $\hat{\lambda}$ as defined in Eq. (2.7). Repeated CV, i.e., k-fold CV, reduces the variance of the estimator and results in a more accurate estimate. There is, as always, a trade-off: the more CV folds, the better the estimate, but more computational time is needed.

Example: Reporting the model assessment (final evaluation)

It can be useful to take the uncertainty due to the choice of hyperparameters values into account, when one reports the performance of learning algorithms. Bergstra and Bengio (2012) present a procedure for estimating test set accuracy, which takes into account any uncertainty in the choice of which trial is actually the best-performing one. To explain this procedure, they distinguish between estimates of performance $\psi^{(\text{val})}$ and $\psi^{(\text{test})}$ based on the validation and test sets, respectively.

To resolve the difficulty of choosing the best configuration, Bergstra and Bengio (2012) reported a weighted average of all the test set scores, in which each one is weighted by the probability that its particular λ_s is in fact the best. In this view, the uncertainty arising from $X^{(\text{val})}$ being a finite sample makes the test-set score of the best model among $\{\lambda_i\}_{i=1,2,\ldots,n}$ a random variable.

2.5 Tunability and Complexity

The term *tunability* is used according to the definition presented in Probst et al. (2019a).

Definition 2.26 (*Tunability*) Tunability describes a measure for modeling algorithms as well as for individual hyperparameters. It is the difference between the model quality for default values (or reference values) and the model quality for optimized values (after tuning is completed).

Or in the words of Probst et al. (2019a): "measures for quantifying the tunability of the whole algorithm and specific hyperparameters based on the differences between the performance of default hyperparameters and the performance of the hyperparameters when this hyperparameter is set to an optimal value". Tunability of individual hyperparameters can also be used as a measure of their *relevance*, *importance*, or *sensitivity*. Accordingly, parameters with high tunability are of greater importance for the model. The model reacts strongly to (i.e., is sensitive to) changes in these hyperparameters.

Definition 2.27 (*Complexity*) The term *complexity* or model complexity generally describes, how many functions of different difficulty can be represented by a model.

Example: Complexity

For linear models, complexity can be influenced by the number of model coefficients. For Support Vector Machines (SVMs), it can be influenced by the parameter `cost`.

2.6 The Basic HPT Process

Now all ingredients are available for defining the basic HPT process.

Definition 2.28 (*The basic HPT process*) For a given space of hyperparameters Λ, a ML or DL model $\mathcal{A}$ with hyperparameters λ, training, validation, and testing data $(X, Y)^{(\text{train})}$, $(X, Y)^{(\text{val})}$, and $(X, Y)^{(\text{test})}$, respectively, a loss function $\mathcal{L}$, and a hyperparameter response surface function ψ, e.g., mean loss, the basic HPT process looks like this

(HPT-1) Set $t = 1$. Hyperparameter selection (at iteration t). Choose a set of hyperparameters from the space of hyperparameters, $\lambda(t) \in \Lambda$.

(HPT-2) ML or DL model building. Build the corresponding ML or DL model $\mathcal{A}_{\lambda(t)}$. Note: The model building step is listed separately, because this corresponds with the steps of the `keras` procedure: first the DL model is specified and compiled (via `compile`), then it is trained (e.g., via `fit`).

(HPT-3) ML or DL model training and evaluation (e.g., via `keras fit`). Fit the model $\mathcal{A}_{\lambda(t)}$ to the training data $(X, Y)^{(\text{train})}$ (see Fig. 2.1) and measure the final performance, e.g., expected loss, on the validation data $(X, Y)^{(\text{val})}$, see Eq. (2.9). Under k-fold CV, the performance measure from Eq. (2.9) can be written as

$$\psi_{\text{CV}}^{(\text{val})} = \frac{1}{k} \sum_{i=1}^{k} \frac{1}{|(X, Y)^{(\text{val})}|} \sum_{x \in (X,Y)_i^{(\text{val})}} \mathcal{L}\left(y, \mathcal{A}_{\lambda^{(t)}}((X, Y)_i^{(\text{train})})\right), \tag{2.10}$$

if the training and validation set partitions are generated k times.

(HPT-4) Hyperparameter update. The next set of hyperparameters to try, $\lambda(t+1)$, is chosen accordingly to minimize the performance, e.g., $\psi^{(\text{val})}$. An infill criterion (acquisition function) is used.

(HPT-5) Looping. Repeat until budget is exhausted.

(HPT-6) Final evaluation of the best hyperparameter set $\lambda^\star$on test (or hold out) data $(X, Y)^{(\text{test})}$, i.e., measuring performance on the test data

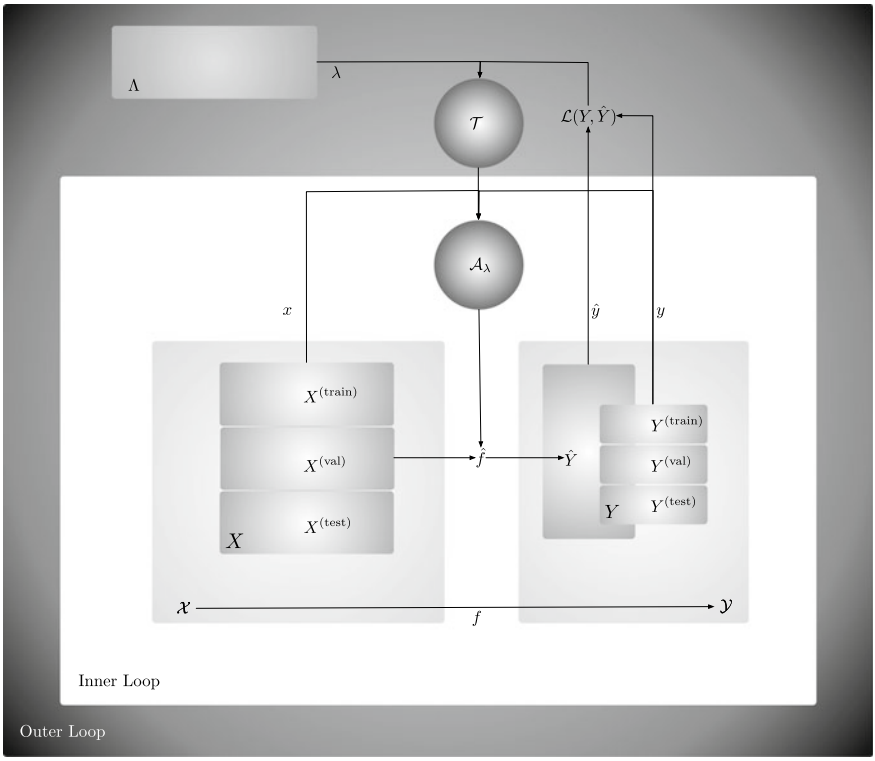

Fig. 2.2 Elements of the HPT process. A *dark gray* background color shows the "outer" optimization loop, whereas a *light gray* background is used for the "inner" optimization loop. $\mathcal{T}$ denotes the tuner, which selects hyperparameters $\lambda \in \Lambda$ to optimize the loss $\mathcal{L}$. The loss is the output from the inner optimization loop, where an algorithm $\mathcal{A}_\lambda$ (also referred to as the model or the method) optimizes the function $\hat{f}$, that builds a relation between input data $X \in \mathcal{X}$ and output data $Y \in \mathcal{Y}$. Note, depending on the three different tasks, i.e., model training, validation (selection), or test (assessment), the loss function $\mathcal{L}$, computes tree different losses: $\psi^{(\text{train})}$, $\psi^{(\text{val})}$, and $\psi^{(\text{test})}$

$$\psi^{(\text{test})} = \frac{1}{|(X,Y)^{(\text{test})}|} \sum_{x \in (X,Y)^{(\text{test})}} \mathcal{L}\left(y, \mathcal{A}_{\lambda^{(*)}}((X,Y)^{(\text{train} \cup \text{val})})\right). \tag{2.11}$$

The HPT process is illustrated in Fig. 2.2.

Essential for this process is the infill criterion (acquisition function) in (HPT-4). It uses the validation performance to determine the next set of hyperparameters to evaluate. and requires building and training a new model. Often, the hyperparameter space Λ is not differentiable or even continuous. Gradient methods are not applicable in Λ. Pattern search, Evolution Strategys (ESs), or other gradient-free methods are used instead.

2.7 Practical Considerations

Unfortunately, training, validation, and test data are used inconsistently in HPO studies: for example, Wilson et al. (2017) selected *training loss*, $\psi^{(\text{train})}$, (and not validation loss) during optimization and reported results on the test set $\psi^{(\text{test})}$.

Choi et al. (2019) considered this combination as a "somewhat non-standard choice" and performed tuning (optimization) on the validation set, i.e., they used $\psi^{(\text{val})}$ for tuning, and reported results $\psi^{(\text{test})}$ on the test set. Their study allows some valuable insight into the relationship of validation and test error:

> For a *relative comparison* between models during the tuning procedure, in-sample error is convenient and often leads to effective model selection. The reason is that the relative (rather than absolute performance) error is required for the comparisons. Choi et al. (2019)

Choi et al. (2019) compared the final predictive performance of NN optimizers after tuning the hyperparameters to minimize validation error. They concluded that their "final results hold regardless of whether they compare final validation error, i.e., $\psi^{(\text{val})}$, or test error, i.e., $\psi^{(\text{test})}$". Figure 1 in Choi et al. (2019) illustrates that the relative performance of optimizers stays the same, regardless of whether the validation or the test error is used. Choi et al. (2019) considered two statistics: (i) the quality of the best solution and (ii) the speed of training, i.e., the number of steps required to reach a fixed validation target.

2.7.1 Some Thoughts on Cross Validation

There are some drawbacks of k-fold CV: at first, the choice of the number of observations to be held out from each fit is unclear: if n denotes the size of the training data set, with $k = n$, which is referred to as Leave One Out Cross Validation (LOOCV), the CV estimator is approximately unbiased for the expected prediction error. But this estimator has high variance, because LOOCV does not mix the observations very much. The estimates from each fold are highly correlated and hence their average can

have high variance. Furthermore, computational costs are relatively high, because n evaluations of the model are necessary.

Furthermore, CV does not fully represent variability of variable selection, because p elements are removed each time from set of n. Kohavi (1995) reviewed accuracy estimation methods and compared CV and bootstrap Efron and Tibshirani (1993). Note that Picard and Cook (1984) proposed Monte Carlo (MC) CV as an improvement over standard CV.

2.7.2 Replicability and Stochasticity

Results from DL and ML tuning runs are noisy, e.g., caused by random sampling of batches and initial parameters. Repeats to estimate means and variances that are required for a sound statistical analysis are costly.

However, even if seeds are provided, full reproducibility cannot be guaranteed. Gramacy (2020) mentioned two important issues:

- First, Random Number Generator (RNG) sequences can vary across software versions.
- Second, conditional expressions involving floating point calculations can change across hardware architectures and lead to different results in stochastic experimentation even with identical pseudorandom numbers.

As a consequence, it is impossible to fully remove randomness from the experiments. López-Ibáñez et al. (2021b) provide guidelines and suggest tools that may help to overcome some of these reproducibility issues.

2.7.3 Implementation in R

Background: Data Types in R

Our implementation is done in the R programming language, where data and functions are represented as objects. Each object has a data type. The basic (or *atomic*) data types are shown in Table 2.3.

In addition to these data types, R uses an *internal storage mode* which can be queried using `typeof()`. Thus, there are two storage modes for the `numeric` data type:

- `integer` for integers and
- `double` for real values.

The corresponding variables are referred to as *numeric*.

Table 2.3 Atomic data types of the programming language R

Data type	Description	Examples
NULL	Empty set	NULL
logical	Boolean values	TRUE, FALSE
numeric	Integer and real values	1, 0.5
complex	Complex values	1+1i
character	Characters and strings of characters	"123", "test"

Factors are used in R to represent nominal (qualitative) features. Ordinal features can also be represented by factors in R (see argument `ordered` of the function `factor()`). However, this case is not considered here. Factors are generated with the generating function `factor()`. Factors are not atomic data types. Internally in R, factors are stored by numbers (integers), externally the name of the factor is used. We call the corresponding variables *categorical*.

Data types of the hyperparameters that are analyzed in this book can be obtained with the function `getModelConf`. The function `spot` can handle the data types `numeric`, `integer`, and `factor`.

Example: Hyperparameters and Their Types

The following code shows how to get the hyperparameter names and their corresponding types of the k-Nearest-Neighbor (KNN) method.

```
library("SPOTMisc")
cfg <- getModelConf(list(model = "kknn"))
cfg$tunepars


  ## [1] "k"         "distance"

cfg$type


  ## [1] "integer" "numeric"
```

The method KNN will be described in detail in Sect. 3.2.