Chapter 1
Automated Machine Learning and Bayesian Optimization

1.1 Automated Machine Learning

1.1.1 Motivation

Automated machine learning (AutoML) is introduced in this chapter to illustrate model selection and hyperparameter tuning and the specific features of the resulting optimization problems.

These issues are important because an algorithm that scores well on one learning task can score poorly on another, as summarized by the "no free lunch" (NFL) theorem (Wolpert and Macready 1995): even the same algorithm, with different hyperparameter values, can show very different performances (Wolpert 2002). NFL theorems have since grown into a significant research domain: their impact on fields like optimization and supervised learning is analysed in a recent survey paper (Adam et al. 2019), in which specific results are given for early stopping and cross-validation rules to conduct the training phase.

While this result emphasizes the role of the ML experts in the design of reliable applications, substantial research activity has been focusing on the possibility to exploit the potential of ML by making it easy to be used off the shelf also by non-experts, taking, in some sense, the "human out of the loop" (Shahriari 2016). This requires automating the configuration of the algorithm and the tuning of their hyperparameters.

The time-honoured approach to solve the problem of hyperparameter optimization, namely grid search, is hardly feasible for more than 2/3 hyperparameters and totally unsuitable in view of the growing complexity of ML models as deep networks which have many hyperparameters and may take hours or days to train the model.

Model selection and hyperparameter optimization are very important applications of Bayesian optimization (BO) (Frazier 2018) and offer a good motivation of the relevance assumed by BO in the ML community (Hutter et al. 2019). Section 1.2 is devoted to illustrating the basic workflow of the sequential model-based optimization (SMBO) and specifically of BO introducing its basic components: the surrogate

F. Archetti and A. Candelieri, *Bayesian Optimization and Data Science*,
SpringerBriefs in Optimization, https://doi.org/10.1007/978-3-030-24494-1_1

model and the acquisition function. Section 1.3 is devoted to a specific application showing the design and the implementation of a complex ML application of predictive analytics.

AutoML was formulated in Kotthoff et al. (2017) as "automatically and simultaneously choosing a learning algorithm and setting its hyperparameters to optimize its empirical performance on a given dataset". The more general definition of this problem is also called combined algorithm selection and hyperparameter (CASH) optimization.

This approach was originally built on WEKA and developed into Auto-WEKA (Thornton et al. 2013) which uses BO for an optimal selection among the components of WEKA for a given dataset and problem addressed (regression or classification). More recently, a new and robust AutoML system auto-sklearn, based on scikit-learn, has been proposed in Feurer et al. (2015).

This system uses several supervised learning algorithms, data preprocessing and feature processing methods, leading to an overall number of up 110 machine learning algorithms which can be compared according to their generalization capability that is to make accurate "prediction" (regression or classification) on new data.

The evaluation of a ML algorithm means evaluating any performance index or metric (e.g. accuracy or root-mean-squared error in classification and regression tasks, respectively) which we shall indicate as loss function.

This evaluation is usually done by a k-fold cross-validation procedure. This procedure consists in randomly dividing the set of data into k groups, or *folds*, of approximately equal size. With an index i iterating from 1 to k, the ith fold is used as a validation or testing set while the ML algorithm is trained on the training set consisting of the remaining $k-1$ folds. This means that each algorithm must be trained and validated k times, making this procedure time consuming and significantly expensive, especially in the case of large databases or for those algorithms whose training is expensive. Usually, a limited "budget" to train and test different configurations of an ML algorithm is available, bounding the available computational resources such as CPU, wall clock time, memory usage.

The successful application of ML in a broad range of domains and the shortage of human experts has recently led to an increase in the demand of AutoML solutions. Indeed, a growing number of commercial enterprises are addressing this demand: Microsoft's Azure Machine Learning, Amazon Machine Learning and Google's platforms (Prediction API, Vizier, Hypertune and the more recent AutoML Tables) offer "machine learning" ecosystems with their own AutoML tools. The need of sample efficient AutoML solutions cannot be overestimated also in view of the growing awareness of the heavy carbon footprint of large scale machine learning models, in particular deep learning. A recent paper (Strubell et al. 2019) argues that the cost of the hyperparameter tuning and the required experimentation (Sect. 1.1.3) can reach a tag of CO_2 emissions in the range of tens of thousands Libs which can reach hundreds of thousands if the neural architecture search (Sects. 1.1.2 and 1.1.4) is also considered. The unprecedented scale of this challenge requires not only better algorithms in the Bayesian framework but revolutionary solutions like quantum computing. Bayesian Optimization, (Zhu et al. 2018), has turned out a good solu-

tion for data -driven quantum circuit training, using the software tool OPTaas, of Mindfoundry which is analysed in Chap. 6.

1.1.2 Model Selection

A learning algorithm A maps training data points $x_1, \ldots, x_n$ to their corresponding "targets" $y_1, \ldots, y_n$, where y_i is continuous in the case of regression or a "label" (categorical values) in the case of classification. All the pairs (x_i, y_i) are organized into a dataset $\mathcal{D} = \{(x_i, y_i)\}_{1:n}$.

The model selection problem is formulated as follows:

$$A^* \in \underset{A \in \mathcal{A}}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\left(A, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)}\right)$$

where $\mathcal{L}\left(A, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)}\right)$ is the loss achieved by A when trained on $\mathcal{D}_{\text{train}}^{(i)}$ and evaluated on $\mathcal{D}_{\text{valid}}^{(i)}$. The set $\mathcal{A}$ contains all the available ML algorithms, $\mathcal{A} = \left\{A^{(1)}, \ldots, A^{(h)}\right\}$. We use k-fold cross-validation which splits the training data into k equal-sized validation folds, $\mathcal{D}_{\text{valid}}^{(1)}, \ldots, \mathcal{D}_{\text{valid}}^{(k)}$, and associated training sets $\mathcal{D}_{\text{train}}^{(1)}, \ldots, \mathcal{D}_{\text{train}}^{(k)}$, where $\mathcal{D}_{\text{train}}^{(i)} = \mathcal{D} \backslash \mathcal{D}_{\text{valid}}^{(i)}$.

The following figure (Fig. 1.1) displays a schematic representation of a model selection problem.

Before going ahead, it is important to clarify the difference between "training" and "validating" a ML model, as well as the difference between "parameters" and "hyperparameters" of a ML algorithm.

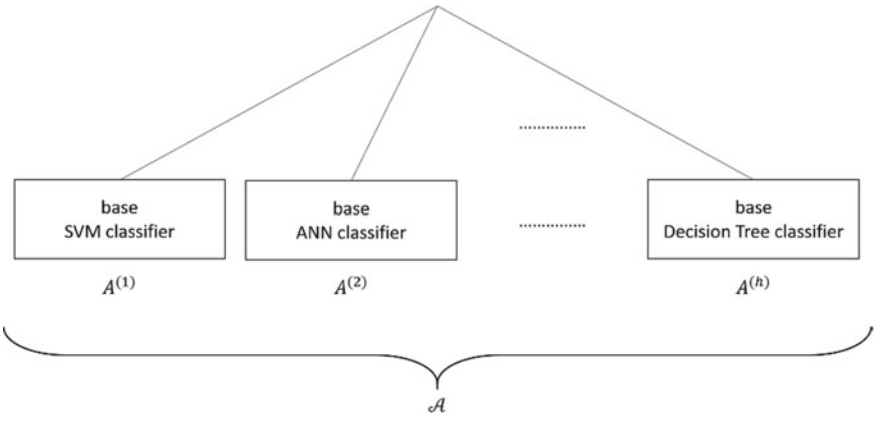


Fig. 1.1 An example of model selection. A set of "base" algorithms are available in the set A, where "base" refers to the adoption of default values for the algorithm's hyperparameters

The algorithm A is characterized by:

- a vector θ of "model parameters", whose value is learned directly from data, during the "training" phase
- a vector of "hyperparameters" $\gamma \in \Gamma$ that change the way the algorithm "learns" the values for θ. Hyperparameters can be set up manually or optimized on the "validation" phase.

For instance, the weights on the connections of an artificial neural network are parameters to be learned, while a number of hidden layers, a number of neurons in each hidden layer, activation function and learning rate are hyperparameters.

The goal of *training* is to estimate the value $\hat{\theta}$ minimizing a given loss function $\mathcal{L}_{\text{train}}$ (e.g. classification error or root-mean-squared error for classification and regression problems, respectively) on training data $\mathcal{D}$. The resulting ML model can be then validated on a validation dataset (or fold, in the case of k-fold cross-validation), measuring the corresponding loss function $\mathcal{L}_{\text{valid}}$. During validation, the estimate $\hat{\theta}$ does not change for each fold. The hyperparameters γ are not estimated during the train, and they must be set up before training. Therefore, $\hat{\theta}$ as well as $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{valid}}$ depend on the value of γ.

Many ML algorithms, such as in artificial neural networks or support vector machines, use an analytical form for $\mathcal{L}_{\text{train}}$, so that $\hat{\theta}$ can be estimated by gradient-based methods. Other ML algorithms have no parameters (e.g. instance-based algorithms such as k-nearest neighbours). On the contrary, $\mathcal{L}_{\text{valid}}$ is black box and its optimization, depending on the hyperparameters γ, requires derivative-free GO approaches, such as BO.

When k-fold cross-validation is concerned, the common way to compute the overall $\mathcal{L}_{\text{valid}}$ is to average the values obtained on the k different folds, as in a randomized experiment.

The loss function computed on each fold $\mathcal{L}^i_{\text{valid}}$ with $i = 1, \ldots, k$, can be considered as an observation from a sample of k elements, whose size can be controlled by some statistical tests. This approach could result in an early stopping when the hyperparameter configuration is unlikely to yield a good result (Florea et al. 2018). The same principle is used in hyperband (Li et al. 2016), halving sequentially the set of configurations "deserving" more sampling.

1.1.3 Hyperparameter Optimization

Given n hyperparameters $\gamma_1, \ldots, \gamma_n$ with domains $\Gamma_1, \ldots, \Gamma_n$, the hyperparameter space Γ is a subset of the product of these domains $\Gamma_1 \times \cdots \times \Gamma_n$. Γ is a subset because certain settings of one hyperparameter render other hyperparameters inactive. For example, the parameters determining the specifics of the third layer of an artificial neural network (ANN) are not relevant if the network depth is set to one or two.

Likewise, the hyperparameters of a support vector machine's (SVM) polynomial kernel are not relevant if we use a different kernel instead. More formally, we say that a hyperparameter γ_i is conditional on another hyperparameter γ_j; that is γ_i is

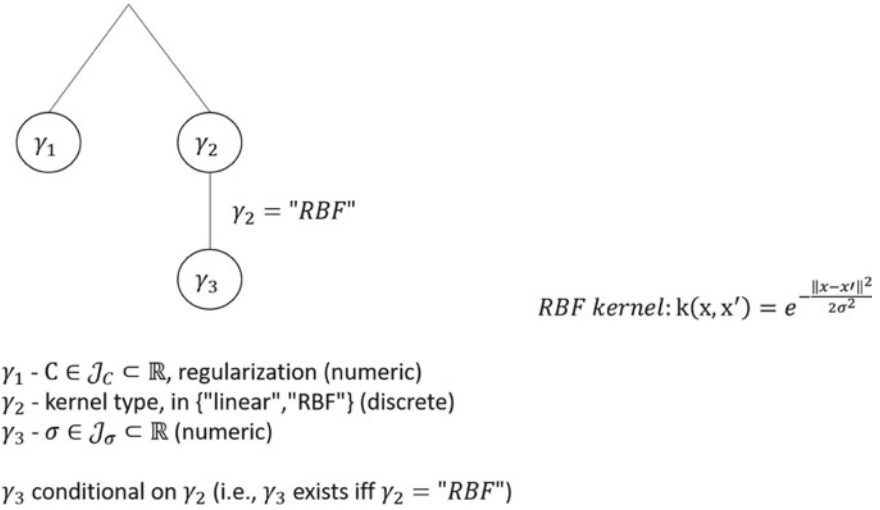


Fig. 1.2 An example of hyperparameter optimization for a support vector machine (SVM) classifier with linear or radial basis function (RBF) kernel. $\mathcal{J}_C$ and $\mathcal{J}_\sigma$ are the ranges for values of the regularization hyperparameter C and the RBF kernel's hyperparameter σ

active if and only if hyperparameter γ_j takes values from a given set $V_i(j) \subset \Gamma_j$; in this case, we call γ_j a parent of γ_i. Conditional hyperparameters generate a tree-structured space or, in some cases, a directed acyclic graph (DAG). Given such a structured space Γ, the (hierarchical) hyperparameter optimization problem can be written as:

$$\boldsymbol{\gamma}^* \in \operatorname{argmin}_{\boldsymbol{\gamma} \in \Gamma} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\left(A_{\boldsymbol{\gamma}}, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)}\right) \tag{1.1}$$

Figures 1.2 and 1.3 propose two examples of hyperparameter optimization for an SVM and an ANN, respectively.

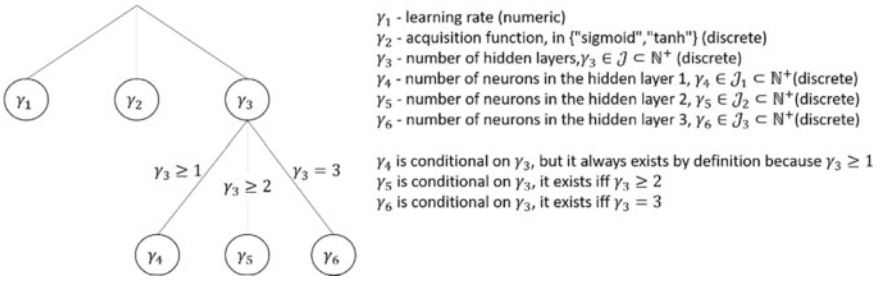


Fig. 1.3 An example of hyperparameter optimization for an artificial neural network (ANN) classifier with at maximum three hidden layers. $\mathcal{J}_1$, $\mathcal{J}_2$ and $\mathcal{J}_3$ are the ranges for number of neurons in the hidden layer 1, 2 and 3, respectively

The problem of hyperparameter optimization of a given learning algorithm is quite similar to model selection. There are still some key differences in that hyperparameters are often continuous, that hyperparameter spaces are often high dimensional, like in deep neural networks, and that we can exploit correlation structure between hyperparameter settings $\boldsymbol{\gamma}, \boldsymbol{\gamma}' \in \Gamma$.

1.1.4 Combined Algorithm Selection and Hyperparameter Optimization

Given a set of algorithms $\mathcal{A} = \left\{A^{(1)}, \ldots, A^{(h)}\right\}$ with associated hyperparameter spaces, $\Gamma^{(1)}, \ldots, \Gamma^{(h)}$, we define the combined algorithm selection and hyperparameter optimization problem (combined algorithm selection and hyperparameter optimization, CASH) as computing:

$$A^*_{\gamma^*} \in \text{argmin}_{A^{(j)} \in \mathcal{A}, \gamma \in \Gamma^{(j)}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\left(A^{(j)}_{\gamma}, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}}\right) \tag{1.2}$$

We note that this problem can be reformulated as a single combined hierarchical hyperparameter optimization problem with hyperparameter space $\Gamma = \Gamma^{(1)} \cup \ldots \cup \Gamma^{(h)} \cup \{\gamma_r\}$, where γ_r is a new root-level hyperparameter that selects between algorithms $A^{(1)}, \ldots, A^{(h)}$. The root-level hyperparameter of each subspace $\Gamma^{(i)}$ is made conditional on γ_r. In principle, the problem can be tackled in various ways: SMBO is a versatile stochastic optimization framework that can work with both categorical and continuous hyperparameters and that can exploit the hierarchical structure stemming from the conditional parameters (Kotthof et al. 2017) (Fig. 1.4).

1.1.5 Why Hyperparameter Optimization Is Important?

Figures 1.5 and 1.6 display a loss function, namely mean squared error (MSE), computed on ten fold cross-validation, with respect to the hyperparameters of a support vector machine (SVM) regression, namely the regularization $C \in [0, 1000]$ and the hyperparameter $\sigma \in [0.002, 0.1]$ of the radial basis function (RBF) kernel. Two different benchmark datasets have been used: CPU dataset (https://archive.ics.uci.edu/ml/datasets/Computer+Hardware) and Yacht dataset (http://archive.ics.uci.edu/ml/datasets/yacht+hydrodynamics). One can see how the performance metrics (i.e. MSE) is impacted by hyperparameter value.

Another critical feature is that the value of the loss function, for each hyperparameter configuration, is the outcome of the randomized process of k-fold cross-validation. An often-overlooked point is to measure the uncertainty of the prediction error esti-

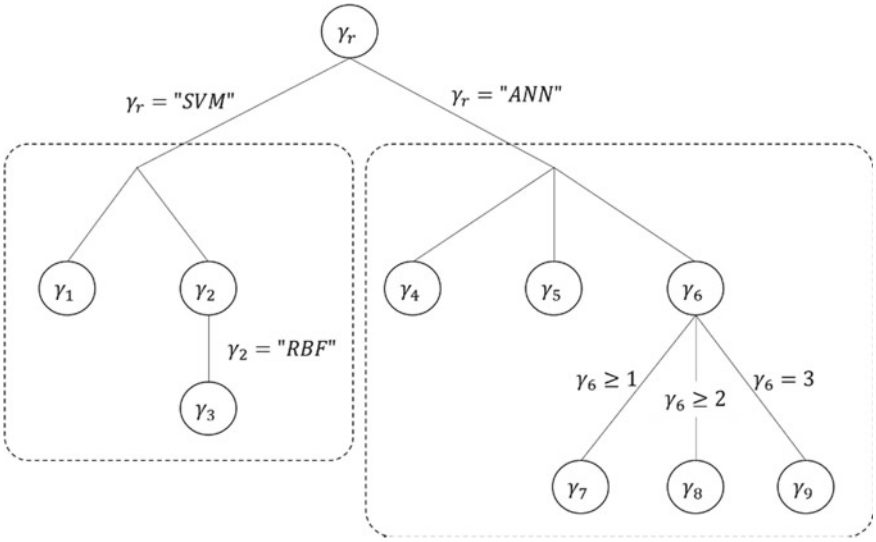


Fig. 1.4 An example of CASH considering an SVM and an ANN classifier. Description of the hyperparameters of each algorithm follows from Figs. 1.2 and 1.3, while γ_r is the further "root" CASH hyperparameter introduced to model the choice between the types of machine learning algorithm

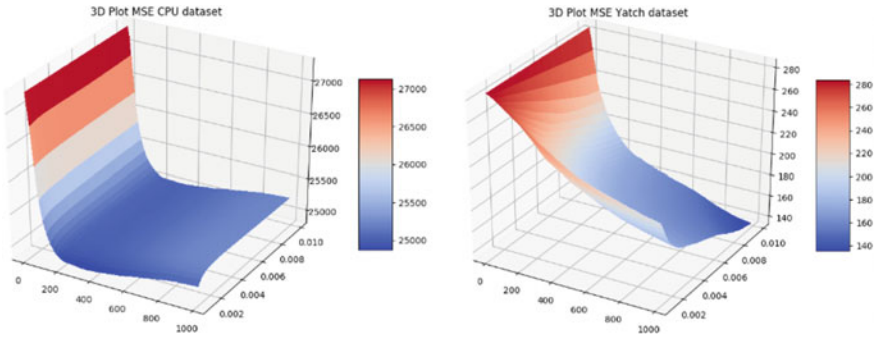


Fig. 1.5 Error on ten fold cross-validation with respect to two hyperparameters of an SVM with RBF kernel on the CPU dataset (left) and Yacht dataset (right)

mators which is important because the accuracy of model selection is limited by the variance of model estimates. Cross-validation provides an unbiased estimate of the prediction error on the training set, but the estimation of the variance is still crucial. Seminal contributions to this problem are given in Nadeau and Bengio (2000) and Bengio and Grandvalet (2004).

In a recent paper (Jiang and Wang 2017), a uniform normalized variance is proposed which not only measures model accuracy but relates it to the number of folds.

Let us have a look at the simplest case in which we keep fixed the value of C of an SVM and can choose among different kernels, represented by the options A to E.

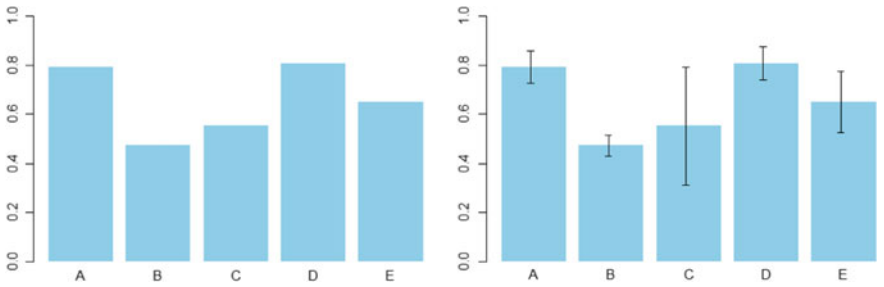


Fig. 1.6 MSE computed over k-fold cross-validation for an SVM classifier with five different kernels (options A to E). On the left, the mean of MSE is reported, and on the right also variance is depicted

If we consider the mean of MSE over the k-folds, we obtain the chart on the left, but we ignore the variance, which is instead depicted in the figure on the right. The variance makes the choice not so obvious. Indeed, there is a nonzero probability that option C could turn out a better choice than E.

1.2 The Basic Structure of Bayesian Optimization

1.2.1 Sequential Model-Based Optimization

This is the reference problem, where we use x for the sake of generality:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in X} f(\mathbf{x})$$

In the cases seen in previous Sect. 1.1 $\mathbf{x} = \boldsymbol{\gamma}$ for the hyperparameter optimization (1.1), while, in the most general case of CASH (1.2), $\mathbf{x}$ is a vector whose first component is the decision variable associated with the ML algorithm to choose in the set $\mathcal{A}$ and the remaining components are the associated hyperparameters $\boldsymbol{\gamma}$.

When we are dealing with this optimization problem, we are faced with two kinds of uncertainty:

1. Measurement uncertainty called "noise". We can not observe $f(x_i)$ but rather a noisy value $y_i = f(x_i) + \varepsilon$, where $\varepsilon$is the observation noise, assumed to be $\varepsilon \sim \mathcal{N}(0, \lambda^2)$ (Fig. 1.7).

Many papers are dealing with noise (Frazier 2018), but most of results in BO can be obtained without explicitly taking measurement noise into account. The "workhorse" GP model can handle it without disrupting the basic algorithm structure.

2. Even removing the noise, as it is often assumed, we still have a problem of "structural uncertainty". For instance, if we have three noise-free evaluations of $f(x)$, $D_{1:3} = \{(x_i, y_i)\}_{i=1,..,3}$, we still have infinite number of functions with different minima and minimizers, compatibly with $D_{1:3}$, as depicted in Fig. 1.8.

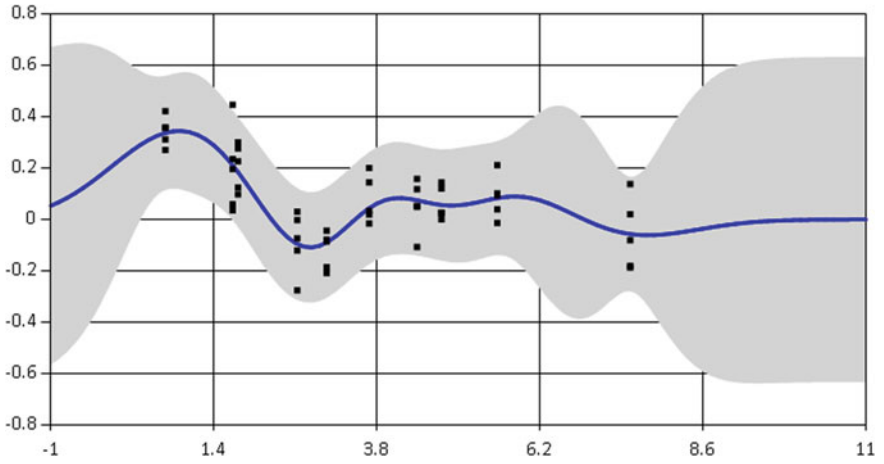
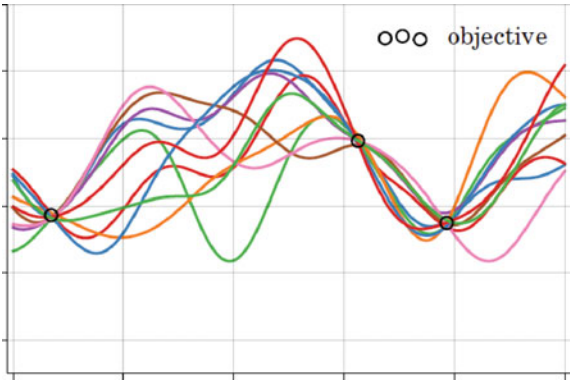


Fig. 1.7 An example of GP model with noisy evaluation: for the same point, x_i is possible to observe different observations y_i

Fig. 1.8 Different functions compatible with the function observations $D_{1:3}$



In many cases, X is assumed a hyper–rectangle (box-bounded or essentially unconstrained optimization). In this chapter, we present the basic structure of BO in which the design variables, the hyperparameters, are assumed to be continuous.

We remark that BO can handle more complex design space: variables can be continuous, integer, categorical and conditional. This is in general the case of automatic algorithm configuration (AAC), hyperparameter optimization in machine learning and CASH.

Nonlinear constraints can be considered including the case in which they are the results of a simulation model and are partially/completely unknown, as will be discussed in Chap. 5.

We do not assume the existence in $f(x)$ of structural properties like linearity/convexity. We also assume that first and second derivatives are not available

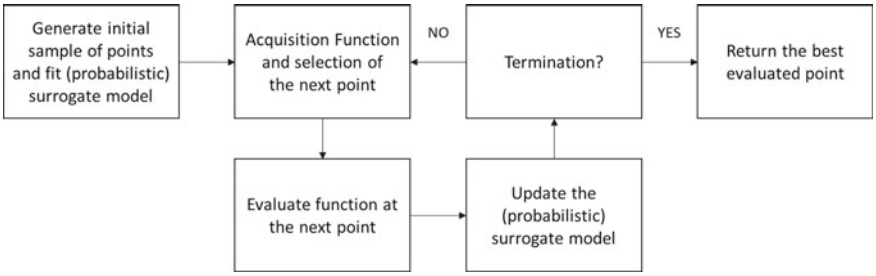


Fig. 1.9 A schematic representation of the Bayesian optimization process

to the optimizer. Where derivative information can be observed along with $f(x)$, it can be embedded in the learning process, as we shall see in Chap. 3.

In situations as the above, an approach is to create a surrogate model of the objective function and build on it a sequential model-based optimizer (SMBO) evaluating $f(x)$ at the points $x_1, x_2, \ldots, x_n$ updating, on the basis of new values, the surrogate and, as shown in Figs. 1.10 and 1.11, suggesting the next evaluation point. This suggestion is performed by an acquisition function, which balances exploitation and exploration (as better detailed in Sect. 1.2.3) and represets the "utility" to select a point. The SMBO process is summarized in the schema and steps (Fig. 1.9).

Steps of the sequential model-based optimization process
STEP 1—Generating initial sample/design—by Random Search (Chap. 2) or other sampling methods—and fitting the first (probabilistic) surrogate model (Sect. 1.2.2 and Chap. 3)
STEP 2—Identifying, depending on the acquisition function, the new promising point(s), evaluating the function (Sect. 1.2.3 and Chap. 4)
STEP 3—Evaluating the objective function
STEP 4—Update the (probabilistic) surrogate model (Chap. 3)
STEP 5—Checking for termination criteria: if at least one is active, go to STEP 6, else go to STEP 2
STEP 6—Return the current best solution (usually the "best seen")

1.2.2 Surrogate Model

In this book we focus on probabilistic surrogate model, we must anyway remark that deterministic surrogate models are also widely studied and applied in academia and industry (Cozad et al. 2014). Probabilistic surrogate models are able to offer an estimate of the uncertainty which is used to balance the trade-off between exploration and exploitation. The Gaussian process is most widely adopted model. The idea behind GP is that the values of the objective function correspond to realizations of a multivariate Gaussian process that for each x returns a mean and a variance, as reported in Figs. 1.10 and 1.11.

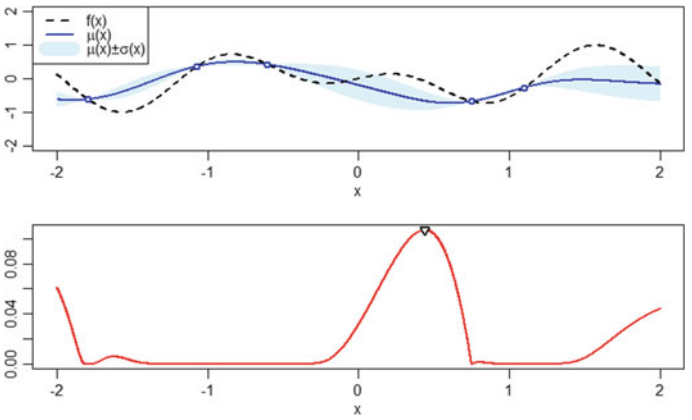


Fig. 1.10 A GP conditioned to five observations (top) and corresponding "utility" (bottom). The maximum indicates the next point where to evaluate the objective function

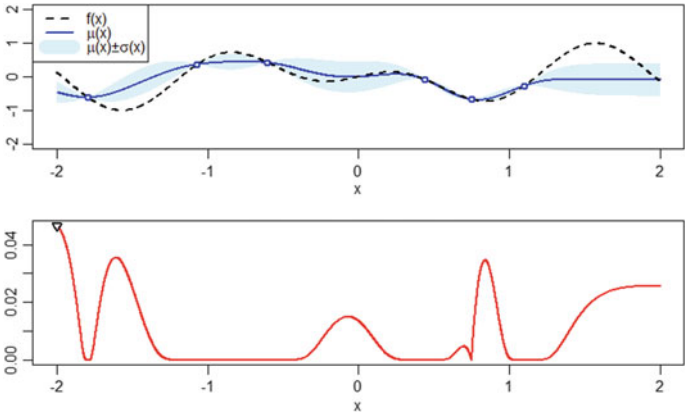


Fig. 1.11 How GP and acquisition function change after one more function evaluation, selected as in Fig. 1.10

One way to interpret a Gaussian process (GP) regression model is to think of it as defining a distribution over functions and with inference taking place directly in the space of functions. A Gaussian process is a collection of random variables, any finite number of which has a joint Gaussian distribution. A Gaussian process is completely specified by its mean function $\mu(x) = \mathbb{E}[f(x)]$ and covariance function $k(x, x') = \mathbb{E}\big[(f(x) - \mu(x))\big(f(x') - \mu(x')\big)\big]$. The covariance function $k(x, x')$ is also named *kernel* and amounts to impose a geometry to our decision space to Gaussian process depending on observations.

The Gaussian process is initially fitted on a set of available observations $D_{1:n}$ (STEP 1 of SMBO).

$$\mu_n(x) = \mathbb{E}[f(x)|D_{1:n}, x] = k(x, X_{1:n})\left[\mathrm{K}(X_{1:n}, X_{1:n}) + \lambda^2 I\right]^{-1} y$$
$$\sigma_n^2(x) = k(x, x) - k(x, X_{1:n})\left[\mathrm{K}(X_{1:n}, X_{1:n}) + \lambda^2 I\right]^{-1} k(X_{1:n}, x)$$

where $D_{1:n} = \{(x_i, y_i)\}_{i=1,..,n}$ is the set of observation performed so far, the vector $\mathbf{y}$ gives the value of the function at the previous points, the covariance matrix K has entries $\mathrm{K}_{ij} = k\left(x_i, x_j\right)$, with $i, j = 1, \ldots, n$, and $k(x, X_{1:n})$ is a n-dimensional vector with components $\mathrm{k}_i = k(x, x_i)$.

GP also allows to build an "acquisition function" (such as the probability of improvement, PI, over the best value observed so far, introduced in the next subsection) which modulates the trade-off between exploration and exploitation and suggests the next promising candidate point x_{n+1}.

The GP is useful also because it provides an analytical expression for the update of mean and variance as a new function evaluation (x_{n+1}, y_{n+1}) arrives, by using the same formulas and simply replacing n with $n + 1$.

In this chapter, the reference kernel for the GP is the squared exponential (SE):

$$k_{\mathrm{SE}}\left(x, x'\right) = \mathrm{e}^{-\frac{\left\|x - x'^2\right\|}{2\ell^2}}$$

With ℓ known as *characteristic length-scale*. The role of this hyperparameter is to rescale any point x by $1/\ell$ before computing the kernel value. A large length scale implies long-range correlations, whereas a short length scale makes function values strongly correlated only if their respective inputs are very close to each other.

One should not see GP as one-size-fit-all recipe: many different kernels, as well as other models, have been suggested and are reviewed in Chap. 3. Also, many acquisition functions have been suggested and will be reviewed in Chap. 4.

1.2.3 Acquisition Function

Besides the surrogate model, the other main building block is the *utility function*, also named *acquisition function* or *infill criterion*.

Acquisition functions (STEP 2 of SMBO) are a critical part of the BO framework: many have been proposed, and new ones are being suggested to cope with new problems and take advantage of more computational power. Chapter 4 is devoted to this topic.

The first utility function was probability of improvement (PI) (Kushner 1964):

$$PI(x) = P\left(f(x) \leq f\left(x^+\right)\right) = \phi\left(\frac{f\left(x^+\right) - \mu(x)}{\sigma(x)}\right)$$

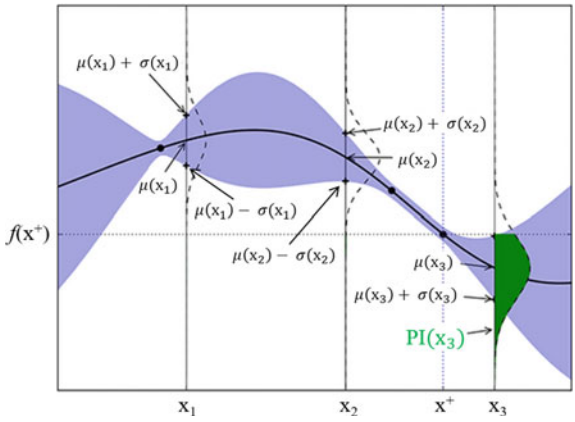


Fig. 1.12 A representation of probability of improvement

where $f\left(x^{+}\right)$ is the best value of the objective function observed so far, $\mu(x)$ and $\sigma(x)$ are mean and standard deviation of the probabilistic surrogate model, such as a GP, and $\phi(\cdot)$ is the probability density function.

This function will be analysed in detail in Chap. 4. This picture shows that the value of PI in x_3, given the best y value obtained in x^{+}, corresponds to the area depicted in green (Fig. 1.12).

GP, as surrogate model, has several advantages, which explain why it is widely used:

1. It provides a closed formula for mean and variance which can be analytically updated through new observations and takes care of measurement uncertainty and structural uncertainty (Figs. 1.4 and 1.5).
2. We can easily define an analytical acquisition function to deal with exploration—exploitation dilemma (this is a defining feature of BO, and its relationship with learning will be considered in Chap. 2).
3. The derivative of a GP is also a GP allowing to estimate in the training process not only the value of the function but also its gradients and hessians, given their, eventually noisy, observations: this enables to speed up the convergence reducing the variance in the GP regression (Chap. 3). This possibility is yet not really exploited in the available software being fraught with numerical instability, as we shall discuss in Chap. 3.

On the other hand, GP-based BO does not scale well with the number of decision variables (dimension of the design space). This is due, according to Kandasamy et al. (2015), to the statistical difficulty to scale up nonparametric regression in high dimensions and the computational challenge in maximizing the acquisition function. In Chap. 4, we shall see that many approaches have been proposed to mitigate these problems: random dropout and random embeddings of relevant search space dimensions (Wang et al. 2016; Chen et al. 2012) and structural assumptions such as the additive form of the objective function (Kandasamy et al. 2015).

1.3 Automated Machine Learning for Predictive Analytics

This section is devoted to an optimized machine learning pipeline that has been designed and developed in the context of research and customer projects and deployed in a number of situations (Candelieri 2017; Candelieri and Archetti 2018).

According to this approach, time-series forecasting is based on two consecutive phases: time-series clustering and artificial neural network (ANN) for regression.

As the loss function, we consider a measure of forecasting error, namely mean absolute percentage error (MAPE).

The problem consists of hyperparameter optimization where the number of ANN predictors to be learned in the second phase depends on the value of one of the hyperparameters of the first phase (i.e. number of clusters). In Fig. 1.13, we provide a representation of this problem:

Figure 1.14 summarizes the organization of the predictive pipeline. With respect to the previous picture, an emphasis on data used for training the overall system is also given.

Data was organized into a distinct time-series dataset $\mathcal{D} = \{x_1, \ldots, x_l\}$ consisting of l vectors, one for each day in the observation period, and where each vector x_i is a set of 24 ordered values, one for each hour of the day. The basic advantage of using time-series clustering as first stage is that it might allow for the identification of typical patterns within the time window of interest and, consequently, for splitting the dataset into subsets (i.e. clusters) which are then used in the second stage.

The clustering algorithm, kernel k-means, is a generalization of the standard k-means: it implicitly maps data from the input space, spanned by the original set of

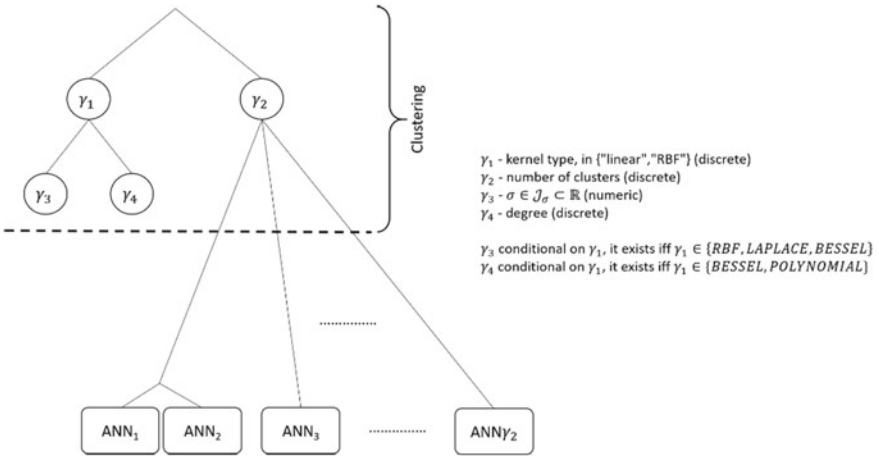


Fig. 1.13 Hyperparameter optimization problem for the predictive analytics pipeline considered. The number of ANN models to be trained in the second phase depends on the value of the hyperparameter defining the number of clusters in the first phase

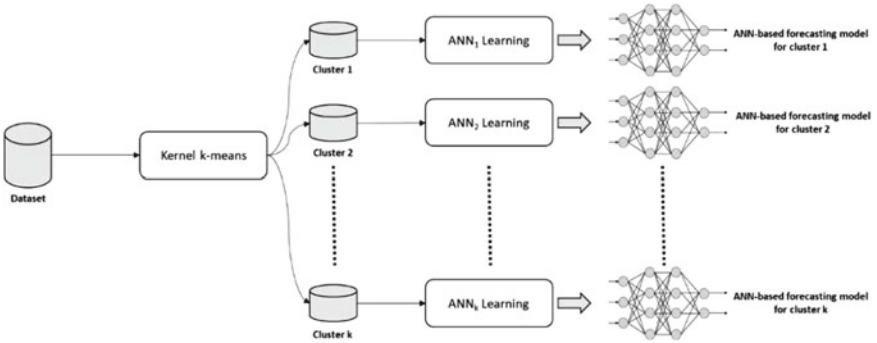


Fig. 1.14 A predictive analytics pipeline for time-series forecasting: first stage is clustering through kernel k-means; in the second stage, an ANN is learned on each cluster

data, to a higher-dimensional space, namely feature space. Therefore, kernel k-means can discover clusters that are nonlinearly separable in input space.

The second stage of the pipeline consists in training $\bar{k}$ different ANNs, one for each one of the $\bar{k}$ clusters resulting from the first stage. The first 6 values of the time series and the remaining $24 - 6 = 18$ are, respectively, the input and output neurons of every ANN. For each ANN, the hyperparameters reported in Table 1.2 are optimized via SMBO. The number of ANNs is not given a priori, and it depends on the number $\bar{k}$ of clusters identified from the first stage. The training process for the $\bar{k}$ ANNs is performed in parallel.

Tables 1.1 and 1.2 summarize, separately for the two stages, the hyperparameters

Table 1.1 Decision variables/hyperparameters for the clustering phase

Hyperparameter	Hyperparameter name	Type	Description
γ_1	$\bar{k}$	Integer	Number of clusters. Possible values are from 3 to 9
γ_2	Kernel type	Categorical	Type of kernel used in the kernel-based clustering. Possible kernels are: linear, spline, RBF, Laplace, Bessel, polynomial
γ_3	σ	Numeric, conditioned	Hyperparameter of the RBF, Laplace and Bessel kernels. Possible values are in $[10^{-5}, 10^{5}]$
γ_4	Degree	Integer, conditioned	Hyperparameter of the Bessel and polynomial kernels. Possible values are 2, 3 or 4

Table 1.2 Decision variables/hyperparameters for the ANN learning phase

Hyperparameter	Hyperparameter name	Type	Description
$\bar{\gamma}_1^k$	Hidden layers	Integer	Number of hidden layers in the artificial neural network. Possible values are 1, 2 or 3
$\bar{\gamma}_2^k$	Neurons in the hidden layer 1	Integer	Number of neurons in the hidden layer 1. Possible values are from 1 to 20
$\bar{\gamma}_3^k$	Neurons in the hidden layer 2	Integer, conditioned	Number of neurons in the hidden layer 2 (if $\bar{x}_1^k > 1$). Possible values are from 1 to 20
$\bar{\gamma}_4^k$	Neurons in the hidden layer 3	Integer, conditioned	Number of neurons in the hidden layer 3 (if $\bar{x}_1^k > 2$). Possible values are from 1 to 20
$\bar{\gamma}_5^k$	Algorithm	Categorical	Type of algorithm used to train the artificial neural network. Possible values are: backprop, rprop+, rprop-, sag, slr
$\bar{\gamma}_6^k$	Learning rate	Numeric, conditioned	Learning rate of the backprop algorithm. Possible values are in the range [0.1, 1.0]
$\bar{\gamma}_7^k$	Error function	Categorical	Function used to compute training error. Possible values are: sse and ce
$\bar{\gamma}_8^k$	Activation function	Categorical	Function used to compute the output of every neuron. Possible values are: logistic and tanh
$\bar{\gamma}_9^k$	Linear output	Logical	This hyperparameter defines whether to use a linear combination in the output layer of the artificial neural network or not. Possible values are TRUE or FALSE

of the predictive analytics pipelines (i.e. decision variables). According to the value of the hyperparameter γ_1 (i.e. the number of clusters), the number of hyperparameters in the second phase is consequently defined: it is γ_1 times 9. Since γ_1 is an integer

ranging from 3 to 9, this means that the overall number of hyperparameters for the pipeline ranges from $4 + 3 \times 9 = 31$ to $4 + 9 \times 9 = 85$.

The performance measure is mean average percentage error (MAPE):

$$\text{MAPE} = \frac{1}{T} \sum_{t=1}^{T} \left| \frac{A_t - F_t}{A_t} \right|$$

where A_t and F_t are the actual and forecasted values, respectively, at time t, and T is the number of time steps predicted. Since we have a MAPE value for every time series and an ANN is trained for every cluster, MAPE is aggregated as:

$$\min_{x \in X} \left\{ f(x) = \max_{k=1,\ldots,\bar{k}} \overline{\text{MAPE}}_k \right\}$$

This predictive pipeline was developed in R by using the following packages:

- "mlrMBO" for BO (described in Chap. 6), by setting a random forest as surrogate model and testing PI, EI and LCB as acquisition functions. The initial design has been sampled according to latin hypercube sampling (LHS) procedure
- "kernlab" for implementing the kernel k-means clustering
- "neuralnet" for implementing the artificial neural networks.

The approach has been validated on two different datasets, one public and related to energy consumption (https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction), and one collected during a European project and related to urban water consumption. In both cases the first six actual values are sufficient to forecast, in one shot, the overall consumption pattern for the day. The resulting value of prediction error was very low (approximately 0.1%) for both the test cases considered.

This means that, when a new data comes (i.e. a vector of consumptions at the first six hours of the day) it is assigned to one of the clusters and, consequently, the corresponding predictor is selected and used to forecast. Thus, the overall pipeline is flexible to adapt the choice of predictor to the incoming data.

References

Adam, S.P., Alexandropoulos, S.A.N., Pardalos, P.M., Vrahatis, M.N.: No free lunch theorem: a review. In: Approximation and Optimization, pp. 57–82. Springer, Cham (2019)

Bengio, Y., Grandvalet, Y.: No unbiased estimator of the variance of k-fold cross-validation. J. Mach. Learning Res. **5**, 1089–1105 (2004, September)

Candelieri, A., Archetti, F.: Global optimization in machine learning: the design of a predictive analytics application. Soft Comput. 1–9 (2018). https://doi.org/10.1007/s00500-018-3597-8

Candelieri, A.: Clustering and support vector regression for water demand forecasting and anomaly detection. Water (Switzerland) **9** (2017). https://doi.org/10.3390/w9030224

Chen, B., Castro, R., Krause, A.: Joint optimization and variable selection of high-dimensional Gaussian processes (2012). arXiv preprint arXiv:1206.6396

Cozad, A., Sahinidis, N.V., Miller, D.C.: Learning surrogate models for simulation-based optimization. AIChE. **60**(6), 2211–2227 (2014)

Feurer, M., Springenberg, J.T., Klein, A., Blum, M., Eggensperger, K., Hutter, F.: Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems, pp. 2962–2970 (2015)

Florea, A.C., Andonie, R.: A dynamic early stopping criterion for random search in SVM hyperparameter optimization. In: IFIP International Conference on Artificial Intelligence Applications and Innovations, pp. 168–180. Springer, Cham (2018)

Frazier, P.I.: Bayesian optimization. In: Recent Advances in Optimization and Modeling of Contemporary Problems, pp. 255–278. INFORMS (2018)

Hutter, F., Kotthoff, L., Vanschoren, J.: Automatic machine learning: methods, systems, challenges. Challenges Mach. Learn. (2019)

Jiang, G., Wang, W.: Error estimation based on variance analysis of k-fold cross-validation. Pattern Recogn. **69**, 94–106 (2017)

Kandasamy, K., Schneider, J., Póczos, B.: High dimensional Bayesian optimisation and bandits via additive models. In: International Conference on Machine Learning, pp. 295–304 (2015, June)

Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA 2.0: automatic model selection and hyperparameter optimization in WEKA. J. Mach. Learn. Res. **18**(1), 826–830 (2017)

Kushner, H.J.: A new method of locating the maximum point of an arbitrary multi-peak curve in the presence of noise. J. Basic Eng. **86**, 97–106 (1964)

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization (2016). arXiv preprint arXiv:1603.06560

Nadeau, C., Bengio, Y.: Inference for the generalization error. In: Advances in neural information processing systems, pp. 307–313 (2000)

Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., De Freitas, N.: Taking the human out of the loop: A review of Bayesian optimization. In: Proceedings of the IEEE, pp. 148–175 (2016)

Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP (2019). arXiv preprint arXiv:1906.02243

Thornton, C., Hutter, F., Hoos, H. H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 847–855. ACM (2013, August)

Wang, Z., Hutter, F., Zoghi, M., Matheson, D., de Freitas, N.: Bayesian optimization in a billion dimensions via random embeddings. J. Artif. Intell. Res. **55**, 361–387 (2016)

Wolpert, D.H., William, G.M.: No Free Lunch Theorems for Search, Vol. 10. Technical Report SFI-TR-95-02-010, Santa Fe Institute (1995)

Wolpert, D.H.: The supervised learning no-free-lunch theorems. In: Soft Computing and Industry, pp. 25–42. Springer, London (2002)

Zhu, D., Linke, N.M., Benedetti, M., Landsman, K.A., Nguyen, N.H., Alderete, C.H., Perdomo-Ortiz, A., Korda, A., Garfoot, A., Brecque, C., Egan, L., Perdomo, O., Monroe, C.: Training of quantum circuits on a hybrid quantum computer (2018). arXiv preprint arXiv:1812.08862