6

Metalearning for Hyperparameter Optimization

Summary. This chapter describes various approaches for the hyperparameter optimization (HPO) and combined algorithm selection and hyperparameter optimization problems (CASH). It starts by presenting some basic hyperparameter optimization methods, including grid search, random search, racing strategies, successive halving and hyperband. Next, it discusses Bayesian optimization, a technique that learns from the observed performance of previously tried hyperparameter settings on the current task. This knowledge is used to build a meta-model (surrogate model) that can be used to predict which unseen configurations may work better on that task. This part includes the description *sequential model-based optimization* (SMBO). This chapter also covers metalearning techniques that extend the previously discussed optimization techniques with the ability to transfer knowledge across tasks. This includes techniques such as *warm-starting* the search, or *transferring previously learned meta-models* that were trained on prior (similar) tasks. A key question here is how to establish how similar prior tasks are to the new task. This can be done on the basis of past experiments, but can also exploit the information gained from recent experiments on the target task. This chapter presents an overview of some recent methods proposed in this area.

6.1 Introduction

Many machine learning algorithms include various hyperparameters that greatly affect their performance (Lavesson and Davidsson, 2006). These hyperparameters can be *numeric*, e.g., the gradient descent learning rate in a neural network, but also *categorical*, e.g., the choice of kernel in an SVM, and some hyperparameters are also *conditional* on the value of other hyperparameters, e.g., when choosing the Gaussian kernel for an SVM, one also needs to choose the kernel width (i.e., gamma).

The effect of hyperparameter configurations on performance can be very complex and greatly dependent on the properties of the dataset at hand. Hence, we want to *learn* – based on prior experimentation – which configurations are likely to work better than others on a particular dataset, or across a group of datasets. Such experience is partly encoded in the *default hyperparameter settings* provided by algorithm designers,

[1] In the first edition, this topic was covered only briefly in Section 2.4, focusing mainly on metalearning approaches used to determine the potentially best parameter settings.

P. Brazdil et al., *Metalearning*, Cognitive Technologies,
https://doi.org/10.1007/978-3-030-67024-5_6

but these will seldom be optimal for a newly given task. Some illustrative examples of this are shown in Chapter 17.

The task of optimizing these hyperparameter settings for a particular task is known as *hyperparameter optimization* (HPO) or *algorithm configuration* (AC).

Algorithm selection (discussed in the previous chapter) can be seen as a special (discrete) form of HPO, simply by encoding the choice of algorithm as an additional hyperparameter (Thornton et al., 2013). That also means that one can optimize the choice of algorithms and their hyperparameters at the same time, also known as *combined algorithm selection and hyperparameter optimization* (CASH). Even more generally, one could define a hyperparameter search space that includes all possible design decisions involved in building a learning model, including the architecture of neural networks or the structure of machine learning pipelines (covered in the next chapter). Since the goal here is to completely automate the process of designing and training machine learning models, this is called *automatic machine learning* (AutoML).

In practice, turning every design decision into a new hyperparameter leads to an explosion of the search space. The larger and more complex the search space becomes, the harder it becomes to optimize it effectively, and the longer it may take until a satisfactory model is found.

In Chapter 8 we discuss general principles that can be followed in the design of the search spaces. This chapter also discusses some methods that can be used in the process of redesigning these spaces on the basis of experience. This is done on the basis of experience with different tasks.

In this chapter, we explore how *metalearning* can allow us to learn from past experimentation and leverage this prior experience to design algorithms and optimize hyperparameters more effectively. Much like a machine learning expert learns through trial and error how to design and optimize models for new tasks, the aim is to learn across tasks to make informed decisions about how to design and tune the best machine learning models.

6.1.1 Overview of this chapter

In Section 6.2, we start by presenting some basic concepts and then cover the basic hyperparameter optimization methods, which do not use metalearning, but form the basis for subsequent methods. These include grid search, random search, racing strategies, evolutionary methods, best-first search, and search with an elimination strategy, which is followed, for instance, in Hyperband.

Next, Section 6.3 focuses on Bayesian optimization, a technique that learns from the observed performance of previously tried hyperparameter settings on the current task to build a meta-model (surrogate model) that can be used to predict which unseen configurations may work better on that task. This section includes the description of the approach known under the name *sequential model-based optimization* (SMBO).

Section 6.4 covers metalearning techniques that extend the previously discussed optimization techniques with the ability to transfer knowledge across tasks. This includes techniques such as *warm-starting* the search for the best hyperparameter with configurations that worked well before, *learning a probability distribution* (a prior) of the best hyperparameter configurations based on previous tasks, or *transferring previously learned meta-models* that were trained on prior (similar) tasks. A key question here is to establish how similar prior tasks are to the new task, since the metaknowledge obtained from very similar tasks is likely much more useful. This can be done on the basis of past

experiments and the accompanying metadata, which include measurable data characteristics (see Chapter 4). Alternatively, this can be based on novel knowledge gained by new experimentation on the new task itself and by observing that the hyperparameter configurations tried on the new task behave similarly as on some previous tasks. Finally, Section 6.5 closes with concluding remarks.

6.2 Basic Hyperparameter Optimization Methods

6.2.1 Basic concepts

Let us first describe the task of hyperparameter optimization formally. Let $M(a, \theta, d_{train})$ represent a trained model M generated by a particular algorithm a with hyperparameter configuration θ on the training portion of the target dataset d, d_{train}. Let $A(M(a, \theta, d_{train}), X_{val})$ represent the application of the trained model to the validation data X_{val}, which returns a set of predictions. The output of $A(..)$ varies as θ is varied. Then, the loss L can be determined using a given loss function $\mathcal{L}$:

$$L = \mathcal{L}(A(M(a, \theta, d_{train}), X_{val}), y_{val}). \tag{6.1}$$

Sometimes it is convenient to use the following short form of the loss function, which only includes the input arguments, namely $\mathcal{L}(a, \theta, d_{train}, d_{val})$. Whenever the algorithm a and the dataset d and train–test splits are fixed, we can simply use $\mathcal{L}(a)$. The aim of CASH is then to determine the values of a and θ from the respective sets of all algorithms A and all possible configurations Θ that minimize the loss.

$$(a*, \theta*) = \underset{\theta \in \Theta,\ a \in A}{\operatorname{argmin}} \mathcal{L}(a, \theta, d_{train}, d_{val}). \tag{6.2}$$

Since algorithm selection can be formulated as hyperparameter selection through a categorical variable representing the choice of algorithm, the pair $(a*, \theta*)$ in the equation above can be substituted by a single hyperparameter $(\theta*)$. The evaluation of various values of θ gives rise to a *observation history* H of losses across hyperparameter settings. It is of the form

$$H_{\theta,L} \equiv (\theta_i, L_i)^n. \tag{6.3}$$

The observation history H can be part of the *metadata* $MetaD$ discussed in Chapter 5. It can be stored for both prior tasks and the current task and leveraged in different ways, as we will discuss in this chapter.

6.2.2 Basic optimization methods

Grid search

One simple method to find θ^* is *grid search*, which searches exhaustively through a predefined set of hyperparameter values of a given algorithm. It requires that the space of alternatives, Θ, is identified and discretized beforehand. This involves identifying the hyperparameters that should be considered. Some have categorical values (e.g., the type of the SVM kernel), while others are real-valued. The latter need to be discretized and the resulting values provided to the system. Figure 6.1 (left) illustrates this. We note that

the choice of hyperparameter values may be conditional on other choices. For instance, if the SVM kernel is *Gaussian*, it makes sense to also tune the *kernel width*.

After the different hyperparameter configurations have been defined, the performance of the given algorithm is evaluated for each configuration. Finally, the configuration with the best performance, $\theta*$, is returned.

Many machine learning libraries use a grid search (or other simple search method) internally and return a configuration that is often better than the default configuration (i.e., the one with default settings). The grid is normally predefined by the designer and includes a relatively small number of values. This is done to limit the search and the corresponding time spent in the search. For instance, the *caret* package (Kuhn, 2008, 2018) runs a predefined grid search with various machine learning algorithms. So we can say that these systems perform a rudimentary form of hyperparameter optimization in an autonomous manner.

Random search

Random search explores the space of configuration randomly. As in the previous case, the space of alternatives Θ needs to be determined beforehand. However, it is not necessary to discretize real-valued hyperparameters. All that is needed is to provide the interval from which the values should be sampled and the type of distribution that should be followed by the sampling process (e.g., uniform).

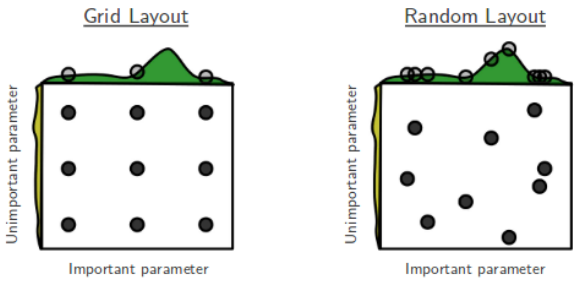

Fig. 6.1: Conceptual difference between random search and grid search. Image taken from Bergstra and Bengio (2012)

Bergstra and Bengio (2012) argue that random search has several advantages over grid search. First, grid search does not scale well with the number of hyperparameters. Adding one hyperparameter can have an exponential influence on the number of points that need to be evaluated. Second, grid search might very well miss the global optimum, as the discretization can remove it from the search space. Finally, when some hyperparameters happen to be irrelevant, i.e. they have little effect on performance, random search often converges to a better configuration. In order to illustrate this, consider Figure 6.1. In this example, we try to optimize two hyperparameters: an important hyperparameter on the horizontal axis, and an unimportant hyperparameter on the vertical

axis. The important hyperparameter has an effect on the performance of the algorithm, while the unimportant one has no such effect. In this example, random search still explores nine different values in the range of the important hyperparameter, whereas grid search only explores three. For this reason, random search has a higher chance of finding a better configuration.

Enhancing search with racing methods

Random search, as well as more sophisticated search methods, can be sped up by using stochastic optimization techniques such as *racing* (Hutter et al., 2011; López-Ibáñez et al., 2011, 2016).

To compute the loss L, one often uses a cross-validation mechanism (see Chapter 3) that evaluates each hyperparameter configuration on multiple train–test splits (folds) to determine the one with the highest average performance across all folds. After evaluating multiple configurations θ_i in parallel on several of these folds, some configurations may show suboptimal performance and hence have a low probability of beating the best configuration identified so far. Consequently, these configurations can be eliminated from further consideration without evaluating them on all folds. In the case of large datasets, one can also use racing of multiple configurations on a single train–test split by letting them predict individual test instances on increasingly more training data until they are statistically unlikely to be optimal.

Various algorithms implement such racing strategies. Originally, it was used to speed up the search for informative feature subsets in classification (Moore and Lee, 1994; John et al., 1994). It was later incorporated into various approaches whose aim is identify the best algorithm configurations. $ROAR$ is an extension of random search that uses a rather aggressive version of *racing* to drop candidates (Hutter et al., 2011). $Irace$ embodies a more conservative racing strategy; it performs a statistical test to determine whether a particular configuration can be dropped. This happens when the test indicates that this configuration has a very low chance of beating the more promising candidates (López-Ibáñez et al., 2011, 2016).

6.2.3 Evolutionary methods

Evolutionary algorithms and population-based methods are also often employed to optimize hyperparameters, because they can optimize many hyperparameters simultaneously, while providing more direction than basic random search. Popular approaches include genetic algorithms (Reif et al., 2012), evolution strategies (Hansen, 2006), tabu search (Gomes et al., 2012), and particle swarm optimization (de Miranda et al., 2012).

One of the most successful techniques is CMA-ES (Hansen, 2006), a population-based method that evaluates a set of randomly sampled configurations, selects the best one, and then iteratively samples new configurations *around* the current best one until it converges. Loshchilov and Hutter (2016) used CMA-ES to optimize the hyperparameters of neural networks.

6.2.4 Heuristic search methods

Heuristic search methods (Russell and Norvig, 2016), such as hill climbing and best-first methods, involve a *heuristic function* that attributes a heuristic value to a given state.

They can be used to optimize hyperparameters by viewing the hyperparameter space as a multi-dimensional space, where each point in this space is associated with a specific heuristic score: the performance of that configuration.

These methods traverse this space by moving to a *neighboring configuration* that has the highest score. That raises the question of which configurations are the neighbors of the current configuration. This could, for example, be all configurations that have one hyperparameter value changed to a different value.

As hill climbing methods can get stuck on local optima, some researchers have used so-called *diversification methods*, such as *restarts* and *random steps*, to avoid getting stuck at local minima. Some examples of works that use similar techniques are *iterated local search* (Lourenço et al., 2003) and ParamILS (Hutter et al., 2009).

Gradient descent methods can be used for adjusting numeric hyperparameters. It assumes that the best hyperparameter setting can be identified by following the gradient of the loss function. Many hyperparameters of machine learning algorithms do not satisfy this assumption, and hence this method can also get stuck at local minima. Maclaurin et al. (2015) compute the gradients of cross-validation performance with respect to all hyperparameters by chaining derivatives.

6.2.5 Hypergradients

When the learning algorithm uses stochastic gradient descent to optimize its model parameters (weights), as in neural networks, it is possible to also optimize certain numeric hyperparameters through gradient descent. Indeed, one can take the derivative of the used loss function $\mathcal{L}$ with respect to certain hyperparameters (e.g., the learning rate), which also appear in the loss function. This is also called a *hypergradient*. Hence, we can use (hyper)gradient descent steps to optimize these hyperparameters bit by bit (Maclaurin et al., 2015; Baydin et al., 2018). Since computing the validation loss requires an inner loop of optimizing the model parameters (weights), this method is quite expensive, but it may still be useful when optimizing many hyperparameters simultaneously with parallel computational resources.

6.2.6 Multi-fidelity techniques

To speed up the search for the best hyperparameter configuration, it is possible to not evaluate every configuration on the entire dataset, which is expensive on larger datasets, but rather to evaluate many configurations on small samples of the training set, and only evaluate the best ones on more training data. Since the performance evaluated on small samples only gives a rough indication of the performance on the full training set, we require an optimization method that can handle noisy probabilistic rewards, such as *multi-armed bandit methods*, which are are discussed in detail in Chapter 8 (Section 8.9). The aim of these methods is to solve the following problem: suppose there are various alternatives to be explored, each with an associated cost and the probability of a certain reward, then which alternatives should be explored to maximize the reward? When dealing with HPO, the cost is processing time and the reward is the performance measured (e.g., accuracy).

Successive halving (SH)

Successive halving (SH) (Jamieson and Talwalkar, 2016; Li et al., 2017) is a multi-armed bandit (MAB) method. It conducts basically best-first search of a given initial number

of alternative configurations. Typically, this number would be quite high. The method differs from ordinary best first in that it includes a *budget* (e.g., computation time) that limits the exploration of each alternative.

It initiates a relatively large pool of candidate configurations that are allocated a low budget. After the budget has been exhausted, the method interrupts and all the nodes (configurations) are ordered by their respective performance (e.g., accuracy). The configurations in the bottom half of this list are eliminated. The search then continues only with the remaining nodes with their budget duplicated. This process continues until only one configuration has been obtained. By running a configuration with increasing budget, implicitly a learning curve is created. Algorithm 6.1 shows the details. While

input : Θ space of hyperparameters of algorithm a
n_{init} - initial number of alternatives
b_{init} - initial budget
output: θ_{best} - algorithm configuration with the best performance
begin
  $\Theta' \leftarrow Sample.uniformly(\Theta, n_{init})$
  $b_c \leftarrow b_{init}$
  $n_c \leftarrow n_{init}$
  while $n_c \geq 2$ **do**
    Run all configurations in Θ' with budget b_c
    $b_c \leftarrow b_c \times 2$
    $n_c \leftarrow n_c \div 2$
    $\Theta' \leftarrow Select.best(\Theta', n_c)$
  end
end

Algorithm 6.1: Successive halving (based on Jamieson and Talwalkar (2016))

many multi-armed bandit methods work by finding a trade-off between exploration and exploitation, SH is a full exploration method.

Regarding the nature of the budget, various alternatives have been suggested (Li et al., 2017): Apart from *runtime*, which was mentioned already, one can consider also the *number of steps*, which obviously affects runtime. It is possible to also consider different settings of some hyperparameter, such as the *number of epochs* in NNs or *the number of trees* in a random forest, that are also correlated with runtime.

Hyperband and extensions to *successive halving* (SH)

One drawback of SH is that the outcome depends on the selection of an initial budget. Also, if the optimal configuration only performs well after a certain budget has been explored, e.g., a certain number of training examples are given, it can happen that it is prematurely eliminated.

Hyperband (Li et al., 2017) is a method that aims to address these issues, by running SH multiple times, each time with a higher initial budget but fewer initial alternatives. The final run of successive halving is in fact an emulation of random search, where a small number of configurations (arms) is run on a single budget, i.e., the maximum

budget. Hyperband is accompanied with several theoretical guarantees: one will never spend more than a log-factor of time more than random search to obtain the same result.

Both SH and Hyperband are very simple yet powerful methods. However, they do not exploit any meta-knowledge obtained on either the current, or other datasets. Some works have attempted to do this. Baker et al. (2017) train a learning curve model on the initial performance data to predict whether the performance of a new alternative would exceed the performance of the incumbent. If the model gives a negative prediction, the alternative is eliminated, resulting in faster execution.

van Rijn and Hutter (2018) propose to sample hyperparameter values that performed well on other datasets more frequently. Falkner et al. (2018) suggested a similar method, but only using the configurations that were obtained from the current dataset. This method is called *Bayesian Optimization with HyperBand (BOHB)* and combines good properties of both paradigms.

6.3 Bayesian Optimization

The term *Bayesian optimization*, first proposed by Mockus et al. (1978), refers to a black-box optimization method that places a prior over the function. The prior models capture a certain belief about the behavior of the function (Brochu et al., 2010). The methods described in the following subsection follow this basic strategy.

6.3.1 Sequential model-based optimization

Model-based search employs functional meta-level models in the search for the best configuration of hyperparameters of a given algorithm. Unlike both grid and random approaches, they take into account the results of previous evaluations. This approach is known under the name *sequential model-based optimization* (SMBO). Algorithm 6.2, which is based on the work of Hutter et al. (2011), summarizes this method.
The aim is to find the optimal configuration θ_{best} that minimizes the loss, as defined in Eq. 6.2. In order to model the prior with our beliefs about the behavior of the function, SMBO employs a model M_L that captures the dependence of loss on hyperparameter settings. Loosely speaking, this surrogate model expresses the probability function $p(y|\theta)$ (Bergstra et al., 2011), where y is the expected performance of configuration θ. Therefore, the surrogate model requires both good predictive power as well as reliable uncertainty estimates. This model is sometimes referred to as a *surrogate model*. The model M_L is used to determine a promising candidate configuration θ_{new} which is used to conduct a test to determine the loss. The value θ_{new} together with the corresponding loss are used to update the model M_L. These two steps are carried out in an iterative fashion.

Acquisition function

In order to generate the next hyperparameter configuration, the method employs a so-called *acquisition function*, a_{M_L}. Several different acquisition functions were proposed in the past (Wistuba, 2018). These include, for instance:

- Probability improvement (PI) (Mockus et al., 1978);
- Expected improvement (EI) (Kushner, 1964; Jones et al., 1998);

input : a - algorithm whose hyperparameters are to be optimized
Θ - space of hyperparameters
d - target dataset
output: θ_{best} - algorithm configuration with the best performance
begin
  $H_{\theta,L} \leftarrow InitialRandomTests(a, \Theta, d)$
  $Initialize\ model\ M_L$
  $(\theta_{best}, L_{best}) \leftarrow SelectBest(H_{\theta,L})$
  while *Not converged* **and** *Time budget not exhausted* **do**
    $\theta_{new} \leftarrow GenConfig(M_L)$
    $L_{new} \leftarrow \mathcal{L}(a, \theta, d_{train}, d_{val})$
    if $L_{new} < L_{best}$ **then**
      $(\theta_{best}, L_{best}) \leftarrow (\theta_{new}, L_{new})$
    end
    $H_{\theta,L} \leftarrow (\theta_{new}, L_{new}) \cup H_{\theta,L}$ *(update history)*
    $M_L \leftarrow update(M_L, H_{\theta,L})$
  end
end

Algorithm 6.2: Sequential model-based optimization

- Entropy search (MacKay, 1992);
- Lower/upper confidence bounds, UCB (Cox and John, 1997; Srinivas et al., 2010).

Here we focus on EI, which is used in various systems, e.g., SMAC (Hutter et al., 2011), Auto-WEKA (Thornton et al., 2013), and Auto-sklearn (Feurer et al., 2015a). The aim is to identify the hyperparameter configuration θ that will most likely have the lowest loss. Good candidate combinations are the ones with both high predicted value (low loss) and high uncertainty. This can be captured by the following equation:

$$I_{L_{min}}(\theta) = max\{L_{min} - L(\theta), 0\}. \tag{6.4}$$

As the value of $L(\theta)$ is not known, Thornton et al. (2013) suggest to calculate the expectation:

$$\mathbb{E}_{L_{min}}[I_{L_{min}}(\theta)] = \int_{-\infty}^{L_{min}} max\{L_{min} - L(\theta), 0\} \cdot p_{M_L}(L|\theta)\ d\theta. \tag{6.5}$$

The form of the acquisition function depends on the underlying models of the loss function. The most common ones are based on *Gaussian processes* (GPs) and *random forests*. More details about both types are given in the following subsections.

Gaussian processes as surrogate models of loss

Gaussian processes (GPs) (Rasmussen and Williams, 2006) have been commonly used by various researchers as surrogate models to model the loss function (e.g., Mockus et al. (1978); Bergstra et al. (2011); Hutter et al. (2011); Snoek et al. (2012); Wistuba (2018)).

The model M_L in Eq. 6.5 is modeled as a posterior GP, given the observation history H. The distribution $p(\boldsymbol{L}|\boldsymbol{\theta})$ is assumed to be distributed as a multivariate Gaussian

$$\mathcal{N}(\boldsymbol{L}|m(\boldsymbol{\theta}), k(\boldsymbol{\theta}, \boldsymbol{\theta}),) \tag{6.6}$$

where $m(\boldsymbol{\theta})$ represents its mean function and $K = k(\boldsymbol{\theta}, \boldsymbol{\theta})$ is a kernel matrix that captures covariance. To simplify matters, the mean $m(\boldsymbol{\theta})$ is often set to 0.

Gaussian process assumption states that $\boldsymbol{L}$ and $\boldsymbol{L*}$ are jointly Gaussian. In other words, Gaussian processes are *closed* under sampling (Bergstra et al., 2011). The predictive posterior distribution $p(\boldsymbol{L*}|\boldsymbol{\theta}, \boldsymbol{L}, \boldsymbol{\theta*})$ can be obtained from the joint distribution.

Bayesian optimization requires that the Gaussian process is frequently updated. Updating it every time from scratch is computationally expensive and dominated by the inversion of the kernel matrix. This operation has a cubic complexity with the number of training instances, i.e., $|H|$. As Wistuba (2018) has shown, the update can be reduced to square complexity.

Random forests as surrogate models of loss

Some researchers preferred *random forest* (RF) models (Breiman, 2001), as they perform well with discrete and high-dimensional data (Thornton et al., 2013). They provide quite accurate predictions and are also fast to train. They have been employed in various systems, including, e.g., SMAC (Thornton et al., 2013) and some predecessor systems (Hutter et al., 2011).

These systems employ a random forest to calculate a predictive mean μ_θ and variance σ_θ on the basis of frequentist estimates for $p(L|\theta)$. So $p_{M_L}(L|\theta)$ is modeled as a Gaussian $\mathcal{N}(\mu_\theta, \sigma_\theta)$. The authors show that the expectation can be computed using a closed-form expression:

$$\mathbb{E}_{L_{min}}[I_{L_{min}}(\theta)] = \sigma_\theta * [u * \Phi(u) + \phi(u)], \tag{6.7}$$

where $u = \frac{(L_{min} - \mu_\theta)}{\sigma_\theta}$, ϕ represents the probability density function and Φ the cumulative density function of a normal distribution.

Note on previous approaches

Previous SMBO methods (see, e.g., Bartz-Beielstein et al. (2005); Hutter et al. (2009)) applied random sampling for the task of finding a new configuration. However, this is not very efficient, particularly in high-dimensional configuration spaces. This led Hutter et al. (2011) to adopt another approach. The method is referred to as *multi-start local search*. Certain locally maximal configurations are gathered and then used in a local search, in which the value of one parameter is varied.

6.3.2 Tree-structured Parzen estimator (TPE)

Instead of using a probabilistic regression model as a surrogate model, one can also use kernel density estimation, leading to a *tree-structured Parzen estimator* (Bergstra et al., 2011). The TPE method defines two probability distributions over the hyperparameter space:

$$p(\theta|y) = \begin{cases} \ell(\theta), & \text{if } y < y* \\ g(\theta) & \text{if } y \geq y* \end{cases} \tag{6.8}$$

where $\ell(\theta)$ defines the density function over all points with a loss lower than the threshold $y*$ (the distribution of "good" configurations), and $g(\theta)$ defines the density function

over all points with a loss higher than the given threshold ("bad" configurations). Assuming that we want to minimize a given measure, we would intuitively want to sample from distribution $\ell(\theta)$. However, there are better options.

The authors suggested to evaluate the configurations that maximize the ratio between $\ell(\theta)$ and $g(\theta)$. Indeed, we want to sample configurations that have a high probability of leading to a low loss and a low probability of leading to a high loss. The configuration that satisfies this cannot be determined analytically. Therefore, this is typically done by sampling a large number of configurations and, for each, establishing the value of $\ell(\theta)/g(\theta)$. Additionally, Bergstra et al. (2011) show that sampling according to this criterion is similar to using the acquisition function *expected improvement*.

Thornton et al. (2013) identified that the tree-structured requirement of configuration spaces makes TPE a very suitable candidate for solving the CASH problem on an extensive set of Weka algorithms. Moreover, it is easy to parallelize, while most Bayesian optimization techniques are sequential in nature.

6.4 Metalearning for Hyperparameter Optimization

In this section, we cover metalearning techniques that extend the previously discussed optimization techniques with the ability to leverage knowledge from previous tasks. This often results in significantly faster anytime performance, since good configurations can be found faster.

6.4.1 Warm-starting: exploiting metaknowledge in initialization

Many optimization techniques start the search with randomly selected points. This can be improved by using metaknowledge to suggest the set of most suitable points to initialize the search.

Reusing best configuration

Bayesian methods suffer from a cold-start problem. That is, when relatively few test results are available, the surrogate model may not deliver good suggestions. This is why some researchers have proposed to reuse the meta-knowledge obtained on other datasets. This technique represents a kind of transfer from past datasets to the current one.

Reif et al. (2012) have done this in conjunction with the search method based on genetic algorithms. These methods usually achieve good results fast, but their performance may not reach the performance of simple grid search. The authors have shown that, by reusing the best configurations identified on *similar datasets*, the process can be speeded up substantially. The similarity of datasets was established with the help of metafeatures (see Chapter 4). A similar approach was used by Gomes et al. (2012).

Feurer et al. (2014, 2015b) have proposed a similar idea for Bayesian optimization. The best configurations identified on past problems were reused to initialize the search on the target datasets. This approach has led again to marked improvements, when compared, for instance, with random initialization. In particular, Feurer et al. (2015b) proposed two distance functions that estimate the distance between two datasets. One of these is based on a *p-norm* between meta-features of these datasets.

The other distance function is based on a *correlation of performance values* of different configurations that have been run on the dataset. The assumption is that configurations that have performed well on datasets similar to the current dataset (i.e., with a low distance between both datasets) will also perform well on the current dataset.[1] The authors have shown that initializing Bayesian optimization with such configurations yields superior performance.

Of course, these initialization procedures are limited to configurations that have already been examined in the past and the existing metadata captures this.

Searching for a globally best configuration

The algorithm of Hutter et al. (2011) identifies several configurations, as the aim is to identify the best configuration for several dataset variants (called *instances*) at the same time. The algorithm includes an *intensify* step, which selects a subset which appears to be the best one for the given set of dataset variants.

A similar approach (Wistuba et al., 2015; Wistuba, 2018) uses an initialization method that generalizes the information found on different datasets. This way, the system can arrive at entirely new configurations that can be used for initialization.

The method uses the concept of *meta-loss*, representing effectively a *meta-level loss* across various datasets $\mathcal{D}$. The aim is to search for a configuration $\theta*$ that minimizes the difference between the global minimum and the best configuration for each dataset.

As this loss is not differentiable, the authors suggest to approximate the minimum function by a differentiable *softmin* function σ. So, the differentiable meta-loss can be expressed as

$$\mathcal{L}(\Theta_I, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \sum_{i=1}^{I} \sigma_{D,i} \, \hat{L}_D(\theta_i), \tag{6.9}$$

where $\sigma_{D,i}$ represents the *softmin* function and $\hat{L}_D(\theta_i)$ the estimate of the loss for the configuration θ_i on dataset D.

The authors then derive an analytic form for the gradient which is then used in the gradient-descent approach. The procedure starts with the best set of configurations for each dataset $\theta_1, \ldots, \theta_I$ to initialize the process. In each cycle this solution is iteratively improved by selecting one of the configurations at a time. An enhanced method also uses dataset similarity, which captures the effect of configurations belonging to similar datasets.

Wistuba (2018) has conducted experiments which showed that this initialization method leads to a lower normalized loss than the method described earlier in Subsection 6.4.1 based on the work of Reif et al. (2012) and Feurer et al. (2014).

Ranking configurations

We note that grid search, discussed in Section 6.2.2, does not really specify the order in which the alternatives should be tested. However, it is well known that some configurations may be better than others. Search methods that exploit metaknowledge from other datasets exploit this.

[1] Chapter 4 provides more details on various ways of establishing similarity between datasets.

All that is required is to define the configuration space beforehand, that is, all possible configurations of interest. This leads to a set of finite alternatives that can be used. Then, it is necessary to populate this space with test results to obtain metaknowledge $MetaD$. The ranking approach discussed in Chapter 2 can be used to construct a ranking of the pre-defined alternatives which can be used in the search.

One early work in this area was presented by Soares et al. (2004). The aim of the authors was to suggest a value of one parameter of SVM, the width of the Gaussian kernel (σ), on the target dataset. The authors have shown that the methodology could be used to select a good configuration accompanied by a relatively low error. Although this approach could exploit metaknowledge acquired on prior datasets, it did not exploit the metaknowledge acquired on the current dataset. This shortcoming was corrected in the system AT* (Abdulrahman et al., 2018), discussed in Chapter 5 (Section 5.8). The result of each new test carried out on the target dataset affects the selection of the subsequent tests.

Some experiments carried out with this approach are described in Chapter 7 (Section 7.4). In one of the experiments reported there, which was carried out by Cachada (2017), the given portfolio included various workflows with different hyperparameter configurations. This approach was able to identify a competitive workflow and compete well with Auto-WEKA.

6.4.2 Exploiting metaknowledge in Bayesian optimization

Recent work tries to reuse metaknowledge from past experiments in the search for the best algorithm configuration. This process can be seen as a kind of *transfer* from past datasets to the target dataset. The aim of this section is to describe some of the approaches that have been taken.

Surrogate collaborative tuning (SCoT/MKL)

Bardenet et al. (2013) were the first to propose that a surrogate model be learned over observations from different datasets. Hence this method undertakes *multi-kernel learning*. They have used a *ranking model* instead of a regression model. This choice was motivated by the fact that, when a given algorithm is applied to different datasets, it tends to incur rather different losses. The relative model avoids this problem. A ranking of hyperparameter configurations per dataset was learned with *SVMRank* with an *RBF kernel*. As the ranking model does not provide the needed uncertainty estimations, the authors fit a Gaussian process to the output of the ranking model.

Gaussian process with multi-kernel learning (MKL-GP)

Yogatama and Mann (2014) also proposed an algorithm for automatic hyperparameter tuning that can generalize across datasets. Their method is an instance of sequential model-based optimization (SMBO) that transfers information by constructing a common response surface for all datasets, similar to Bardenet et al. (2013). They did not use a *ranking model*, as Bardenet et al. (2013), but overcame the problem of losses of rather different magnitude by normalizing the data. After this, they could just use a regression model. The authors use a linear combination of two kernels:

- A squared exponential kernel with automatic relevance determination (k_{SE-ARD}) for points belonging to the target dataset,

- A nearest-neighbor kernel (k_{NN}) for modeling similarities between datasets.

The time complexity of reconstructing the response surface at every SMBO iteration is linear in the number of trials, allowing the method to scale up to many more datasets.

Multi-task and multi-fidelity Bayesian optimization

Swersky et al. (2013) employed a *multi-task Gaussian process* (Bonilla et al., 2008) as a surrogate model. This model does not only model the performance of the algorithm across various configurations, but also the performance of the various configurations across other auxiliary tasks. The hope is that, if there are auxiliary tasks where similar configurations appear to have a similar performance as on task t, they can be used to speed up the process of learning. This is useful particularly when the experiments with the auxiliary task are faster to execute than the experiments on task t. This can happen, for example, if the auxiliary task is simpler, that is, if it includes fewer observations or attributes. This way it plays a similar role to *landmarkers* discussed in Chapter 4. The authors used the acquisition function *expected improvement per second* to emphasize the need for fast experimentation.

Klein et al. (2017) took this notion one step further and proposed a *multi-task Bayesian optimization* method, based on the notion of *multi-fidelity*. They argue that configurations perform similarly on subsets of the given dataset, and employ a GP to model the performance of the algorithm across various configurations and across various dataset sizes.

Ensemble of individual surrogate models (SGPT)

As we have pointed out earlier, the aim of recent work on SMBO is to also exploit metadata concerning the effects of different hyperparameter configurations on different datasets in the search for the best configuration on the target dataset. However, Gaussian processes do not scale up well with growing metadata (Wistuba et al., 2018). This is due to the fact that the method involves an inversion of a kernel matrix, which represents a bottleneck.

To overcome this difficulty, Wistuba et al. (2016), Wistuba (2018), and Wistuba et al. (2018) have proposed to learn individual surrogate models on a set of different datasets. The target dataset is included in this set. Different surrogate models are then combined into a joint model using an ensembling technique.

The final surrogate is represented by a weighted sum of the individual surrogate models. In Wistuba et al. (2018) the authors call this approach the *scalable GP transfer surrogate framework* (SGPT).[2] Three different variants have been defined, both for $SGPT$ and the corresponding TST. The variant SGTP-R, which uses pairwise hyperparameter performance descriptors, obtained the best experimental results.

Transfer acquisition function (TAF)

The proposal discussed in the previous section suffers from some shortcomings. One major one is that the weights of different components do not change as tests proceed.

[2] In Wistuba (2018) (Chapter 7), this type of solution is referred to as the *two-stage transfer surrogate model, TST.*

This is counter-intuitive, because as tests proceed on the target dataset, the metadata on this dataset is more informative.

These observations led the authors (Wistuba et al., 2016; Wistuba, 2018; Wistuba et al., 2018) to propose another variant of the surrogate framework that exploits *transfer acquisition function* (TAF).[3]

The transfer acquisition function is defined as a weighted average of two components. The first one represents the expected improvement on the new target dataset. It tends to be rather unreliable in the early trials. The second component captures the predicted improvement on all other datasets used in previous experiments. This second component provided by the metadata is followed in the early trials. This favors hyperparameter configurations that have led to good performance on different datasets.

As time proceeds and as more information about the new target dataset has been gathered, the prediction obtained by the first component becomes more reliable, and consequently, the metadata starts to play a minor role.

Similarly, as with SGPT, the authors have defined three different variants. Here again, the variant TAF-R, which uses pairwise descriptors, has obtained better experimental results than the other two.

The authors have compared their system against others on two problems. One of them involved running 19 different Weka classifiers on 59 datasets with 21,871 hyperparameter configurations. TAF-R obtained competitive results when compared with the other approaches.

Focusing on high-performance regions with QRF

Eggensperger et al. (2018) did not use Gaussian processes, but rather a regression algorithm as a surrogate model. The regression algorithm used was *quantile regression forest (QRF)* (Meinshausen, 2006) based on quantile regression (Koenker, 2005; Takeuchi et al., 2006).

Certain datasets were used as training data to generate the model. This involved results of various hyperparameter configurations obtained on the training datasets.

This method thus permits to focus on the high-performance regions of the parameter configuration space, in a way somewhat similar to *irace* (López-Ibáñez et al., 2011).

6.4.3 Adaptive dataset similarity

Chapter 5 (Section 5.8) describes various variants of active testing (AT*), where the similarity is determined dynamically by combining the information gathered on both the new and past datasets. Some of the enhanced variants have led to significant improvements in performance on a combined algorithm selection and hyperparameter optimization (CASH) problem. It is foreseeable that this approach could compete with other existing approaches discussed in this section.

[3] In another publication (Wistuba, 2018) (Chapter 8), a similar system is discussed and referred to as *adaptive transfer hyperparameter learning* (AHT).

6.5 Concluding Remarks

Relation to experiment design, exploration, and exploitation

The topics discussed in this chapter build on the work of many other areas, including, for instance, the area of *experiment design* (Robbins, 1952). Another area is the area of *reinforcement learning*, which has introduced the concepts of *exploration* and exploitation. The exploration phase can be equated to the process of conducting tests involving given algorithms and datasets, that is, the process of gathering metadata. As was shown in Chapters 2, 5 and 6, the metadata that was gathered is then used to construct a metamodel. So, the exploitation phase can be compared to the process of applying a given meta-level model to the target dataset so as to identify the best possible algorithm (or workflow).

The research in the area of multi-armed bandits (MABs) is in many ways related to the problems addressed in this chapter. The process of gathering test results can be compared to the process of gathering knowledge about different "arms" in multi-armed bandit (MAB) problems. The aim is to find a good compromise between exploration (i.e., examining different arms) and exploitation (using the best arm(s) for the target problem).

Summary

In this chapter we have briefly reviewed various AutoML methods that successfully address the hyperparameter optimization (HPO) problem, as well as the combined algorithm selection and hyperparameter optimization (CASH) problem. We have started the exposition with simple uninformed search methods (grid search and random search) and then continued with more intelligent approaches, such as hill-climbing, best-first methods, successive halving, Hyperband and Bayesian optimization.

In general, the simple methods do not make use of metadata across tasks, but utilize knowledge that was acquired during the search process.

It has been shown that these techniques can be improved with metalearning. Section 6.4 provides an overview of the techniques that permit to incorporate metaknowledge obtained from previous tasks, often dramatically speeding up the search for the best models for new tasks.

Discussion

One obvious question that arises is whether we have not replaced the original problem of algorithm selection for a specific dataset with a meta-level algorithm selection problem. This is due to the fact that various meta-level approaches exist. Several of those were discussed in this chapter.

However, we note that most of the hyperparameter search and optimization techniques enable users to automatically explore multiple algorithms and hyperparameter configurations. Even when these configurations are configured suboptimally, they enable data scientists to make better informed decisions regarding which configuration to use for their problem.

It is foreseeable that new comparative studies will provide a better insight in the future and enable us to identify fewer methods as generally useful and others that are useful in certain specific circumstances.

Chapter 7 continues with the topic of this chapter. The focus there is on how to construct solutions that include various algorithms, each with its own hyperparameters, usually referred to as *workflows* or *pipelines*.

References

Abdulrahman, S., Brazdil, P., van Rijn, J. N., and Vanschoren, J. (2018). Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine Learning*, 107(1):79–108.

Baker, B., Gupta, O., Raskar, R., and Naik, N. (2017). Accelerating neural architecture search using performance prediction. In *Proc. of ICLR 2017*.

Bardenet, R., Brendel, M., Kégl, B., and Sebag, M. (2013). Collaborative hyperparameter tuning. In *Proceedings of the 30th International Conference on Machine Learning*, ICML'13, pages 199–207. JMLR.org.

Bartz-Beielstein, T., Lasarczyk, C., and Preuss, M. (2005). Sequential parameter optimization. In *Proceedings of CEC-05*, page 773–780. IEEE Press.

Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M., and Wood, F. (2018). Online learning rate adaptation with hypergradient descent. In *Sixth International Conference on Learning Representations (ICLR), Vancouver, Canada, April 30 – May 3, 2018*.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyperparameter optimization. In *Advances in Neural Information Processing Systems 24*, NIPS'11, pages 2546–2554.

Bonilla, E. V., Chai, K. M., and Williams, C. (2008). Multi-task Gaussian process prediction. In *Advances in Neural Information Processing Systems 21*, NIPS'08, pages 153–160.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599.

Cachada, M. (2017). Ranking classification algorithms on past performance. Master's thesis, Faculty of Economics, University of Porto.

Cox, D. and John, S. (1997). SDO: A statistical method for global optimization. In *Multidisciplinary Design Optimization: State-of-the-Art*, page 315–329.

de Miranda, P. B., Prudêncio, R. B., de Carvalho, A. C. P., and Soares, C. (2012). Combining a multi-objective optimization approach with meta-learning for SVM parameter selection. *Systems, Man, and Cybernetics (SMC)*, page 2909–2914.

Eggensperger, K., Lindauer, M., Hoos, H., Hutter, F., and Leyton-Brown, K. (2018). Efficient benchmarking of algorithm configuration procedures via model-based surrogates. *Special Issue on Metalearning and Algorithm Selection, Machine Learning*, 107(1).

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *ICML'18*, pages 1437–1446. JMLR.org.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015a). Efficient and robust automated machine learning. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, NIPS'15, pages 2962–2970. Curran Associates, Inc.

Feurer, M., Springenberg, J., and Hutter, F. (2015b). Initializing Bayesian hyperparameter optimization via meta-learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1128–1135.

Feurer, M., Springenberg, J. T., and Hutter, F. (2014). Using meta-learning to initialize Bayesian optimization of hyperparameters. In *ECAI Workshop on Metalearning and Algorithm Selection (MetaSel)*, pages 3–10.

Gomes, T. A., Prudêncio, R. B., Soares, C., Rossi, A. L., and Carvalho, A. (2012). Meta-learning for evolutionary parameter optimization of classifiers. *Neurocomputing*, 75(1):3–13.

Hansen, N. (2006). The CMA evolution strategy: a comparing review. In *Towards a New Evolutionary Computation*, pages 75–102. Springer.

Hutter, F., Hoos, H., Leyton-Brown, K., and Stützle, T. (2009). ParamILS: an automatic algorithm configuration framework. *JAIR*, 36:267–306.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. *LION*, 5:507–523.

Jamieson, K. and Talwalkar, A. (2016). Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pages 240–248.

John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant feature and the subset selection problem. In Cohen, W. and Hirsch, H., editors, *Machine Learning Proceedings 1994: Proceedings of the Eighth International Conference*, pages 121–129. Morgan Kaufmann.

Jones, D., Schonlau, M., and Welch, W. (1998). Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017). Fast Bayesian optimization of machine learning hyperparameters on large datasets. In *Proc. of AISTATS 2017*.

Koenker, R. (2005). *Quantile regression*. Cambridge University Press.

Kuhn, M. (2008). Building predictive models in R using the caret package. *J. of Statistical Software*, 28(5).

Kuhn, M. (2018). Package caret: Classification and regression training.

Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106.

Lavesson, N. and Davidsson, P. (2006). Quantifying the impact of learning algorithm parameter tuning. In *AAAI*, volume 6, pages 395–400.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: Bandit-Based Configuration Evaluation for Hyperparameter Optimization. In *Proc. of ICLR 2017*.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

López-Ibáñez, M., Dubois-Lacoste, J., Stüzle, T., and Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Technical report, IRIDIA, Université libre de Bruxelles.

Loshchilov, I. and Hutter, F. (2016). CMA-ES for hyperparameter optimization of deep neural networks. In *Proc. of ICLR 2016 Workshop*.

Lourenço, H., Martin, O., and Stützle, T. (2003). Iterated local search. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers.

MacKay, D. (1992). Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604.

Maclaurin, D., Duvenaud, D., and Adams, R. P. (2015). Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *ICML'15*, pages 2113–2122.

Meinshausen, N. (2006). Quantile regression forests. *Journal of Machine Learning Research*, 7:983–999.

Mockus, J., Tiešis, V., and Žilinskas, A. (1978). The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2:117–129.

Moore, A. W. and Lee, M. S. (1994). Efficient algorithms for minimizing cross-validation error. In Cohen, W. and Hirsch, H., editors, *Machine Learning Proceedings 1994: Proceedings of the Eighth International Conference*, pages 190–198. Morgan Kaufmann.

Rasmussen, C. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

Reif, M., Shafait, F., and Dengel, A. (2012). Meta-learning for evolutionary parameter optimization of classifiers. *Machine learning*, 87(3):357–380.

Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 55:527–535.

Russell, S. J. and Norvig, P. (2016). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, NIPS'12, page 2951–2959.

Soares, C., Brazdil, P., and Kuba, P. (2004). A meta-learning method to select the kernel width in support vector regression. *Machine Learning*, 54:195–209.

Srinivas, N., Krause, A., Seeger, M., and Kakade, S. M. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning*, ICML'10, page 1015–1022. Omnipress.

Swersky, K., Snoek, J., and Adams, R. P. (2013). Multi-task Bayesian optimization. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, NIPS'13, pages 2004–2012. Curran Associates, Inc.

Takeuchi, I., Le, Q., Sears, T., and Smola, A. (2006). Nonparametric quantile estimation. *Journal of Machine Learning Research*, 7:1231–1264.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). 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*, pages 847–855. ACM.

van Rijn, J. N. and Hutter, F. (2018). Hyperparameter importance across datasets. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM.

Wistuba, M. (2018). *Automated Machine Learning: Bayesian Optimization, Meta-Learning & Applications*. PhD thesis, University of Hildesheim, Germany.

Wistuba, M., N. Schilling, L., and Schmidt-Thieme (2018). Scalable Gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1):43–78.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. (2015). Learning hyperparameter optimization initializations. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015*, pages 1–10.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. (2016). Two-stage transfer surrogate model for automatic hyperparameter optimization. In *Machine Learning and Knowl-*

edge Discovery in Databases - European Conference, ECML-PKDD 2016, Proceedings, pages 199–214.

Yogatama, D. and Mann, G. (2014). Efficient transfer learning method for automatic hyperparameter tuning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.