Using Meta-Learning to Initialize Bayesian Optimization of Hyperparameters

Matthias Feurer and **Jost Tobias Springenberg** and **Frank Hutter**[1]

Abstract. Model selection and hyperparameter optimization is crucial in applying machine learning to a novel dataset. Recently, a subcommunity of machine learning has focused on solving this problem with Sequential Model-based Bayesian Optimization (SMBO), demonstrating substantial successes in many applications. However, for expensive algorithms the computational overhead of hyperparameter optimization can still be prohibitive. In this paper we explore the possibility of speeding up SMBO by transferring knowledge from previous optimization runs on similar datasets; specifically, we propose to initialize SMBO with a small number of configurations suggested by a metalearning procedure. The resulting simple MI-SMBO technique can be trivially applied to any SMBO method, allowing us to perform experiments on two quite different SMBO methods with complementary strengths applied to optimize two machine learning frameworks on 57 classification datasets. We find that our initialization procedure mildly improves the state of the art in low-dimensional hyperparameter optimization and substantially improves the state of the art in the more complex problem of combined model selection and hyperparameter optimization.

1 Introduction

Hyperparameter optimization is a crucial step in the process of applying machine learning algorithms in practice. Depending on the training time of the algorithm at hand, finding good hyperparameter settings manually is often a time-consuming, tedious process requiring many ad-hoc choices by the practitioner. As a result, much recent work in machine learning has focused on the development of better hyperparameter optimization methods [14, 3, 29, 4, 26, 21, 10, 33, 5].

Recently, Sequential Model-based Bayesian Optimization (SMBO) [16, 7, 14] has emerged as a successful hyperparameter optimization method in machine learning. It has been conclusively shown to yield better performance than both grid and random search [3, 29, 33, 9]. In practical applications, it has unveiled new state-of-the-art performance on the challenging CIFAR-10 object recognition benchmark by tuning the hyperparameters of a deep neural network [29] and was repeatedly found to match or outperform human practitioneers in tuning complex neural network models [3] as well as computer vision architectures with up to 238 hyperparameters [5]. It has also enabled AutoWEKA [33], which performs combined algorithm selection and hyperparameter optimization in the space of algorithms defined by the WEKA package [11]. We describe SMBO in detail in Section 2.

However, SMBO is defined as a generic function optimization framework, and—like any other generic optimization method—it requires a substantial number of function evaluations to detect high-performance regions when started on a new optimization problem. The resulting overhead is computationally infeasible for expensive-to-evaluate machine learning algorithms. To combat this problem, metalearning has been applied in two ways. Firstly, a method similar to SMBO that reasons across datasets has been developed[19]. Secondly, metalearning was used to initialize hyperparameter optimization methods with hyperparameter configurations that previously yielded good performance on similar datasets [26, 21, 10]. We follow this latter approach to yield a simple and effective initialization procedure that applies generically to all variants of SMBO; we refer to the resulting SMBO approach with Meta-learning-based Initialization as MI-SMBO. In contrast to another recent line of work on collaborative SMBO methods [1, 35, 32], MI-SMBO does not require any adaptation of the underlying SMBO procedure. It is hence easy to implement and can be readily applied to several off-the-shelf hyperparameter optimizers.

Using a comprehensive suite of 57 datasets and 46 metafeatures, we empirically studied the impact of our meta-learning-based initialization procedure to two SMBO variants with complementary strengths. First, we applied it to optimize the 2 hyperparameters C and γ of a support vector machine (SVM), which control the SVM's learning process. Here, our MI-Spearmint variant of the Gaussian-process-based SMBO method Spearmint [29] (a state-of-the-art approach for low-dimensional hyperparameter optimization) yielded mild improvements: in particular, MI-Spearmint performed better than Spearmint initially, but after 50 function evaluations the differences levelled off. Second, we applied our method to optimize 10 hyperparameters describing a choice between three classifiers from the prominent Scikit-Learn package [22] and their hyperparameters. Here, our MI-SMAC variant of the random-forest-based SMBO method SMAC [14] (a state-of-the-art approach for high-dimensional hyperparameter optimization) yielded substantial improvements, significantly outperforming the previous state of the art for this problem. To enable other researchers to reproduce and build upon our results, we will provide our software on the first author's github page.[2]

2 Foundations

Before we describe our MI-SMBO approach in detail we formally describe hyperparameter optimization and SMBO.

2.1 Hyperparameter Optimization

Let $\theta_1, \dots, \theta_n$ denote the hyperparameters of a machine learning algorithm, and let $\Theta_1, \dots, \Theta_n$ denote their respective domains. The al-

[1] University of Freiburg, Germany, {feurerm,springj,fh}@cs.uni-freiburg.de

[2] https://github.com/mfeurer

Algorithm 1: Generic Sequential Model-based Optimization. SMBO(f^D, T, $\boldsymbol{\Theta}$, $\boldsymbol{\theta}_{1:t}$)

```
Input: Target function $f^D$; limit $T$; hyperparameter space $\boldsymbol{\Theta}$;
       initial design $\boldsymbol{\theta}_{1:t} = \langle \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_t \rangle$
Result: Best hyperparameter configuration $\boldsymbol{\theta}^*$ found
for $i \leftarrow 1$ to $t$ do $y_i \leftarrow$ Evaluate $f^D(\boldsymbol{\theta}_i)$
for $j \leftarrow t+1$ to $T$ do
    $\mathcal{M} \leftarrow$ fit model on performance data $\langle \boldsymbol{\theta}_i, y_i \rangle_{i=1}^{j-1}$
    Select $\boldsymbol{\theta}_j \in \arg\max_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} a(\boldsymbol{\theta}, \mathcal{M})$
    $y_j \leftarrow$ Evaluate $f^D(\boldsymbol{\theta}_j)$
return $\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}_j \in \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T\}} y_j$
```

gorithm's hyperparameter space is then defined as $\boldsymbol{\Theta} = \Theta_1 \times \cdots \times \Theta_n$. When trained with $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ on data $\mathcal{D}_{\text{train}}$, we denote the algorithm's validation error on data $\mathcal{D}_{\text{valid}}$ as $\mathcal{V}(\boldsymbol{\theta}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}})$. Using k-fold cross-validation, the hyperparameter optimization problem for a given dataset D then is to minimize:

$$f^D(\boldsymbol{\theta}) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{V}(\boldsymbol{\theta}, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)}). \tag{1}$$

Hyperparameters θ_i can be numerical (real or integer, as, e.g., the strength of a regularizer) or categorical (unordered, with finite domain, as, e.g., the choice between different kernels). Furthermore, there can be *conditional* hyperparameters, which are only active if another hyperparameter takes a certain value; for example, the hyperparameter "number of principal components" only needs to be instantiated when the hyperparameter "preprocessing method" is set to PCA.

The space of hyperparameter configurations can be searched either manually or automatically. Since manual search is tedious, time-consuming, and often not sample-efficient, much recent work in machine learning has focused on the development of automated methods. Grid search, the most frequently used automated method, does not scale to high-dimensional hyperparameter spaces, and has been shown to be outperformed by random search in the presence of low effective dimensionality [3]. Various types of direct search have been applied to the problem as well, such as genetic algorithms [26], particle swarm optimization [21], and tabu search [10]. Most recently, several SMBO algorithms have been presented for hyperparameter optimization [14, 3, 29]; we discuss these in the following section.

2.2 Sequential Model-based Bayesian Optimization

Sequential Model-based Bayesian Optimization (SMBO) [16, 7, 14] is a powerful method for the global optimization of expensive blackbox functions f. As described in Algorithm 1, SMBO starts by querying the function f at the t values in an initial design and recording the resulting $\langle$input, output$\rangle$ pairs $\langle \boldsymbol{\theta}_i, f(\boldsymbol{\theta}_i) \rangle_{i=1}^{t}$. Afterwards, it iterates the following three phases: (1) fit a probabilistic model $\mathcal{M}$ to the $\langle$input, output$\rangle$ pairs collected so far; (2) use the probabilistic model $\mathcal{M}$ to select a promising input $\boldsymbol{\theta}$ to evaluate next by quantifying the desirability of obtaining the function value at arbitrary inputs $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ through a so-called acquisition function $a(\boldsymbol{\theta}, M)$; (3) evaluate the function at the new input $\boldsymbol{\theta}$.

The SMBO framework offers several degrees of freedom to be instantiated, including the procedure's initialization, the type of probabilistic model to use, and the acquisition function. We discuss three prominent hyperparameter optimization methods in terms of these components: SMAC [14], Spearmint [29], and TPE [3].

The role of the acquisition function $a(\boldsymbol{\theta}, \mathcal{M})$ is to trade off *exploration* in hyperparameter regions where the model $\mathcal{M}$ is uncertain with *exploitation* in regions with low predicted validation error. The most commonly-used acquisition function is the Expected positive improvement (EI) over the best input found so far [16]:

$$a_{EI}(\boldsymbol{\theta}, \mathcal{M}) = \int_{-\infty}^{\infty} \max(y^* - y, 0) p_{\mathcal{M}}(y|\boldsymbol{\theta}) d_y. \tag{2}$$

Other prominent acquisition functions are Upper Confidence Bounds [31] and Entropy Search [12]. All of SMAC, Spearmint, and TPE use the expected improvement criterion.

Several different model types can be used inside of SMBO. The most popular choice, used for example by Spearmint, are Gaussian processes [24] because they provide good predictions in low-dimensional numerical input spaces and allow the computation of the posterior Gaussian process model in closed form. The other popular model type are tree-based approaches, which are particularly well suited to handle high-dimensional input spaces and partially categorical input spaces. In particular, SMAC uses random forests [6], modified to yield an uncertainty estimate [15]. Random forests are particularly well suited for SMBO in high dimensions due to their robustness and automated feature selection. Another tree-based approach, applied by TPE, is to use the Tree Parzen Estimator [3] in order to construct a density estimate over good and bad instantiations of each hyperparameter: instead of predicting $p(y \mid \boldsymbol{\theta})$ directly, TPE constructs estimates of $p(\boldsymbol{\theta} \mid y \geq q)$ and $p(\boldsymbol{\theta} \mid y < q)$ for a given quantile q. Expected improvement can then be shown to be proportional to $\frac{p(\boldsymbol{\theta}|y>q)}{p(\boldsymbol{\theta}|y<q)}$.

The final degree of freedom in SMBO is its initialization. To date, this component has not received much attention, and is instantiated in a fairly ad-hoc manner: Spearmint evaluates f at two pre-defined input points, SMAC evaluates it at a pre-defined 'default' input, and TPE evaluates 20 points selected at random according to a user-defined prior distribution. It is this initialization procedure that our MI-SMBO approach aims to improve.

An empirical evaluation of Bayesian hyperparameter optimization methods in the framework of the hyperparameter optimization library (HPOlib [9]) has shown Spearmint to yield the best results for low-dimensional continuous hyperparameter optimization problems, and SMAC to perform best for high-dimensional hyperparameter optimization problems and problems with categorical and/or conditional hyperparameters.

3 Initializing SMBO With Configurations Suggested by Meta-Learning

Building on the foundations from Section 2 we will now describe our proposed MI-SMBO method that uses meta-learning to initialize SMBO.

The core idea behind MI-SMBO is to follow the common practice machine learning experts employ when applying a known machine learning method to a new dataset D_{new}: they first study D_{new}, relating it to datasets they previously experienced. When manually optimizing hyperparameters for D_{new}, they would begin the search with hyperparameter configurations that were optimal for the most similar previous datasets. Our MI-SMBO method automates this approach and uses it to initialize an SMBO method. In addition to eliminating the need for manual exploration, this can lead to better results as more time can be spend on improving known configurations. We note that in settings where only a few performance evaluations of the algorithm to be optimized are feasible using additional information

Algorithm 2: SMBO with Meta-Learning Initialization.
MI-SMBO(D_{new}, $f^{D_{\text{new}}}$, $D_{1:N}$, $\hat{\boldsymbol{\theta}}^{1:N}$, d, t, T, $\boldsymbol{\Theta}$)

```
Input: new dataset $D_{\text{new}}$; target function $f^{D_{\text{new}}}$; training datasets
    $D_{1:N} = (D_1, \dots, D_N)$; best configurations for training
    datasets, $\hat{\boldsymbol{\theta}}^{1:N} = \hat{\boldsymbol{\theta}}^1, \dots, \hat{\boldsymbol{\theta}}^N$; distance metric $d$; number
    of configurations to include in initial design, $t$; limit $T$;
    hyperparameter space $\boldsymbol{\Theta}$
Result: Best hyperparameter configuration $\boldsymbol{\theta}^*$ found
Sort dataset indices $\pi(1), \dots \pi(N)$ by increasing distance to
    $D_{\text{new}}$, i.e.: $(\pi(i) \leq \pi(j)) \Leftrightarrow (d(D_{\text{new}}, D_i) \leq d(D_{\text{new}}, D_j))$
for $i \leftarrow 1$ to $t$ do $\boldsymbol{\theta}_i \leftarrow \hat{\boldsymbol{\theta}}^{\pi(i)}$
$\boldsymbol{\theta}^* \leftarrow$ SMBO($f^D$, $T$, $\boldsymbol{\Theta}$, $\boldsymbol{\theta}_{1:t}$)
return $\boldsymbol{\theta}^*$
```

from other datasets might be the only possibility to achieve reasonable performance.

Formally, MI-SMBO can be stated as follows. Let $\hat{\boldsymbol{\theta}}^1, \dots, \hat{\boldsymbol{\theta}}^N$ denote the best known hyperparameters for the previously encountered datasets $D_1, \dots, D_N$, respectively. These may originate from an arbitrary source, e.g., a manual search or the application of an SMBO method during an offline training phase. Further, let D_{new} denote a new dataset, let d denote a distance metric between datasets, and let π denote a permutation of $(1, \dots, N)$ sorted by increasing distance between D_{new} and D_i (i.e., $(\pi(i) \leq \pi(j)) \Leftrightarrow (d(D_{\text{new}}, D_i) \leq d(D_{\text{new}}, D_j))$). Then, MI-SMBO with an initial design of t configurations initializes SMBO with configurations $\hat{\boldsymbol{\theta}}^{\pi(1)}, \dots, \hat{\boldsymbol{\theta}}^{\pi(t)}$. Algorithm 2 provides pseudocode for the approach.

We would like to highlight the fact that MI-SMBO is agnostic of the SMBO algorithm used, as long as the algorithm's implementation accepts an initial design as input or can be warmstarted with a given list of performance data $\langle \boldsymbol{\theta}_i, y_i \rangle_{i=1}^t$. All of SMAC, TPE, and Spearmint fulfill these criteria. We would also like to highlight that SMBO is a particularly good match for initialization with meta-learning: in contrast to existing approaches that initialize other types of hyperparameter optimization algorithms via metalearning [10, 21, 26], SMBO can make effective use of all performance data it receives as input (and does not have to adapt population sizes or alike to the size of the initial design).

To implement MI-SMBO, we still need to define a distance metric between datasets. This is a well studied problem which was, to our knowledge, first discussed by Soares and Brazdil [30]. For the purpose of this work we assume that each dataset D_i can be described through a set of F metafeatures $\boldsymbol{m}^i = (m_1^i, \dots, m_F^i)$. We discuss the metafeatures we used in Section 3.1. In practice, we precompute the metafeatures for all training datasets $D_1, \dots, D_N$ along with the best configurations $(\hat{\boldsymbol{\theta}}^1, \dots, \hat{\boldsymbol{\theta}}^N)$. We then measure the distance between a new dataset D_{new} and a previous dataset D_i as the norm of the distance between their metafeatures:

$$d(D_{\text{new}}, D_j) = \|\boldsymbol{m}^{\text{new}} - \boldsymbol{m}^j\|. \tag{3}$$

3.1 Implemented Metafeatures

To evaluate our approach in a realistic setting we implemented the 46 metafeatures from the literature listed in Table 1. Based on their types and underlying assumptions, these metafeatures can be divided into at least five groups:

- *Simple metafeatures*, such as the number of features, patterns or classes, describe the basic dataset structure [20, 17, 1, 35].
- *PCA metafeatures* [1] perform principal component analysis and compute various statistics of the principal components.
- The *information-theoretic metafeature* measures the class entropy in the data [20].
- *Statistical metafeatures* [20] attempt to characterize the data distribution via descriptive statistics such as the kurtosis or the dispersion of the label distribution.
- *Landmarking metafeatures* [23, 2] are computed by running fast machine learning algorithms to characterize properties of the dataset. Since they characterize which simple approaches work well (and, in combination, also which simple approaches work better than others) they are intuitively very relevant for determining which hyperparameter configuration of a given algorithm would perform well.

Table 1. List of implemented metafeatures

Simple metafeatures:
number of patterns
log number of patterns
number of classes
number of features
log number of features
number of patterns with missing values
percentage of patterns with missing values
number of features with missing values
percentage of features with missing values
number of missing values
percentage of missing values
number of numeric features
number of categorical features
ratio numerical to categorical
ratio categorical to numerical
dataset dimensionality
log dataset dimensionality
inverse dataset dimensionality
log inverse dataset dimensionality
class probability min
class probability max
class probability mean
class probability std

Information-theoretic metafeature:
class entropy

Statistical metafeatures:
min # categorical values
max # categorical values
mean # categorical values
std # categorical values
total # categorical values
kurtosis min
kurtosis max
kurtosis mean
kurtosis std
skewness min
skewness max
skewness mean
skewness std

PCA metafeatures:
pca 95%
pca skewness first pc
pca kurtosis first pc

Landmarking metafeatures:
One Nearest Neighbor
Linear Discriminant Analysis
Naive Bayes
Decision Tree
Decision Node Learner
Random Node Learner

While most of the metafeatures can be computed for a whole dataset, some of them (e.g., skewness) are defined for each attribute of a dataset. In this case, we compute the metafeature for each attribute of the dataset and use the mean, standard deviation, minimum and maximum of the resulting vector as proposed in [27]. Importantly, as our datasets are relatively small, the metafeatures for one dataset can be computed within less than one minute. Furthermore, for every dataset we use, the time needed to compute the metafeatures is less than the average time it takes to evaluate a hyperparameter configuration.

4 Experimental Methodology

We now discuss the datasets we used in our experiments, as well as the machine learning algorithms and their hyperparameters we optimized for them.

Table 2. List of the 57 datasets used for the experiments; the names refer to the names on the OpenML project website[34].

abalone	anneal.ORIG	arrhythmia
audiology	autos	balance-scale
braziltourism	breast-cancer	breast-w
car	cmc	credit-a
credit-g	cylinder-bands	dermatology
diabetes	ecoli	eucalyptus
glass	haberman	heart-c
heart-h	heart-statlog	hepatitis
ionosphere	iris	kr-vs-kp
labor	letter	liver-disorders
lymph	mfeat-factors	mfeat-fourier
mfeat-karhunen	mfeat-morphological	mfeat-pixel
mfeat-zernike	mushroom	nursery
optdigits	page-blocks	pendigits
postoperative-patient-data	primary-tumor	satimage
segment	sonar	soybean
spambase	tae	tic-tac-toe
vehicle	vote	vowel
waveform-5000	yeast	zoo

Table 3. Hyperparameters of the SVM. We optimized the base-2 logarithm of C and γ.

Hyperparameter	Values	Steps
$\log_2(C)$	$\{-5, -4, \ldots, 15\}$	21
$\log_2(\gamma)$	$\{-15, -14, \ldots, 3\}$	19

4.1 Datasets and Preprocessing

For our experiments, we obtained the 57 datasets listed in Table 2 from the OpenML project website[34]. We first shuffled each dataset and then split it in stratified fashion into 2/3 training and 1/3 test data. Validation performance for Bayesian optimization was then computed by ten-fold crossvalidation on the training dataset.

To use the same dataset for each classification algorithm, we coded categorical features using a one-hot (aka 1-in-k) encoding, replacing each categorical feature f with domain $\{v_1, \ldots, v_k\}$ by k binary variables, only the i-th of which is set to true for data points where f is set to v_i. To retain sparsity, we replaced any missing values with zero. Finally, we scaled numerical features linearly to the range $[0, 1]$ by subtracting the minimum value and dividing by the maximum. [3]

4.2 Machine Learning Algorithms and Their Hyperparameters

We empirically evaluated our MI-SMBO approach to optimize two practically relevant machine learning frameworks, one with few and one with many hyperparameters. The first framework are Support Vector Machines (SVMs) [28], namely the SVM implementation in Scikit-Learn (short sklearn) [22]. We used an RBF kernel and, in accordance with the LibSVM user guide [8] optimized two hyperparameters: the complexity penalty C and the kernel width of the RBF kernel γ. We chose the range of allowed values according to the LibSVM user guide; see Table 3 for details.

Our second machine learning framework comprises a range of machine learning algorithms in sklearn [22]. We combined all algorithms into a single hierarchical optimization problem using the Combined Algorithm Selection and Hyperparameter optimization (CASH) setting by Thornton et al. [33]: there was one top-level hyperparameter $\theta_{\text{classifier}}$ choosing between several classification algorithms and all hyperparameters of classification algorithm A_i were conditional on $\theta_{\text{classifier}}$ being set to A_i. This CASH problem is of high practical relevance since it describes precisely the problem an end user faces when given a new dataset.[4] To keep the computation bearable and the results interpretable, we only included three classification algorithms: an SVM with an RBF kernel (as in our first experiment), a linear SVM, and random forests [6] (one of the most robust classifiers available). Since we expected noise and redundancies in the training data, we also allowed the optimization procedure to use Principal Component Analysis (PCA) for preprocessing, with the number of PCA components being conditional on PCA being applied. In total this lead to 10 hyperparameters, as detailed in Table 4.

Table 4. Hyperparameters for the CASH problem in sklearn. All hyperparameters except $\theta_{\text{classifier}}$ and preprocessing are conditional. Hyperparameters not mentioned were set to their default value.

Component	Hyperparameter	Values	# Values
Main	$\theta_{\text{classifier}}$	{RF, SVM, LinearSVM}	3
Main	preprocessing	{PCA, None}	2
SVM	$\log_2(C)$	$\{-5, -4, \ldots, 15\}$	21
SVM	$\log_2(\gamma)$	$\{-15, -14, \ldots, 3\}$	19
LinearSVM	$\log_2(C)$	$\{-15, -14, \ldots, 15\}$	21
LinearSVM	penalty	$\{L_1, L_2\}$	2
RF	min splits	$\{1, 2, 4, 7, 10\}$	5
RF	max features	$\{1\%, 4\%, \ldots, 100\%\}$	10
RF	criterion	{Gini, Entropy}	2
PCA	variance to keep	$\{80\%, 90\%\}$	2

4.3 Experimental Setup

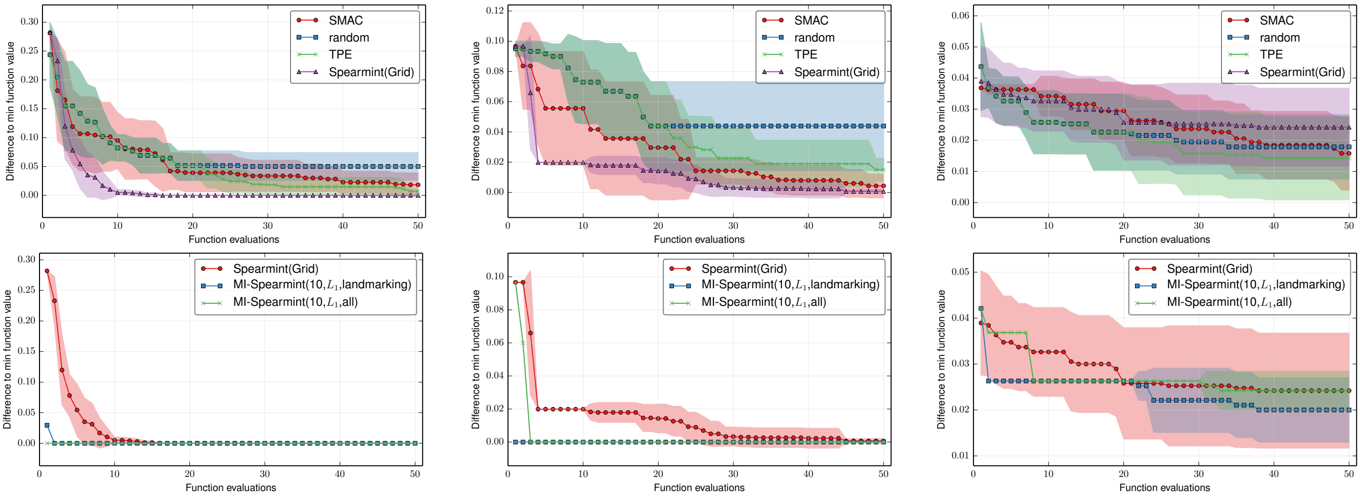
For both machine learning frameworks, we precomputed the 10-fold crossvalidation error on all 57 datasets over a grid with all possible hyperparameter combinations. For the SVM, this grid contained all 399 combinations of the 19 values for C and 21 values for γ listed in Table 3. For sklearn, it contained an additional 1 224 possible hyperparameter configurations, due to the additional flexibility of preprocessing and the two other model classes (linear SVMs and random forests, see Table 4). Therefore, in total, we evaluated 1 623 hyperparameter configurations on each dataset. Although the classification datasets were no larger than medium-sized ($< 30\,000$ data points), calculating the grid took up to three days per dataset on a modern CPU. This extensive precomputation allowed us to run all our experiments in simulation, by using a lookup table in lieu of running an actual algorithm. We will make the gathered algorithm performance data publicly available to facilitate both reproducibility of our experiments and follow-up work using the same data.

We evaluated our MI-SMBO approach in a leave-one-dataset-out fashion: to evaluate it on a dataset D_{new}, we assumed knowledge of the other 56 datasets and their best hyperparameter settings. Because Bayesian optimization contains random factors, we repeated each optimization run ten times on each dataset. In total, we thus executed each optimization procedure 570 times.

Our metalearning initialization approach has several free design choices we had to instantiate for our experiments. Firstly, we had to

[3] This is the standard practice for SVMs, as for example advised in the LibSVM user guide: `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

[4] We note that several others have also studied variants of the CASH problem in sklearn [13, 18].



dataset: heart-c

dataset: arrhythmia

dataset: breast-cancer

Figure 1. Difference in SVM validation error between the best found hyperparameters at evaluation t and the best value obtained via a full grid search on three datasets. MI-Spearmint$(10, L_1, X)$ stands for MI-Spearmint with an initial design of $t = 10$ configurations suggested by metalearning using metafeatures X. Note the differently scaled y-axes in the top and bottom plots.

choose a norm for the distance metric in the space of meta-features in Equation 3; we experimented with both the L_1 and L_2 norms. Next to using the full set of metafeatures, we experimented with various subsets. Since previous empirical results suggested that landmarking metafeatures are superior to other metafeatures [23, 25, 26], we experimented with using only the landmarking features used in the first experiment of Pfahringer et al. [23]. We also experimented with the subsets of metafeatures used in previous works on collaborative SMBO [1, 35]. Lastly, we had to decide how many hyperparameter configurations to evaluate as part of the warmstart phase before switching to the SMBO algorithm. Here we tried the values $t \in \{5, 10, 20, 25\}$. In total, we evaluated 40 different instantiations of our metalearning procedure.

Due to space restrictions, we only report results for a subset of these instantiations. Concerning distance measures, in preliminary experiments the results with the L_1 and L_2 norms were qualitatively similar, with slightly better results for the L_1 norm. Thus, all distances in the experiments reported here were calculated with the L_1-norm. Preliminary results with different subsets of metafeatures showed that the metafeatures used by existing collaborative SMBO methods did not match the performance of the other sets; we therefore restricted ourselves to only show results for all metafeatures and for only the landmarking metafeatures. Finally, we report performance only for MI-SMBO with $t = 10$ hyperparameter configurations suggested by metalearning; however, preliminary results suggest that for larger configuration spaces larger values of t improve results.

5 Experimental Results

We now report our results for optimizing SVMs and sklearn. For each of the two machine learning frameworks we studied, we first assessed the state of the art and then improved it with MI-SMBO. Specifically, we evaluated the base performance of the hyperparameter optimization procedures random search, Spearmint, TPE, and SMAC (described in Section 2; note that for TPE the prior distributions were uniform) on our 57 datasets and then added warmstarts via MI-SMBO to the best of these.

5.1 Warmstarting Spearmint for Optimizing SVMs

For the low-dimensional problem of optimizing SVMs, the Spearmint optimizer tended to perform best. Figure 1 (top) compares its qualitative performance on three representative datasets to that of TPE, SMAC, and random search, showing that it typically performed best, but that there was still room for improvement. A statistical analysis using a two-sided t-test on the performances for each of the 57 datasets shows that Spearmint indeed significantly outperformed TPE, SMAC, and random search in 35%, 44%, and 52% of the datasets, respectively, and only lost in 4%, 4%, and 8% of the cases, respectively. These findings are in line with previous results showing Spearmint to be the best choice for hyperparameter optimization benchmarks with a small number of continuous hyperparameters [9].

We thus applied our MI-SMBO approach to Spearmint, using either all meta-features or just the landmarking features, to suggest the first $t = 10$ hyperparameter settings Spearmint should evaluate. Figure 1 (bottom) compares the resulting warm-start versions of Spearmint against vanilla Spearmint on the same three representative datasets as above. For the two datasets on the left, metalearning directly identified one of the optimal hyperparameter configurations in the first function evaluation; this is in contrast to vanilla Spearmint, which required 17 and 45 function evaluations, respectively, to eventually reach a configuration of equal performance. In contrast, for the dataset on the right, metalearning only yielded small improvements (a comparison to the right top plot in Figure 1 shows that neither variant of Spearmint performed better than random search in this case).

Next, we analyzed the performance of MI-Spearmint using the same ranking-based evaluation as Bardenet et al. [1] to aggregate over datasets. For each dataset and for each function evaluation budget from 1 to 50, we computed the ranks of the four baselines (random search, SMAC, TPE, and Spearmint) and the two MI-Spearmint variants. More precisely, since we had available 10 runs of each of the 6 methods for each dataset (which give rise to 10^6 possible com-

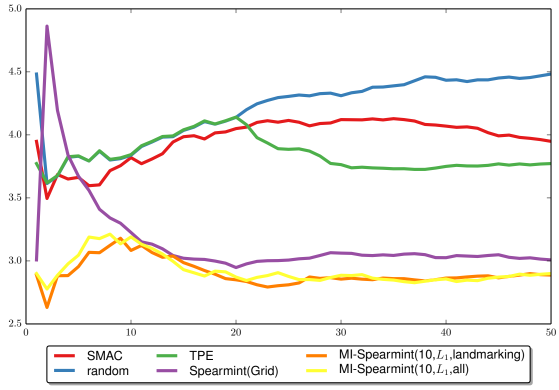


Figure 2. Average rank of each optimizer, computed over all datasets, for the SVM classification experiment. (WS10,l1,X) denotes MI-Spearmint warm-started with 10 configurations suggested by metalearning using metafeatures X.

binations), we drew a bootstrap sample of 1 000 joint runs of the six optimizers and computed the average ranks across these runs. We then further averaged these average ranks across the 57 datasets and show the results in Figure 2. We remind the reader that the rank is a measure of performance *relative* to the performance of the other optimizers; thus, a method's rank can increase over time (with larger function evaluation budgets) even though its error decreases if the other methods achieve greater error reductions. Furthermore, we note that this plot simply ranks raw function values and does not include information about how much the errors of the various methods differ. With this disclaimer noted, the results are as expected: random search performed worst, followed by SMAC and TPE, which are known to be outperformed by Spearmint for low-dimensional continuous problems [9]. The three variants of Spearmint performed best, converging to a similar rank with larger function evaluation budgets; meta-learning yielded dramatically better results for very small function evaluation budgets, and after about 10 function evaluations Spearmint (almost) caught up. We note, however, that even after 50 function evaluations Spearmint still had not fully caught up to its warmstart variants, indicating that an initialization with 10 configurations determined by meta learning provided not only good performance with few function evaluations but also a good basis for Spearmint to improve upon further.

To complement the above ranking analysis, Figure 3 quantifies on how many datasets MI-Spearmint (based on the landmarking features) performed better and worse than the other methods according to a two-sided t-test (over the ten repetitions of runs per dataset). The upper plot of Figure 3 shows the ratio of datasets for which MI-Spearmint performed significantly better than the other methods, and the lower plot shows the statistically significant losses. Both of these quantities are plotted over time, as the function evaluation budget increases. We observe that MI-Spearmint started off much better than all other methods. Given larger function evaluation budgets, using its Spearmint part, it even increased the performance advantage over random search, TPE, and SMAC. Compared to Spearmint, MI-Spearmint started off significantly better in 70% of the datasets, but these differences leveled off over time. There was very little difference between the two MI-Spearmint variants (based on landmarking features vs. based on all features).

5.2 Warmstarting SMAC for Optimizing sklearn

We used the same approach as in the above experiment to assess MI-SMBO's performance on the combined algorithm selection and hyperparapeter optimization problem in sklearn. First, we assessed the state of the art for this problem. Due to the conditional hyperparameters in the sklearn space, we excluded Spearmint (which does not natively support them and is known to perform poorly in their presence [9]) and only evaluated SMAC, TPE, and random search. Figure 4 (top) presents the qualitative performance of these optimizers on three representative datasets, showing that both SMAC and TPE performed better than random search. Overall, in line with the results of Eggensperger et al. [9] for large hyperparameter spaces, we found SMAC and TPE to perform best. We applied our metalearning initialization to SMAC, but would also expect TPE to benefit from it.

Figure 4 (bottom) shows the qualitative results of MI-SMAC compared to vanilla SMAC. In the left plot, the metalearning suggestions were reasonable and MI-SMAC's second part could improve on these over time. In the middle plot the second configuration suggested by metalearning was already the best, leaving no room for improvement by SMAC. The right plot highlights the fact that metalearning can also fail and decrease the performance of SMAC.

Figure 5 shows the percentage of statistically significant wins of MI-SMAC against the other optimizers. As before, we evaluated two different versions of MI-SMAC, based on all features and based on only the landmarking metafeatures from Pfahringer [23]; the figure shows that MI-SMAC based on the landmarking features alone worked somewhat better than based on all features, winning statistically significantly on 11% of the datasets (and loosing on 8%). Compared to the optimizers without metalearning, MI-SMAC performed much better from the start. Even after 50 iterations, it performed significantly better than TPE on 14% of the datasets (in 8% worse), better than SMAC on 25% of the datasets (in 10% worse), and better than random search on 35% of the dataset (in 9% worse). We would like to point out that the improvement MI-SMAC yielded over SMAC is nearly as large as the improvement that SMAC yielded over random search (in 29% better). This is in contrast to the (only) slight improvements MI-Spearmint yielded over Spearmint for optimizing SVMs. We attribute the success for sklearn to its much larger search space, which not even SMAC can effectively search in as little as 50 function evaluations. Drawing on successful optimizations from previous datasets clearly helped SMAC in this complex search space.

6 Conclusion

We have presented a simple, yet effective, method for improving Sequential Model-based Bayesian Optimization (SMBO) of hyperparameters by transferring knowledge from previous optimization runs. Our method combines ideas from both the metalearning and the Bayesian optimization community by initializing SMBO with configurations suggested by a metalearning procedure. We dub the resulting metalearning-initialized SMBO variant MI-SMBO. Importantly, MI-SMBO is agnostic of the actual SMBO method used and can thus be applied to the method best suited for a particular problem.

We demonstrated MI-SMBO's efficacy by improving the initialization of two quite different SMBO methods for optimizing two machine learning frameworks on a total of 57 datasets. For optimization in the low-dimensional hyperparameter space of a support vector machine, our MI-Spearmint variant of the best-performing SMBO method Spearmint mainly improved upon Spearmint in the

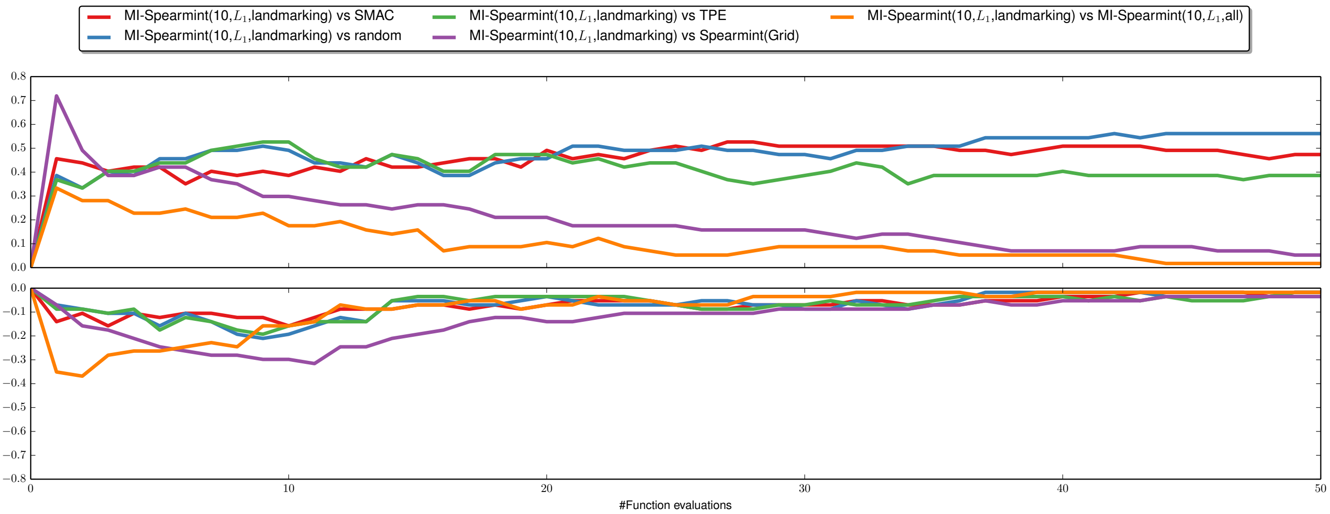
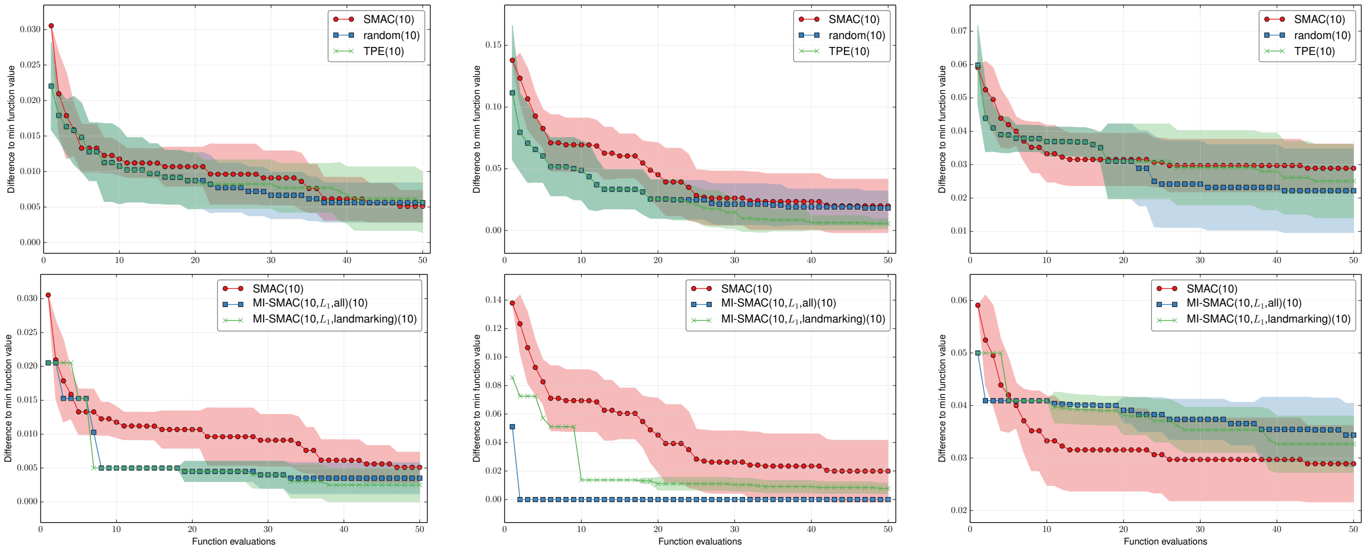


Figure 3. Percentage of wins of MI-Spearmint with an initial design of $t = 10$ configurations suggested by metalearning using the L_1 distance on the metafeature subset from Pfahringer [23]. The upper plot shows significant wins of MI-Spearmint against each other approach according to the two-sided t-test while the lower plot shows the statistically significant losses.



dataset: sonar

dataset: heart-h

dataset: hepatitis

Figure 4. Difference in validation error between sklearn instantiated with the best found hyperparameters and the best value obtained via a full grid search, for three datasets. (WS10,l1,X) stands for MI-SMAC with an initial design of $t = 10$ configurations suggested by metalearning using metafeatures X. Note the differently scaled y-axes in the top and bottom plots.

early stages of optimization, thus helping it find good configurations quickly. For a large configuration space describing a combined algorithm selection and hyperparameter optimization problem in scikit-learn, our MI-SMAC variant of the best-performing SMBO variant SMAC substantially improved over SMAC (and all other optimizers we tested) across a range of function evaluation budgets, showing the potential of our approach especially for large scale hyperparameter optimization.

In future work, we plan to evaluate MI-SMAC for even larger configuration spaces, such as those of Auto-WEKA [33] and Hyperopt-Sklearn[18]. We also noticed the lack of a canonical implementation of metafeatures and are aiming to provide such an implementation. Finally, we plan to integrate metalearning into the SMBO procedure and compare the result with recent work on collaborative SMBO [1, 35, 32].

REFERENCES

[1] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, ‘Collaborative hyperparameter tuning’, in *Proc. of ICML*, (2013).

[2] H. Bensusan and C. Giraud-Carrier, ‘Discovering task neighbourhoods through landmark learning performances’, in *Proc. of 4th PKDD*. Springer, (September 2000).

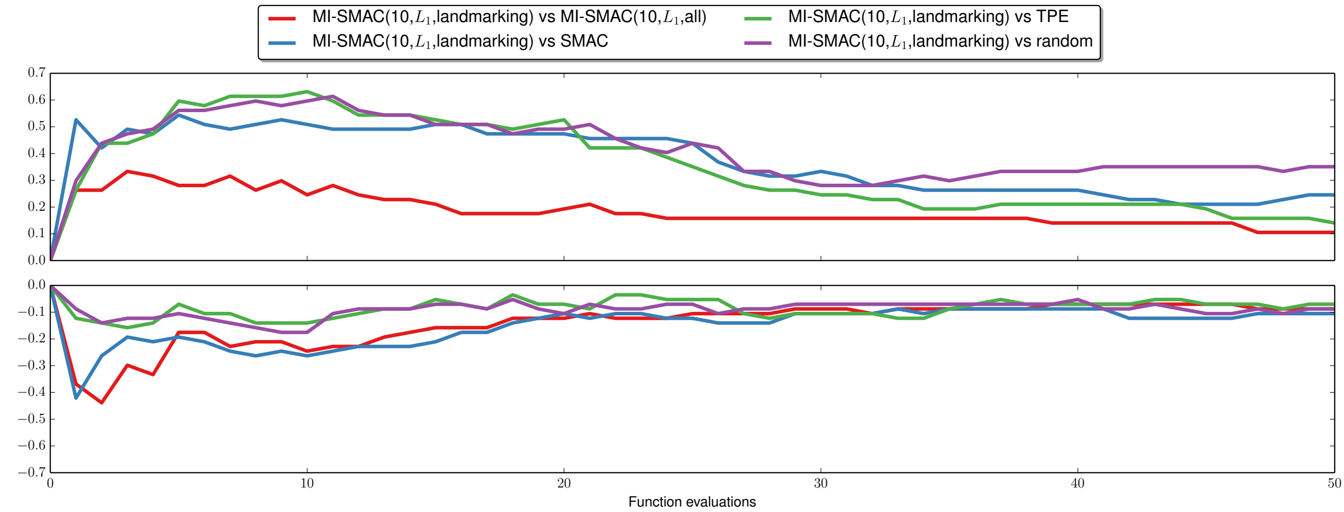


Figure 5. Percentage of wins of MI-SMAC with an initial design of $t = 10$ configurations suggested by metalearning using the L_1 distance on the metafeature subset from Pfahringer [23]. The upper plot shows the number of significant wins of MI-SMAC over competing approaches according to the two-sided t-test while the lower plot shows the statistically significant losses.

[3] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, ‘Algorithms for hyper-parameter optimization’, in *Proc. of NIPS*, (2011).

[4] J. Bergstra and Y. Bengio, ‘Random search for hyper-parameter optimization’, *JMLR*, **13**, (February 2012).

[5] J. Bergstra, D. Yamins, and D. D. Cox, ‘Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures’, in *Proc. of ICML*, (2013).

[6] L. Breiman, ‘Random forests’, *Machine Learning*, **45**, (2001).

[7] E. Brochu, V. M. Cora, and N. de Freitas, ‘A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning’, *CoRR*, **abs/1012.2599**, (2010).

[8] Chih-Chung Chang and Chih-Jen Lin, ‘LIBSVM: A library for support vector machines’, *ACM Transactions on Intelligent Systems and Technology*, **2**, (2011).

[9] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. H. Hoos, and K. Leyton-Brown, ‘Towards an empirical foundation for assessing bayesian optimization of hyperparameters’, in *NIPS workshop on Bayesian Optimization*, (2013).

[10] T.A.F. Gomes, R.B.C. Prudêncio, C. Soares, A. Rossi, and A. Carvalho, ‘Combining meta-learning and search techniques to select parameters for support vector machines’, *Neurocomputing*, **75**(1), (2012).

[11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten, ‘The WEKA data mining software: an update’, *ACM SIGKDD Explorations Newsletter*, **11**(1), 10–18, (2009).

[12] P. Hennig and C. Schuler, ‘Entropy search for information-efficient global optimization’, *JMLR*, **13**, (2012).

[13] M. W. Hoffman, B. Shahriari, and N. de Freitas, ‘Exploiting correlation and budget constraints in Bayesian multi-armed bandit optimization’, *ArXiv e-prints*, (March 2013).

[14] F. Hutter, H. H. Hoos, and K. Leyton-Brown, ‘Sequential model-based optimization for general algorithm configuration’, in *Proc. of LION-5*, (2011).

[15] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, ‘Algorithm runtime prediction: Methods and evaluation’, *JAIR*, **206**(0), 79 – 111, (2014).

[16] D.R. Jones, M. Schonlau, and W. Welch, ‘Efficient global optimization of expensive black box functions’, *Journal of Global Optimization*, **13**, (1998).

[17] A. Kalousis, *Algorithm Selection via Meta-Learning. University of Geneve, Department of Computer Science*, Ph.D. dissertation, University of Geneve, 2002.

[18] B. Komer, J. Bergstra, and C. Eliasmith, ‘Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn’, in *ICML workshop on AutoML*, (2014).

[19] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren, ‘Selecting classification algorithms with active testing on similar datasets’, in *5th PLANLEARN WORKSHOP at ECAI*, (2012).

[20] *Machine Learning, Neural and Statistical Classification*, eds., Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell, Ellis Horwood, 1994.

[21] P.B.C. Miranda, R.B.C. Prudêncio, A. Carvalho, and C. Soares, ‘Combining meta-learning with multi-objective particle swarm algorithms for SVM parameter selection: An experimental analysis’, in *Brazilian Symposium on Neural Networks*, (2012).

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, ‘Scikit-learn: Machine learning in Python’, *JMLR*, **12**, (2011).

[23] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier, ‘Meta-learning by landmarking various learning algorithms’, in *Proc. of ICML*, (2000).

[24] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*, The MIT Press, 2006.

[25] M. Reif, F. Shafait, and A. Dengel, ‘Prediction of classifier training time including parameter optimization’, in *KI 2011: Advances in Artificial Intelligence*, (2011).

[26] M. Reif, F. Shafait, and A. Dengel, ‘Meta-learning for evolutionary parameter optimization of classifiers’, *Machine Learning*, **87**, (2012).

[27] M. Reif, F. Shafait, and A. Dengel. Meta2-features: Providing meta-learners more information, 2012. Poster and Demo Track of the 35th German Conference on AI.

[28] Bernhard Scholkopf and Alexander J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, Cambridge, MA, USA, 2001.

[29] J. Snoek, H. Larochelle, and R.P. Adams, ‘Practical bayesian optimization of machine learning algorithms’, in *Proc. of NIPS*, (2012).

[30] C. Soares and P.B. Brazdil, ‘Zoomed ranking: Selection of classification algorithms based on relevant performance information’, in *Proc. of PKDD’00*, Springer, (2000).

[31] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, ‘Gaussian process optimization in the bandit setting: No regret and experimental design’, in *Proc. of ICML*, (2010).

[32] K. Swersky, J. Snoek, and R.P. Adams, ‘Multi-task bayesian optimization’, in *Proc. of NIPS*, (2013).

[33] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, ‘Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms’, in *Proc. of KDD’13*, (2013).

[34] J. N. van Rijn, B. Bischl, L. Torgo, B. Gao, V. Umaashankar, S. Fischer, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren, ‘OpenML: a collaborative science platform’, in *Proc. of ECML/PKDD’13*, (2013).

[35] D. Yogatama and G. Mann, ‘Efficient transfer learning method for automatic hyperparameter tuning’, in *Proc. of AISTATS*, (2014).