Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Initializing Bayesian Hyperparameter Optimization via Meta-Learning

Matthias Feurer and **Jost Tobias Springenberg** and **Frank Hutter**
{feurerm,springj,fh}@cs.uni-freiburg.de
Computer Science Department, University of Freiburg
Georges-Köhler-Allee 52
79110 Freiburg, Germany

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 computationally expensive algorithms the overhead of hyperparameter optimization can still be prohibitive. In this paper we mimic a strategy human domain experts use: speed up optimization by starting from promising configurations that performed well on similar datasets. The resulting initialization technique integrates naturally into the generic SMBO framework and can be trivially applied to any SMBO method. To validate our approach, we perform extensive experiments with two established SMBO frameworks (Spearmint and SMAC) with complementary strengths; optimizing two machine learning frameworks on 57 datasets. Our initialization procedure yields mild improvements for low-dimensional hyperparameter optimization and substantially improves the state of the art for the more complex combined algorithm selection and hyperparameter optimization problem.

Introduction

Hyperparameter optimization is a crucial step in the process of applying machine learning algorithms in practice. 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 (Hutter, Hoos, and Leyton-Brown 2011; Bergstra et al. 2011; Snoek, Larochelle, and Adams 2012; Bergstra and Bengio 2012).

Recently, Sequential Model-based Bayesian Optimization (SMBO, see, e.g., Brochu, Cora, and de Freitas (2010) for an overview) has emerged as a successful hyperparameter optimization method in machine learning. SMBO has been conclusively shown to yield better performance than both grid and random search and matched or outperformed the state-of-the-art performance for several challenging machine learning problems (Snoek, Larochelle, and Adams 2012; Bergstra et al. 2011; Bergstra, Yamins, and Cox 2013). It has also enabled AutoWEKA (Thornton et al. 2013), which performs combined algorithm selection and hyperparameter optimization in the space of algorithms defined by the WEKA package (Hall et al. 2009).

However, as a generic function optimization framework, SMBO requires a substantial number of 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. A promising approach to combat this problem is to apply meta-learning (Brazdil et al. 2008) to the hyperparameter search problem. The key concept behind meta-learning for hyperparameter search is to suggest good configurations for a novel dataset based on configurations that are known to perform well on similar, previously evaluated, datasets. We follow this strategy 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. Importantly, MI-SMBO does not require any adaptation of the underlying SMBO procedure. It is hence easy to implement and can be readily applied to off-the-shelf hyperparameter optimizers.

We empirically studied the impact of our meta-learning-based initialization procedure on two SMBO variants, using a comprehensive suite of 57 classification datasets and 46 metafeatures. First, we applied our method to a combined algorithm selection and hyperparameter (CASH) optimization problem on this benchmark: choosing between three classifiers from the prominent scikit-learn package (Pedregosa et al. 2011) and simultaneously optimizing their hyperparameters. Second, to demonstrate the generality of our approach, we applied MI-SMBO to the lower-dimensional problem of optimizing the 2 hyperparameters of a support vector machine (SVM) on the same datasets. We found that for the lower-dimensional problem our MI-Spearmint variant of Spearmint (Snoek, Larochelle, and Adams 2012) (a state-of-the-art approach for low-dimensional hyperparameter optimization) yielded mild improvements. For the more challenging CASH problem our MI-SMAC variant of the SMBO method SMAC (Hutter, Hoos, and Leyton-Brown 2011) (a state-of-the-art approach for CASH optimization) yielded substantial improvements, significantly outperforming the previous state of the art for this problem.

This paper is an improved and extended version of a previous workshop submission (Feurer, Springenberg, and Hutter 2014)

Foundations

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

Hyperparameter Optimization

Let $\theta_1, \ldots, \theta_n$ denote the hyperparameters of a machine learning algorithm, and let $\Theta_1, \ldots, \Theta_n$ denote their respective domains. The algorithm'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 is to minimize:

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

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, setting the "number of principal components" is conditioned on the usage of PCA as a preprocessing method.

The most frequently used hyperparameter optimization method is grid search, which often performs poorly and does not scale to high dimensions. Therefore, a large body of recent work has focused on better-performing methods, in particular SMBO, which we describe in the following section.

Sequential Model-based Bayesian Optimization

Sequential Model-based Bayesian Optimization (SMBO) (Jones, Schonlau, and Welch 1998; Brochu, Cora, and de Freitas 2010; Hutter, Hoos, and Leyton-Brown 2011) is a powerful method for 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 acquisition function to use, and the type of probabilistic model. We discuss three prominent hyperparameter optimization methods in terms of these components: SMAC (Hutter, Hoos, and Leyton-Brown 2011), Spearmint (Snoek, Larochelle, and Adams 2012), and TPE (Bergstra et al. 2011).

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 acquisition function (used by all three SMBO methods we discuss) is the expected positive improvement (EI) over the best input found so far (Jones, Schonlau, and Welch 1998):

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

Several different model types can be used inside of SMBO. The most popular choice, used for example by Spearmint, are Gaussian processes (Rasmussen and Williams 2006) 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 and partially categorical input spaces. In particular, SMAC uses random forests (Breiman 2001) modified to yield an uncertainty estimate (Hutter et al. 2014). Another tree-based approach is the Tree Parzen Estimator (TPE) (Bergstra et al. 2011), which constructs a density estimate over good and bad instantiations of each hyperparameter.

The final degree of freedom in SMBO is its initialization. A classic approach is to initialize SMBO with a space-filling design (Jones, Schonlau, and Welch 1998). While this can greatly improve the quality of the model, the corresponding function evaluations are also costly and for expensive hyperparameter optimization problems a cheaper solution is needed. To date, this initialization component has not received much attention, and it is typically instantiated in a fairly ad-hoc manner: Spearmint evaluates f at the first two points of a Sobol sequence, SMAC evaluates it at a user-defined 'default' input, and TPE evaluates 20 points selected at random according to a user-defined prior distribution. It is this initialization component that our MI-SMBO approach improves by starting from a list of hyperparameter configurations suggested by meta-learning.

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 $\boldsymbol{\theta}_{1:t} = \langle \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_t \rangle$
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 $\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}_j \in \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T\}} y_j$

Initializing SMBO With Configurations Suggested by Meta-Learning

Building on the foundations from the previous section we now describe our proposed MI-SMBO method.

The core idea behind MI-SMBO is to follow the common practice machine learning experts employ when applying a

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

Input: new dataset D_{N+1}; target function $f^{D_{N+1}}$; training datasets $D_{1:N} = (D_1, \ldots, D_N)$; best configurations for training datasets, $\hat{\boldsymbol{\theta}}^{1:N} = \hat{\boldsymbol{\theta}}^1, \ldots, \hat{\boldsymbol{\theta}}^N$; 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 $\pi(1), \ldots \pi(N)$ by increasing distance to D_{N+1}, i.e.:
$(\pi(i) \leq \pi(j)) \Leftrightarrow (d(D_{N+1}, D_i) \leq d(D_{N+1}, D_j))$

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 $\boldsymbol{\theta}^*$

known machine learning method to a new dataset D_{N+1}: they first study D_{N+1}, relating it to datasets they previously experienced. When manually optimizing hyperparameters for D_{N+1}, they would begin the search with hyperparameter configurations that were optimal for the most similar previous datasets (see, e.g., Dahl, Sainath, and Hinton (2013); Goodfellow et al. (2013)). Our MI-SMBO method automates this approach and uses it to initialize an SMBO method.

Formally, MI-SMBO can be stated as follows. Let $\hat{\boldsymbol{\theta}}^1, \ldots, \hat{\boldsymbol{\theta}}^N$ denote the best known hyperparameters for the previously encountered datasets $D_1, \ldots, D_N$, 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_{N+1} denote a new dataset, let d denote a distance metric between datasets, and let π denote a permutation of $(1, \ldots, N)$ sorted by increasing distance between D_{N+1} and D_i (i.e., $(\pi(i) \leq \pi(j)) \Leftrightarrow (d(D_{N+1}, D_i) \leq d(D_{N+1}, D_j))$). Then, MI-SMBO with an initial design of t configurations initializes SMBO with configurations $\hat{\boldsymbol{\theta}}^{\pi(1)}, \ldots, \hat{\boldsymbol{\theta}}^{\pi(t)}$. 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. Furthermore, in contrast to existing approaches that initialize direct search algorithms via meta-learning (Gomes et al. 2012; Reif, Shafait, and Dengel 2012), SMBO is a particularly good match for initialization with meta-learning as it can make effective use of all performance data it receives as input. In practice, this procedure replaces SMACs and Spearmints initialization and delays the start of the actual SMBO procedure until all configurations $\hat{\boldsymbol{\theta}}^{\pi(1)}, \ldots, \hat{\boldsymbol{\theta}}^{\pi(t)}$ are evaluated.

The last component needed to implement MI-SMBO is the definition of a distance metric between datasets. This problem was, to our knowledge, first discussed by Soares and Brazdil (2000). For the purpose of this work we assume that each dataset D_i can be described by a set of F metafeatures $\boldsymbol{m}^i = (m_1^i, \ldots, m_F^i)$. We discuss the metafeatures we used in the next section. In practice, we precompute the metafeatures for all training datasets $D_1, \ldots, D_N$, along with the best configurations $(\hat{\boldsymbol{\theta}}^1, \ldots, \hat{\boldsymbol{\theta}}^N)$. Given a new dataset D_{N+1}, we then measure its distances to all previous datasets D_i using a distance measure $d : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$.

We experimented with two different instantiations of this distance measure $d(\cdot, \cdot)$. The first measure (denoted as d_p) we used is the commonly-used p-norm of the difference between the datasets' metafeatures:

$$d_p(D_i, D_j) = \|\boldsymbol{m}^i - \boldsymbol{m}^j\|_p. \tag{3}$$

Next to this standard metric we aimed for a metric that reflects how similar the datasets are with respect to the performance of different hyperparameter settings. The measure we use (in the following denoted as d_c) is the negative Spearman correlation coefficient between the ranked results of a fixed set of n hyperparameter configurations on both datasets[1]:

$$\begin{aligned} d_c(D_i, D_j) = 1 - Corr([f^{D_i}(\boldsymbol{\theta}_1), \ldots, f^{D_i}(\boldsymbol{\theta}_n)], \\ [f^{D_j}(\boldsymbol{\theta}_1), \ldots, f^{D_j}(\boldsymbol{\theta}_n)]). \end{aligned} \tag{4}$$

Of course, this distance measure cannot be computed directly for the new dataset D_{N+1} since we have not yet evaluated $f^{D_{N+1}}(\boldsymbol{\theta}_1), \ldots, f^{D_{N+1}}(\boldsymbol{\theta}_N)$. However, we *can* compute $d_c(D_i, D_j)$ for all $1 \leq i, j \leq N$ and use regression to learn a function $R : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}$, mapping from pairs of meta-features $\langle \boldsymbol{m}^i, \boldsymbol{m}^j \rangle$ to $d_c(D_i, D_j)$.

Using this pre-trained regressor the distance metric can then be approximated as

$$d_c(D_{N+1}, D_j) \approx R(\boldsymbol{m}^{N+1}, \boldsymbol{m}^j). \tag{5}$$

In our experiments, we implemented R using a random forest because of its robustness and speed.

Implemented Metafeatures

To evaluate our approach in a realistic setting we implemented 46 metafeatures from the literature; they are 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 (Michie et al. 1994; Kalousis 2002)
- *PCA metafeatures* (Bardenet et al. 2013) compute various statistics of the datasets principal components.
- The *information-theoretic metafeature* measures the class entropy in the data (Michie et al. 1994).
- *Statistical metafeatures* (Michie et al. 1994) characterize the data via descriptive statistics such as the kurtosis or the dispersion of the label distribution.
- *Landmarking metafeatures* (Pfahringer, Bensusan, and Giraud-Carrier 2000) are computed by running several fast machine learning algorithms on the dataset. Based on their learning scheme they can capture different properties of the dataset, like e.g. linear separability.

[1] In practice, we used all n hyperparameter configurations for which we had results available on all training datasets.

Simple metafeatures:	**Statistical metafeatures:**
number of patterns	min # categorical values
log number of patterns	max # categorical values
number of classes	mean # categorical values
number of features	std # categorical values
log number of features	total # categorical values
number of patterns with missing values	kurtosis min
percentage of patterns with missing values	kurtosis max
number of features with missing values	kurtosis mean
percentage of features with missing values	kurtosis std
number of missing values	skewness min
percentage of missing values	skewness max
number of numeric features	skewness mean
number of categorical features	skewness std
ratio numerical to categorical	
ratio categorical to numerical	**PCA metafeatures:**
dataset dimensionality	pca 95%
log dataset dimensionality	pca skewness first pc
inverse dataset dimensionality	pca kurtosis first pc
log inverse dataset dimensionality	
class probability min	**Landmarking metafeatures:**
class probability max	One Nearest Neighbor
class probability mean	Linear Discriminant Analysis
class probability std	Naive Bayes
	Decision Tree
Information-theoretic	Decision Node Learner
metafeature:	Random Node Learner
class entropy	

Table 1: List of implemented metafeatures

For each dataset, metafeatures are only computed on the training set. In our experiments, for each dataset this required less than one minute and less than the average time it took to evaluate one hyperparameter configuration on that dataset.

Application to Machine Learning Algorithms

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

ML Algorithms and Hyperparameters

We empirically evaluated our MI-SMBO approach to optimize two practically relevant machine learning frameworks.

We focused on supervised classification because it is the most widely studied problem in metalearning, with a large body of literature and readily available metafeatures and datasets.[2]

The large configuration space for our main experiment is spanned by a range of machine learning algorithms from scikit-learn (Pedregosa et al. 2011). We combined all algorithms into a single hierarchical optimization problem using the Combined Algorithm Selection and Hyperparameter optimization (CASH) setting by Thornton et al. (2013): we used one top-level hyperparameter $\theta_{\text{classifier}}$ for choosing between classification algorithms, and set all hyperparameters of classification algorithm A_i as 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.[3] To keep the computation bearable and the results interpretable, we only included three classification algorithms: an SVM with an RBF kernel, a linear SVM, and random forests. 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 2. We discretized these 10 hyperparameters to obtain a manageable number of 1 623 hyperparameter configurations that allowed the exhaustive precomputation of classification errors for the entire grid.

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

Table 2: Hyperparameters for the CASH problem in scikit-learn. All hyperparameters except $\theta_{\text{classifier}}$ and preprocessing are conditional. Hyperparameters not mentioned were set to their default value.

We performed a second experiment to test the suitability of our method for a low-dimensional hyperparameter optimization problem: optimizing the complexity penalty C and the kernel width γ of an SVM using an RBF kernel. As above, we discretized these two hyperparameters to a grid of $19 \cdot 21 = 399$ combinations; these constitute a subset of the configurations considered in the CASH problem above.

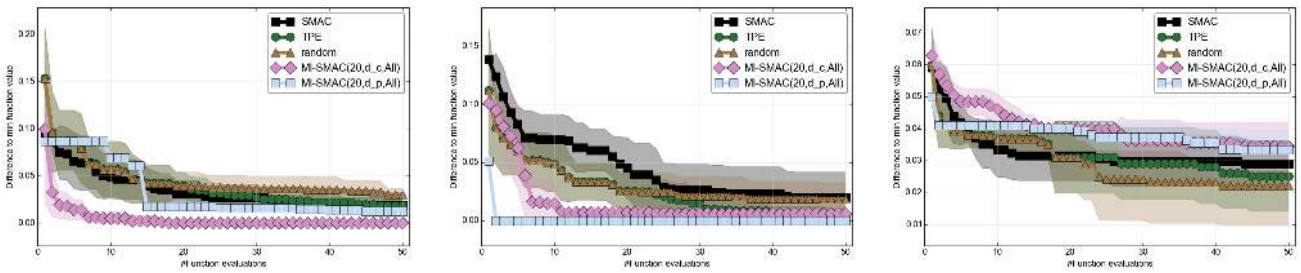
Datasets and Preprocessing

For our experiments, we aimed for a large number of high-quality classification datasets. We found the OpenML project (Vanschoren et al. 2013) to be the best source of datasets and used the 60 classification datasets it contained in April 2014. For computational reasons we had to exclude three datasets, leaving us with a total of 57. We first shuffled each dataset and then split it in stratified fashion into 2/3 training and 1/3 test data. Then, we computed the validation performance for Bayesian optimization by ten-fold crossvalidation on the training dataset.

To use the same dataset for each classification algorithm,

[2] We note, however, that in principle, our procedure is applicable to every optimization problem that is concerned with minimizing a measurable objective and has a set of metafeatures describing the problem. For example, one possible use in the field of unsupervised learning could be representation learning, with reconstruction error as the objective.

[3] We note the existence of previous work on CASH variants for scitkit-learn (Hoffman, Shahriari, and de Freitas 2014; Komer, Bergstra, and Eliasmith 2014).



dataset: liver-disorders dataset: heart-h dataset: hepatitis

Figure 1: Difference in validation error between hyperparameters found by SMBO and the best value obtained via full grid search for three datasets with scikit-learn. (20,d,X) stands for MI-SMAC with an initial design of $t = 20$ configurations suggested by meta-learning with distance measure d using metafeatures X.

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 $[0, 1]$.

Experiments

Experimental Setup

We precomputed the 10-fold crossvalidation error on all 57 datasets for each of the 1 623 hyperparameter configurations in our CASH problem. Because the configurations for the SVM benchmark form a subset of these configurations, the corresponding results were reused for the second experiment. Although the classification datasets were no larger than medium-sized ($<$ 20 000 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 evaluated our MI-SMBO approach in a leave-one-dataset-out fashion: to evaluate it on one dataset, 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 meta-learning initialization approach has several free design choices we had to instantiate for our experiments. These are: the distance metric d, the used metafeatures (we experimented with several subsets suggested in the literature (Pfahringer, Bensusan, and Giraud-Carrier 2000; Bardenet et al. 2013; Yogatama and Mann 2014)) and the number $t \in \{5, 10, 20, 25\}$ of configurations used for initializing SMBO. In total, we evaluated 40 different instantiations of our meta-learning procedure. Due to space restrictions, we only report results for the best of these instantiations; for more results, please see the supplementary material: *www.automl.org/aaai2015-mi-smbo-supplementary.pdf*

Concerning distance measures, we found the results with d_p and d_c distance to be qualitatively similar, with slightly better results for the d_c measure. We thus restrict the plots to d_c in several experiments to avoid clutter in the plots. Our experiments with different metafeatures showed that there is no general best set of metafeatures; thus, we only report results using all metafeatures.

Warmstarting SMAC for Optimizing scikit-learn

We now report our results for solving the CASH problem in scikit-learn. First, we evaluated the base performance of the hyperparameter optimization procedures random search, TPE, and SMAC (note that for TPE the prior distributions were uniform) on all 57 datasets and then added meta-learning-initialization to the best of these. Due to the conditional hyperparameters in the scikit-learn space we excluded Spearmint, which – without modification – is known to perform poorly in their presence (Eggensperger et al. 2013).

Figure 1 presents the qualitative performance of all optimizers on three representative datasets. The plots show the mean of the best function values for one optimizer obtained up to a given number of function evaluations. Overall, we found SMAC to outperform both TPE and random search for this large hyperparameter space, confirming the results of Eggensperger et al. (2013). We thus applied our meta-learning initialization to SMAC, but would also expect TPE to benefit from it.

Figure 1 also compares qualitative results of MI-SMAC to the three baselines. In the left plot, the meta-learning suggestions were reasonable and thus lead to MI-SMAC successively improving them over time. In the middle plot the second configuration suggested by meta-learning was already the best, leaving no room for improvement by SMAC. The right plot highlights the fact that meta-learning can also fail and decrease SMAC's performance.

Next, we analyzed MI-SMAC's performance using the same ranking-based evaluation as Bardenet et al. (2013) to aggregate over datasets. For each dataset and each function evaluation, we computed the ranks of the three baselines and the two MI-SMAC variants. More precisely, since we had 10 runs of each of the five methods available for each dataset (which give rise to 10^5 possible combinations), we drew a bootstrap sample of 1000 joint runs of the five 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

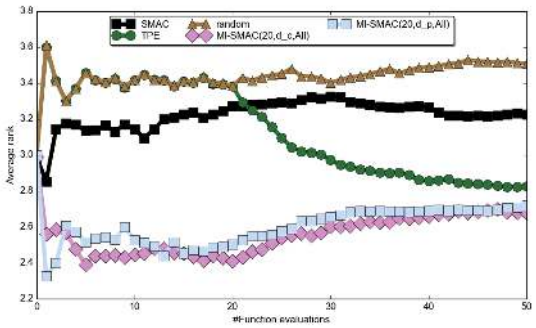


Figure 2: Ranks of various optimizers averaged over all datasets for the CASH problem in scikit-learn.

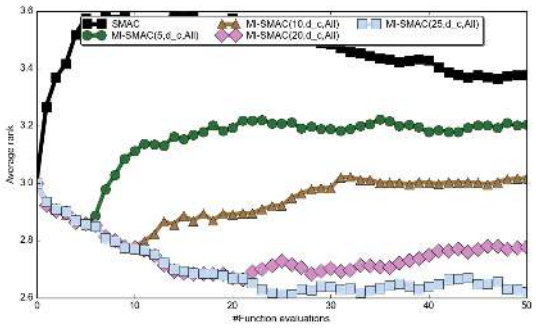


Figure 3: Ranks of SMAC and various MI-SMAC variants averaged over all datasets for the CASH problem in scikit-learn.

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 the ranks do not reflect the magnitude of the difference between raw function values.

As Figure 2 shows, the two variants of MI-SMAC performed best, converging to similar ranks with larger function evaluation budgets; and meta-learning yielded dramatically better results for very small function evaluation budgets. We also note that even after 50 function evaluations no SMBO method had fully caught up to the MI-SMBO results. This indicates that meta-learning initialization provided not only good performance with few function evaluations but also a good basis for SMAC to improve upon further.

To demonstrate the effect of varying the number of initial configurations t selected by meta-learning, we plotted the ranks of different instantiations of MI-SMAC in Figure 3. We observe that within the range of t we studied MI-SMAC performs better with more initial configurations.

To complement the above ranking analysis, Figure 4 (top) quantifies on how many datasets MI-SMAC with a learned distance performed significantly better than the other methods according to a two-sided t-test, while Figure 4 (bottom) shows the statistically significant losses. Both of these quantities are plotted over time, as the function evaluation budget increases.

Compared to the optimizers without meta-learning, MI-SMAC performed much better from the start. Even after 50 iterations, it performed significantly better than TPE on 28% of the datasets (in 11% worse), better than SMAC on 35% of the datasets (in 7% worse), and better than random search on 43% of the datasets (in 9% worse). We would like to point out that the improvement MI-SMAC yielded over SMAC is larger than the improvement that SMAC yielded over random search (in 20% better). We attribute this success to the large search space for this problem, which not even SMAC can effectively search in as little as 50 function evaluations. Leveraging successful optimizations from previous datasets clearly helped SMAC in this complex search space.

Warmstarting Spearmint for Optimizing SVMs

To test the generality of our approach we performed an additional experiment on a lower dimensional problem; optimizing the hyperparameters of an SVM on all 57 datasets using Spearmint. We expected Spearmint to yield the best results for this problem as it is known to perform well in cases where the hyperparameters are few and real-valued (Eggensperger et al. 2013). A statistical analysis using a two-sided t-test on the performances for each of the 57 datasets confirms this hypothesis, as Spearmint indeed significantly outperformed TPE, SMAC, and random search in 32%, 44%, and 52% of the datasets, respectively, and only lost in 7%, 8%, and 9% of the cases, respectively.

The ranking plot in Figure 5 shows the performance of Spearmint and two MI-Spearmint variants compared to SMAC, TPE and random search. As this plot shows, the three variants of Spearmint performed best, converging to a similar rank with larger function evaluation budgets. While meta-learning yielded considerably better results for small function evaluation budgets, after about 10 evaluations Spearmint caught up.

As for the scikit-learn benchmark, we also evaluated the effect of using different values of t and plotted these in Figure 6. In contrast to the results for scikit-learn, for this benchmark it was better to use less configurations suggested by meta-learning. In both benchmarks, however, MI-SMBO yielded substantial performance gains over SMBO during the first function evaluations.

Related Work and Possible Extensions

Existing work on using meta-learning for hyperparameter optimization roughly follows two different directions of research. Firstly, Leite, Brazdil, and Vanschoren (2012) developed Active Testing, a method similar to SMBO that reasons across datasets. In contrast to SMBO, Active Testing is a pure algorithm selection method which does not model the effect of hyperparameters (and algorithms) on the results and is limited to a finite number of algorithms. Secondly, meta-learning was used to initialize model-free hyperparameter optimization methods with configurations that previously yielded good performance on similar datasets (Reif, Shafait, and Dengel 2012; Gomes et al. 2012). While similar to our

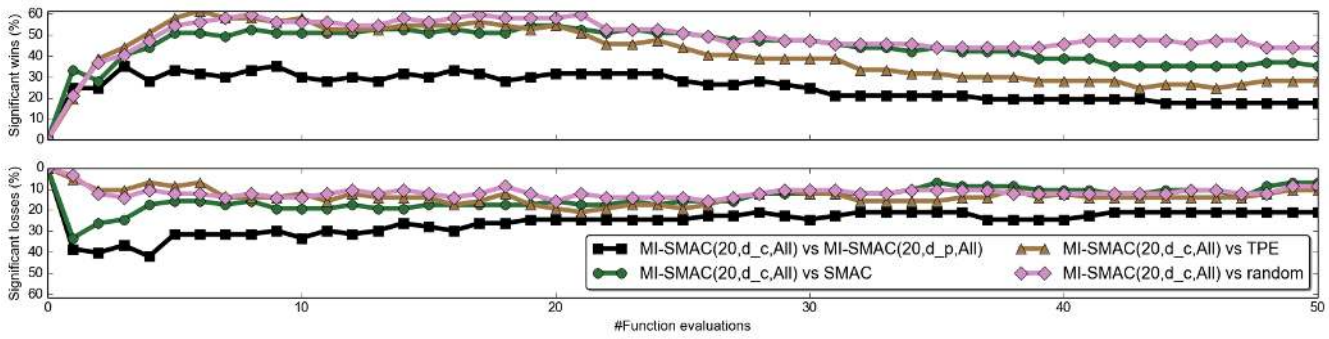


Figure 4: Percentage of wins of MI-SMAC with an initial design of $t = 20$ configurations suggested by meta-learning using the learned distance on all metafeatures. The upper plot shows the number of significant wins of MI-SMAC over competing approaches according to the two-sided t-test while the lower plot shows the statistically significant losses.

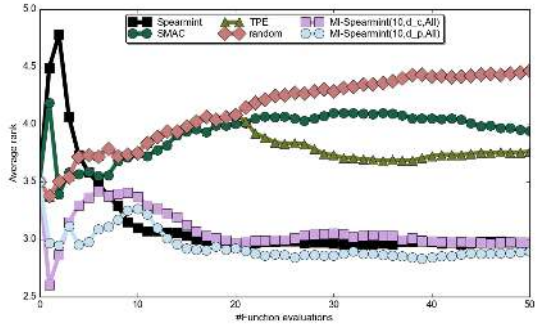


Figure 5: Ranks of various optimizers averaged over all datasets for optimizing the SVM.

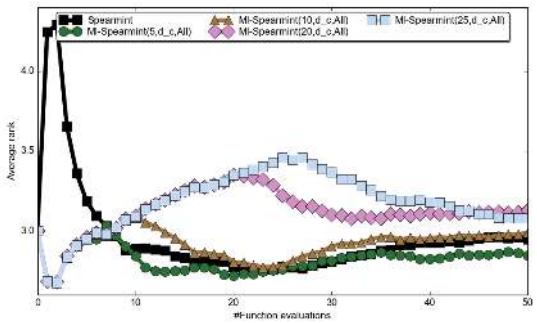


Figure 6: Ranks of Spearmint and various MI-Spearmint variants averaged over all datasets for optimizing the SVM.

work, these methods were limited by their search mechanism and did not improve the state of the art in hyperparameter optimization.

There also exist first attempts to formalize SMBO across several datasets. These collaborative SMBO methods (Bardenet et al. 2013; Swersky, Snoek, and Adams 2013; Yogatama and Mann 2014) address the knowledge transfer directly in the SMBO procedure. However, to date they are limited to small-scale problems with few continuous hyperparameters and a handful of meta-features. In contrast to MI-SMBO they are dependent on the specific SMBO implementation and cannot be readily applied to off-the-shelf hyperparameter optimizers.

Our method's generality opens several avenues for future work. Here, we evaluated MI-SMBO on small and medium-sized hyperparameter optimiziation problems, and an important open research question is to extend it to even larger configuration spaces, such as those of Auto-WEKA (Thornton et al. 2013) and Hyperopt-Sklearn (Komer, Bergstra, and Eliasmith 2014). We also plan to extend collaborative SMBO methods to overcome their limitation to small-scale problems. Finally, it would be interesting to extend our work to general algorithm configuration (Hutter, Hoos, and Leyton-Brown 2011) and to the life-long learning setting (Gagliolo and Schmidhuber 2005; Hutter and Hamadi 2005; Arbelaez, Hamadi, and Sebag 2010).

Conclusion

We have presented a simple, yet effective, method for improving Sequential Model-based Bayesian Optimization (SMBO) by leveraging knowledge from previous optimization runs. Our method combines SMBO with configurations suggested by a meta-learning procedure. It 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 SMBO methods on a collection of 57 datasets. For a low-dimensional hyperparameter optimization problem, for small optimization budgets MI-Spearmint improved upon the current state of the art algorithm Spearmint. For a large configuration space describing a CASH problem in scikit-learn, MI-SMAC substantially improved over the current state of the art CASH algorithm SMAC (and all other tested optimizers), showing the potential of our approach especially for large-scale hyperparameter optimization.

Acknowledgments.

This work was supported by the German Research Foundation (DFG) under grant HU 1900/3-1.

References

Arbelaez, A.; Hamadi, Y.; and Sebag, M. 2010. Continuous search in constraint programming. In *Proc. ICTAI*, 53–60.

Bardenet, R.; Brendel, M.; Kégl, B.; and Sebag, M. 2013. Collaborative hyperparameter tuning. In *Proc. of ICML*, 199–207.

Bergstra, J., and Bengio, Y. 2012. Random search for hyper-parameter optimization. *JMLR* 13:281–305.

Bergstra, J.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Proc. of NIPS*, 2546–2554.

Bergstra, J.; Yamins, D.; and Cox, D. D. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proc. of ICML*.

Brazdil, P.; Giraud-Carrier, C.; Soares, C.; and Vilalta, R. 2008. *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1 edition.

Breiman, L. 2001. Random forests. *Machine Learning* 45:5–32.

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. *CoRR* abs/1012.2599.

Dahl, G. E.; Sainath, T. N.; and Hinton, G. E. 2013. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *Proc. of ICASSP*, 8609–8613.

Eggensperger, K.; Feurer, M.; Hutter, F.; Bergstra, J.; Snoek, J.; Hoos, H. H.; and Leyton-Brown, K. 2013. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization*.

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

Gagliolo, M., and Schmidhuber, J. 2005. Towards life-long meta learning. *Inductive Transfer : 10 Years Later* — NIPS 2005 workshop.

Gomes, T.; Prudêncio, R.; Soares, C.; Rossi, A.; and Carvalho, A. 2012. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing* 75(1):3–12.

Goodfellow, I. J.; Warde-Farley, D.; Mirza, M.; Courville, A.; and Bengio, Y. 2013. Maxout networks. In *Proc. of ICML*, 1319–1327.

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

Hoffman, M.; Shahriari, B.; and de Freitas, N. 2014. On correlation and budget constraints in model-based bandit optimization with application to automatic machine learning. In *Proc. of 15th AISTATS*, volume 33, 365–374.

Hutter, F., and Hamadi, Y. 2005. Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. Technical Report MSR-TR-2005-125, Microsoft Research, Cambridge, UK.

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

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, 507–523.

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

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

Komer, B.; Bergstra, J.; and Eliasmith, C. 2014. Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*.

Leite, R.; Brazdil, P.; and Vanschoren, J. 2012. Selecting classification algorithms with active testing. In *Machine Learning and Data Mining in Pattern Recognition*. Springer. 117–131.

Michie, D.; Spiegelhalter, D. J.; Taylor, C. C.; and Campbell, J., eds. 1994. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *JMLR* 12:2825–2830.

Pfahringer, B.; Bensusan, H.; and Giraud-Carrier, C. 2000. Meta-learning by landmarking various learning algorithms. In *Proc. of ICML*, 743–750.

Rasmussen, C. E., and Williams, C. K. I. 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* 87:357–380.

Snoek, J.; Larochelle, H.; and Adams, R. 2012. Practical bayesian optimization of machine learning algorithms. In *Proc. of NIPS*, 2951–2959.

Soares, C., and Brazdil, P. 2000. Zoomed ranking: Selection of classification algorithms based on relevant performance information. In *Proc. of PKDD'00*. Springer. 126–135.

Swersky, K.; Snoek, J.; and Adams, R. 2013. Multi-task bayesian optimization. In *Proc. of NIPS*, 2004–2012.

Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD'13*, 847–855.

Vanschoren, J.; van Rijn, J. N.; Bischl, B.; and Torgo, L. 2013. OpenML: Networked science in machine learning. *SIGKDD Explorations* 15(2):49–60.

Yogatama, D., and Mann, G. 2014. Efficient transfer learning method for automatic hyperparameter tuning. In *Proc. of AISTATS*, 1077–1085.