Collaborative hyperparameter tuning

Rémi Bardenet[1,2,3] REMI.BARDENET@GMAIL.COM
Mátyás Brendel[2,4] MATTHIAS.BRENDEL@GMAIL.COM
Balázs Kégl[2,3] BALAZS.KEGL@GMAIL.COM
Michèle Sebag[3] SEBAG@LRI.FR

[1]Dept. of Statistics, University of Oxford, 1 South Parks Road, Oxford, OX1 3TG, UK

[2]Linear Accelerator Laboratory (LAL), CNRS/University of Paris Sud, 91405 Orsay, France

[3]Computer Science Laboratory (LRI), CNRS/University of Paris Sud, 91405 Orsay, France

[4]Ferchau Engineering, 88046 Friedrichshaven, Germany

Abstract

Hyperparameter learning has traditionally been a manual task because of the limited number of trials. Today's computing infrastructures allow bigger evaluation budgets, thus opening the way for algorithmic approaches. Recently, surrogate-based optimization was successfully applied to hyperparameter learning for deep belief networks and to WEKA classifiers. The methods combined brute force computational power with model building about the behavior of the error function in the hyperparameter space, and they could significantly improve on manual hyperparameter tuning. What may make experienced practitioners even better at hyperparameter optimization is their ability to generalize *across* similar learning problems. In this paper, we propose a generic method to incorporate knowledge from previous experiments when simultaneously tuning a learning algorithm on new problems at hand. To this end, we combine surrogate-based ranking and optimization techniques for surrogate-based collaborative tuning (SCoT). We demonstrate SCoT in two experiments where it outperforms standard tuning techniques and single-problem surrogate-based optimization.

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

1. Introduction

Hyperparameter tuning is a crucial step in machine learning practice. Recently, it was shown that the state of the art on image classification benchmarks can be improved by configuring existing techniques better rather than inventing new learning paradigms (Pinto et al. 2009, Coates et al. 2011, Bergstra et al. 2011, Snoek et al. 2012, Thornton et al. 2012). Hyperparameter tuning is often carried out by hand, progressively refining a grid over the hyperparameter space. Several automatic hyperparameter tuning methods are already available, including local-search based methods (ParamILS of Hutter et al. 2009), estimation of distribution methods (REVAC of Nannen & Eiben 2007), and surrogate-based methods (Hutter, 2009). Recently, Bergstra et al. (2011) successfully applied surrogate-based optimization methods to tuning numerous hyperparameters of deep belief networks. The method combined brute force computational power with building a model of the behavior of the cost function (validation error) in the hyperparameter space, and it could significantly improve on manual hyperparameter tuning. In a similar setup, Thornton et al. (2012) applied surrogate-based optimization for a large number of classifiers from the WEKA package, outperforming the state of the art on a large set of problems.

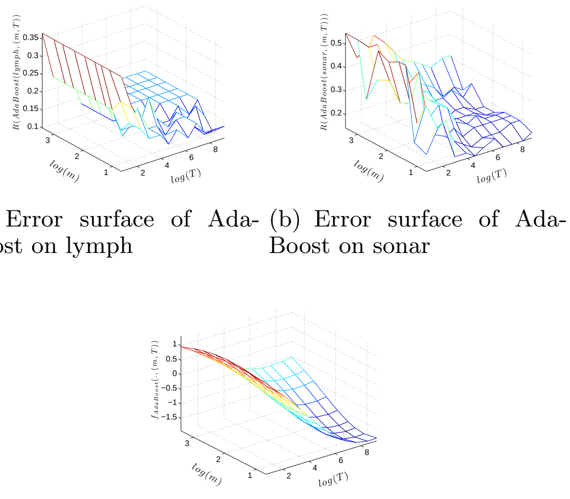
What may still make experienced practitioners better at hyperparameter optimization is their ability to generalize *across* similar learning problems. For example, if somebody in the past successfully applied a classification algorithm $\mathcal{A}$ to the popular MNIST dataset with a given set of hyperparameters x, he or she would certainly use this set as a hint (or "prior") to choose the hyperparameters of $\mathcal{A}$ when tuning $\mathcal{A}$ on a slightly noisy or rotated version of MNIST. Our main contribution is to propose a way to mimic this human behavior when performing an automatic, surrogate-based hyperparameter search.

Brendel & Schoenauer (2011) has recently proposed a similar idea in artificial planning. They tuned an evolutionary algorithm by learning a mapping from problems to hyperparameters using a neural network. This setting may work well if the problem descriptors determine the optimal hyperparameters with high certainty. In machine learning on diverse datasets, however, we cannot hope to come up with a set of easily measurable problem descriptors that can predict the best hyperparameters with no uncertainty. What we propose here instead is to build a model from past experience that can *bias* the search on a new problem towards regions in the hyperparameter space where the optimal hyperparameters are likely to be found. Surrogate-based methods suit well this setup. They also solve efficiently the exploration/exploitation dilemma. Furthermore, combining surrogate-based ranking and optimization techniques, we can define a novel Bayesian optimization method that can be used to collaboratively optimize quantitatively different but similarly behaving objective functions.

1.1. The formal setup

To tackle the problem, we first build a *quality function* $f : \mathbb{D} \times \mathbb{H} \to \mathbb{R}$ that takes a dataset $D \in \mathbb{D}$ and a set of hyperparameters $x \in \mathbb{H}$ as inputs, and outputs the quality $f(D, x)$ of the result $\mathcal{A}(D, x)$ obtained by applying the algorithm $\mathcal{A}$ on the dataset D with hyperparameters x. The hyperparameters are usually numerical so the space $\mathbb{H}$ is naturally $\mathbb{R}^{d_H}$ where d_H is the number of hyperparameters. To apply surrogate optimization on the joint space $\mathbb{D} \times \mathbb{H}$, we will also represent datasets with a real vector of size d_D. These d_D numerical descriptors should be such that datasets with similar descriptors give rise to similar optimal hyperparameters. Coming up with such features is itself an interesting (and, in general, open) research problem. We will describe our approach in Section 4.

Once the feature and dataset representation $\mathbb{D} \times \mathbb{H}$ is fixed, we place a prior over f that incorporates knowledge from previous experiments. The Gaussian process (GP) prior is now a common choice in sequential model-based optimization (SMBO; Jones, 2001; Lizotte, 2008). Since GPs are closed under sampling, they provide a simple and elegant way of accumulating more and more knowledge about the surrogate function as the observations arrive. On the other hand, specifying the quality function f is more subtle. Typically, in single-set hyperparameter optimization, f is a (cross-)validation error $f(D, x) = R\big(\mathcal{A}(D, x)\big)$ as in the setup of Bergstra et al. (2011). The problem with this choice in our multi-set setup is that when applying the same algorithm $\mathcal{A}$ to different problems $D_1, \dots, D_M \in \mathbb{D}$, the errors can differ significantly in scale, which means that f can be non-smooth in its dimensions coming from $\mathbb{D}$ (Figure 1). The raw validation error is thus a poor choice for f. At the same time, similar problems may share a model about *where* the error is minimized in the hyperparameter space. To deal with this issue, we need a quality function that encodes knowledge on the *ranking* of hyperparameters on individual problems, and can convey information such that "if $f(D_1, x_1) < f(D_1, x_2)$ and D_1 is similar D_2, then probably $f(D_2, x_1) < f(D_2, x_2)$", even though the ranges of the errors $R\big(\mathcal{A}(D_1, \cdot)\big)$ and $R\big(\mathcal{A}(D_2, \cdot)\big)$ are very different. We propose therefore to *rank* hyperparameters x_j on each problem D_i by the validation error $R\big(\mathcal{A}(D_i, x_j)\big)$, and take $f : \mathbb{D} \times \mathbb{H} \to \mathbb{R}$ as the surrogate model output by surrogate-based ranking algorithms. This results in a novel and rather uncommon sequential model-based optimization algorithm that redefines its training set at each time step, since new rankings yield a new model.



(a) Error surface of AdaBoost on lymph (b) Error surface of AdaBoost on sonar

(c) The common latent ranker

Figure 1. (a,b) Error surfaces on two similar datasets have similar shapes although the errors are quantitatively different. (c) The similar shapes can be captured by a latent ranker.

The rest of the paper is organized as follows: in Section 2 we give a brief overview of sequential model-based optimization. Section 3 contains our contributions, describing in detail the quality function f and the prior we place over it. Finally, in Section 4 we present two experimental case studies on AdaBoost and on multilayer perceptrons, where we compare our approach to several standard tuning techniques and single-problem surrogate-based optimization.

2. Sequential model-based optimization

On today's large databases, obtaining a validation error for a given set of hyperparameters requires a training phase that can typically take hours or days. Sequential model-based optimization (SMBO) is a surrogate optimization method suitable for such expensive-to-evaluate target functions. SMBO replaces the target function $f(z)$ by a cheaper-to-evaluate surrogate model $\mathcal{M}$, and iteratively 1) tunes the model and 2) optimizes an auxiliary criterion function $S(z;\mathcal{M})$. The criterion $S(z;\mathcal{M})$ measures the interest of asking the target function value at a new point z, given model $\mathcal{M}$. The optimization paradigm is presented in Figure 2. The ingredients that the user has to provide are a model $\mathcal{M}$ and a criterion S, for which we now present two common choices (which we also used in our experiments).

```
SMBO$\big(f, \mathcal{M}, T, S\big)$
1  $\mathcal{O} \leftarrow \emptyset$
2  for $t \leftarrow 1$ to $T$
3      $z^* \leftarrow \arg\max_z \; S(z;\mathcal{M})$
4      Evaluate $f(z^*)$        $\triangleright$ Expensive step
5      $\mathcal{O} \leftarrow \mathcal{O} \cup (z^*, f(z^*))$
6      Fit a new model $\mathcal{M}$ to $\mathcal{O}$
7  return $\mathcal{O}$
```

Figure 2. The pseudo-code of generic sequential model-based optimization. f is the function to be optimized, $\mathcal{M}$ is an initial surrogate model that is updated when new evaluations of f become available. T is the number of steps, and S is the auxiliary criterion that measures the interest of asking the target function value at a new point z.

2.1. A model: Gaussian processes

Gaussian processes (GPs; Rasmussen & Williams, 2006) are a convenient way of putting priors over functions. It is the most common model of choice for SMBO, mainly since it is *closed under sampling*. This means that if we put a GP prior on f, then the posterior distribution of f given a set $\mathcal{O} \triangleq \{(z_i, f(z_i)), i = 1, ..., N\}$ of observed points is still a GP. Formally, a GP is defined by its mean function $\mu(.)$ and its covariance function $k(.,.)$. The latter is a positive kernel that encodes the degree of smoothness of the sample functions, while the mean function encodes a trend shared by these functions. Basically, writing $f \sim \mathrm{GP}(\mu, k)$ means that marginally, for any M, we have $\big(f(z_1), ..., f(z_M)\big)^T \sim \mathcal{N}(0, Q)$ with $Q \triangleq \big(k(z_i, z_j)\big)_{1\leq i,j\leq M}$. It is common to assume $\mu \equiv 0$ after centering the data. If $f \sim \mathrm{GP}\big(0, k(.,.)\big)$, then the posterior of f given $\mathcal{O}$ is $\mathrm{GP}\big(\widetilde{\mu}, \widetilde{k}(.,.)\big)$, where

$$\begin{aligned} \widetilde{\mu}(z) &= \mathbf{k}_z^T K^{-1}\mathbf{f} \\ \widetilde{k}(z_1, z_2) &= k(z_1, z_2) - \mathbf{k}_{z_1}^T K^{-1}\mathbf{k}_{z_2}, \end{aligned} \qquad (1)$$

with $K \triangleq \big(k(z_i, z_j)\big)_{1\leq i,j\leq N}$, $\mathbf{f} \triangleq \big(f(z_i)\big)_{1\leq i\leq N}$ and $\mathbf{k}_v \triangleq \big(k(v, z_i)\big)_{1\leq i\leq N}$. This property allows to analytically derive the posterior distribution on the value $f(z)$ of f at any test point z, which is univariate Gaussian with mean $\widetilde{\mu}(z)$ and variance $\widetilde{k}(z,z)$. To conclude, note that it is easy to incorporate input-independent evaluation noise by adding a scalar matrix to K in the regression equations (1). Note also that kernel functions have their own hyperparameters, which are usually chosen by maximizing the marginal likelihood of the data given the hyperparameters (see (Rasmussen & Williams, 2006) for details).

2.2. A criterion for SMBO: expected improvement

Armed with this analytical posterior on $f(z)$, we can design criteria that can predict the quality of a new point z. Expected improvement (EI; Jones, 2001) is the most common such criterion, but others also include the probability of improvement of (Jones, 2001), minimizing the conditional entropy of the minimizer (Villemonteix et al., 2006), or, more recently, a criterion based on multi-armed bandits (Srinivas et al., 2010). We will use here the EI criterion, which we now describe formally. Assume we want to minimize a function f on which we put a $\mathrm{GP}(0,k)$ prior. Conditioning on the target function evaluated at some training points $\mathcal{O} \triangleq \{\big(z_i, f(z_i)\big), i = 1, ..., N\}$, we obtain a posterior GP model $\mathcal{M}$ as described in Section 2.1, and the *expected improvement* at z over the best minimum found so far is defined by

$$\mathrm{EI}(z;\mathcal{M}) \triangleq \mathbb{E}\big(\max(m_N - f(z), 0)|\mathcal{F}_N\big),$$

where $\mathcal{F}_N$ is the σ-algebra generated by the previous fitness evaluations $\mathcal{O}$, and $m_N \triangleq \min_{1\leq i\leq N} f(z_i)$.

3. The quality function f and its prior

We will now apply the SMBO framework of Section 2 in the joint space $\mathbb{D} \times \mathbb{H}$ of (dataset, hyperparameters) pairs. To explain our motivation for designing the target "quality" function f and choosing a ranking-based surrogate method, we start by an example. Figures 1(a) and 1(b) show the two-dimensional error surfaces when running AdaBoost.MH of Schapire & Singer (1999) using product weak learners (Kégl & Busa-Fekete, 2009) with two hyperparameters (the number of terms m and the number of boosting iterations T) on two benchmark datasets (the data selection and the experimental setup will be described in detail in Section 4). The datasets are similar in terms of some high-level features (such as the number of instances, number of classes, or number of attributes) and the shapes of the error surfaces are also similar, so it intuitively makes sense to try to fit a common surrogate function onto the surfaces. The errors $R\big(\text{AdaBoost}(D,(m,T))\big)$, however, are quantitatively different in scale for these two different datasets, so direct fitting would fail. To overcome this problem, we will use a ranking surrogate $f_{\text{AdaBoost}}\big(\mathcal{D},(m,T)\big)$ that only defines the relative ordering of error values.

3.1. An SMBO algorithm targeting a latent ranker

Let us fix a learning algorithm $\mathcal{A}$. Let $R\big(\mathcal{A}(D,x)\big)$ be a validation error of algorithm $\mathcal{A}$ applied with hyperparameters $x \in \mathbb{H}$ to problem $D \in \mathbb{D}$. Define a partial order $\prec$ on $\mathbb{D} \times \mathbb{H}$ by

$$(D,x) \prec (D,x') \Leftrightarrow R\big(\mathcal{A}(D,x)\big) \leq R\big(\mathcal{A}(D,x')\big). \quad (2)$$

We say that a function g preserves rankings iff $(D,x) \prec (D,x')$ implies $g(D,x) \leq g(D,x')$. Assume that we are given a smooth function $f : \mathbb{D} \times \mathbb{H} \to \mathbb{R}$ that preserves rankings and whose range does not vary across datasets. f is a better target for an SMBO-based tuning algorithm than raw validation error, since the arbitrary scale factors corresponding to different datasets would not influence the stochastic search. Surrogate-based ranking algorithms, such as SVM$^{\text{RANK}}$ of Joachims (2002) or the GP-based ranking algorithm of Chu & Ghahramani (2005), build an *estimate* $\widehat{f}$ by trying to preserve as many as possible of the *empirical* rankings they are given while keeping $\widehat{f}$ smooth and flat. Thus, our strategy is to feed such a surrogate-based ranking algorithm $\mathcal{B}$ with all available rankings defined by (2). The output $\widehat{f}$ will not estimate f in a classical (e.g., L_2) sense, but it will be a reasonable approximation of f up to a monotonic transformation.

Assuming that $f(D,x)$ is smooth in x is quite natural, and most of the local search and surrogate optimization algorithms are designed based on this hypothesis. In addition, and this is our key assumption, we also suppose that $f(D,x)$ is smooth in D, which means that similar problems produce similarly-looking error surfaces. How problem similarity is defined is a deep and generally unanswered question. In Section 4 we propose two simple setups, but the algorithm described in the following section is generic in the sense that it only assumes that there exists a positive semidefinite kernel representing problem similarity.

We can now describe an SMBO algorithm that approximately optimizes f up to a monotonic transformation. Let us start with a collection $\mathcal{O} = \big((D_i,x_i,R_i)\big)_i$ of (dataset, hyperparameter, validation error) triplets. Note that in Figures 3 and 4, we also abusively denote $\mathcal{O} = (\mathbf{D},\mathbf{x},\mathbf{R})$ to be able to underline the separate roles of each triplet component. Consider the repetition of the following steps. First compute all pairwise rankings $\mathcal{P}$ among pairs (D_i,x_i) in $\mathcal{O}$ according to (2). Then apply a surrogate-based ranking algorithm $\mathcal{B}$ to rankings $\mathcal{P}$, outputting $\widehat{f}$. Evaluate $\widehat{f}$ on all points (D_i,x_i) in $\mathcal{O}$ to yield $\widehat{\mathbf{f}}$, and perform GP regression on $\widehat{f}$ using $\widehat{\mathbf{f}}$ as input. Pick the next point to add to $\mathcal{O}$ by maximizing EI, as described in Section 2.

The procedure described above yields a first Bayesian optimization algorithm presented in Figure 3, named ST for surrogate-based tuning. What makes it a particularly unusual Bayesian optimization algorithm is that the regressed function is different at each time step because $\widehat{f}$ depends on the rankings $\mathcal{P}$ and the size of $\mathcal{P}$ is increasing in each iteration.

```
ST$\big(D,T,\mathcal{O},\mathcal{A},\mathcal{B}\big)$
1  for $t \leftarrow 0$ to $T-1$
2      Compute pairwise rankings $\mathcal{P}$ from $\mathcal{O}$ as in (2)
3      $\widehat{\mathbf{f}} \leftarrow$ evaluation on $(\mathbf{D},\mathbf{x})$ of the surrogate $\widehat{f}$
       built by $\mathcal{B}$ with rankings $\mathcal{P}$
4      $\mathcal{M} \leftarrow$ posterior GP on $\widehat{f}$ knowing $\big((\mathbf{D},\mathbf{x}),\widehat{\mathbf{f}}\big)$
5      $x^* \leftarrow \text{argmax}_{x\in\mathbb{H}}\ EI(D,x;\mathcal{M})$
6      $R^* \leftarrow R\big(\mathcal{A}(D,x^*)\big)$     ▷ Run learning algorithm
7      $\mathcal{O} \leftarrow \mathcal{O} \cup (D,x^*,R^*)$
8  return $\mathcal{O}$
```

Figure 3. The pseudo-code of the surrogate-based tuning algorithm. Input $\mathcal{B}$ denotes a surrogate-based ranking algorithm. $\mathcal{O} = \big((D_i,x_i,R_i)\big)_i \triangleq (\mathbf{D},\mathbf{x},\mathbf{R})$ summarizes available data. The input $\mathcal{O}$ comes from results from past experiments with the same algorithm, and $\mathcal{O}$ is then updated along the algorithm. See text for details.

3.2. Collaborative tuning

Since ST is now independent of the scale of validation error $R(\cdot)$, we can apply it to several problems $D_1, ..., D_M$ at the same time. Instead of repeatedly applying ST (Figure 3), we propose here to tune all problems simultaneously by spending one iteration on each problem in turn, thus making use of all information gained so far *on all problems*. This gives birth to the SCoT algorithm, for surrogate-based collaborative tuning, presented in Figure 4. In Section 5, we come back on other collaborative strategies.

```
SCoT$\big((D_1,\ldots,D_M),T,\mathcal{O},\mathcal{A},\mathcal{B}\big)$
1  for $t \leftarrow 0$ to $T-1$
2    for $i \leftarrow 1$ to $M$
3      Compute pairwise rankings $\mathcal{P}$ from $\mathcal{O}$ as in (2)
4      $\widehat{\mathbf{f}} \leftarrow$ evaluation on $(\mathbf{D},\mathbf{x})$ of the surrogate $\widehat{f}$
       built by $\mathcal{B}$ with rankings $\mathcal{P}$
5      $\mathcal{M} \leftarrow$ posterior GP on $\widehat{f}$ knowing $\big((\mathbf{D},\mathbf{x}),\widehat{\mathbf{f}}\big)$
6      $x^* \leftarrow \operatorname{argmax}_{x\in\mathbb{H}} EI(D_i,x;\mathcal{M})$
7      $R^* \leftarrow R\big(\mathcal{A}(D_i,x^*)\big)$   ▷ Run learning algorithm
8      $\mathcal{O} \leftarrow \mathcal{O} \cup (D_i,x^*,R^*)$
9  return $\mathcal{O}$
```

Figure 4. The pseudo-code of the surrogate-based collaborative tuning algorithm. See the caption of Figure 3 for notations, the only difference being the simultaneous tuning on several datasets $D_1,\ldots,D_M$. See text for details.

3.3. On the choice of a surrogate-based ranking algorithm

The method of choice for Algorithm $\mathcal{B}$ in Figure 4 is the Gaussian process-based ranking algorithm of Chu & Ghahramani (2005) since it provides a way of simultaneously estimating the values of a hidden $\widehat{f}$ and tuning the GP hyperparameters in a double optimization loop. This method, applied with a squared exponential kernel with automatic relevance determination (SE-ARD; Rasmussen & Williams, 2006, pp. 83 and 106), would avoid the need for Step 5 of SCoT and provide an easier interpretation for $\widehat{f}$. However, both our implementation and the one available on the web proved to be too slow in the regime presented in Section 4 to be realistically incorporated in the for loop of SCoT. We thus chose to limit ourselves in Step 4 to an isotropic squared exponential kernel, and used the efficient optimization routines available for the SVM$^{\text{RANK}}$ of Joachims (2002), while Step 5 is carried out with an SE-ARD kernel. Note that SVM$^{\text{RANK}}$ requires that all hyperparameter vectors $x \in \mathbb{H}$ be comparable when applied to the same problem, which is the case here by definition of $\prec$ in (2).

4. Experiments

To assess the performance of SCoT as a practical hyperparameter tuning algorithm, we now present two experiments of tuning classification algorithms. In the first experiment we apply multi-class AdaBoost.MH with two hyperparameters to a large number of heterogeneous UCI problems. In the 2-dimensional hyperparameter space we can pre-compute the validation errors on a finite grid, and we can perform a meta-cross-validation over datasets to assess whether a model tuned on a set of training problems can aid the hyperparameter optimization on a hold-out set of test problems. In the second experiment we tune five hyperparameters of single-hidden-layer neural nets on binary classification problems. In this case precomputing the errors on a grid is impossible. Furthermore, to make the point that problem similarity can help collaborative tuning, we use 20 datasets obtained by perturbing the same five datasets, four times each.

For the sake of clarity, we henceforth call *instance* a point to be classified, and we call *attribute* one of the elements of the input vector. The term *feature* is reserved to numerical descriptors of the whole classification problem, that is, a point in $\mathbb{D}$ is described by a vector of features. Finally note that all GP hyperparameters were tuned maximizing the marginal likelihood (Rasmussen & Williams, 2006).

4.1. A case study on AdaBoost

We describe here a setup that mimics an experienced AdaBoost user facing multi-class classification problems. We used the implementation of AdaBoost.MH available at `multiboost.org` (Benbouzid et al., 2012).

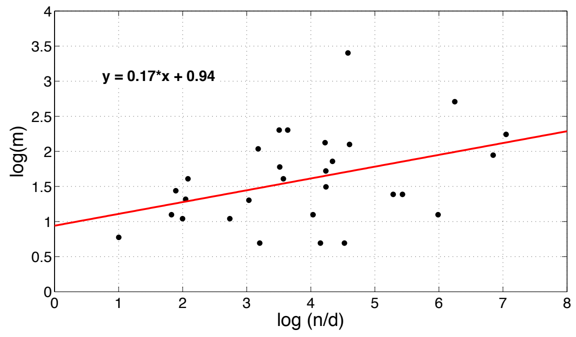


Figure 5. Optimal number of product terms versus the problem feature $\log(n/d)$.

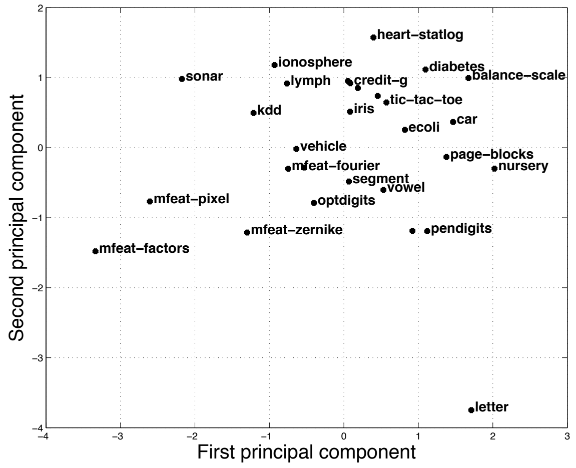


Figure 6. Projection of the problems onto the first two principal components of the feature space. For the sake of visibility some problem names are omitted.

4.1.1. Setup

We downloaded 29 multi-class classification problems from WEKA. We converted nominal attributes to numerical (binary) values using a one-hot encoding. We split each set 80-20% into training and validation sets. We ran AdaBoost.MH with decision products as weak learners (Kégl & Busa-Fekete, 2009), so the two hyperparameters to tune were the number of iterations T and the number of product terms m. Our target measure $R\big(\text{AdaBoost}(\mathcal{D}, (m, T))\big)$ was the classical multi-class 0-1 error. In this setup, because $\mathbb{H}$ has dimension 2, we decided to precompute all validation errors on a 12×9 grid with values $m \in \{2, 3, 4, 5, 7, 10, 15, 20, 30\}$ and $T \in \{2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000\}$, giving us 108 trained models for each dataset. The inputs of the GP kernel were the logarithms of the hyperparameters, rescaled to belong to $[0, 1]$. Figure 1 shows two example error surfaces and a smooth latent ranker learned by the GP.

To embed the problems into a Euclidean space $\mathbb{D}$ in which a GP can be used with a classical squared exponential kernel, we first extracted three simple measures from each dataset, the number of training instances n, the number of classes K, and the number of attributes d, and defined three features K, $\log d$, and $\log(n/d)$ that we found indicative about the value of the best hyperparameters. For example, Figure 4.1 shows that $\log(n/d)$ is correlated to the optimal number of product terms m. This makes sense: the more instances we have compared to the number of attributes, the more complex the optimal classifier can be.

In order to describe some statistical properties of each dataset, a fourth feature ρ was derived through PCA: we extracted the first d' principal components that explained 95% of the variance of each dataset, and divided d' by the number of attributes d to obtain ρ. Table 1 summarizes the statistics of the features, and Figure 6 visualizes the datasets in the feature space projected onto the first two principal components. The first two components explain 73.5% of the variance, which means that the four-dimensional distribution is not degenerate but not completely uniform either. Finally, the features were also rescaled to belong to $[0, 1]$.

feature	min	1st q.	med	3rd q.	max
K	2	2	5	10	26
$\log d$	1.61	2.3	2.94	4.01	5.49
$\log(n/d)$	1	3.03	4.04	4.57	7.05
ρ	0.14	0.5	0.67	0.75	0.86

Table 1. Minimum value, first quartiles, medians, third quartiles and maximum value of the features of the problems.

4.1.2. Different tuning strategies

We designed five experiments, each one mimicking a different user behavior. In all experiments, we used a 5-fold cross-validation over the 29 datasets. To avoid confusion with training and testing in each problem, we call *meta-train* and *meta-test*, respectively, the train and test sets made out of problems.

Global default This experiment mimics the tuning strategy of a non-expert in machine learning who chooses the same hyperparameter vector for every problem. Although this sounds unreasonable, a typical WEKA user often runs classification algorithms using the default hyperparameters. Here we assume that the designer of the classification algorithm is an expert, so he sets the default hyperparameters to those that perform the best on average. Formally, we select the hyperparameter vector that minimizes the average error over the problems in the meta-train sets.

Collaborative default This experiment mimics the tuning strategy of a more experienced user, who builds a model according to his or her experience with the meta-train set, but runs out of time before the conference deadline. She thus uses the surrogate model to predict one vector of hyperparameters for each problem, but does no tuning on the problems. In practice, this strategy is similar to a single iteration of the outer loop of SCoT (Figure 4) and consists in (i) taking the surrogate output by SVM$^{\text{RANK}}$, (ii) regressing it with a GP and

(iii) taking the hyperparameter with the best posterior mean on each meta-test problem.

Separate surrogate tuning This experiment mimics a user who has time for a sequential algorithm, but does not learn either from the meta-train set or from the knowledge acquired in tuning on other meta-test problems simultaneously. State-of-the-art surrogate tuning methods are of this kind. This experiment consists in running a different SMBO algorithm on each meta-test problem, thus using an independent two-dimensional GP for each meta-test problem. To avoid numerical problems, we start by four iterations picking random hyperparameters.

SCoT (surrogate collaborative tuning) This experiment mimics an expert user, who tries to learn a relation between features, hyperparameters, and quality from the meta-train problems he encountered in the past. Here we combine the static model and the surrogate technique, and we add collaborative tuning as described in Section 3 and Figure 4.

Random search This is the baseline experiment that samples points from predefined independent priors on each hyperparameter (uniform in Section 4.1, defined in Table 2 for Section 4.2). This is probably the most common strategy when exhaustive grid search is out of computational reach. Bergstra & Bengio (2012) demonstrated that random search is competitive in hyperparameter tuning for deep belief networks.

Note that the three last strategies were used *without replacement*. For surrogate-based methods, when EI is maximized at a point that has already been evaluated in the past, we replace it by the unevaluated point in the grid with the highest EI. In practice, all strategies should then converge to the best point when the number of evaluations reaches the size of the grid.

4.1.3. Results

We present results in terms of average generalization error in the supplementary material, obtained through a 5-fold cross validation on the set of datasets. Comparing the methods in terms of average generalization error is however questionable for the exact same reason that made us use a ranking-based surrogate: the classification datasets may not be commensurable in terms of test error rate (Demšar, 2006; Lacoste et al., 2012). Hence we computed an average rank score as follows: in each iteration and for each problem, the results of the different strategies were ranked with the so-called fractional ranking ("1 2.5 2.5 4" ranking), ties being rewarded by the average of their ordinal rankings. The ranking points of all meta-test problems were then averaged to get the final score. The average rank of each strategy can then be plotted against the number of trials. Note that the average rank of a single method depends on the results of the others, which means that the score of non-tuning methods (global and collaborative defaults) will increase while their quality values remain constant. Also, note that the lower the average rank score, the better is the method.

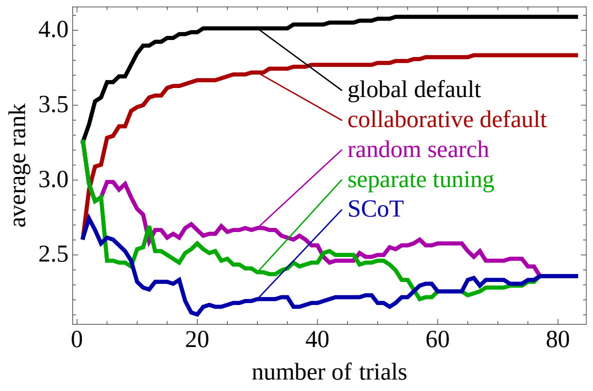


Figure 7. The average rank of the different methods as a function of the number of trials. Collaborative methods start from the value 2.62 and all non-collaborative methods start from 3.26, the first four iterations of separate tuning are the same as random search, see text for details.

Figure 7 shows the results in terms of ranking. The first observation is that the collaborative default achieves a better score than the global default, validating therefore our hypothesis that past experience can help to find better hyperparameters. Secondly, separate tuning beats random search, confirming the results of Bergstra et al. (2011). Finally, SCoT seems to robustly outperform all methods, which means that combining surrogate optimization and collaborative tuning gets the best of both worlds. Note that as the number of iterations grow, all three tuning methods start to saturate the search space, so their average rank converges to the same value.

4.2. A controlled experiment with MLPs

In this experiment we tested SCoT on tuning five hyperparameters of multi-layer perceptrons (MLPs) with a single hidden layer. The hyperparameters and their prior distributions are summarized in Table 2. The goal of the experiment was to test SCoT on a setup where exhaustive grid search is not possible at all due to the higher number of hyperparameters. This experiment also underlines the cotuning abilities of SCoT.

Hyperparameter	Distribution
learning rate	$\log U(10^{-3}, 10)$
ℓ_1 penalty	$\log U(10^{-7}, 10^{-4})$
ℓ_2 penalty	$\log U(10^{-7}, 10^{-4})$
batch size	uniform on $\{1, \dots, 30\}$
number of hidden units	uniform on $\{10, \dots, 100\}$

Table 2. Hyperparameters of a single-layer MLP. The distributions used in the random strategy are given in the second column.

4.2.1. Setup

MLPs are widely used in practice, however, tuning the larger number of hyperparameters is a more important issue than in boosting and SVM, so automatic tuning methods can have a greater practical impact. To demonstrate that SCoT learns a latent structure when such a structure exists, we picked five different binary classification problems out of the collection presented in Section 4.1, and perturbed them in three different ways: *(i)* we suppressed one attribute, *(ii)* we halved the number of samples, and *(iii)* both. In total, we tuned MLPs simultaneously on 20 datasets.

The dataset space $\mathbb{D}$ was built as follows. Since all classification problems were binary, the number of classes K was not relevant here. Besides the three features $\log d$, $\log(n/d)$, and ρ described in Section 4.1.1, we used two more statistical descriptors: the skewness and the kurtosis of each dataset projected onto its first principal component. In the end, $\mathbb{H} \times \mathbb{D}$ has ten dimensions. In the supplementary material, we provide an illustration of a PCA in $\mathbb{D}$ of our 20 datasets.

Unlike in the case study on AdaBoost of Section 4.1, we did not perform meta-cross-validation, rather, we acted as if we used SCoT to tune neural networks simultaneously on these 20 datasets. Methods that build models (collaborative default, separate tuning, and collaborative tuning in Section 4.1.2) started only after the first 10 random points were evaluated on each dataset. The global default strategy was taken here to be the constant choice of the best hyperparameters on average among these first 10 points.

4.2.2. Results

As in Section 4.1.3, results based on generalization error are deferred to the supplementary material. Figure 8 depicts the results in terms of average ranking, as in Section 4.1. Soon after the initial 10 training points, SCoT clearly outperforms all other methods. Separate tuning comes second, but sharing information among problems obviously helps SMBO on this controlled benchmark.

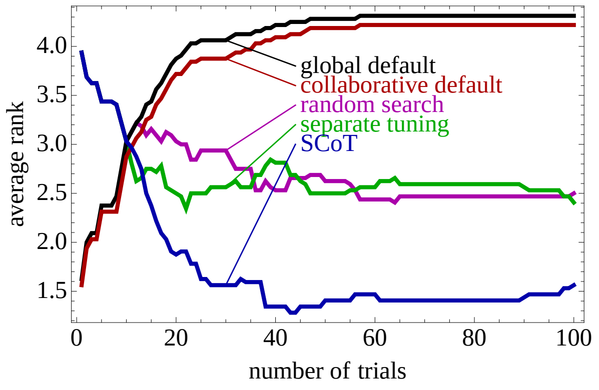


Figure 8. Results on the MLP benchmark in terms of the average "fractional" rankings (see Section 4.1.3).

5. Conclusion

We presented SCoT, a surrogate-based optimization algorithm for hyperparameter tuning. It builds on previous approaches but adds a memory of past experiments and a collaborative tuning ability. Developing these two points led to the combination of surrogate-based optimization and ranking, resulting in a novel Bayesian optimization algorithm whose target is moving with the iteration number. We demonstrated SCoT in two experiments on AdaBoost and MLPs: it outperformed a variety of common tuning strategies, including vanilla surrogate-based optimization.

In order to eventually yield an automatic hyperparameter tuner, some methodological and theoretical efforts are needed. On the methodological side, a comprehensive database of learning problems should be compiled, progressively closing the gap between benchmark problems and real-world applications. On the theoretical side, some efforts of feature construction are needed to define for instance compound parameters, more amenable to build the surrogate quality model. Such augmentations of the training set of SCoT will require careful choices in the methods, maybe leading to lighten the computational burden with cheaper models such as approximate GPs. Finally, we feel that there is room for improvement in designing asynchronous strategies for the collaborative tuning of SCoT, which currently tunes problems in a synchronous way, spending one point on each problem in turn, while it should intuitively spend more evaluations on difficult problems.

Acknowledgments

This work was supported by the ANR-2010-COSI-002 grant of the French National Research Agency.

References

Benbouzid, D., Busa-Fekete, R., Casagrande, N., Collin, F.-D., and Kégl, B. MultiBoost: a multi-purpose boosting package. *Journal of Machine Learning Research*, 13:549–553, 2012.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.

Bergstra, J., Bardenet, R., Kégl, B., and Bengio, Y. Algorithms for hyperparameter optimization. In *Advances in Neural Information Processing Systems (NIPS)*, volume 24. The MIT Press, 2011.

Brendel, M. and Schoenauer, M. Instance-based parameter tuning for evolutionary AI planning. In *Proceedings of the 20th Genetic and Evolutionary Computation Conference*, 2011.

Chu, W. and Ghahramani, Z. Preference learning with Gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning*, pp. 137–144, 2005.

Coates, A., Lee, H., and Ng, A. Y. An analysis of single-layer networks in unsupervised feature learning. In *14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

Demšar, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

Hutter, F. *Automated Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, University of British Columbia, 2009.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, October 2009.

Joachims, T. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2002.

Jones, D. R. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001.

Kégl, B. and Busa-Fekete, R. Boosting products of base classifiers. In *International Conference on Machine Learning*, volume 26, pp. 497–504, Montreal, Canada, 2009.

Lacoste, A., Laviolette, F., and Marchand, M. Bayesian comparison of machine learning algorithms on single and multiple datasets. In *15th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

Lizotte, D. *Practical Bayesian Optimization*. PhD thesis, University of Alberta, 2008.

Nannen, V. and Eiben, A. E. Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 975–980, 2007.

Pinto, N., Doukhan, D., DiCarlo, J. J., and Cox, D. D. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Computational Biology*, 5(11), 11 2009.

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

Schapire, R. E. and Singer, Y. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. Auto-WEKA: Automated selection and hyper-parameter optimization of classification algorithms. Technical report, `http://arxiv.org/abs/1208.3719`, 2012.

Villemonteix, J., Vazquez, E., and Walter, E. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 2006.