Efficient Transfer Learning Method for Automatic Hyperparameter Tuning

Dani Yogatama[*]
Carnegie Mellon University
Pittsburgh, PA 15213, USA
dyogatama@cs.cmu.edu

Gideon Mann
Google Research
New York, NY 10011, USA
gideon.mann@gmail.com

Abstract

We propose a fast and effective algorithm for automatic hyperparameter tuning that can generalize across datasets. Our method is an instance of sequential model-based optimization (SMBO) that transfers information by constructing a common response surface for all datasets, similar to Bardenet et al. (2013). The time complexity of reconstructing the response surface at every SMBO iteration in our method is linear in the number of trials (significantly less than previous work with comparable performance), allowing the method to realistically scale to many more datasets. Specifically, we use deviations from the per-dataset mean as the response values. We empirically show the superiority of our method on a large number of synthetic and real-world datasets for tuning hyperparameters of logistic regression and ensembles of classifiers.

1 Introduction

In order to get a machine learning algorithm to work well in practice, there are often a number of hyperparameters that need to be tuned when training the model, and the best hyperparameter setting is often different for different datasets. Users are also faced with non-trivial choices of which machine learning algorithm to use for the problem at hand. Machine learning experts can draw on their expertise and find a reasonable setting for a new dataset relatively quickly, although for a large dataset it is still imperative to get to the best setting with as few trials as possible. More importantly, end users without much background in machine learning often struggle to tune these hyperparameters.

Over the past few years, the machine learning community has increased exploration into ways to simplify hyperparameter tuning (Bergstra et al., 2011; Snoek et al., 2012; Thornton et al., 2013). Surrogate based model optimization is one promising approach that has been shown to work well provided that we can run enough trials for a given dataset. However, there are many practical scenarios that require an algorithm to return a good hyperparameter setting with very few trials, often less than ten. For example, when providing a machine learning service, it is unattractive in terms of user wait time and provider resource time to run a large number of trials for each user.

One way to speed up the search process is by transferring information from previous trials on other datasets, as has been explored in Bardenet et al. (2013). As noted in their work, a key challenge is that evaluation metrics (e.g., accuracies) are not comparable across datasets. For example, a simple algorithm for a highly skewed classification with 90 percent of the instances from one class can yield 90 percent accuracy, whereas the same algorithm on a harder task with a more balanced class distribution might only yield 60 percent accuracy. However, while the accuracies of the best models may be different in these cases, the *region* of a good hyperparameter setting may still be the same. Brochu et al. (2010) consider a similar setup of learning from multiple sessions in the context of procedural animation design.

The method proposed by Bardenet et al. (2013) is based on a ranking surrogate to approximate the response surface. One drawback of this approach is that constructing the response surface requires running a ranker inside the outer loop of sequential model-based

Appearing in Proceedings of the 17^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2014, Reykjavik, Iceland. JMLR: W&CP volume 33. Copyright 2014 by the authors.

[*]Work done while at Google Research.

optimization. The time complexity of constructing the response surface is then equal to running a surrogate-based ranking algorithm (e.g., SVM Rank (Joachims, 2002), GP-based ranking (Chu & Ghahramani, 2005)) on at most $D\text{Choose}(T, 2)$ points, where D is the number of datasets and T is the number of trials for each dataset. For example, the time complexity of learning SVM Rank with a linear kernel and $D\text{Choose}(T, 2)$ points is the number of iterations times $\mathcal{O}(DT^2)$ with the cutting-plane algorithm (Joachims et al., 2009). As the number of stored datasets increases, reconstructing the response surface may add significantly to the time per trial.

In this work, we propose a novel method of transferring knowledge from previous experiments using deviations from the per-dataset mean. Our method is simple and cheap, requiring only $\mathcal{O}(T)$ time to reconstruct the response surface, *without* needing to run a surrogate-based ranking algorithm. We demonstrate our method's superiority in terms of accuracy compared to other methods that do not transfer knowledge from previous experiments, and in terms of actual runtime compared to the method of Bardenet et al. (2013).

2 Background

2.1 Sequential model-based optimization

Sequential model-based optimization (SMBO) is a framework for optimizing an expensive target function by optimizing a surrogate that is cheaper to evaluate (Hutter et al., 2011). There are two components that need to be specified to use SMBO: the evaluation criterion (acquisition function) and the surrogate model $\mathcal{S}$. For the evaluation criterion of SMBO, we can use: Expected Improvement (EI) (Jones, 2001), maximum probability of improvement (Jones, 2001), minimum conditional entropy (Villemonteix et al., 2006), GP upper confidence bound (Srinivas et al., 2010), or a combination of them (Hoffman et al., 2011). In this work, we use Expected Improvement, one of the most commonly used criteria for SMBO, which has been shown to give good empirical results for hyperparameter tuning. EI is defined as:

$$EI_{y^*}(\mathbf{x}, \mathcal{S}) = \int_{-\infty}^{\infty} \max(y - y^*, 0) p_{\mathcal{S}}(y|\mathbf{x}) d_y$$

where $\mathbf{x}$ is the evaluation point, y^* is the optimal value found so far, and $p_{\mathcal{S}}$ is the probability under model $\mathcal{S}$. Similar to other work on automatic hyperparameter tuning (e.g., Bergstra et al. (2011); Snoek et al. (2012); Bardenet et al. (2013), inter alia), we use a Gaussian Process (Rasmussen & Williams, 2006) as the surrogate model $\mathcal{S}$. Algorithm 1 outlines a typical SMBO algorithm.

Algorithm 1 Sequential Model Based Optimization

```
Input: number of trials $T$, target function f
$\mathcal{S}_0$ = initial surrogate model
$\mathcal{O}_0 = \{\}$
for $t = 1$ to $T$ do
    $\mathbf{x}_t = \text{argmax}_{\mathbf{x}} EI(\mathbf{x}, \mathcal{S}_{t-1})$
    $y_t$ = evaluate $\text{f}(\mathbf{x}_t)$
    $\mathcal{O}_t = \langle \mathbf{x}_i, y_i \rangle_{i=1}^t$
    Add $y_t$ and $\mathbf{x}_t$ to $\mathcal{S}_{t-1}$ to get $\mathcal{S}_t$
end for
```

2.2 Gaussian Process

A Gaussian Process (GP) provides a convenient way to define a distribution over functions. An important characteristic that makes it attractive as a (surrogate) model for sequential-based model optimization is that it satisfies the consistency requirement (Rasmussen & Williams, 2006). The posterior distribution of samples from a GP prior is still a GP. A GP is defined by a mean function $\text{m}(\mathbf{x})$ and a covariance function $\text{k}(\mathbf{x}_i, \mathbf{x}_j)$. If we assume that a function $\text{f}(\mathbf{x})$ follows a Gaussian Process:

$$\text{f}(\mathbf{x}) \sim GP(\text{m}(\mathbf{x}), \text{k}(\mathbf{x}_i, \mathbf{x}_j)),$$

the mean and covariance of the posterior distribution at $\mathbf{z}$ (assuming the prior mean $\text{m}(\mathbf{x}) = \mathbf{0}$, without loss of generality) after $t \geq i, j$ observations become:

$$\text{m}(\mathbf{z}) = \text{k}(\mathbf{z}, \mathbf{x}_{1:t})^\top \mathbf{K}^{-1} \text{f}(\mathbf{x}_{1:t})$$
$$\text{k}(\mathbf{z}, \mathbf{x}_t) = \text{k}(\mathbf{z}, \mathbf{x}_t) - \text{k}(\mathbf{z}, \mathbf{x}_{1:t})^\top \mathbf{K}^{-1} \text{k}(\mathbf{z}, \mathbf{x}_{1:t})$$

where $\mathbf{K}$ is the kernel matrix, $\text{f}(\mathbf{x}_{1:t}) = [\text{f}(\mathbf{x}_i)]_{i=1}^t$ (i.e., $\text{f}(\mathbf{x}_i)$ stacked t times to form a column vector), and $\text{k}(\mathbf{z}, \mathbf{x}_{1:t}) = [\text{k}(z, x_i)]_{i=1}^t$.

3 Transfer learning from previous experiments

3.1 Problem Setup

Unlike Bardenet et al. (2013), we consider the streaming setup where we see one dataset at a time, process the dataset, and move to another dataset. We assume that we cannot store the datasets that we have seen in the past, except for high-level descriptive statistics of these datasets (e.g., number of instances, number of features, number of classes, etc.). In practice, this turns out to be a realistic scenario since privacy and memory concerns would prevent a system from storing all prior datasets.

Notationally, dataset index d is always given as a superscript, whereas trial index t is given as a subscript. We assume that we are only allowed to run T trials for each dataset. The goal is to find a good hyperparameter setting $\mathbf{x}$ for a dataset d to maximize $\mathrm{f}^d(\mathbf{x})$ in as few trials as possible. In addition to dataset statistics, we are also allowed to store pairs of hyperparameter settings and function evaluations resulting from previous experiments (prior datasets and prior trials on the current dataset) to be used by future experiments (future datasets). We outline the main idea of our transfer learning method in §3.2 and discuss it in more detail subsequently.

3.2 Method

At each new dataset d, we fit a surrogate Gaussian Process to a new black-box function f^d, in order to find $\max_{\mathbf{x}} \mathrm{f}^d(\mathbf{x})$. Instead of modeling $\mathrm{f}^d(\mathbf{x})$ directly, we use the deviations from the per-dataset mean as the common response surface values. In our setup, we are then able to directly retain the observations from prior datasets in the surrogate function.

The response value that is used by our GP model is simply:

$$y_t^d = \frac{\mathrm{f}^d(\mathbf{x}_t) - \mu^d}{\sigma^d},$$

where μ^d and σ^d are the dataset d mean and standard deviation respectively. Since we do not know μ^d and σ^d, we use

$$\hat{\mu}^d = \frac{1}{t}\sum_t \mathrm{f}^d(\mathbf{x}_t)$$

$$\hat{\sigma}^d = \sqrt{\frac{1}{t}\sum_t (\mathrm{f}^d(\mathbf{x}_t) - \hat{\mu}^d)^2}$$

based on dataset trial results up to t. Therefore, similar to Bardenet et al. (2013), at each iteration our response surface changes.

Algorithm 2 outlines our transfer learning SMBO algorithm and Figure 1 illustrates the target response surface that we try to reconstruct using deviations from the per-dataset mean.

While this work was under review, a related paper was published by Swersky et al. (2013). In that work, the authors proposed a multi-task Bayesian optimization framework similar in spirit to our work. The main difference is that their method estimates posterior probabilities of parameter estimates (the per-dataset mean and variance) using Monte Carlo methods, whereas we use maximum likelihood estimates.

3.3 Assumptions and Justification

Besides standard assumptions for GP-based methods, we need one additional assumption to justify the above

Algorithm 2 Transfer Learning Sequential Model Based Optimization

```
**Input:** surrogate model $\mathcal{S}^{d-1}$, number of trials $T$, dataset $d$, target function $\mathrm{f}^d$
**Output:** updated surrogate model $\mathcal{S}^d$
$\mathcal{S}_0 = \mathcal{S}^{d-1}$
$\mathcal{O}_0^d = \{\}$
**for** $t = 1$ **to** $T$ **do**
  $\mathbf{x}_t = \mathrm{argmax}_{\mathbf{x}}\, \mathrm{EI}(\mathbf{x}, \mathcal{S}_{t-1})$
  Evaluate $\mathrm{f}^d(\mathbf{x}_t)$
  $\mathcal{O}_t^d = \langle \mathbf{x}_i, \mathrm{f}^d(\mathbf{x}_i)\rangle_{i=1}^t$
  Update $\hat{\mu}^d$ and $\hat{\sigma}^d$
  Reconstruct response surface: recompute $\mathbf{y}_{1:t}^d$
  Replace $\mathbf{y}_{1:t-1}^d$ in the surrogate model $\mathcal{S}_{t-1}$ with the recomputed values
  Add $y_t^d$ and $\mathbf{x}_t$ to $\mathcal{S}_{t-1}$ to get $\mathcal{S}_t$
**end for**
Output $\mathcal{S}_T$
```

method: different datasets produce similarly looking functions, although the location and the scale parameters can be different. If this assumption is satisfied, one explanation of using deviation from the dataset mean as the response surface is that we are using a Gaussian Process model with different means for different datasets. The information transfer occurs from reusing the covariance function across datasets. Let $\mathbf{x}_d$ and $\mathbf{x}_{d+1}$ be trial points for datasets d and $d+1$. Since they are jointly Gaussian random vectors, we have:

$$\begin{bmatrix} \mathrm{f}^d(\mathbf{x}_{1:T}^d) \\ \mathrm{f}^{d+1}(\mathbf{x}_{1:T}^{d+1}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_{1:T}^d \\ \mu_{1:T}^{d+1} \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{\mathbf{x}_{1:T}^d, \mathbf{x}_{1:T}^d} & \mathbf{K}_{\mathbf{x}_{1:T}^d, \mathbf{x}_{1:T}^{d+1}} \\ \mathbf{K}_{\mathbf{x}_{1:T}^{d+1}, \mathbf{x}_{1:T}^d} & \mathbf{K}_{\mathbf{x}_{1:T}^{d+1}, \mathbf{x}_{1:T}^{d+1}}, \end{bmatrix} \right)$$

where $\mu_{1:T}^d$ denotes μ^d stacked T times to form a vector and $\mathbf{K}_{\mathbf{x}_{1:T}^d, \mathbf{x}_{1:T}^d}$ denotes the submatrix of $\mathbf{K}$ for all trials points on dataset d. The conditional distribution of $\mathrm{f}^{d+1}(\mathbf{x}_{1:T}^{d+1}) \mid \mathrm{f}^d(\mathbf{x}_{1:T}^d)$ is therefore a Gaussian distribution with mean:

$$\mu^{d+1} + \mathbf{K}_{\mathbf{x}_{1:T}^{d+1}, \mathbf{x}_{1:T}^d} \mathbf{K}_{\mathbf{x}_{1:T}^d, \mathbf{x}_{1:T}^d}^{-1} (\mathrm{f}^d(\mathbf{x}_{1:T}^d) - \mu^d)$$

and covariance:

$$\mathbf{K}_{\mathbf{x}_{1:T}^{d+1}, \mathbf{x}_{1:T}^{d+1}} - \mathbf{K}_{\mathbf{x}_{1:T}^{d+1}, \mathbf{x}_{1:T}^d} \mathbf{K}_{\mathbf{x}_{1:T}^d, \mathbf{x}_{1:T}^d}^{-1} \mathbf{K}_{\mathbf{x}_{1:T}^d, \mathbf{x}_{1:T}^{d+1}}.$$

However, even if we take into account that different datasets can have different means, the scales of the target function values can still vary greatly. Since covariance is scale dependent, we need to divide $\mathrm{f}^d(\mathbf{x}_{1:T}^d) - \mu^d$ by the dataset standard deviation σ^d to obtain the response value $\mathbf{y}_{1:T}^d$ that is used by our GP model in order to compare results from different datasets.

One key assumption of using GP as the surrogate model is that the function $\mathrm{f} : \mathbb{R}^p \rightarrow \mathbb{R}$ is reasonably

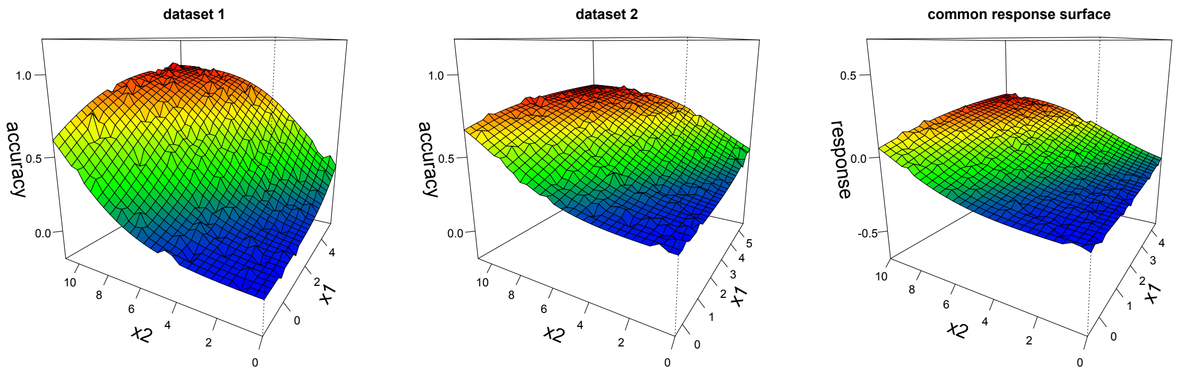


Figure 1: A toy example of a common response surface reconstructed by our method from response surfaces of two datasets. Dataset 1 (left) and dataset 2 (center) produce similar-looking response surfaces, but they are on very different scales. The right plot is an illustration of the common response surface constructed by our method from sample points of these two datasets.

smooth in $\mathbf{x} \in \mathbb{R}^p$.[1] In the context of hyperparameter optimization with an independent GP model for each dataset, this assumption translates to nearby hyperparameter values have similar performance.

For hyperparameter optimization with transfer learning across datasets, in practice, we typically augment the hyperparameter space with dataset features and let $\mathbf{x}$ live in this extended space, i.e., $\mathbf{x} \in \mathbb{R}^{p+q}$, where p is the number of hyperparameters and q is the number of dataset features. This implies that we assume that $\mathrm{f}(\mathbf{x})$ is also smooth in the dataset features (Bardenet et al., 2013). One drawback of this approach is that the performance depends greatly on the dataset features. It is also unclear what dataset features to use, and further research is still needed to investigate the best features. Furthermore, the response surface that is constructed by transfer learning algorithms is an approximation of the true response surface. In our method, we only have access to $\hat{\mu}^d$ and $\hat{\sigma}^d$ instead of μ^d and σ^d, and this approximation might interact with dataset features. We need an approach that is more robust to poor estimates of the response surfaces of *some* datasets.

We propose to lessen the dependencies on dataset features by using multiple kernels, which we describe below.

3.4 Multiple kernel framework

Having access to more information (trials from previous experiments) allows us to define a better kernel than typical kernels used for hyperparameter tuning, and opens up the possibility to bring multiple kernel learning framework into GP-based SMBO. For example, in this work, we consider a convex combination of two kernels, where the first is simply the squared exponential kernel for points in the same dataset, and the second is a nearest neighbor kernel that looks at points from the n nearest neighbor datasets from previous experiments. In both kernels, we have $\mathbf{x} \in \mathbb{R}^p$ (i.e., only in the hyperparemeter space).

The first kernel (SQE) is:

$$\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \theta_0^2 \exp\left(-\frac{1}{2}\frac{\sum_p (x_{i,p} - x_{j,p})^2}{\theta_p^2}\right)$$

We set $\theta_0 = 1$ and optimize θ_p by maximizing the likelihood of the data. As a result, our SQE kernel is an ARD kernel.

The second kernel is:

$$\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 1 - \frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2}{B} & \text{if } \mathbf{x}_j \in \text{nearest neighbor datasets} \\ 0 & \text{otherwise,} \end{cases}$$

where $\|\mathbf{x}\|_2 \leq B$. In other words, the second kernel is one minus the normalized Euclidean distance if the point belong to one of the nearest neighbor datasets, and zero otherwise. We find nearest neighbors by using Euclidean distance in the dataset feature space $\mathbb{R}^q$. The goal is to combine the power of nearest neighbor search and squared exponential kernel with automatic relevance determination.

Notice that we only use the SQE kernel for points in the same dataset, relying on the nearest neighbor kernel to capture information from other datasets. As a result, the transfer of information is more direct. We

[1]This is mainly true if we use the Automatic Relevance Determination (ARD) kernel. There has been work that has used other kernels such as the Matern 5/2 kernel (Snoek et al., 2012). Our discussion focuses on the ARD kernel since it is the most widely used kernel in hyperparameter optimization.

only use dataset features to find the nearest neighbors – instead of estimating exactly how similar two datasets are, which is harder and thus more error-prone. Since each dataset in the nearest neighbors contributes equally, it can also give the algorithm more tolerance to error in approximating the response surface of prior datasets.

We also need not assume that $f(\mathbf{x})$ is smooth in dataset features, although the actual function space becomes more complex since we have a combination of kernels. We view our work as a first step towards using multiple kernel learning framework in hyperparameter tuning.

3.5 Time complexity

We assume that evaluating $f(\mathbf{x})$ is the most expensive step in the algorithm. Our method, similar to Bardenet et al. (2013), needs to reconstruct the response surface at every iteration. Therefore, we want this step to be as cheap as possible.

Our method is very effective since at each iteration t we only need to recompute the current dataset mean and standard deviation. Therefore, the time complexity of reconstructing the response surface is $\mathcal{O}(T)$, where T is the number of trials in one dataset.

On the other hand, Bardenet et al. (2013) needs to run SVM Rank on pairwise differences for per-dataset trials for *all* datasets. There can be $D\text{Choose}(T, 2)$ such pairs in total, where D is the number of datasets used to reconstruct the response surface. For example, using the cutting-plane algorithm (Joachims et al., 2009), the SVM-Rank-iteration time complexity is $\mathcal{O}(DT^2)$ with a linear kernel and $\mathcal{O}(D^2T^4)$ with a non-linear kernel (e.g., RBF) without approximation. Note that the *actual* time complexity of reconstructing the response surface is the number of SVM Rank iterations times the per-iteration complexity above. As we collect more samples from many datasets, this step becomes prohibitively expensive, sometimes even more expensive in practice than the function evaluation step. One possible solution is to limit the number of datasets that contribute to the reconstruction of the response surface. We do not explore this option when comparing our method to Bardenet et al. (2013).

4 Experiments

4.1 Setup

Dataset We collected 42 publicly available classification datasets from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/` (Chang & Lin, 2011). Out of time considerations, we restricted ourselves to datasets that have fewer than 40,000 training instances and 300 features. We assume that the datasets arrive in a fixed order and are processed one by one. Since we do not know the true ordering of the datasets, we randomly shuffle the order of the datasets many times and average the results. For each dataset, we extract the (log) number of instances, the (log) number of features, and the number of classes to be used as dataset features. We split each dataset into training, development, and (blind) test sets. The hyperparameters are tuned on the development set. Assessing performance on only the development sets can be misleading since we might overfit to the development sets. We report the results in terms of performance on both the development and test sets. Table 1 offers descriptive statistics about our collection of datasets.

Table 1: Descriptive statistics about the datasets. The dataset features in our experiments are the number of classes, the log number of features, and the log number of training instances.

Number of	Avg	Min	Max
classes	3.71	2	26
features	93.71	2	300
training instances	5912.12	150	32,561

Baselines We consider three baseline methods for comparison with our algorithm: random search (Bergstra & Bengio, 2012), an SMBO model that only uses information from current dataset (independent GP), and an SMBO model that transfers knowledge from previous experiments using surrogate-based ranking (Bardenet et al., 2013). We implement Bardenet et al. (2013) method with SVM Rank as the ranking algorithm. Due to runtime considerations, we use a linear SVM Rank kernel instead of the isotropic squared exponential kernel used in their paper. One limitation of SVM Rank is that it introduces additional parameters, namely the convergence threshold and another constant for the error penalty of the slack variables. We set the values to 10^{-3} and 1 after slight tuning to get a reasonable trade-off between runtime and performance. These parameters also require extensive tuning in order to get the best performance. Our method, on the other hand, is parameter free and can be applied without separate tuning of additional hyperparameters.

For models that transfer knowledge from previous experiments, we consider two kernels: a squared exponential kernel with automatic relevance determination (SQE) and the multiple kernel framework that we describe in §3.4 (MKL). Therefore, there are 6 models in total that we compare in our experiments. For the models with multiple kernels, we arbitrarily set the kernel weights to 0.3 (SQE) and 0.7 (20 nearest neighbors). Future work can try to learn the optimal kernel weights on the fly. We would also like to note that we ran independent GP using the nearest neighbor kernel on preliminary experiments and found that it gave comparable performance to SQE, so we do not include it here.

Evaluation metric Similar to Bardenet et al. (2013), we use average rank as the evaluation metric. Every method gets a rank for every dataset. The best performing method gets rank 1, the second best 2, and so on. Since there are 6 methods that we compare, for every dataset, each method can get rank 1 to 6. Each competing method is evaluated on *every* dataset that is assumed to arrive in an unknown random order. We compute the average for every run, where a run corresponds to a random order of the datasets.

4.2 Synthetic datasets

In order to test our hypothesis that we can find a good hyperparameter setting faster by using deviations from the per-dataset mean, we perform an experiment using artificially generated datasets. For each of the 42 datasets that we collected, we create 5 perturbed datasets. A perturbed dataset is created by randomly removing features or instances from a source dataset. As a result, we have 210 synthetic datasets, each 5 of these are obviously related since they are generated from one source dataset. If the methods that we use are able to transfer information from previous experiments, they will surely perform better than the independent GP method and random search. The relationships among synthetic datasets generated from different real-world datasets are less clear. However, in this experiment, if a method can make use of information from previous datasets, we still expect a performance improvement over non-transfer learning methods.

We run our experiment using logistic regression as the classifier. Logistic regression is one of the most widely used machine learning algorithms. In our setup, there are four hyperparameters to tune: the ℓ_1 penalty, the ℓ_2 penalty, the maximum number of iterations, and the objective value convergence threshold. Therefore, our logistic regression is an elastic net (Zou & Hastie, 2005). For each dataset, each method gets 1+5 trials. A trial corresponds to a function evaluation for a given hyperparameter setting. We set the first trial point to be always the same for non-transfer learning methods: random search and independent GP. Transfer learning methods can make use of information from previous datasets for choosing the first trial point if they have seen more than one dataset ($d > 1$), so the evaluated point can be different. The goal of this experiment is to simulate a machine learning service that receives numerous requests and needs to obtain reasonably good performance as fast as possible. We compute the average for 5 different random orders of the datasets (5 runs).

Table 2: Average ranks and standard deviations on the development and test sets for logistic regression experiments on synthetic datasets. They are averaged over 5 different random orders of the datasets. Note that standard errors are $\sqrt{5}$ times smaller than standard deviations here. The improvements for our methods (MKL and SQE) over the independent GP method for the test set are statistically significant ($p < 0.05$, student's t-test).

Method	**Logistic regression**	
	Dev	**Test**
Random	3.98 ± 0.11	3.94 ± 0.12
Independent GP	3.55 ± 0.11	3.58 ± 0.13
Bardenet et al., 2013 (SQE)	3.76 ± 0.19	3.75 ± 0.17
Bardenet et al., 2013 (MKL)	3.15 ± 0.48	3.18 ± 0.43
Our method (SQE)	3.36 ± 0.14	3.34 ± 0.15
Our method (MKL)	3.17 ± 0.22	3.19 ± 0.20

Table 2 shows the average rank of each competing method on these synthetic datasets. The results clearly indicate that deviations from the per-dataset mean is an effective method to transfer information from previous experiments. As we can see, both the SQE and MKL variants of our method significantly outperformed independent GP. The Bardenet et al. (2013) method with MKL also performed well, but the SQE variant did not. Notice also the high standard deviation of the MKL variant, indicating that it is less robust to the dataset ordering.

Figure 2 shows the average rank over time for each of the competing methods on one run of the experiment. In general, we can see that after seeing about 75 datasets the average ranks level off. An interesting observation is that the average rank of our method with SQE kernel still decreases, whereas our method with MKL does not. The MKL method only looks at 20 nearest neighbor datasets, so the effect of seeing more datasets is not as significant as the SQE method which looks at all datasets. One possible direction for future work is to investigate the effect of increasing the size of the nearest neighbors to take advantage of more datasets.

4.3 Real-world datasets

We perform two experiments using 42 real-world datasets described in §4.1. Note that unlike in the synthetic dataset experiment, we have no idea beforehand if any of these datasets are related and there is information that can be transferred across them.

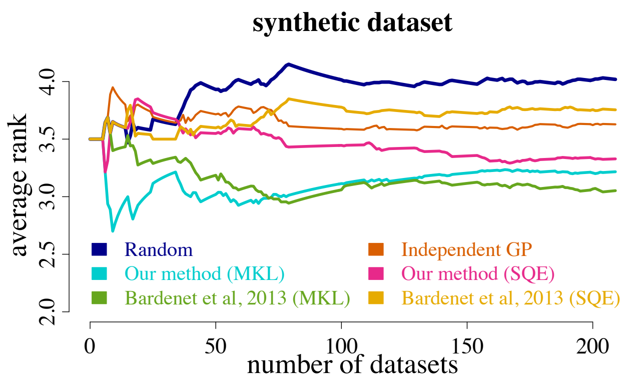


Figure 2: Average ranks over time for one of the runs on 210 synthetic datasets.

Logistic regression In the first experiment, we use logistic regression as our classifier. Similar to the synthetic dataset experiment, there are four hyperparameters to tune: the ℓ_1 penalty, the ℓ_2 penalty, the maximum number of iterations, and the objective value convergence threshold. In this experiment, each method gets 1+10 trials for each dataset. The treatment of the first trial for each method in each dataset is similar to that in the synthetic dataset experiment. Table 3 shows the average rank on 42 datasets, averaged over 10 different random orders of the datasets (10 runs), on the development set and the test set.

Ensemble of classifiers In the second experiment, we consider an ensemble of seven classifiers; with 23 hyperparameters to tune. The classifiers that we consider are: Winnow (Littlestone, 1988), perceptron, logistic regression, Naive Bayes, online Support Vector Machines, an online passive-aggresive classifier (Crammer et al., 2006), and IKPamir.[2] Each classifier has one to six hyperparameters to tune. The classifiers are combined using weighted majority voting, so in addition to individual algorithm hyperparameters, we also search for classifier voting weights. Therefore, we have 30 hyperparameters in total; see Table 4 for details. Similar to the logistic regression experiments, each method gets 1+10 trials for each dataset. We report average rank on 42 datasets over 10 different random orders of the datasets (10 runs) in Table 3.

Results and discussion The results clearly demonstrate the benefits of information from prior datasets, especially in the MKL case. In the logistic regression experiments, our method and Bardenet et al. (2013) with the SQE kernel failed to improve over independent GP. One possible explanation is that since there are only 4 hyperparameters, 11 trials are sufficient. Another possible explanation is that SQE kernel re-

[2]IKPamir is a combination of Crammer et al. (2006) and Maji et al. (2008). (Hector Ye; personal correspondence).

Table 4: 23 hyperparameters in our ensemble of classifiers experiments. The minimum and maximum values were chosen with runtime and performance trade-off as a criterion. There are additional 7 hyperparameters corresponding to classifier weights in the probability simplex for each of these classifiers.

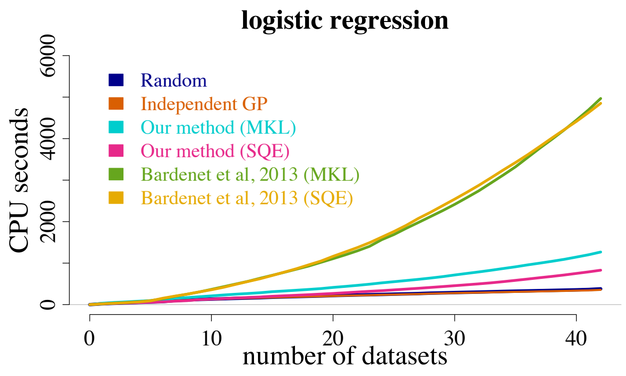
Classifier	Hyperparameter	Min	Max
Winnow	number of epochs	2	20
Winnow	conv. threshold	10^{-4}	1
Winnow	bias conv. threshold	10^{-4}	1
Winnow	margin	10^{-4}	1
perceptron	average	0	1
perceptron	number of epochs	5	15
log. reg.	ℓ_1 penalty	0	10
log. reg.	ℓ_2 penalty	0	10
log. reg.	max iteration	50	500
log. reg.	conv. threshold	10^{-7}	10^{-3}
naive bayes	min probability	10^{-4}	10^{-2}
online SVM	polynomial degree	1	20
online SVM	polynomial bias	10^{-2}	10
online SVM	lambda	0.8	1.2
online SVM	max iteration	100	20,000
online SVM	mini batch size	1	5
online SVM	ℓ_2 penalty	0	100
pass. aggr.	sensitivity	0.5	1.5
pass. aggr.	relaxation	0.5	1.5
pass. aggr.	number of epochs	5	15
IKPamir	number of bins	5	100
IKPamir	learning rate	0.1	0.9
IKPamir	number of epochs	3	8

quires a crucial assumption that the function is also smooth in dataset features, as described in §3.3. We only used three dataset features: (log) number of instances and features, and number of classes, which might not satisfy this assumption in practice. The method based on multiple kernel learning, however, is more robust since it only uses dataset features to find nearest neighbors and does not use the features directly in the GP kernel. We also found that for our method, randomizing some elements in $\mathbf{x}$ (which is returned after maximizing EI) improved the performance. One difficulty in our method is obtaining good estimates of the mean $\hat{\mu}^d$ and standard deviation $\hat{\sigma}^d$ while not wasting trials. Simply using unperturbed $\mathbf{x}^d_{1:t}$ to estimate the dataset mean and standard deviation introduced bias, since the $\mathbf{x}$ that maximizes the EI is biased towards points that are predicted to have good performance (accuracy). The rationale for randomization is to reduce the bias. For all our experiments, we use uniform randomization with replacement probability 0.25 (i.e., an element of $\mathbf{x}$ is replaced, with probability 0.25, by a new value uniformly sampled from the corresponding hyperparameter space).

Figure 3 shows runtime comparisons in CPU seconds for one of the runs of logistic regression and ensemble

Table 3: Average ranks and standard deviations on the development and test sets for logistic regression experiments (left) and ensemble of classifiers experiments (right). They are averaged over 10 different random orders of the datasets. Note that standard errors are $\sqrt{10}$ times smaller than standard deviations here. The improvement for our method (MKL) over the independent GP method for the test set is statistically significant ($p < 0.05$, student's t-test).

Method	Logistic regression		Ensemble of classifiers	
	Dev	Test	Dev	Test
Random	3.93 ± 0.19	3.89 ± 0.16	3.85 ± 0.13	3.77 ± 0.17
Independent GP	3.42 ± 0.09	3.43 ± 0.11	3.62 ± 0.19	3.55 ± 0.29
Bardenet et al., 2013 (SQE)	3.51 ± 0.18	3.50 ± 0.18	3.75 ± 0.26	3.57 ± 0.20
Bardenet et al., 2013 (MKL)	3.20 ± 0.46	3.18 ± 0.40	3.11 ± 0.24	3.31 ± 0.24
Our method (SQE)	3.85 ± 0.31	3.78 ± 0.20	3.58 ± 0.13	3.50 ± 0.24
Our method (MKL)	3.08 ± 0.17	3.23 ± 0.21	3.10 ± 0.31	3.29 ± 0.26



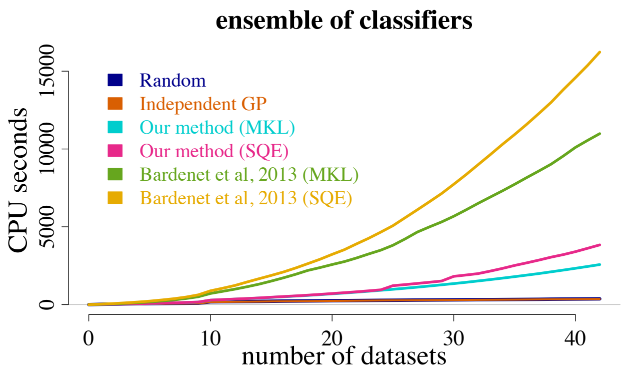


Figure 3: Runtime comparisons (CPU seconds) for one of the runs in logistic regression experiments (left) and ensemble of classifiers experiments (right).

of classifiers. We can see that the runtime of Bardenet et al. (2013) grows significantly faster with the number of datasets, whereas the runtime of our method grows almost linearly. For example, in the logistic regression experiment, increasing the number of seen datasets from 20 to 40 leads to about five times increase in time (≈1,000 CPU seconds to ≈5,000 CPU seconds) for Bardenet et al. (2013), whereas our method (MKL) goes from ≈400 CPU seconds to ≈1,000 CPU seconds.

In practice, running a surrogate ranking algorithm to reconstruct the common response surface can be very expensive with a large number of datasets. The goal of SMBO is to optimize an expensive target function by optimizing its surrogate that is *cheaper* to evaluate. When constructing the surrogate becomes too expensive, it nullifies the benefits of SMBO.

An interesting direction is to compare our method and Bardenet et al. (2013) using a fixed computation resource budget (e.g., time, memory). Since our method is more effective, it can make use of more datasets to potentially provide better predictions. We leave this for future work.

5 Conclusion

We proposed an algorithm for automatic hyperparameter tuning that can generalize across datasets. Our method is an instance of sequential model-based optimization that uses deviations from the per-dataset mean as the response values. We theoretically and empirically showed that the time complexity of reconstructing the response surface at every iteration in our method is significantly less than previous work, with comparable performance. We demonstrated the superiority of our method compared to non-transfer learning methods on a large number of synthetic and real-world datasets for tuning hyperparameters of logistic regression and ensembles of classifiers.

Acknowledgements

The authors thank anonymous reviewers, Thomas Fu, `abe` research group, Nathan Schneider, Brendan O'Connor, and Noah A. Smith for helpful comments and discussions.

References

Bardenet, Remi, Brendel, Matyas, Kegl, Balazs, and Sebag, Michele. Collaborative hyperparameter tuning. In *Proc. of ICML*, 2013.

Bergstra, James and Bengio, Yoshua. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

Bergstra, James, Bardenet, Remi, Bengio, Yoshua, and Kegl, Balazs. Algorithms for hyper-parameter optimization. In *Proc. of NIPS*, 2011.

Brochu, Eric, Brochu, Tyson, and de Freitas, Nando. A Bayesian interactive optimization approach to procedural animation design. In *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2010.

Chang, Chih-Chung and Lin, Chih-Jen. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3): 27:1–27:27, 2011.

Chu, Wei and Ghahramani, Zoubin. Preference learning with gaussian processes. In *Proc. of ICML*, 2005.

Crammer, Koby, Dekel, Ofer, Keset, Joseph, Shalev-Shwarts, Shai, and Singer, Yoram. Onlive passive-aggresive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

Hoffman, Matthew, Brochu, Eric, and de Freitas, Nando. Portfolio allocation for bayesian optimization. In *Proc. of UAI*, 2011.

Hutter, Frank, Hoos, Holger H., and Leyton-Brown, Kevin. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, 2011.

Joachims, Thorsten. Optimizing search engines using clickthrough data. In *Proc. of KDD*, 2002.

Joachims, Thorsten, Finley, Thomas, and Yu, Chun-Nam John. Cutting-plane training of structural SVMs. *Machine Learning Journal*, 77(1):27–59, 2009.

Jones, Donald R. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–385, 2001.

Littlestone, Nick. Learning quickly when irrelevant attributes are abound: A new linear threshold algorithm. *Machine Learning*, 2:285–318, 1988.

Maji, Subhransu, Berg, Alexander C., and Malik, Jitendra. Classification using intersection kernel support vector machines is efficient. In *Proc. of CVPR*, 2008.

Rasmussen, Carl Edward and Williams, Christopher K. I. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Snoek, Jasper, Larrochelle, Hugo, and Adams, Ryan P. Practical bayesian optimization of machine learning algorithms. In *Proc. of NIPS*, 2012.

Srinivas, Niranjan, Krause, Andreas, Kakade, Sham, and Seeger, Matthias. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. of ICML*, 2010.

Swersky, Kevin, Snoek, Jasper, and Adams, Ryan P. Multi-task bayesian optimization. In *Proc. of NIPS*, 2013.

Thornton, Chris, Hutter, Frank, Hoos, Holger H., and Leyton-Brown, Kevin. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD*, 2013.

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

Zou, Hui and Hastie, Trevor. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67(2):301–320, 2005.