Multi-Fidelity Automatic Hyper-Parameter Tuning via Transfer Series Expansion [*]

Yi-Qi Hu[1,2], Yang Yu[1], Wei-Wei Tu[2], Qiang Yang[3], Yuqiang Chen[2], Wenyuan Dai[2]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[2]4Paradigm Inc., Beijing, China
[3]Hong Kong University of Science and Technology, Hong Kong, China
{huyq,yuy}@lamda.nju.edu.cn, tuweiwei@4paradigm.com

Abstract

Automatic machine learning (AutoML) aims at automatically choosing the best configuration for machine learning tasks. However, a configuration evaluation can be very time consuming particularly on learning tasks with large datasets. This limitation usually restrains derivative-free optimization from releasing its full power for a fine configuration search using many evaluations. To alleviate this limitation, in this paper, we propose a derivative-free optimization framework for AutoML using multi-fidelity evaluations. It uses many low-fidelity evaluations on small data subsets and very few high-fidelity evaluations on the full dataset. However, the low-fidelity evaluations can be badly biased, and need to be corrected with only a very low cost. We thus propose the *Transfer Series Expansion (TSE)* that learns the low-fidelity correction predictor efficiently by linearly combining a set of base predictors. The base predictors can be obtained cheaply from down-scaled and experienced tasks. Experimental results on real-world AutoML problems verify that the proposed framework can accelerate derivative-free configuration search significantly by making use of the multi-fidelity evaluations.

Introduction

Machine learning is an important subfield of artificial intelligence which discovers knowledge via experience. During the past decades, machine learning has achieved great successes in many applications, e.g., computer vision (Bradski 1998), recommender system (Broder 2008), financial market analysis (Ball 2013), and so on. But the model configuration with best performance has to be customized for different learning problems. And it crucially relies on human machine learning experts. Lack of machine learning expert limits the range of machine learning applications. Automatic machine learning (AutoML) is proposed to choose model configuration without any human participation. Recently, AutoML is often considered as the *combined algorithm selection and hyper-parameter optimization* (CASH) problem (Feurer et al. 2015). Some derivative-free optimization methods (Liu et al. 2006; Bergstra and Bengio 2012; Shahriari et al. 2015; Hansen, Müller, and Koumoutsakos 2003; Hutter, Hoos, and Leyton-Brown 2011; Munos 2011; Bergstra et al. 2011; Yu, Qian, and Hu 2016) have been applied to solve this problem successfully. Further more, some open-source tools (Feurer et al. 2015; Thornton et al. 2013) based on those methods make application of AutoML easier.

Despite of the preliminary success on AutoML applications, derivative-free optimization still suffers low efficiency issue. Without gradient, derivative-free optimization explores search space by evaluating samples. But evaluations on AutoML are extremely expensive with k-fold cross validation as criterion. For limited time consuming, a small number of evaluations makes it impossible to get a good enough model configuration. Some previous works (Lindauer and Hutter 2018; Hu, Yu, and Zhou 2018) were proposed to reduce the evaluation requirement of derivative-free optimization. On the other hand, some methods were proposed to make evaluations cheaper. One of the most popular method is multi-fidelity optimization (March and Willcox 2012; Fernández-Godino et al. 2016). It decreases total evaluation cost by combing many cheap low-fidelity evaluations and few expensive high-fidelity evaluations. Multi-fidelity optimization has been deeply studied on surrogate model optimization such as Bayesian optimization (Huang et al. 2006; Kandasamy et al. 2016; 2017). Few works (Sen, Kandasamy, and Shakkottai 2018) involve other optimization methods.

In this paper, we propose a general framework to extend multi-fidelity optimization to any derivative-free optimization methods. It should be noticed that AutoML follows multi-fidelity setting naturally by evaluating model configuration on small data subset. But evaluation on a part of dataset is badly biased. Thus, it is necessary to fix the biases for low-fidelity evaluations. In this work we propose to learn a predictor Φ to fix the bias with some observations of low and high-fidelity evaluations. Because of high cost, lack of high-fidelity observations makes it hard to learn the predictor. To tackle this issue, we propose *Transfer Series Expansion (TSE)* method. TSE linearly combines a series of base predictors to Ψ. The base predictors can be obtained from down-scaled and experienced tasks. Because base predictors contain meta knowledge, they can be even transfered among different AutoML problems. From experiments on Light-

[*]This work is supported by the National Key R&D Program of China (2017YFB1001903), NSFC (61876077), Jiangsu SF (BK20160066), and Collaborative Innovation Center of Novel Software Technology and Industrialization. Yang Yu is the corresponding author. This work is partially done when Yi-Qi Hu was an intern in 4Paradigm Inc.

GBM (Ke et al. 2017) hyper-parameter tuning tasks, TSE can effectively fix the biases with only few high-fidelity observations. Multi-fidelity optimization with TSE shows good performance on most of AutoML problems.

The rest of five sections present the background & related works, problem setting, proposed method, experiments and conclusion.

Background & Related Works

Model configuration is often considered as model selection (Brazdil, Soares, and Da Costa 2003; Chapelle, Vapnik, and Bengio 2002; Maron and Moore 1994; Zhao and Yu 2006) and hyper-parameter tunning (Bergstra and Bengio 2012; Snoek, Larochelle, and Adams 2012). Let $A \in \mathcal{A}$ denote a learning model, where $\mathcal{A}$ denotes the model space. Let $\boldsymbol{\delta}_A \in \Delta_A$ denote a hyper-parameter setting of A, where Δ_A is the hyper-parameter space. k-fold cross validation is a popular criterion of model configuration:

$$f(A, \boldsymbol{\delta}_A) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\left(A, \boldsymbol{\delta}_A, \mathcal{D}_i^{\text{train}}, \mathcal{D}_i^{\text{valid}}\right),$$

where $\mathcal{L}(\cdot)$ is the loss function, $\mathcal{D}_i^{\text{train}}$, $\mathcal{D}_i^{\text{valid}}$ denote the train and validation dataset in the i-th fold. In this paper, we focus on hyper-parameter optimization problem for algorithm A: $\boldsymbol{\delta}_A^* = \operatorname{argmin}_{\boldsymbol{\delta}_A \in \Delta_A} f_A(\boldsymbol{\delta}_A)$.

Because of non-differentiable, non-convex and non-continuous properties of AutoML formulation, it is usually solved by derivative-free optimization. Derivative-free optimization is designed for solving sophisticated problems. We consider the minimization problem which is to find $\boldsymbol{x}^* \in \mathcal{X}$ s.t. $\forall \boldsymbol{x} \in \mathcal{X} : f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$, where $\mathcal{X}$ denotes the compact search space, $f : \mathcal{X} \rightarrow \mathbb{R}$ denotes the objective function. In constraint, derivative-free optimization can only query f but can't get any other optimization information. Popular derivative-free optimization methods such as Bayesian optimization (Shahriari et al. 2015), evolutionary optimization (Hansen, Müller, and Koumoutsakos 2003), classification-based optimization (Yu, Qian, and Hu 2016; Hu, Qian, and Yu 2017), etc. are model-based. For examples, Bayesian optimization models the search space by Gaussian process (GP) or tree-structure (Hutter, Hoos, and Leyton-Brown 2011). And classification-based optimization learns a classifier to model search space.

Derivative-free optimization always suffers from efficiency issue. It means that many evaluations will be spent to get a good solution. However, evaluation in AutoML is extremely expensive. Some previous works were proposed to improve the sample efficiency. In (Lindauer and Hutter 2018), the authors proposed warm start techniques. It makes the optimization start from good enough initialization points and models. (Hu, Yu, and Zhou 2018) proposed an experienced directional model which was learned from the previous optimization processes to predict the optimization direction. On the other hand, the methods which decrease the evaluation cost is attacking the researchers' attention. The multi-fidelity optimization (March and Willcox 2012; Fernández-Godino et al. 2016) is one of them. Multi-fidelity setting is applicable where a much cheaper approximated evaluation is available. Most of multi-fidelity optimization methods are studied on Bayesian optimization. Recently, (Sen, Kandasamy, and Shakkottai 2018) extended multi-fidelity step to a tree-search based derivative-free optimization DOO (Munos 2011).

Focusing on big data hyper-parameter tuning tasks, evaluation on full dataset is high-fidelity. Evaluation on a small data subset is low-fidelity. It is biased, but just only takes seconds to minutes. According to PAC learning theorem (Valiant 1984), the evaluation will approach real generalization performance when the training dataset is large enough. Thus, the low-fidelity evaluation is too coarse to apply on optimization directly. In this paper, we propose a general multi-fidelity optimization framework with transfer series expansion (TSE). Based on this framework, optimization with corrected low-fidelity evaluations is practical.

Problem Setting

In this section, we will introduce the multi-fidelity optimization problem setting, and formulate multi-fidelity optimization on hyper-parameter tunning problems.

Multi-fidelity optimization

In multi-fidelity optimization, samples can be evaluated in several different levels. For the simplest situation, there can be two evaluation functions. One function outputs precise evaluation values, e.g., from the full data set, but is quite time-consuming. This is called as the high-fidelity evaluation, denoted as $f_H : \mathcal{X} \rightarrow \mathbb{R}$. Meanwhile, the other function is much cheaper to calculate, e.g., from a data subset, but usually is badly biased. We call this as low-fidelity evaluation, denoted as $f_L : \mathcal{X} \rightarrow \mathbb{R}$. The motivation of multi-fidelity optimization is to use f_L many times instead of f_H, so that the total evaluation cost can be reduced. But because f_L is biased, for each $\boldsymbol{x} \in \mathcal{X}$, the simple regret is used to measure the residual between f_H and f_L:

$$R(\boldsymbol{x}) = f_H(\boldsymbol{x}) - f_L(\boldsymbol{x}).$$

From the simple regret formulation, we can see that, once R is available, we can avoid evaluating f_H by applying $f_L + R$ as a substitution. When we have collected a set of coupled low-fidelity/high-fidelity samples, we can make a dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$ about the low-fidelity and the simple regret, where $y_i = R(\boldsymbol{x}_i)$. It is easy to see that the simple regret function R can be learned by a supervised regression learner Ψ from D. Because of the high evaluation cost of f_H, we can only able to collect a few high-fidelity evaluations in multi-fidelity optimization. The number of instances in D is very small. Therefore, the challenge of this work is to learn an accurate residual predictor from a very small dataset.

Multi-fidelity on hyper-parameter tuning

Hyper-parameter tuning problem meets multi-fidelity setting perfectly. In machine learning process, dataset $\mathcal{D}$ is often split as training set $\mathcal{D}^{\text{train}}$, validation set $\mathcal{D}^{\text{valid}}$ and testing set $\mathcal{D}^{\text{test}}$. For a hyper-parameter configuration $\boldsymbol{\delta}$, the high-fidelity evaluation can be defined as:

$$f_H(\boldsymbol{\delta}) = \mathcal{L}(\boldsymbol{\delta}, \mathcal{D}^{\text{train}}, \mathcal{D}^{\text{valid}}).$$

When $\mathcal{D}$ is a huge dataset, evaluation of f_H is expensive. If we train learning model on a small subset of $\mathcal{D}$, evaluation turns to be very cheap. Let $\mathcal{D}_r^{\text{sub}}$ denote a random chosen subset of $\mathcal{D}$, where r is the subsample ratio. Let $0 < r_L \ll 1$ denote a small subsample ratio. Low-fidelity evaluation can be defined as:

$$f_L(\boldsymbol{\delta}) = \mathcal{L}(\boldsymbol{\delta}, \mathcal{D}_{r_L}^{\text{sub}}, \mathcal{D}^{\text{valid}}).$$

With f_H and f_L, hyper-parameter tuning problem follows multi-fidelity optimization setting. Thus, the core task is how to learn the residual predictor Ψ with some observations of f_H.

Proposed method

We propose a general multi-fidelity optimization framework, which can be easily applied into any derivative-free optimization methods. With some high-fidelity observations, the main idea of proposed framework is to learn a residual predictor to correct the biases of the low-fidelity evaluations during optimization. Because of few high-fidelity evaluations, it is hard to train a accurate predictor. To tackle this issue, we propose the Transfer Series Expansion (TSE) method. TSE trains predictor by transferring a series of base predictors. Let Ψ denote the final predictor and $\boldsymbol{\psi} = \{\psi_1, \psi_2, \ldots, \psi_k\}$ denote a series of base predictors. Linear combination is used to expand $\boldsymbol{\psi}$ to Ψ.

We will present our proposed method as: multi-fidelity optimization framework, details of TSE, acquisition of base predictors and discussion about why TSE works.

Multi-fidelity optimization framework

Previous works on multi-fidelity optimization often rely upon some special optimization structure. In our work, we design a general multi-fidelity framework for any derivative-free optimization methods. We focus on minimization problem. The key step of derivative-free optimization is how to generate a new sample $\boldsymbol{x}$. Let $\texttt{Sample}_{\mathcal{O}}$ denote it, where $\mathcal{O}$ is a derivative-free optimization method. Most of the derivative-free optimization methods are model-based. `Sample` step includes the processes of modeling on $(\boldsymbol{X}, f)$ and sampling based on model. Different methods have different `Sample` steps. In multi-fidelity optimization, the low-fidelity evaluation f_L is introduced to decrease the total evaluation cost. The proposed framework learns a predictor Ψ to estimated the residual between high and low-fidelity evaluations. And then, optimization on the corrected evaluations $(f_L + \Psi)$ can find a good sample which performance is still good on high-fidelity evaluation.

General multi-fidelity derivative-free optimization framework is Algorithm 1. $\boldsymbol{X}_H$ is a sample set. Each $\boldsymbol{x} \in \boldsymbol{X}_H$ is evaluated by f_H. $\boldsymbol{y}$ is a set of regression target according to $\boldsymbol{X}_H$. At the beginning, $\boldsymbol{X}_H$ and $\boldsymbol{y}$ are empty (line 1). Ψ is a predictor to estimate the residual between f_H and f_L. At the beginning (line 2), Ψ can only output 0, because of no learning information. $\boldsymbol{X}_L$ is a set to store all samples that method generates. Initialization step (line 3) is to sample several solutions from $\mathcal{X}$ uniformly. In each iteration, firstly, the corrected low-fidelity evaluation $(f_L + \Psi)$ is considered as the objective function. And T_L samples is generated in this loop (line 5 to 8). And then, a sub-process `Find` chooses a sample to be evaluated by f_H (line 9 to 10). With the high-fidelity evaluated sample, framework constructs the regression dataset (line 11) to re-train the predictor Ψ (line 12). With the growth of t_H, the Ψ is approaching to the real simple regret function. Optimization on corrected low-fidelity evaluation is similar to which on high-fidelity function. At last, algorithm returns the best-so-far sample (line 14).

Algorithm 1 Multi-Fidelity Optimization Framework

```
Input:
    $\mathcal{X}$: The optimization space;
    $f_L, f_H$: Low and high-fidelity evaluation functions;
    $T_H$: The budget of high-fidelity evaluations;
    $T_L$: The low-fidelity evaluation times between high-fidelity evaluations;
    Initialization: Initialization step;
    Sample$_{\mathcal{O}}$: The sample step in optimization method $\mathcal{O}$;
    Find: Select a sample to be high-fidelity evaluated;
    Train: Predictor training process.
Procedure:
 1: $\boldsymbol{X}_H, \boldsymbol{y} = \emptyset, \emptyset$
 2: $\Psi$ = predictor Initialization, $\forall \boldsymbol{x} \in \mathcal{X} : \Psi(\boldsymbol{x}) = 0$
 3: $\boldsymbol{X}_L$ = Initialization($\mathcal{X}$)
 4: for $t_H = 1$ to $T_H$ do
 5:     for $t_L = 1$ to $T_L$ do
 6:         $\boldsymbol{x}$ = Sample$_{\mathcal{O}}$ $(\boldsymbol{X}, f_L + \Psi)$
 7:         $\boldsymbol{X}_L = \boldsymbol{X}_L \cup \boldsymbol{x}$
 8:     end for
 9:     $\boldsymbol{x}'$ = Find($\boldsymbol{X}_L$)
10:     $\gamma' = f_H(\boldsymbol{x}')$
11:     $\boldsymbol{X}_H, \boldsymbol{y} = \boldsymbol{X}_H \cup \boldsymbol{x}', \boldsymbol{y} \cup (\gamma' - f_L(\boldsymbol{x}'))$
12:     $\Psi$ = Train($\boldsymbol{X}_H, \boldsymbol{y}$)
13: end for
14: return $\text{argmin}_{\boldsymbol{x} \in \boldsymbol{X}_H} f_H(\boldsymbol{x})$
```

Transfer Series Expansion (TSE)

In Algorithm 2, the instance number of dataset for training Ψ is small because of the high evaluation cost of f_H. TSE is proposed to make Ψ converge when training dataset is small. We assume that there are a series of pre-trained base predictors $\boldsymbol{\psi} = \{\psi_1, \psi_2, \ldots, \psi_k\}$. A simple way to aggregate the base predictors is linear combination:

$$\Psi(\boldsymbol{x}) = \sum_{i=1}^{k} w_i \psi_i(\boldsymbol{x}) + b.$$

$\boldsymbol{w} = \{w_1; w_2; \ldots; w_k; b\}$ denotes the weight vector of base predictors. And $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$ denotes the raw regression training dataset, where $y_i = R(\boldsymbol{x}_i)$. For each $\boldsymbol{x}_i$, after predicting by each base predictor, the new instance for linear combination is $\boldsymbol{z}_i = \{\psi_1(\boldsymbol{x}_i), \psi_2(\boldsymbol{x}_i), \ldots, \psi_k(\boldsymbol{x}_i), 1\}$. Let $\boldsymbol{Z} = \{\boldsymbol{z}_1; \boldsymbol{z}_2; \ldots; \boldsymbol{z}_m\}$ denote the input matrix for training linear combination, and $\boldsymbol{y} = \{y_1; y_2; \ldots; y_m\}$ denote the learning target. With the mean squared error, the linear combination of base predictors can be defined:

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} \left(\boldsymbol{y} - \boldsymbol{Z}\boldsymbol{w}\right)^{\mathrm{T}} \left(\boldsymbol{y} - \boldsymbol{Z}\boldsymbol{w}\right).$$

If the $\boldsymbol{Z}$ is a full-rank matrix, $\boldsymbol{w}^* = \left(\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\right)^{-1}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{y}$. So that, to learn the combination predictor Ψ, firstly, we use the base predictors to predict on D. And then, the combination regression dataset $\boldsymbol{Z}$ can be got. Secondly, Ψ can be trained on $\boldsymbol{Z}$ with a close-form solution. On training and prediction phases, linear combination has the closed-form solution and the regression dataset is small. Thus, it has high-efficiency to train the linear combination predictor.

The linear combination in TSE makes Ψ have simple structure and easy to train. Another important part of TSE is the set of base predictors. If base predictor contains meta-knowledge of optimization, it can make Ψ converge faster even when training dataset is small. Thus, the next step is how to get the base predictors.

Acquisition of base predictors

In fact, base predictor is a mapping from search space to a real number. A direct idea is training predictors on easier and related regression problems. In this paper, we decompose residual regression into some middle-level problems. In hyper-parameter tuning problem, middle-level problem is easy to construct by introducing middle-fidelity evaluation: $f_M(\boldsymbol{\delta}) = \mathcal{L}(\boldsymbol{\delta}, \mathcal{D}^{\text{sub}}_{r_M}, \mathcal{D}^{\text{valid}})$, where r_M is subsample ratio and $0 < r_L < r_M \ll 1$. The middle level problem is to estimate the residual between f_M and f_L. Because $r_M \ll 1$, evaluations on f_M are still much cheaper than f_H. We can get many middle-fidelity observations. A good enough base predictor can be trained on middle-level regression problem. If we need to train k base predictors, there are k middle-regression problems should be constructed as $\left\{\left(\mathcal{D}^{\text{sub}1}_{r_L}, \mathcal{D}^{\text{sub}1}_{r_M}\right), \left(\mathcal{D}^{\text{sub}2}_{r_L}, \mathcal{D}^{\text{sub}2}_{r_M}\right), \ldots, \left(\mathcal{D}^{\text{sub}k}_{r_L}, \mathcal{D}^{\text{sub}k}_{r_M}\right)\right\}$. In addition, an extra $\mathcal{D}^{\text{sub}}_{r_L}$ is needed to construct the final regression problem $(\mathcal{D}^{\text{sub}}_{r_L}, \mathcal{D}^{\text{train}})$. So that, there are $k+1$ $\mathcal{D}^{\text{sub}}_{r_L}$ and k $\mathcal{D}^{\text{sub}}_{r_M}$ should be sampled randomly.

On a middle-level regression problem $\left(\mathcal{D}^{\text{sub}}_{r_L}, \mathcal{D}^{\text{sub}}_{r_M}\right)$, we will organize regression training dataset D^{M} for ψ. Considering derivative-free optimization property, optimization is a searching process on optimization space. We care more about the regression performance of samples on optimization trajectory. So that, the instances in D^{M} can be collected by applying derivative-free optimization methods to optimize on objective function f_M. After optimization, for each sample $\boldsymbol{\delta}$, it will be evaluated by f_L. The regression target of $\boldsymbol{\delta}$ can be got by $f_M(\boldsymbol{\delta}) - f_L(\boldsymbol{\delta})$. With the labeled regression dataset, it is the supervised regression problem to train ψ. On hyper-parameter optimization problems, the hyper-parameter space is probably discrete, categorical or even mixed. Thus, we choose the random forest regressor as base predictor. Because the datasets for training base predictors can be collected in parallel. It will not spend much time to train base predictors.

However, on huge datasets, accurate base predictors are unavailable because of extremely high time-cost of f_M. For example, we tune hyper-parameters for a learning model on two different datasets $\mathcal{D}_1$ and $\mathcal{D}_2$. $\mathcal{D}_1$ is too huge to get the base predictors. $\mathcal{D}_2$ is a small dataset that we can get the base predictors on it. Because the hyper-parameter space is only defined by learning model. The regression space is same between $\mathcal{D}_1$ and $\mathcal{D}_2$. The base predictors of $\mathcal{D}_2$ contain meta knowledge about the learning model. Thus, they can be transfered to combine the final predictor when optimizing hyper-parameters on $\mathcal{D}_2$. We will verify the effectiveness of base predictors transfer on experiment section.

Algorithm 2 Multi-Fidelity optimization with TSE

```
Input: (extra input than Algorithm 1)
    $\boldsymbol{\psi} = \{\psi_1, \psi_2, \ldots, \psi_k\}$: The base predictor set.
Procedure:
$\boldsymbol{X}_H, \boldsymbol{Z}, \boldsymbol{y} = \emptyset, \emptyset, \emptyset$
$\Psi^{\boldsymbol{\psi}}_{\boldsymbol{w}} = \Psi^{\boldsymbol{\psi}}_{\boldsymbol{0}}$
$\boldsymbol{X}_L$ = Initialization($\mathcal{X}$)
$(\tilde{\boldsymbol{x}}, \tilde{\gamma}) = (\boldsymbol{0}, +\infty)$
for $t_H = 1$ to $T_H$ do
    for $t_L = 1$ to $T_L$ do
        $\boldsymbol{x} = \mathtt{Sample}_{\mathcal{O}}\left(\boldsymbol{X}_L, f_L + \Psi^{\boldsymbol{\psi}}_{\boldsymbol{w}}\right)$
        $\boldsymbol{X}_L = \boldsymbol{X}_L \cup \boldsymbol{x}$
    end for
    $\boldsymbol{x}' = \operatorname{argmin}_{\boldsymbol{x} \in (\boldsymbol{X}_L - \boldsymbol{X}_H)} f_L(\boldsymbol{x}) + \Psi^{\boldsymbol{\psi}}_{\boldsymbol{w}}(\boldsymbol{x})$
    $\gamma' = f_H(\boldsymbol{x}')$
    if $\gamma' < \tilde{\gamma}$ then
        $(\tilde{\boldsymbol{x}}, \tilde{\gamma}) = (\boldsymbol{x}', \gamma')$
    end if
    $\boldsymbol{X}_H,\ \boldsymbol{y} = \boldsymbol{X}_H \cup \boldsymbol{x}',\ \boldsymbol{y} \cup (\gamma' - f_L(\boldsymbol{x}'))$
    $\boldsymbol{Z} = \boldsymbol{Z} \cup \{\psi_1(\boldsymbol{x}'), \psi_2(\boldsymbol{x}'), \ldots, \psi_k(\boldsymbol{x}')\}$
    $\boldsymbol{w} = \left(\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\right)^{-1}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{y}$
end for
return $(\tilde{\boldsymbol{x}}, \tilde{\gamma})$
```

Discussion

The proposed multi-fidelity optimization framework focuses on the evaluation phase of derivative-free optimization. Only evaluation phase is changed in optimization. Thus, this framework can be applied to any derivative-free optimization methods easily. In addition, we propose the transfer series expansion (TSE) to learn the residual predictor. Because high-fidelity evaluation is expensive, training dataset is not big enough to learn a accurate predictor. TSE applied a linear combination of base predictors to simplify the regression model. And the base predictors make the combination predictor avoid starting from nothing. We get the base predictors by constructing middle-level regression problems which is related with the final regression problem. On hyper-parameter tuning problems, middle-level regression problem is to estimate the simple regret between f_M and f_L. The base predictors are aligned by learning model. They can be transfered among different datasets. We just focus on the optimization local trajectory to sample regression training dataset. Thus, only few instances are enough for combination predictor convergence. With precise but cheap estimated evaluations as substitution, optimization can explore more to get a better solution with affordable cost.

Experiments

We implement the proposed multi-fidelity framework with TSE based on classification-based optimization method SRacos (Yu, Qian, and Hu 2016; Hu, Qian, and Yu 2017), and name it TseSRacos. In this section, we apply TseSRacos to tune hyper-parameters of LightGBM on some real datasets. Comparing with other methods, five main conclusions about the proposed framework that we should investigate as follows:

- Optimization on corrected low-fidelity evaluations is necessary. That is to say, the solution with the best low-fidelity evaluation value is usually not best on high-fidelity space;
- Low-fidelity evaluations can be corrected by learning a residual predictor. In another word, the regression predictor can effectively predict the residual betweenf_H and f_L;
- We should investigate that TSE can learn a good predictor even when regression training dataset is extremely small;
- The base predictors have transfer ability among different datasets. It is meaningful for huge dataset hyper-parameter tuning problem in which base predictors are hard to obtain;
- The last but most important conclusion we have to verify is that multi-fidelity optimization with TSE can fast the optimization process. That is to verify, when TseSRacos has the same evaluation budget with high-fidelity optimization, it can get a similar performance but spends less time comparing with high-fidelity optimization.

We firstly show the experiment settings. And then, we verify the conclusions based on empirical result analysis.

Experiment settings

We empirically investigate the proposed method on LightGBM (Ke et al. 2017) hyper-parameter tuning problems. All 11 hyper-parameters in LightGBM are selected including learning rate, number of leaves, tree depth, round number, etc. We optimize hyper-parameters of LightGBM on 12 selected datasets. The details of datasets are showed in Table 1. Some of datasets such as Musk, HTRU2, Magic04, Adult, Sensorless, Connect and Higgs are benchmark datasets from UCI. Rest of them such as Credit, Miniboone, Airline, MovieLens, Criteo come from machine learning competition. They are all real machine learning application datasets. The size of datasets ranges from thousands to 40 million. The subsample ratio settings are showed in Table 1 too. For small datasets which number of instances is less then 100 thousand, the low-fidelity subsample ratio r_L is 0.05, and the middle-fidelity subsample ratio r_M is 0.2. But for big datasets, r_L and r_M depend on dataset size.

Table 1: The information of datasets. $|\mathcal{D}|$ means the number of instances in dataset $\mathcal{D}$. The validation datasets are constructed by sampling 10% instances from $\mathcal{D}^{\text{train}}$. r_L and r_M are subsample ratios of $\mathcal{D}^{\text{sub}}_{r_L}$ and $\mathcal{D}^{\text{sub}}_{r_M}$.

Dataset	$\|\mathcal{D}^{\text{train}}\|$	$\|\mathcal{D}^{\text{test}}\|$	r_L	r_M
Musk	4,991	2,083	0.05	0.2
HTRU2	14,318	3,580	0.05	0.2
Magic04	15,215	3,805	0.05	0.2
Credit	24,000	6,000	0.05	0.2
Adult	32,561	16,281	0.05	0.2
Sensorless	40,883	17,525	0.05	0.2
Connect	47,504	20,053	0.05	0.2
Miniboone	104,052	26,012	0.01	0.04
Airline	773,469	215,358	0.005	0.02
Higgs	10,000,000	1,000,000	0.001	0.004
MovieLens	16,000,210	4,000,053	0.001	0.004
Criteo	40,000,000	4,840,617	0.0005	0.002

We propose other three compared methods besides TseSRacos. First, TseTrans is the TseSRacos but which base predictors are transfered from other dataset. In this section, the base predictors of TseTrans are all transfered from Miniboone. Second, we replace the linear combination predictor with random forest regressor, and name it RFSRacos. Third, we optimize hyper-parameter setting only on the low-fidelity evaluation f_L, and name it LF-Only. In addition, we only optimize on the high-fidelity evaluation f_H, and name it HF-Only. HF-Only is the upper bound of compared methods. Because of high time-cost, HF-Only are only used on some small datasets. For all five methods, the base derivative-free optimization method is SRacos. TseSRacos, RFSRacos and TseTrans follow the proposed multi-fidelity optimization framework.

The evaluation criterion of experiments is AUC score (Fawcett 2006). For multi-fidelity optimization methods, we get one high-fidelity evaluation for every 100 low-fidelity evaluations. That is to say, $T_L = 100$. The total high-fidelity evaluation budget is 50 ($T_H = 50$) in optimization. Thus, there are all 5000 low-fidelity evaluations and 50 high-fidelity evaluations on a multi-fidelity optimization process. For TseSRacos, TseTrans and RFSRacos, there are 5 base predictors on combination predictor. For LF-Only, the low-fidelity evaluation budget is 5000 as the same as it in multi-fidelity optimization. For HF-Only, we test it on small datasets which size is less than 100,000 (Musk, HTRU2, Magic04, Credit, Adult, Sensorless, Connect, Miniboone) with 5000 evaluation times. On big datasets (Airline, Higgs, MovieLens, Criteo), we early stop them when the time they spend is more than three times that TseTrans spends (HF-Only*). For TseTrans, the base regressors are transfered from Miniboone. For huge datasets (MovieLens, Criteo), even middle-fidelity evaluation is unavailable. Thus, we didn't test TseSRacos, but apply TseTrans on them directly.

Empirical analysis

Table 2 shows the AUC score and running wall-clock time of compared methods on each dataset. Especially, for TseSRacos, time cost of training base predictors is added in the total running time. Comparing TseSRacos and LF-Only, Figure 1 illustrates the AUC score curves with running time within limited time zoom (LF-Only gets). To investigate the effectiveness of TSE, we compare the mean regression

Table 2: The AUC performance and wall-clock time of compared methods. LF-Eval and HF-Eval mean the best solution's low and high-fidelity evaluation values. Test means the generalization performance of best solution. The bold number means the best AUC score among compared methods. On TseTrans, the base predictors of Miniboone are transfered to other datasets. Thus, results of TseTrans on Miniboone are empty. HF-Only* means the early-stopped HF-Only on huge datasets.

Dataset	Method	LF-Eval	HF-Eval	Test	Time	Dataset	Method	LF-Eval	HF-Eval	Test	Time
Musk	TseSRacos	0.9018	**0.9991**	0.9977	0:07:31	HTRU2	TseSRacos	0.9733	**0.9841**	0.9632	0:02:44
	TseTrans	0.9204	**0.9991**	**0.9985**	0:07:16		TseTrans	0.9650	0.9758	**0.9636**	0:01:41
	RFSRacos	0.9220	0.9990	0.9980	0:06:40		RFSRacos	0.9773	0.9814	0.9616	0:01:27
	LF-Only	0.9294	0.9989	0.9974	0:05:46		LF-Only	0.9750	0.9791	0.9613	0:02:11
	HF-Only	-	1.0000	0.9978	1:49:08		HF-Only	-	0.9871	0.9645	0:08:48
Magic04	TseSRacos	0.8859	**0.9446**	**0.9236**	0:04:40	Credit	TseSRacos	0.6407	**0.7451**	**0.7612**	0:03:46
	TseTrans	0.9013	0.9438	0.9227	0:03:04		TseTrans	0.6654	0.7432	0.7531	0:01:36
	RFSRacos	0.8994	0.9387	0.9225	0:02:16		RFSRacos	0.6889	0.7404	0.7579	0:01:21
	LF-Only	0.9092	0.9296	0.9201	0:02:45		LF-Only	0.7270	0.7324	0.7531	0:00:59
	HF-Only	-	0.9495	0.9203	0:20:06		HF-Only	-	0.7554	0.7643	0:04:26
Adult	TseSRacos	0.8961	**0.9261**	**0.9219**	0:04:20	Sensorless	TseSRacos	0.9974	**0.9999**	**0.9999**	0:33:29
	TseTrans	0.8896	0.9224	0.9206	0:02:17		TseTrans	0.9973	**0.9999**	**0.9999**	0:21:33
	RFSRacos	0.9086	0.9190	0.9181	0:02:37		RFSRacos	0.9978	**0.9999**	0.9998	0:54:30
	LF-Only	0.9070	0.9157	0.9156	0:02:37		LF-Only	0.9973	0.9997	0.9997	0:19:38
	HF-Only	-	0.9281	0.9234	0:26:02		HF-Only	-	0.9999	0.9999	2:23:44
Connect	TseSRacos	0.8604	0.9318	**0.9374**	0:11:02	Miniboone	TseSRacos	0.9664	**0.9789**	**0.9785**	0:27:53
	TseTrans	0.8650	**0.9319**	0.9353	0:11:14		TseTrans	-	-	-	-
	RFSRacos	0.8630	0.9284	0.9330	0:09:15		RFSRacos	0.9674	0.9787	0.9781	0:12:44
	LF-Only	0.8684	0.9219	0.9272	0:10:32		LF-Only	0.9694	0.9779	0.9771	0:14:54
	HF-Only	-	0.9367	0.9404	1:20:28		HF-Only	-	0.9814	0.9797	0:51:00
Airline	TseSRacos	0.6392	**0.6801**	**0.8893**	0:44:06	Higgs	TseSRacos	0.7743	0.8037	0.8023	14:09:18
	TseTrans	0.6462	0.6674	0.8696	0:40:31		TseTrans	0.7770	**0.8046**	**0.8044**	11:37:50
	RFSRacos	0.6519	0.6674	0.8762	0:35:55		RFSRacos	0.7847	0.8025	0.8035	12:57:22
	LF-Only	0.6566	0.6600	0.8693	0:41:29		LF-Only	0.7872	0.7991	0.7988	8:53:33
	HF-Only*	-	0.6900	0.8961	2:00:00		HF-Only*	-	0.8145	0.8140	45:00:00
MovieLens	TseTrans	0.6344	**0.6682**	0.6476	11:53:56	Criteo	TseTrans	0.7258	**0.7513**	**0.7496**	62:00:25
	RFSRacos	0.6444	0.6543	**0.6591**	11:35:42		RFSRacos	0.7289	0.7454	**0.7496**	65:41:30
	LF-Only	0.6443	0.6477	0.6361	11:10:26		LF-Only	0.7298	0.7480	0.7480	60:52:23
	HF-Only*	-	0.6767	0.6591	36:00:00		HF-Only*	-	0.7652	0.7584	180:00:00

error of TseSRacos and RFSRacos in each regression step, and the results are shown in Figure 2. We will verify conclusions based on those results.

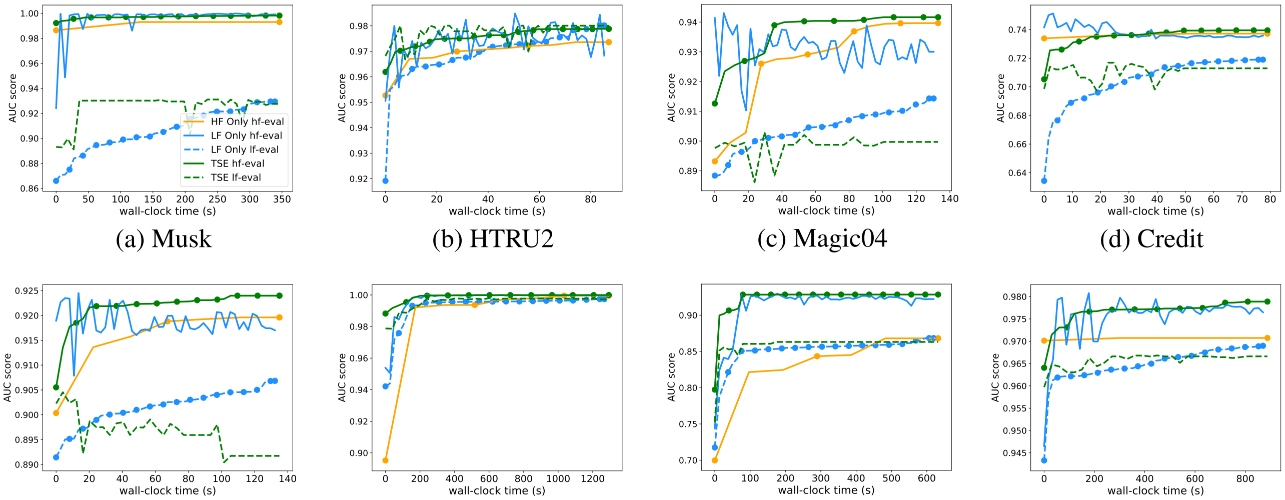
Low-fidelity evaluation correction is necessary. From Table 2, LF-Only usually gets the best low-fidelity evaluation values. However, corresponding high-fidelity evaluations are not good. On Figure 1, comparing the same color solid and dash lines, it is easy to discover that a sample with good low-fidelity evaluation value probably has the bad high-fidelity evaluation value. Focusing on blue solid line, high-fidelity curve is extremely unstable during optimizing on low-fidelity evaluations. Thus, It is necessary to correct the low-fidelity evaluation in optimization.

Correction by regression predictor is effective. From Table 2, the best AUC score that optimizations with correction (TseSRacos, RFSRacos, TseTrans) get is near to the upper bound score (HF-Only gets). They are much better than LF-Only in most of datasets.

TSE makes regression converge fast. On Figure2, we compare the regression error on TSE with random forest regression. At beginning (regression training dataset only has one instance), TSE has big regression error. But the error of TSE decreases fast when data size is more than 5. Especially, the error variance of TSE is much smaller than random forest regression. It means TSE has good stability.

Base predictors can transfer among datasets. From Table 2, except for TseSRacos, TseTrans receives 1st rank for 10 times in all 11 datasets. Comparing with TseSRacos, TseTrans gets the similar optimization performance, but spends less time because of no pre-train base predictors phase. It verifies that base regressors can be transfered to other datasets easily. It is meaningful for huge dataset which is hard to train the base predictors.

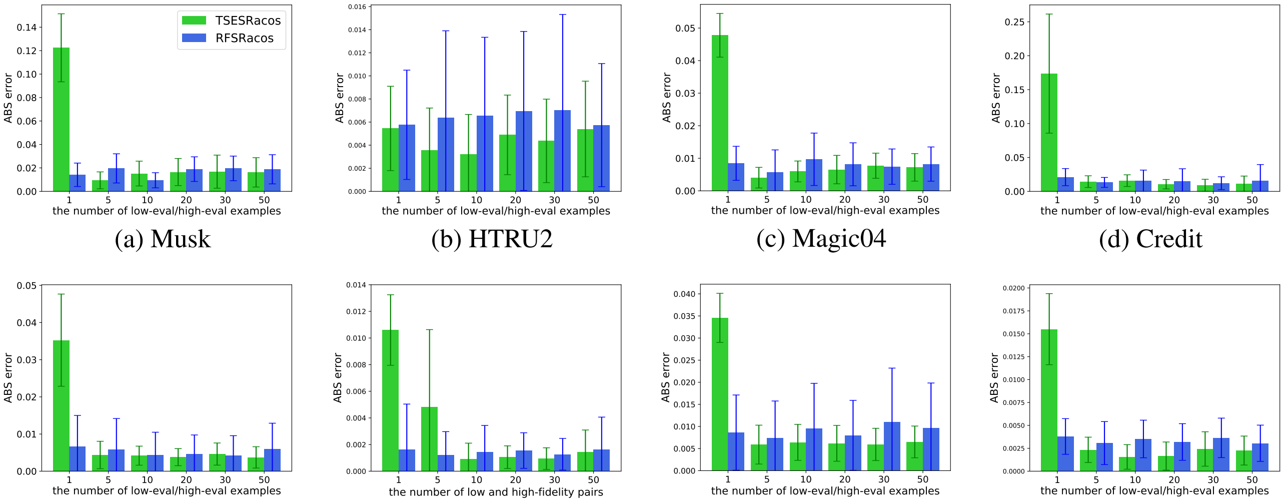
Multi-fidelity optimization with TSE is effective. From Table 2, TseSRacos outperforms others in most cases (8 times 1st rank in all 10 dataset). Comparing with LF-Only and HF-Only, TseSRacos can get performance nears to HF-Only meanwhile spend similar time with LF-Only. On Figure 1, TesSRacos is always better than HF-Only within limited time (the green solid line is over the yellow line all the time). It verifies that multi-fidelity optimization with TSE can improve optimization performance clearly with acceptable extra time-cost.



(a) Musk (b) HTRU2 (c) Magic04 (d) Credit

(e) Adult (f) Sensorless (g) Connect (h) Miniboone

Figure 1: The AUC curves under wall-clock time. Solid lines are the high-fidelity value curves. Dash lines are the low-fidelity value curves. The lines with circle mean that they are objective function curves on optimization. Solid and dash lines with same color are high and low fidelity evaluation values on same samples. X axis spans to the time used by LF-ONLY.



(a) Musk (b) HTRU2 (c) Magic04 (d) Credit

(e) Adult (f) Sensorless (g) Connect (h) Miniboone

Figure 2: The histograms of mean regression prediction error $|f_L + \Psi - f_H|$ in each training step. It only compares the prediction error on TSESRACOS (green) and RFSRACOS (blue). The X axis means the number of instances in regression training dataset.

Conclusion

Multi-fidelity optimization is a popular technique to tackle expensive sophisticated optimization problems. Especially, on AutoML problems, low-fidelity evaluation can be get by validating model configuration on a small data subset. And it is cheap but badly biased. Previous works of multi-fidelity optimization are often based on a specific method. In this paper, we propose a general multi-fidelity optimization framework for derivative-free optimization. A correction predictor is trained to estimate the residual between high and low-fidelity evaluations. And then, derivative-free optimization optimizes on the corrected low-fidelity evaluations. But, high-fidelity evaluations are hard to obtain. The regression dataset is too small to train a accurate predictor. We propose the transfer series expansion (TSE) to tackle this issue. TSE linearly combines some pre-trained base predictors to make regression converge faster. In addition, base predictors are trained on the middle-level regression problems which training datasets are more easier to get. Experiments on LightGBM hyper-parameter tuning problems verify that the multi-fidelity optimization with TSE can effectively fast the optimization process.

References

Ball, P. 2013. Counting google searches predicts market movements. *Nature* 12879.

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

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

Bradski, G. R. 1998. Computer vision face tracking for use in a perceptual user interface.

Brazdil, P. B.; Soares, C.; and Da Costa, J. P. 2003. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. *Machine Learning* 50(3):251–277.

Broder, A. Z. 2008. Computational advertising and recommender systems. In *Proceedings of the ACM Conference on Recommender systems*, 1–2. ACM.

Chapelle, O.; Vapnik, V.; and Bengio, Y. 2002. Model selection for small sample regression. *Machine Learning* 48(1):9–23.

Fawcett, T. 2006. An introduction to roc analysis. *Pattern recognition letters* 27(8):861–874.

Fernández-Godino, M. G.; Park, C.; Kim, N.-H.; and Haftka, R. T. 2016. Review of multi-fidelity models. *arXiv preprint arXiv:1609.07196*.

Feurer, M.; Klein, A.; Eggensperger, K.; , J.; Blum, M.; and Hutter, F. 2015. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems (NIPS'15)*, 2962–2970.

Hansen, N.; Müller, S. D.; and Koumoutsakos, P. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation* 11(1):1–18.

Hu, Y.-Q.; Qian, H.; and Yu, Y. 2017. Sequential classification-based optimization for direct policy search. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, 2029–2035.

Hu, Y.-Q.; Yu, Y.; and Zhou, Z.-H. 2018. Experienced optimization with reusable directional model for hyper-parameter search. In *Proceeding of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*, 2276–2282.

Huang, D.; Allen, T. T.; Notz, W. I.; and Miller, R. A. 2006. Sequential kriging optimization using multiple-fidelity evaluations. *Structural and Multidisciplinary Optimization* 32(5):369–382.

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

Kandasamy, K.; Dasarathy, G.; Oliva, J. B.; Schneider, J.; and Poczos, B. 2016. Multi-fidelity gaussian process bandit optimisation. *arXiv preprint arXiv:1603.06288*.

Kandasamy, K.; Dasarathy, G.; Schneider, J.; and Poczos, B. 2017. Multi-fidelity bayesian optimisation with continuous approximations. *arXiv preprint arXiv:1703.06240*.

Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; and Liu, T.-Y. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems (NIPS'17)*, 3146–3154.

Lindauer, M., and Hutter, F. 2018. Warmstarting of model-based algorithm configuration. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18)*, 1355–1362.

Liu, R.; Liu, E.; Yang, J.; Li, M.; and Wang, F. 2006. Optimizing the hyper-parameters for svm by combining evolution strategies with a grid search. In *Intelligent Control and Automation*. Springer. 712–721.

March, A., and Willcox, K. 2012. Provably convergent multifidelity optimization algorithm not requiring high-fidelity derivatives. *AIAA journal* 50(5):1079–1089.

Maron, O., and Moore, A. W. 1994. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in Neural Information Processing Systems (NIPS'94)*, 59–66.

Munos, R. 2011. Optimistic optimization of a deterministic function without the knowledge of its smoothness. In *Advances in Neural Information Processing Systems (NIPS'11)*, 783–791.

Sen, R.; Kandasamy, K.; and Shakkottai, S. 2018. Multi-fidelity black-box optimization with hierarchical partitions. In *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, 4545–4554.

Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and Freitas, N. D. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE* 104(1):148–175.

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

Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'13)*, 847–855.

Valiant, L. G. 1984. A theory of the learnable. *Communications of the ACM* 27(11):1134–1142.

Yu, Y.; Qian, H.; and Hu, Y.-Q. 2016. Derivative-free optimization via classification. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI'16)*, 2286–2292.

Zhao, P., and Yu, B. 2006. On model selection consistency of lasso. *Journal of Machine Learning Research* 7(Nov):2541–2563.