Machine Learning Hyperparameter Tuning via Bayesian Optimization Exploiting Monotonicity

by

Wenyi Wang

H.B.Sc., Mathematics, The University of Utah, 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies
(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

April 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Machine Learning Hyperparameter Tuning via Bayesian Optimization Exploiting Monotonicity

submitted by Wenyi Wang in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Examining Committee:

William J. Welch, Department of Statistics
Co-supervisor

Alan Wagner, Department of Computer Science
Co-supervisor

David Poole, Department Of Computer Science
Supervisory Committee Member

Abstract

We propose an algorithm for a family of optimization problems where the objective can be decomposed as a sum of functions with monotonicity properties. The motivating problem is optimization of hyperparameters of machine learning algorithms where we argue that the objective, validation error, can be decomposed into two approximately monotonic functions of the hyperparameters, along with some theoretical justification. Our proposed algorithm adapts Bayesian optimization methods to incorporate monotonicity constraints. We illustrate the improvement in search efficiency for applications of hyperparameter tuning in machine learning on an artificial problem and Penn Machine Learning Benchmarks.

Lay Summary

Machine learning (ML) algorithms have been successfully applied to a wide range of valuable real-world problems. However, in practice, applying ML methods often requires a large amount of work on tuning hyperparameters of the algorithm. Bayesian optimization (BO) is a state-of-art framework on ML hyperparameter tuning. In this thesis, we propose an algorithm that adapts the BO framework and incorporates a monotonicity prior motivated by a manual ML hyperparameter turning strategy. We present experimental results showing our algorithm finds settings of the hyperparameters with good performance faster than standard BO algorithms on a collection of problems.

Preface

The work of this thesis was performed in collaboration with the author's supervisor William J. Welch. All code was written by the author, Wenyi Wang. This preprint was written by both Wenyi Wang and William J. Welch. A shorter version of this preprint was accepted by the International Workshop on Automatic Machine Learning (AutoML 2018 at ICML/IJCAI-ECAI) workshop as "Bayesian Optimization Exploiting Monotonicity and Its Application in Machine Learning Hyperparameter Tuning". It was presented as a poster and in a 1-minute spotlight presentation during the workshop. Another version of this preprint was submitted to the Conference on Uncertainty in Artificial Intelligence (UAI 2019).

Table of Contents

List of Figures

Glossary

BO Bayesian optimization

GP Gaussian process

MCMC Markov chain Monte Carlo

MLP Multilayer perceptron

PMLB Penn Machine Learning Benchmarks

MSPE Mean squared prediction error

ML Machine learning

Chapter 1

Introduction

Bayesian optimization (BO) has been successfully applied to many global optimization problems [7, 9, 13, 18]. Typically, it makes few assumptions though a Bayesian prior about the objective function, treating it as a black box. When prior knowledge is available, however, it may be possible to improve the efficiency of the optimization search. In particular, function monotonicity has been successfully exploited to improve accuracy in related statistical modeling for the analysis of computer experiments [e.g., 5]. The methods proposed here employ such monotonicity information when tuning a Machine learning (ML) algorithm with respect to its hyperparameters. We analyze the applicability to ML hyperparameter tuning problems and present positive experimental results.

Our algorithm incorporates monotonicity information in the Gaussian process (GP) model underlying Bayesian optimization. Bayesian optimization sequentially adds new objective function evaluations based on a probabilistic emulation of the objective trained using all current evaluations. At each iteration the emulator guides the choice of the next evaluation in a way that balances global and local search. A typical choice for the statistical emulator is GP regression. In this work, we study optimization problems, where the objective function can be represented as a sum of functions with monotonicity properties. Instead of modeling the objective function directly by a GP, we model its components separately, each emulated by approximately monotonic Gaussian processes (GPs).

There are two main contributions in this thesis. First, we propose an algorithm

for optimization problems where the objective function can be represented by a sum of functions with monotonicity information. Second, we illustrate with theoretical and empirical evidence that our algorithm is suitable for machine learning hyperparameter tuning.

The rest of this thesis is organized as follows. In chapter 2 we provide a brief overview of BO using GPs and previous work on monotonicity constraints. Chapter 3 describes the basics of the proposed method: objective decomposition and monotonicity. In chapter 4 we illustrate how these ideas can be applied to ML hyperparameter tuning with theoretical justifications. Empirical results comparing the proposed algorithm with a standard BO approach are presented in chapter 5. Finally, in chapter 6 we make some concluding remarks and discuss future work.

Chapter 2

Related Works

2.1 Bayesian Optimization

BO is a sequential algorithm for global optimization. At each iteration it builds a statistical model of the objective function based on the evaluations so far; the model guides the location of the next function evaluation. Compared with local search, it tends to use more (often all) available function evaluations to determine the next. This makes BO particular useful when objective evaluations are limited by computational cost. Furthermore, it requires little prior knowledge of the properties (e.g., convexity) of the objective function.

Algorithm 1 describes a typical procedure for Bayesian optimization. The algorithm takes the region of interest $\mathscr{X}$, the objective function $f : \mathscr{X} \to \mathbb{R}$, a set I of initial points from $\mathscr{X}$, a model fitting function M, and an acquisition function A as input, and outputs a list of objective function evaluations. First it initializes objective function evaluation set D at I. At each iterative step it builds a model h based on D, finds the next evaluation point $\mathbf{x}^{(i)}$ that optimizes $A(\cdot, h, D)$, and append $(\mathbf{x}^{(i)}, f(\mathbf{x}^{(i)}))$ to D. Common choices of M include GP regression and random forests. Expected improvement, probability of improvement, and upper confidence bound are typically used for the acquisition function. For a comprehensive overview see Brochu et al. [3]. In this thesis we focus on GP regression for fitting probabilistic models, and adapt the expected improvement acquisition function.

Algorithm 1 Bayesian Optimization

Input: objective function f, region of interest $\mathscr{X}$, initial points I, model fitting function M and acquisition function A
Output: D: a list of objective function evaluations
Initialize $D = \{(\mathbf{x}, f(\mathbf{x})) : \mathbf{x} \in I\}$
repeat
  Fit a probabilistic model $h = M(D)$
  Determine the next evaluation point at
    $\mathbf{x} = \text{argmax}_{\mathbf{x}' \in \mathscr{X}} A(\mathbf{x}', h, D)$
  Append the new data: $D = D \cup (\mathbf{x}, f(\mathbf{x}))$
until maximum number of iterations reached or other stopping criteria satisfied
return D

2.2 Gaussian Processes with Monotonicity Constraints

As discussed in section 2.1, GP regression builds a probabilistic model of a function based on limited evaluations. It is a robust model that can be applied to a large class of functions; it only assumes the target function behaves like a random sample from the GP. The GP itself has parameters, which are trained using the observed data D via maximum likelihood or Bayesian Markov chain Monte Carlo (MCMC). The trained GP leads to predictions by considering sample paths conditional on D. For more details see Rasmussen and Williams [16].

When prior knowledge of the target function is available, performance of the model can be improved. In this thesis, we study functions that can be decomposed as a sum of monotonic functions, and model the decomposed functions separately. There are various approaches to GP modeling subject to (approximate) monotonicity constraints [10, 17, 19]. We employ Riihimäki and Vehtari [17]'s implementation because it is readily accessible.

Riihimäki and Vehtari [17] incorporate monotonicity by introducing derivative information at v virtual points, $\tilde{\mathbf{x}}^{(i)}$ for $i = 1, \ldots, v$, in $\mathscr{X}$. The objective function is not evaluated at the virtual points, rather they are used to favor a GP $\hat{f}(\mathbf{x})$ with positive (or negative) derivatives at these points in all directions for which monotonicity is assumed. For definiteness, take a target function that increases with respect to x_j. Subject to some differentiability conditions, the derivative of a GP is a related GP, and hence has statistical properties. The contribution to the likelihood

from the derivative of $\hat{f}(\mathbf{x})$ with respect to x_j at $\tilde{\mathbf{x}}^{(i)}$ is taken to be

$$P\left(\frac{\partial \hat{f}}{\partial x_j}(\tilde{\mathbf{x}}^{(i)}) > z\right) = \Phi(z/\nu),$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution and $\nu > 0$ is a constant scalar. The likelihood increases with a more positive derivative. For the experiments of this thesis, we set $\nu = 0.1$ so that monotonicity is only approximate ($\nu \to 0$ would strictly enforce monotonicity). The likelihood incorporating all such derivative contributions as well as those from the actual function evaluations is used to train the GP parameters and for prediction.

Chapter 3

Problem Formulation and Algorithm

Our focus is an objective function f that may be written as a sum of functions,

$$f(\mathbf{x}) = \sum_{k=1}^{K} f_k(\mathbf{x}), \tag{3.1}$$

with monotonicity constraints

$$\begin{aligned} & f_k(x_1,\ldots,x_{j-1},x_j,x_{j+1},\ldots,x_n) \\ \leq & f_k(x_1,\ldots,x_{j-1},x'_j,x_{j+1},\ldots,x_n) \end{aligned} \tag{3.2}$$

or

$$\begin{aligned} & f_k(x_1,\ldots,x_{j-1},x_j,x_{j+1},\ldots,x_n) \\ \geq & f_k(x_1,\ldots,x_{j-1},x'_j,x_{j+1},\ldots,x_n) \end{aligned} \tag{3.3}$$

for some k, j and all feasible $x_j < x'_j$ and $(x_1,\ldots,x_{j-1},x_{j+1},\ldots,x_n)$. Whether the function $f_k(\mathbf{x})$ is monotonically non-decreasing or non-increasing with respect to x_j must be known. Our algorithm needs to observe each function $f_k(\mathbf{x})$ at chosen $\mathbf{x}$ values in $\mathscr{X}$. Even if $f_k(\mathbf{x})$ is observed without noise, our implementation assumes a GP with an additional white-noise term for computational robustness.

Algorithm 2 Bayesian Optimization with Monotonicity Information

Input: the objective function decomposed as $\{f_k\}$, the region of interest $\mathscr{X}$, n initial points $\mathbf{x}^{(i)}$, v virtual points $\tilde{\mathbf{x}}^{(i)}$ for monotonicity information, and the acquisition function $A(\mathbf{x}, \hat{f}, D)$
Output: X and Y, lists for the points where the objective is evaluated and the respective function values
Initialize $X = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\}$
For all i, k, evaluate $y_{i,k} = f_k(\mathbf{x}^{(i)})$, hence $y_i = \sum_k y_{i,k}$ for all i
Initialize $Y_k = \{y_{k,1}, \ldots, y_{k,n}\}$ for all k, and $Y = \{y_1, \ldots, y_n\}$
repeat
  For all k, train the GP $\hat{f}_k$ using X, Y_k and the monotonicity constraints at $\tilde{\mathbf{x}}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(v)}$
  Let $\hat{f}(\mathbf{x}) = \sum_k \hat{f}_k(\mathbf{x})$
  Determine the next evaluation point
    $\mathbf{x}^{(n+1)} = \text{argmax}_{\mathbf{x} \in \mathscr{X}} A(\mathbf{x}, \hat{f}, (X, Y))$
  For all k, evaluate $y_{n+1,k} = f_k(\mathbf{x}^{(n+1)})$, hence $y_{n+1} = \sum_k y_{n+1,k}$
  Append $\mathbf{x}^{(n+1)}$ to X, $y_{n+1,k}$ to Y_k for all k, y_{n+1} to Y, and increment n by 1
until maximum number of iterations reached or some stopping criterion is satisfied
return X and Y

BO (chapter 2.1) is adapted so that instead of modeling $f(\mathbf{x})$ directly, we model each $f_k(\mathbf{x})$ separately using independent monotonicity-constrained GP regressions (chapter 2.2). The predictive mean of $\hat{f}(\mathbf{x})$ at any trial $\mathbf{x}$ is then the sum of the predictive means of the $\hat{f}_k(\mathbf{x})$, and because of the assumed independence of the component GPs, the predictive variance of the sum is similarly the sum of the variances. Once the predictive mean and variance of $\hat{f}(\mathbf{x})$ have been computed in this way, the acquisition function $A(\mathbf{x}, \hat{f}, D)$ of a new evaluation at $\mathbf{x}$ can be computed in the standard way [9]. Algorithm 2 summarizes this procedure.

For all the experiments presented, each component GP is assumed to have zero mean, constant variance for the correlated process, an isotropic squared-exponential correlation function, and constant variance for the white-noise. Monotonicity information is also incorporated if known. The search starts from $n = 4$ initial evaluations since there are three parameters to be estimated for each GP regression: the constant variance of the correlated process, the scale parameter of the squared exponential correlation function, and the constant variance of the white noise.

3.1 Illustrative Example

A simple example will demonstrate the proposed algorithm and show that it can be more efficient than a standard Bayesian optimization approach. To examine the roles of function decomposition and monotonicity, we also compare with a modification of our algorithm without monotonicity modeling. Thus, the methods compared are referred to as (1) standard, (2) decomposition and monotonicity (the proposed algorithm), and (3) decomposition without monotonicity.

The illustrative example is to maximize

$$\frac{1}{60}(w_1\phi(b_1-x;\sigma_1)+w_2\phi(b_2-x;\sigma_2)-w_3\phi(b_3-x;\sigma_3)), \quad \text{for } 0\leq \mathbf{x}\leq 1$$

where $\phi(z;\sigma)=1/(\sigma\sqrt{2\pi})e^{-z^2/(2\sigma^2)}$ is a basis function with the shape of a probability density function of the normal distribution with mean zero and standard deviation σ. The b_i, w_i and σ_i are set to $(b_1,b_2,b_3)=(0.78,0.44,0.69)$, $(w_1,w_2,w_3)=(3,10,1)$ and $(\sigma_1,\sigma_2,\sigma_3)=(0.05,0.2,0.1)$. The parameters are chosen to make the objective function have two modes, where the sub-optimal mode is wide, and the optimal mode has a narrower range, which is generally hard to optimize. The fraction $1/60$ is for scaling purposes.

One can show that here, without changing the optimization problem, by adding a constant (i.e. adding $w_1\phi(0;\sigma_1)+w_2\phi(0;\sigma_2)-w_3\phi(0;\sigma_3)$), the objective function f can be written as the sum of two monotonic functions f_1 and f_2, where

$$\begin{aligned}
f_1(\mathbf{x}) = \frac{1}{60}(&w_1\phi(\max(b_1-\mathbf{x},0);\sigma_1)\\
&+w_2\phi(\max(b_2-\mathbf{x},0);\sigma_2)\\
&-w_3\phi(\min(b_3-\mathbf{x},0);\sigma_3)), \quad \text{and}\\
f_2(\mathbf{x}) = \frac{1}{60}(&w_1\phi(\min(b_1-\mathbf{x},0);\sigma_1)\\
&+w_2\phi(\min(b_2-\mathbf{x},0);\sigma_2)\\
&-w_3\phi(\max(b_3-\mathbf{x},0);\sigma_3)).
\end{aligned}$$

The objective function and its decomposition are shown in Figure 3.1. In general, the user need to decompose the objective function into a sum of approximately

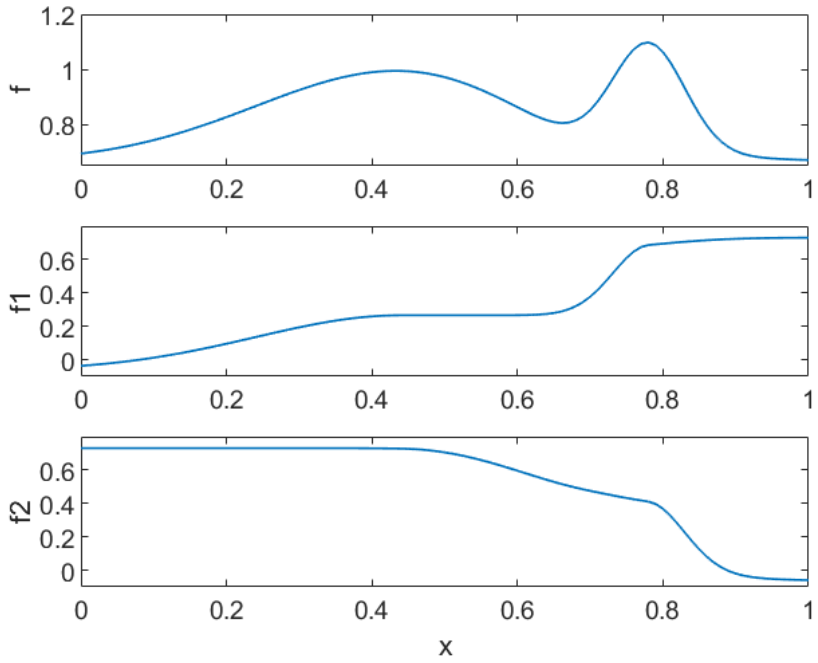


Figure 3.1: Illustrative objective function $f(x)$ and its decomposition into $f_1(x) + f_2(x)$.

monotonic functions that can be evaluated.

Figure 3.2 shows the result of applying the three algorithms 10 times from four random initial objective function evaluations and adding a further six evaluations. For this problem, we employ a uniform grid of 10 virtual points at $(0, \frac{1}{9}, \frac{2}{9}, \ldots, 1)$, where approximate monotonicity is enforced. We see that modeling the decomposed objective functions separately makes the search less efficient, but decomposition plus monotonicity improves the performance. At Step 6, the decomposition and monotonicity algorithm on average finds objective function value close to the optimum (about 1.1). However, the average objective values found by the other two methods are closer to the maximum of the sub-optimal mode (about 1.0).

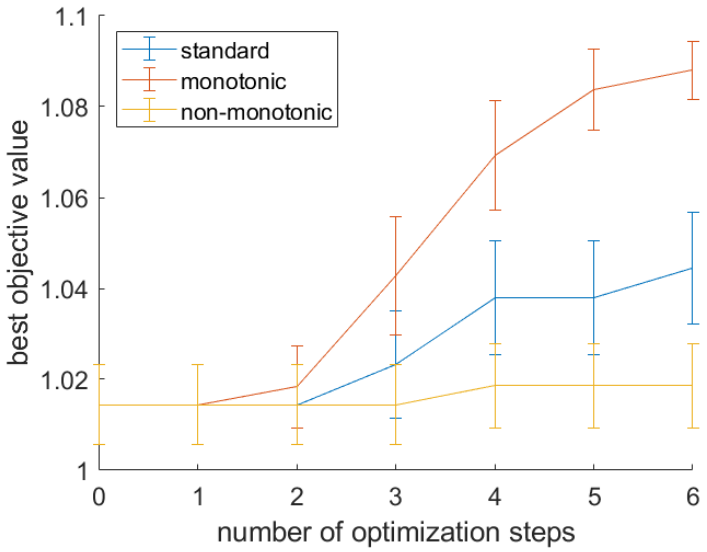


Figure 3.2: Best objective function evaluated versus number of optimization steps for three algorithms. Ten trials are averaged, and the error bars show ± 1 standard error of the mean.

Chapter 4

Application to Machine Learning Hyperparameter Tuning

In practice, users minimize the validation error of an ML algorithm with respect to its hyperparameters, as the generalization error is unavailable. Usually it is unreasonable to assume that the validation error is monotonic with respect to any given hyperparameter. Rather than optimizing validation error directly, we decompose it as training error plus the difference between validation and training errors. We claim that these component functions are reasonably assumed to be monotonic with respect to ML model complexity and hence any hyperparameter x_j controlling complexity, i.e., $K = 2$ in (3.1), and x_j satisfies (3.2) or (3.3) for $k = 1, 2$. If the user is not prepared to make such assumptions about behavior with respect to x_j, there is no contribution to the likelihood from x_j at the virtual points.

For many ML algorithms, the training error is monotonically decreasing as model complexity increases. The training error is also monotonic with respect to regularization parameters, which can also be thought of as hyperparameters. Hence monotonicity of training error with respect to hyperparameters widely exists.

Analysis of monotonicity of the difference between validation and training errors is more subtle, and to the best of our knowledge there are no general results. However the notion that the generalizability (inverse of the difference between generalization and training error) is decreasing with respect to effective model complexity is accepted in the ML community. An intuitive idea supporting this

argument is described as follows. Notice that the training error and generalization error are evaluated as losses of a selected function applied to two datasets (i.e. loosely speaking the training set and test set) from an identical distribution. Since the function is selected from a hypothesis set to fit the training set, the decrease in training error would be expected to be more sensitive (i.e. faster) compared to generalization error, when the hypothesis set expands.

That view is used implicitly by experts in manual ML tuning. One would increase or decrease hyperparameters using monotonicity assumptions and relativities between target error, training error, and validation error minus training error [6, pp. 424, 425]. It leads to a heuristic argument that assuming monotonicity of validation error minus training error is reasonable. In the rest of this chapter, we present theoretical results for an idealized view of this monotonicity argument.

4.1 Framework for Theoretical Justification of Monotonicity

To facilitate analysis, we take a simplified representation of increasing ML model complexity, where the notion of an expanding hypothesis set is is represented by a sequence of fixed ML models. Specifically, consider a binary classification problem that chooses a classifier from a finite set $\{C_i\}$. Classifier C_i has generalization error (e.g., misclassification rate) G_i. Let $D = \{X_j, Y_j\}_{j=1}^m$ be a training set, i.i.d. from some distribution, where X_j is a vector of features for instance j and $Y_j = 0$ or 1 denotes the class. Classifier C_i returns a 0 or 1 classification $C_i(X_j)$ for all X_j.

The training data will be used to choose one of the C_i. The error from classifier C_i and training instance j is $T_{i,j} = |C_i(X_j) - Y_j|$, a Bernoulli random variable. For an arbitrary i, since C_i is fixed (without training), by the definition of generalization error, each of the $T_{i,j}$ takes value 1 with probability G_i. Since $\{X_j, Y_j\}_{j=1}^m$ are i.i.d., given C_i, $\{T_{i,j}\}_{j=1}^m$ are i.i.d. for all i. Let $T_i = \text{mean}_j(T_{i,j})$, where $mT_i \sim \text{Bin}(m, G_i)$, a binomial random variable. This model can be interpreted as the Bayesian network shown in Figure 4.1. We further assume the elements of the set $\{C_i\}$ are independently drawn from some distribution. Then the G_i are i.i.d. random variables too.

We study the learning algorithm that minimizes training error among the first n

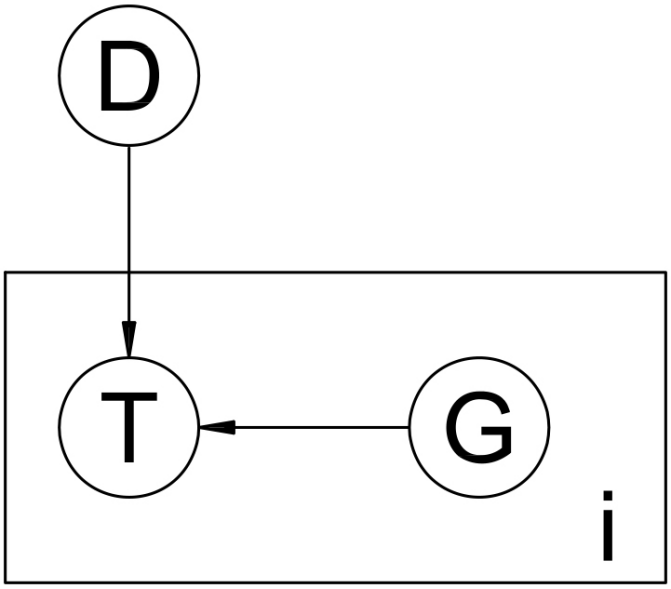


Figure 4.1: This figure shows a simplified probabilistic model of the classification learning problem in plate notation. The generalization error G_i of classifier C_i follows some distribution i.i.d. over i. The training error T_i conditionally depends on G_i and the dataset D.

classifiers. Classifier C_{I_n} is chosen, where $I_n = \text{argmin}_{i \leq n} T_i$. Thus, model complexity of the learning algorithm increases with n in the sense that more classifiers are available for learning. In this formulation, the hypothesis that generalization error minus training error is non-decreasing with respect to learning algorithm model complexity translates to the concrete statement that $G_{I_n} - T_{I_n}$ is statistically non-decreasing with n.

Our result in section 4.2 shows that for this formulation $G_{I_n} - T_{I_n}$ is non-decreasing in expectation as n increases for G_i's following any beta prior distribution. The result is also extended to any mixture of beta distributions. We are not claiming that a mixture of beta distributions is a universal approximator of any prior distribution subject to certain conditions. However, they define a rich family of distributions, and experimentally outperform mixtures of Gaussian distributions when modeling bounded variables in a variety of real-world dataset [11]. Detailed analysis is presented in the next section.

4.2 Monotonicity Result with Beta-Mixture Priors on Generalization Errors

The formulation in section 4.1 requires the following result to be shown. Let G_i be i.i.d. random variables for $i = 1, \ldots, n$. Let T_i be the mean of m i.i.d. Bernoulli random variables with probability G_i for the event of interest (an error) and $I_n = \text{argmin}_{i \leq n} T_i$. We want to show that $G_{I_n} - T_{I_n}$ is non-decreasing in expectation as n increases. We do so in two steps.

1. It is sufficient to consider a simpler problem, namely that the expectation of $G_2 - T_2$ given T_2 is not less than that of $G_1 - T_1$ given T_1 when $T_2 < T_1$.

2. Show that the required difference of expectations for the simplified problem holds with independent mixture of beta priors on G_1 and G_2.

4.2.1 Simplify the Problem

For $I_n = I_{n+1}$, the quantity of interest does not change. The case $I_{n+1} \neq I_n$ arises if and only if $T_{n+1} < T_{I_n}$ so that $I_{n+1} = n+1$. Therefore, we are interested in the conditional event $G_{n+1} - T_{n+1} > G_{I_n} - T_{I_n} \mid T_{n+1} < T_{I_n}$

By conditioning on the n possible T_i giving T_{I_n} and applying the law of total probability, we have

$$
\begin{aligned}
& P(G_{n+1} - T_{n+1} > G_{I_n} - T_{I_n} \mid T_{n+1} < T_{I_n}) \\
=& \sum_{i=1}^{n} P(G_{n+1} - T_{n+1} > G_{I_n} - T_{I_n} \mid I_n = i, T_{n+1} < T_{I_n}) P(I_n = i) \\
=& \sum_{i=1}^{n} \iint_{t_{n+1} < t_i} P(G_{n+1} - t_{n+1} > G_{I_n} - t_{I_n} \mid I_n = i, T_i = t_i, T_{n+1} = t_{n+1}) \\
& \qquad \times P(I_n = i, T_i = t_i, T_{n+1} = t_{n+1}) d(t_i) d(t_{n+1}).
\end{aligned}
$$

Since G_i is independent of any other variable given T_i for all $i = 1, \ldots, n+1$, we also have G_i, I_n, and G_{n+1} are mutually independent given T_i and T_{n+1}. Therefore $P(G_{n+1} - t_{n+1} > G_{I_n} - t_{I_n} \mid I_n = i, T_i = t_i, T_{n+1} = t_{n+1})$ represents the probability of one random variable being greater than the other, where the random variables are $G_{n+1} - t_{n+1} \mid T_{n+1} = t_{n+1}$ and $G_i - t_i \mid T_i = t_i$ when $t_{n+1} < t_i$, and these two random

variables are independent. If we can conclude that $P(G_{n+1} - t_{n+1} > G_{I_n} - t_{I_n} \mid I_n = i, T_i = t_i, T_{n+1} = t_{n+1}) > c_1$ when $t_{n+1} < t_i$ for all i and some constant c_1, then $P(G_{n+1} - T_{n+1} > G_{I_n} - T_{I_n} \mid T_{n+1} < T_{I_n}) > c_1$. Similarly if $\mathbb{E}(P(G_{n+1} - t_{n+1} - (G_{I_n} - t_{I_n}) \mid I_n = i, T_i = t_i, T_{n+1} = t_{n+1})) > c_2$ when $t_{n+1} < t_i$, then $\mathbb{E}(P(G_{n+1} - T_{n+1} - (G_{I_n} - T_{I_n}) \mid T_{n+1} < T_{I_n})) > c_2$. The problem of studying the relationship between $G_{n+1} \mid T_{n+1} = t_{n+1}$ and $G_i \mid T_i = t_i$ when $t_{n+1} < t_i$ therefore reduces to the following problem.

Let G_1 and G_2 be i.i.d. random variables, and let T_1 and T_2 be the means of m Bernoulli random variables with expectations G_1 and G_2, respectively. Then $G_1 \mid T_1$ and $G_2 \mid T_2$ are independent random variables. We need to verify that $G_2 - t_2 \mid T_2 = t_2$ is statistically greater than $G_1 - t_1 \mid T_1 = t_1$ when $t_2 < t_1$.

4.2.2 Generalization Errors Sampled from A Mixture of Beta Distributions

The number of errors, mT_i, is a binomial random variable with probability of error G_i. A standard Bayesian approach for inference about G_i is to assign a beta prior distribution for G_i: the beta family has a wide variety of shapes and is conjugate for the binomial likelihood. For G_i's following Beta(a,b) for some $a,b > 0$, the posterior distribution $G_i \mid T_i = t_i \sim \text{Beta}(a + t_i m, b + (1 - t_i)m)$. Hence, $G_i \mid T_i = t_i$ has mean $(a + t_i m)/(a + b + m)$. Since

$$\frac{d((a + t_i m)/(a + b + m)}{dt_i} < 1,$$

we have the result that $\mathbb{E}(G_i - t_i \mid T_i = t_i)$ is increasing as t_i decreases.

Furthermore, as this result holds for any values of the parameters a, b of the beta distribution, other priors can be approximated by a mixture of beta distributions of different shapes, i.e., different values of a, b. In this way the argument extends, for example, to multimodal priors.

Chapter 5

Experimental Results

In this chapter, we present experimental results comparing our algorithm and standard algorithms when applied to an artificial elastic net regression problem and the Penn Machine Learning Benchmarks. The results show our algorithm outperforms standard algorithms.

5.1 Elastic Net Regularized Linear Regression

Consider the elastic net regularized linear regression algorithm. Let X_{tr} and y_{tr} be training sets. The elastic net regularized linear regression tries to find a linear predictor with sparse coefficients; nonzero coefficients are penalized to be smaller in magnitude. It optimizes over all linear functionals a combination of goodness of fit, the 2-norm of the coefficients, and the 1-norm of the coefficients. Specifically, it finds $\hat{\beta}$ to minimize the loss

$$l(\beta) = \frac{1}{2|y_{tr}|}\|y_{tr} - \beta X_{tr}\|_2^2 + \lambda P_\alpha(\beta),$$

where $P_\alpha(\beta) = \frac{1-\alpha}{2}\|\beta\|_2^2 + \alpha\|\beta\|_1$. We adopt the common performance measure mean square error scaled by $\frac{1}{2}$ for consistency

$$\text{mse}(\beta, X, y) = \frac{1}{2|y|}\|y - \beta X\|_2^2.$$

Note that the loss function l consists of two parts: the first is the mean square error on the training set, which measures the goodness of fit, while the second, $\lambda P_\alpha(\beta)$, is a regularization term. In the regularization the 1-norm forces sparsity of β, and the 2-norm constrains the magnitude of β. The regularization term can be considered as representing prior knowledge of the underlining function and can prevent over fitting for many cases. For more details of elastic net regression, one may refer to Zou and Hastie [20].

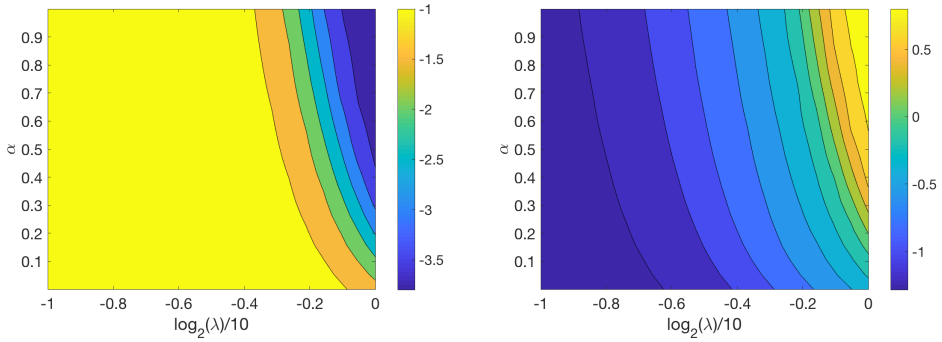
Given the validation sets X_v and y_v, the goal of hyperparameter tuning for this problem is to minimize $\text{mse}(\hat{\beta}, X_v, y_v)$ with respect to α and λ, where $\hat{\beta} = \text{argmin}_\beta\, l(\beta)$.

In the following experiment, all elements of X_{tr} and X_v are sampled independently from the standard normal distribution, with 200 examples in each set and 100 features for each example. We define the underlining linear transformation by a 100×1 vector C with 50 zero elements; the remaining 50 nonzero elements are drawn from a normal distribution with mean zero and standard deviation 0.22. Data y_{tr} and y_v follow independent normal distributions with means CX_{tr} and CX_v, respectively, and standard deviation 1. The parameters for generating the data are chosen to make the optimization problem nontrivial. The hyperparameter ranges of interest are $\alpha \in [0,1]$, and $\lambda \in 2^{([-10,0])}$. Figure 5.1c shows the validation error as a function of α and λ. Since we describe Bayesian optimization for maximization and validation error is to be minimized, negative error is shown.

As we described in the general case, to optimize the validation error, we decompose it as the sum of the training error and the difference between validation error and training error. From Figures 5.1a and 5.1b we see that the two decomposed functions are (at least roughly) monotonic, which satisfies our assumptions. Negative errors are shown due to the fact that we described optimization algorithms for maximization, and in experiments we optimize negation of the errors.

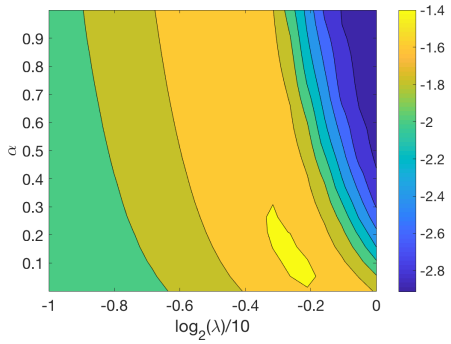
Again, we compare three algorithms: standard, decomposition and monotonicity, and decomposition without monotonicity. Each is applied 10 times, starting from different initial function evaluations at four random points. A 10×10 grid of virtual points is used in monotonicity modeling.

Figure 5.2 shows the results from the three algorithms. The mean of the best validation error averaged over the 10 trials is plotted versus the number of evalua-



(a) Training error

(b) Validation error - training error



(c) Validation error

Figure 5.1: Negative validation error of the elastic net linear regression problem as a function of the hyperparameters α and λ, and its decomposition. (a) and (b) show the training error and validation error minus training error, respectively, which are assumed to be monotonic. (c) shows the validation error. The yellow region is desired for this hyperparameter turning problem.

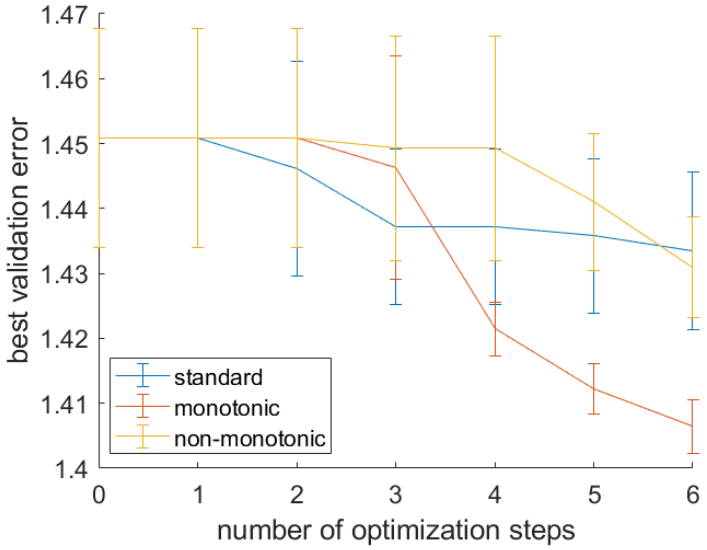


Figure 5.2: Best validation error of elastic net regularized linear regression found versus number of optimization steps for three algorithms. Ten trials are averaged, and the error bars show ± 1 standard error of the mean.

tions. The error bars show ± 1 standard error of the sample mean. We see that the performance of the algorithm employing decomposition and monotonicity dominates that of the other two statistically. From the experiment we also observe that the standard and decomposition-only algorithms start by evaluating the functions at the corners. In contrast, the algorithm exploiting monotonicity does not always do so. This is reasonable since the monotonicity modeling algorithm has more prior knowledge of the objective function. In addition, its GPs usually have less variance, which is reasonable due to incorporating more prior knowledge.

5.2 Three ML algorithms on Penn Machine Learning Benchmarks

In this subsection, we apply and compare Bayesian optimization algorithms when tuning the hyperparameters of three ML algorithms: decision tree regression [2], random forest regression [1] and Multilayer perceptron (MLP) with backpropa-

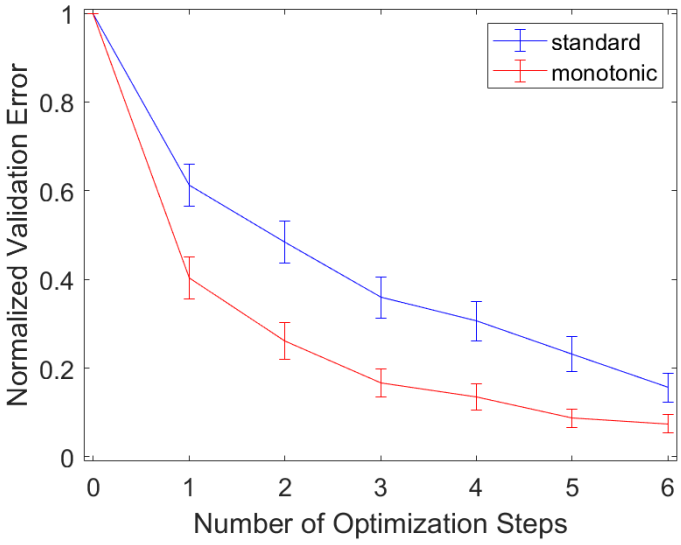
gation [8]. These algorithms are applied to the Penn Machine Learning Benchmarks (PMLB). The PMLB consists of hundreds of datasets, most of them real-world data, and is designed to identify the respective strengths and weaknesses when comparing ML algorithms [15]. We choose PMLB for its comprehensiveness and easy accessibility. Experiments are performed on the regression sets of PMLB. To avoid high instability due to a small training set and to keep computation time manageable, we filter out the datasets with number of instances less than 100 and greater than 10000, leaving 81 sets.

Decision tree, random forest, and MLP are off-the-shelf regression algorithms. We choose these three methods because it is reasonable to use a fixed search space of hyperparameters for an algorithm across all datasets. More specifically, decision tree regression, random forest regression with 100 trees and MLP with two hidden layers are applied. To balance the goodness of fit and generalizability, we control the model complexity by tuning three hyperparameters for each algorithm. The hyperparameters are generically denoted θ_1, θ_2, and θ_3, all in $[0,1]$, and they control the ranges of the actual respective hyperparameters as follows. For decision tree regression and random forest regression we optimize: (1) maximum split nodes $\left\lfloor |X_{tr}|^{\theta_1} \right\rceil$, (2) minimum leaf size $\left\lfloor |X_{tr}|^{1-\theta_2} \right\rceil$, and (3) number of features to use at each split $\left\lfloor n^{\theta_3} \right\rceil$, where n is the number of features of the problem. Here $\lfloor \cdot \rceil$ denotes round to the nearest integer. For the two-layered MLP, we optimize (1) the number of neurons in the first hidden layer $\left\lfloor 512^{\theta_1} \right\rceil$, (2) the number of neurons in the second hidden layer $\left\lfloor 512^{\theta_2} \right\rceil$, and (3) the L_2 regularization coefficient $1-\theta_3$.

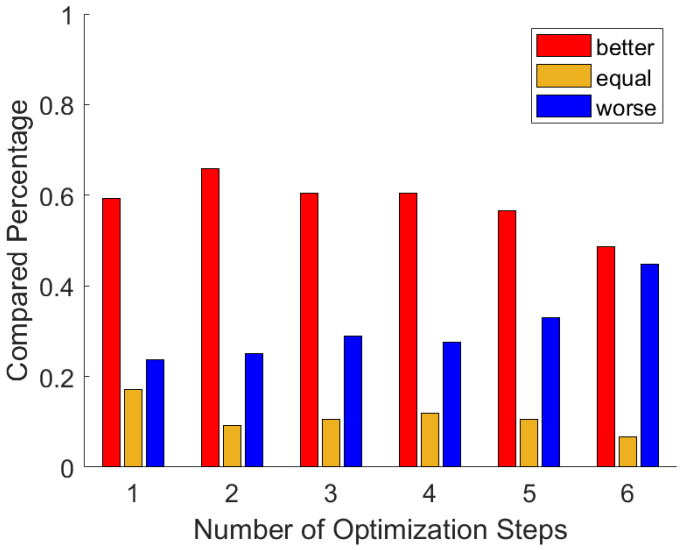
For each dataset, the training set X_{tr}, y_{tr} and validation set X_{val}, y_{val} comprise 70% and 20% of the data, respectively. (10% of the data was reserved as a test set but not used for the results presented here.) Mean squared prediction error (MSPE) is used as the hypothesis performance metric. The hyperparameter optimization problem is to find the best $\theta_1, \theta_2, \theta_3 \in [0,1]$ such that

$$\begin{aligned} \text{error}(h, X_{val}, y_{val}) &= \frac{1}{|X_{val}|} \sum_i (h(x_i) - y_i)^2 \\ &= \text{error}(h, X_{tr}, y_{tr}) + (\text{error}(h, X_{val}, y_{val}) - \text{error}(h, X_{tr}, y_{tr})) \end{aligned}$$

is minimized, where h is the predictor trained by the regression algorithm.

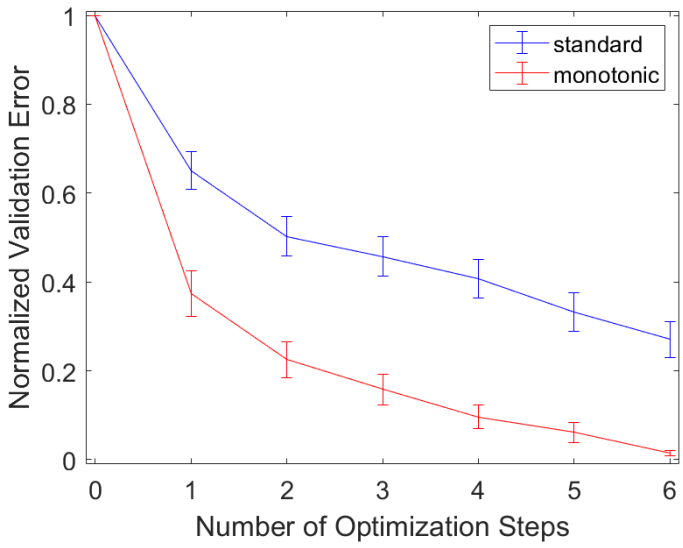


(a) Normalized validation errors

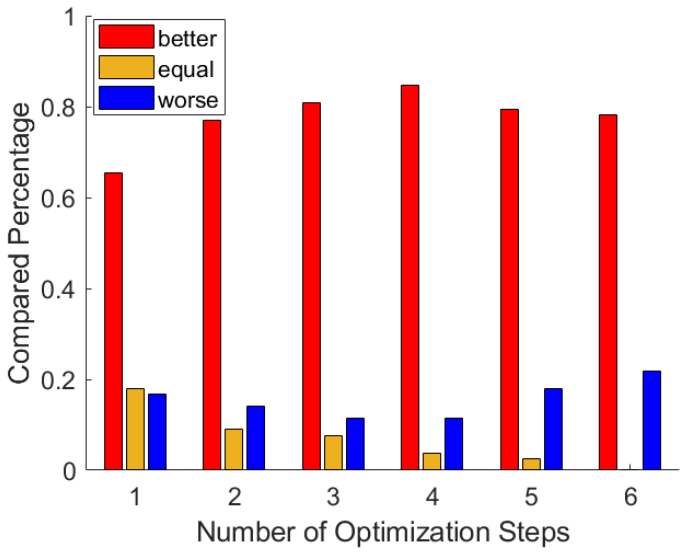


(b) Comparison chart

Figure 5.3: Performance summary plots of standard and our algorithm tested on decision tree hyperparameter turning.

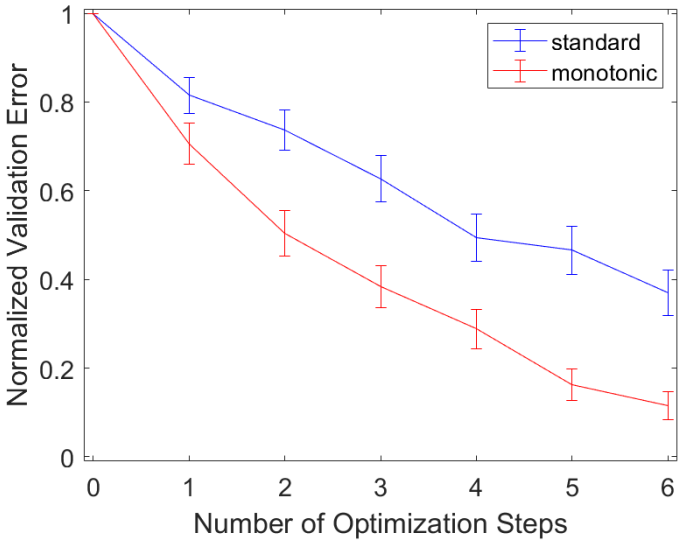


(a) Normalized validation errors

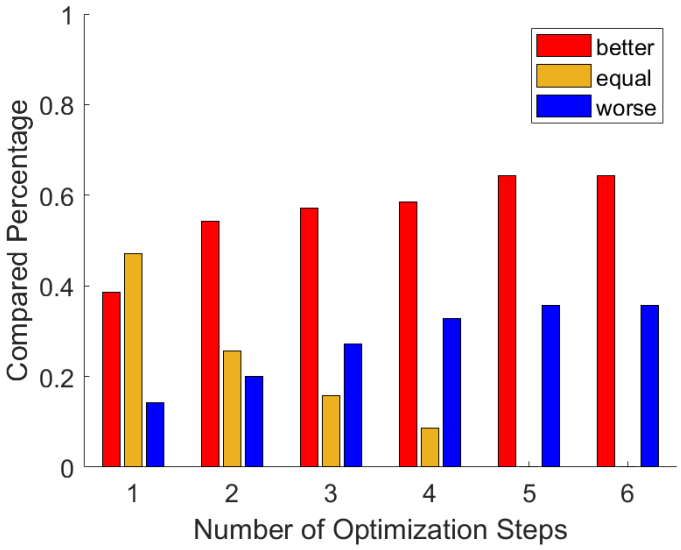


(b) Comparison chart

Figure 5.4: Performance summary plots of standard and our algorithm tested on random forest hyperparameter turning.



(a) Normalized validation errors



(b) Comparison chart

Figure 5.5: Performance summary plots of standard and our algorithm tested on MLP hyperparameter turning.

For each combination of regression method and dataset, we compare our decomposition and monotonicity algorithm with the standard BO. (Decomposition without monotonicity was not included to save computational time.) Virtual points to enforce monotonicity are on a $4 \times 4 \times 4$ grid. The same four random initial objective function evaluations are used for both optimization methods, with six optimization steps.

The results are summarized in Figures 5.3, 5.4 and 5.5 for the three ML regression methods respectively. Sub-figures 5.3a, 5.4a and 5.5a on the left show the best validation MSPE found at each optimization step averaged across the 81 datasets (with 0 steps indicating only the four initial points are used). Because the achievable magnitude of error varies substantially across the 81 datasets—some are just harder problems than others—for any ML algorithm, the raw validation MSPE values for a dataset are normalized such that the worst and best observed validation MSPE values across the two optimization algorithms and all steps become 0 and 1, respectively. Then the normalized scores are averaged across datasets. Hence every subfigure starts at 1 on the y-axis because the two optimization algorithms start with a common set of four points. For steps 1–6 the error bars show ± 1 standard error of the mean. It is clear that the decomposition and monontonicity algorithm achieves better validation MSPE on average for all three ML methods. The differences between averages remains large relative to the standard errors of the means at all steps for the more competitive random forest and MLP algorithms. Similar conclusions may be drawn from sub-figures 5.3b, 5.4b and 5.5b on the right side, which show for each optimization step the percentage of datasets where our algorithm finds better, equal or worse validation error compared with standard BO. In particular, at termination, our algorithm achieves better scores for random forests in over 80% of the datasets.

Chapter 6

Conclusion and Future Works

This work is inspired by manual strategies for hyperparameter tuning of an ML algorithm. That strategy adjusts the hyperparameters for more model complexity when observing training error greater than target error, or reduces complexity if training error is reasonably small but validation error is not satisfied. One can do this because of prior knowledge that training error and the generalizability are both decreasing with respect to algorithm model complexity. The strategy implicitly decomposes the validation error into training error plus training error minus validation error (generalizability is the inverse of this difference). Our algorithm modifies the modeling part of Bayesian optimization to exploit this prior knowledge and the decomposition. The hope is that the decomposition of validation error simplifies the tuning problem by creating two functions that are easier to model: less nonlinear, for example. Moreover, approximate monotonicity provides further useful information for modeling and optimization. The experiments indicate that for ML hyperparameter tuning, monotonicity information can be exploited to tune more efficiently.

There exist at least two approaches for monotonic GP regression. Riihimäki and Vehtari [17] and Wang and Berger [19] enforced monotonicity by inserting virtual observations of derivative processes, and Lin and Dunson [10] projected GPs onto a constrained space. An important question is where to insert the virtual points. The trivial method that we employed using a grid results in having exponentially many virtual points with respect to the number of hperparameters in the objective

function. Training a GP requires $O(n^3)$ computation with a large constant, however, where n is the number of observations. With limited computational power, methods that can assign virtual points as requested to fill the space more sparsely is more preferred, for example, Latin hypercube sampling [14]. A domain-specific adaptive method was proposed by Wang and Berger [19]. It sequentially determines one virtual point at a time at the location where the derivative process has the lowest (or highest) value. However this method itself requires repeating the training process and optimization of the learned process a linear number of times with respect to requested number of virtual points. Thus, it is more computationally expensive that non-adaptive methods.

Lin and Dunson [10]'s method first learns the posterior process without monotonicity information. Then it samples functions from the posterior process, and projects them into the monotonicity constrained space. The projection is defined to minimize an integral of square difference. It is not clear how samples from the posterior process are represented, but we believe a (possibly approximate) closed form projection can be derived with compact sample representation. If a closed formed solution is available, the monotonic GP regression required by our algorithm should share the same order of computation as standard GP regression.

Rather than adapting a GP model, one may exploit monotonicity constraints when using other surrogate models. A common alternative to GPs for Bayesian optimization is random forests [7]. A typical random forest ensembles many decision trees where each tree has a constant numerical prediction within each leaf. Model trees are an extension of decision trees with linear models in the leaves [12]. An easy way to combine monotonicity and random forests as the surrogate model for Bayesian optimization is to use an ensemble of model trees with the linear model within a leaf constrained to have non-negative (or non-positive) linear coefficients. With this approach, the curse of dimensionality is handled by the model trees' splits, which has been well studied and shown to give effective practical results.

For ML hyperparameter tuning problems, since the monotonicity of validation error minus training error is not guaranteed, one may wonder if imposing an approximate monotonicity constraint is appropriate. Here we make an argument that for an objective function that decomposes into two functions where one of them is known to be monotonic, there is an extra reason to model the other function as

monotonic beside the behavior of the function itself. Denote the decomposed functions by f_1 and f_2, with f_1 known to be monotonic. Consider the simple case where the domain of the objective function has dimension one. Without loss of generality assume f_1 to be monotonically increasing. Then for the interval $[a,b]$ where f_2 is increasing, $f_1(x)+f_2(x) \leq f_1(b)+f_2(b)$ for all $x \in [a,b]$. Hence a region where f_2 is not monotonically decreasing is not of interest. However, since we do not know in what region f_2 is increasing, enforcing f_2 to be monotonic may have a negative effect on accuracy of prediction when f_2 is in its decreasing region. This argument can be easily generalized to higher dimensions.

Although this work is motivated by and focused on machine learning hyperparameter problems, our algorithm is designed for any optimization problem with an objective function that can be decomposed as a sum of functions with monotonicity constraints. An example is to optimize a product's price for maximum gain. A simple but widely used model for this problem is that gain equals $d \times (p-c)$ with d for demand, p for price and c for cost. Cost is assumed to be constant, demand is a monotonically decreasing function of price. Hence optimizing $d \times (p-c)$ is equivalent to optimizing $\log(d)+\log(p-c)$ which are both monotonic with respect to p. Evaluating this is sometimes expensive since d given p may be modeled using complicated game theory models. A more complex example was given by Goettler and Gordon [4], where they proved three monotonicities of decomposed expected gain given the current product price and amount of investment for CPU companies.

Another constraint on the objective function that might be useful is to restrict the objective function value to a certain interval. For example, with many machine learning problems, the loss will never be negative. We believe this constraint is particularly useful with the monotonicity constraint. One indication is that we observed from experiments that although our algorithm evaluates corners of the search space less often compared to the standard algorithm, it still does so mainly because the GP model predicts large negative training error at one or more corners. The manual hyperparameter turning strategy that inspires our work would not increase model complexity when observing training error close to zero, because it utilizes both monotonic and training constraints. A similar argument would apply to the difference between validation error and training error. Furthermore, we believe that with more constrains on the objective function, it would be worth exploit-

ing a specialized acquisition function and its optimizer based on the constraints.

In conclusion, we propose a Bayesian optimization algorithm for objective functions that can be decomposed as a sum of functions with monotonicity constraints. We demonstrate that many machine learning hyperparameter tuning problems fall into this family. We present experimental results for optimizing an artificially designed problem, regularized linear regression on artificial data and decision tree regression, random forest regression and MLP on PMLB, which is a comprehensive collection of machine learning problems. They show our algorithm outperforms the standard Bayesian optimization method based on direct modeling of the validation error.

Bibliography

[1] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. → page 19

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Chapman and Hall/CRC, Boca Raton, 1984. → page 19

[3] E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010. → page 3

[4] R. L. Goettler and B. R. Gordon. Does amd spur intel to innovate more? *Journal of Political Economy*, 119(6):1141–1200, 2011. → page 27

[5] S. Golchi, D. R. Bingham, H. Chipman, and D. A. Campbell. Monotone emulation of computer experiments. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):370–392, 2015. → page 1

[6] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016. → page 12

[7] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. *LION*, 5:507–523, 2011. → pages 1, 26

[8] E. M. Johansson, F. U. Dowla, and D. M. Goodman. Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method. *International Journal of Neural Systems*, 2(04):291–301, 1991. → page 20

[9] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4): 455–492, 1998. → pages 1, 7

[10] L. Lin and D. B. Dunson. Bayesian monotone regression using Gaussian process projection. *Biometrika*, 101(2):303–317, 2014. → pages 4, 25, 26

[11] Z. Ma and A. Leijon. Bayesian estimation of beta mixture models with variational inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2160–2173, 2011. → page 13

[12] D. Malerba, A. Appice, M. Ceci, and M. Monopoli. Trading-off local versus global effects of regression nodes in model trees. In *International Symposium on Methodologies for Intelligent Systems*, pages 393–402. Springer, 2002. → page 26

[13] R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and Systems*, volume 3, pages 334–341, 2007. → page 1

[14] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979. → page 26

[15] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, Dec 2017. ISSN 1756-0381. doi:10.1186/s13040-017-0154-4. URL https://doi.org/10.1186/s13040-017-0154-4. → page 20

[16] C. E. Rasmussen and C. K. Williams. *Gaussian process for machine learning*. MIT press, 2006. → page 4

[17] J. Riihimäki and A. Vehtari. Gaussian processes with monotonicity information. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 645–652, 2010. → pages 4, 25

[18] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012. → page 1

[19] X. Wang and J. O. Berger. Estimating shape constrained functions using Gaussian processes. *SIAM/ASA Journal on Uncertainty Quantification*, 4(1): 1–25, 2016. → pages 4, 25, 26

[20] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005. → page 17