Automated Hyper-parameter Tuning for Machine Learning Models in Machine Health Prognostics

Wang-Chi Cheung [1], Weiwen Zhang[2] Yong Liu[3], Feng Yang[4], Rick-Siow-Mong, Goh [5]

[1, 2, 3, 4, 5] *Institute of High Performance Computing, Agency for Science, Technology and Research, Singapore*
cheungwc@ihpc.a-star.edu.sg, zhangww@ihpc.a-star.edu.sg, liuyong@ihpc.a-star.edu.sg,
yangf@ihpc.a-star.edu.sg, gohsm@ihpc.a-star.edu.sg

ABSTRACT

Recent studies have revealed the success of data-driven machine health monitoring, which motivates the use of machine learning models in machine health prognostic tasks. While the machine learning approach to health monitoring is gaining importance, the construction of machine learning models is often impeded by the difficulty in choosing the underlying hyper-parameter configuration (HP-config), which governs the construction of the machine learning model. While an effective choice of HP-config can be achieved with human effort, such an effort is often time consuming and requires domain knowledge. In this paper, we consider the use of Bayesian optimization algorithms, which automate an effective choice of HP-config by solving the associated hyper-parameter optimization problem. Numerical experiments on the data from PHM 2016 Data Challenge demonstrate the salience of the proposed automatic framework, and exhibit improvement over default HP-configs in standard machine learning packages or chosen by a human agent.

1. INTRODUCTION

With the prevalence of Machine Learning and the availability of low-cost sensors, data-driven machine health monitoring is gaining importance in modern manufacturing systems. While popular machine learning models, such as deep neural networks and random forests, give rise to highly accurate predictive models, the success of these models hinges on judicious choices of their underlying *hyper-parameter configurations* (HP-configs), which governs the construction of these models.

For example, in the case of deep neural networks, an HP-config corresponds to the choice of network architecture, such as the number of layers as well as the number of neurons in each layer, and the choice of stochastic gradient descent algorithms for training a network, such as Adagrad or ADAM. After the network architecture and the training algorithm are chosen, the decision maker optimizes the network weights in order to minimize the prediction error on a training dataset. While the decision maker desires to minimize the prediction error, the error crucially depends on the choice of the HP-config. Unfortunately, the size of the hyper-parameter space for a machine learning model is often too big for a brute-force search for a competent HP-config. The identification of a competent HP-config is often based on the experience of the decision maker, and the identification could be daunting and time-consuming for a new machine prognostic task.

We develop an automatic framework that identifies competent hyper-parameter configurations without conducting a brute-force search on the underlying hyper-parameter spaces. Our framework is based on Bayesian Optimization algorithms, which allows quick convergence to a competent hyper-parameter configuration. A Bayesian Optimization algorithm involves the construction of a certain stochastic surrogate function on the performance of every hyper-parameter configuration. We implement our automatic framework on the PHM 2016 Data Challenge Task (PHM, 2016) for fine-tuning popular machine learning models such as random forest regression model and multi-layer perceptrons. We witness reduction in prediction errors under our automatic framework, in comparison to the machine learning models constructed using default HP-configs postulated by practitioners in standard machine learning packages (Pedregosa et al., 2011).

1.1. Literature Review

There has been a line of research work studying the prognostics health modeling with machine learning technology. Baptista et al. in (Baptista et al., 2016) leveraged Support Vector Machines (SVM) to estimate the lifetime for maintenance in aeronautics. Experiment results show that the proposed method can have better estimation than traditional autoregressive moving average (ARMA) method. Liu et al. in

(Liu, Zuo, & Qin, 2016) also leveraged SVM to perform the health state assessment and remaining useful lifetime (RUL) prediction modeling for rolling element bearings. The health states are classified into multiple classes, and individual regression model is separately built for each class to predict the RUL. Deutsch et al. in (Deutsch & He, 2016) leveraged Restricted Boltzman Machine (RBM) to model vibration data to predict the RUL of bearings. The feature vectors are manually designed, with the root mean square function to capture the bearings degradation over time. However, their deep learning approach achieved lower accuracy than the particle filter based approach. One way to improve the accuracy is to use stacked RBM structure with hyperparameter tuning. Chen et al. in (Chen, Lucas, Lee, & Buehner, 2015) leveraged neural network to predict gear failure. Similarly, Yang et al. in (Yang et al., 2016) leveraged neural network and ensemble multiple models to predict the remaining useful lifetime of electrical machines.

Another line of research leveraged deep learning for fault diagnosis and prognostics. He et al. in (He, He, & Bechhoefer, 2016) proposed a deep learning approach for feature extraction, called large memory storage retrieval neural network (LAMSTAR), to perform bearing fault diagnosis. Babu et al. in (Babu, Zhao, & Li, 2016) adopted Convolutional Neural Network (CNN) for RUL estimation in prognostics. The proposed deep architecture is claimed to learn the high-level feature efficiently from the low-level raw sensor signals, which can result in the higher accuracy for RUL estimation. Gugulothu et al. in (Gugulothu et al., 2017) proposed to use Recurrent Neural Networks (RNNs) for predicting remaining useful life of the engine and pump.

However, neither of those previous work considered the hyperparamter tuning of the machine learning algorithms for training the models.

In recent years, automated hyperparameter tuning has been gaining importance in the machine learning and artificial intelligence literature. A basic way to automatically tune hyperparmaeters is by the Random Search algorithm (Bergstra & Bengio, 2012), while more sophisticated Bayesian optimization algorithms such as SMAC (Hutter, Hoos, & Leyton-Brown, 2011) and GP (Srinivas, Krause, Kakade, & Seeger, 2010) are also proposed in the literature. For a survey on hyper-parameter tuning and its applications, the readers are welcome to consult the survey (Shahriari, Swersky, Wang, Adams, & de Freitas, 2016).

1.2. Organization and Notations.

In Section 2, we introduce the concept of hyper-parameters in constructing machine learning models. In Section 3, we define the hyper-paramater optimization problem, and outline Bayesian optimization algorithms and the Random Search algorithms, which solve the optimization problem to near-optimality efficiently. In Section 4, we consider the PHM 2016 Data Challenge (PHM, 2016), and we evaluate the effectiveness of the algorithms in searching for a good hyper-parameter configuration automatically. Finally, we conclude in Secoin 5. Throughout the manuscript, we denote $\mathbf{R}$ as the set of real numbers, and denote $\mathbf{R}_{\geq 0}$ as the set of non-negative real numbers.

2. HYPER-PARAMETERS FOR MACHINE LEARNING

In this Section, we define the notion of a *hyper-parameter configuration* for a machine learning (ML) model, and provide illustrating examples with well-known ML models. Then, we define the notion of out-of-sample validation error, which paths our way to define the hyper-parameter optimization problem.

2.1. Machine Learning Models and Hyper-parameters

Let's consider a supervised machine learning (ML) task involving a collection of training dataset $\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}$, where $\boldsymbol{X}_i^{\text{tr}} \in \mathcal{X} \subseteq \mathbf{R}^D$ is the feature vector of the ith training sample, $y_i^{\text{tr}} \in \mathcal{Y} \subseteq \mathbf{R}$ as the label of the ith training sample, and N^{tr} is the number of training samples. In the context of machine health monitoring, feature vector $\boldsymbol{X}$ could be the time series sensor data measured on a machine part in a time interval, and label y could be the corresponding remaining useful life. The goal of an ML task is to construct an ML model $M : \mathcal{X} \to \mathcal{Y}$, so that the output $M(\boldsymbol{X})$ accurately predicts the corresponding label y. The ML model M belongs to a parameterized class of ML models, and the construction is to identify an effective choice of the parameters that makes the ML model M accurate.

The construction procedure $\mathcal{A}$ of an ML model M crucially depends on the training dataset $\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}$, as well as the underlying HP-config $\boldsymbol{c}$. An HP-config $\boldsymbol{c}$ governs the way M is constructed. For example, an HP-config $\boldsymbol{c}$ determines the pre-processing procedure on the training dataset $\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}$, and the algorithmic details in determining M, which often involve solving an optimization problem. Altogether, we express the procedure as a function on both the training dataset and the HP-config:

$$\mathcal{A}\left(\{\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}}\}_{i=1}^{N^{\text{tr}}}, \boldsymbol{c}\right) = M.$$

To provide a more concrete discussion, we illustrate the HP-configs involved in linear regression models and multi-layer perceptron models.

Linear regression: A linear regression model M_{LR} is parameterized by a vector $\boldsymbol{\theta} \in \mathbf{R}^D$. For a given feature vector $\boldsymbol{X} \in \mathcal{X} \subseteq \mathbf{R}^D$, the model postulates a linear prediction, i.e. $M_{\text{LR}}(\boldsymbol{X}) = \boldsymbol{\theta}^\top \boldsymbol{X}$, which is the dot product between $\boldsymbol{X}, \boldsymbol{\theta}$.[1]

[1] Sometimes, a linear regression model requires appending a "y-intercept",

With a given training dataset $\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}$, the vector of parameters $\boldsymbol{\theta}$ is the optimal solution of the following empirical risk minimization problem:

$$\min_{\theta \in \mathbf{R}^D} \frac{1}{N^{\text{tr}}} \sum_{i=1}^{N^{\text{tr}}} \|\boldsymbol{\theta}^\top \boldsymbol{X}_i^{\text{tr}} - y_i^{\text{tr}}\|_2 + \kappa \|\boldsymbol{\theta}\|_p. \tag{1}$$

The regularization term $\kappa\|\boldsymbol{\theta}\|_p$, where $(\kappa, p) \in \mathbf{R} \times [1, \infty]$, serves to stabilize the optimal solution $\boldsymbol{\theta}$. For example, the choice of $p = 1$ corresponds to the LASSO regularization, and the choice of $p = 2$ corresponds to the Euclidean regularization.

The HP-config for constructing M_{LR} is $\boldsymbol{c}_{\text{LR}} = (\kappa, p)$, and the construction procedure $\mathcal{A}_{\text{LR}}(\{\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}}\}_{i=1}^{N^{\text{tr}}}, \boldsymbol{c}_{\text{LR}})$ involves solving optimization problem (1) for the parameter vector $\boldsymbol{\theta}$.

Multi-layer perceptrons: A multi-layer perceptron (MLP) model M_{MLP} is a feed-forward neural network, which inputs a feature vector $\boldsymbol{X}$ and outputs a prediction $M_{\text{MLP}}(\boldsymbol{X})$ for the label y. The model M_{MLP} is parameterized by the edge weights in the network. Each internal node in the network carries an activation function, which is a uni-variate nonlinear function that takes as input a linearly weighted sum of the preceding layer's outputs, and returns a scalar value for the next layer.

Compared to the case of M_{LR}, the construction of ML model M_{MLP} involves a more complicated HP-config $\boldsymbol{c}_{\text{MLP}}$. The HP-config $\boldsymbol{c}_{\text{MLP}}$ consists of HPs in the following two categories. The first category concerns the architecture of the network, such as the number of layers, the number of nodes in each layer, as well as the type of activation function in each layer. The second category concerns the training procedure of the MLP model. It is well known that an MLP is trained by the Backward Propagation algorithm $\mathcal{A}_{\text{MLP}}$, which is a stochastic gradient descent algorithm that incrementally adjusts the network weights in a series of iterations. Relevant HPs include the mini-batch size, the learning rate, etc.

Finally, while the discussions above focus on the HPs regarding the algorithmic details in determining an ML model, an HP-config could also contain hyper-parameters regarding the pre-processing of the training dataset, as shown in our study on the PHM 2016 prognostic challenge in Section 4.

2.2. Out-of-Sample Validation Error

As exemplified in the preceding examples, the choice of an HP-config has a profound impact on the resulting machine learning model. More precisely, an HP-config $\boldsymbol{c}$ (together with the training data) determines the resulting machine learning model M, which we wish to evaluate its prediction accuracy on a validation dataset. We evaluate the effectiveness of an HP-config $\boldsymbol{c}$ by considering its *out-of-sample validation error*, which is denoted as $\text{OOSV}(\boldsymbol{c})$.

To define $\text{OOSV}(\boldsymbol{c})$, we first denote $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbf{R}_{\geq 0}$ as the prediction error function, where $\ell(M(\boldsymbol{X}), y)$ is the prediction error of $M(\boldsymbol{X})$ on the true label y. For example, for $\ell(y, y') = |y - y'|$, the error $\ell(M(\boldsymbol{X}), y)$ is the absolute error of the prediction $M(\boldsymbol{X})$ on the label y.

Suppose that we are given a training dataset $\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}$, as well as a validation dataset $\{(\boldsymbol{X}_i^{\text{va}}, y_i^{\text{va}})\}_{i=1}^{N^{\text{va}}}$ that is separate from the training dataset. For a given HP-config $\boldsymbol{c}$, the corresponding out-of-sample validation error $\text{OOSV}(\boldsymbol{c})$ is equal to

$$\text{OOSV}(\boldsymbol{c}) := \frac{1}{N^{\text{va}}} \sum_{i=1}^{N^{\text{va}}} \ell(M(\boldsymbol{X}_i^{\text{va}}), y_i^{\text{va}}),$$

where $M = \mathcal{A}(\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}, \boldsymbol{c})$. Essentially, $\text{OOSV}(\boldsymbol{c})$ evaluates the performance of the resulting ML model M, which is constructed with the training dataset $\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}$, on another set of unseen validation data $\{(\boldsymbol{X}_i^{\text{va}}, y_i^{\text{va}})\}_{i=1}^{N^{\text{va}}}$. To identify an effective HP-config $\boldsymbol{c}$, we would like to identify a $\boldsymbol{c}$ such that $\text{OOSV}(\boldsymbol{c})$ is small by using Bayesian optimization algorithms, which are introduced and motivated in the next section.

3. BAYESIAN OPTIMIZATION FOR HYPER-PARAMETER OPTIMIZATION

In this Section, we define the hyper-parameter optimization problem, and illustrate its intractability. Then, we outline Bayesian optimization algorithms for solving the hyper-parameter optimization problem to near-optimality efficiently, and highlight two prominent Bayesian optimization algorithms, SMAC and GP. Finally, we also provide the description of Random Search Algorithm, which is a simple but effective algorithm for identifying a good HP-config.

Suppose that we are given a fixed training dataset $\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}$ and validation dataset $\{(\boldsymbol{X}_i^{\text{va}}, y_i^{\text{va}})\}_{i=1}^{N^{\text{va}}}$, and we restrict our search of HP-config $\boldsymbol{c}$ in the search space C. The hyper-parameter optimization problem is defined as follows:

$$\min_{\boldsymbol{c} \in C} \text{OOSV}(\boldsymbol{c}). \tag{2}$$

Given an optimal solution $\boldsymbol{c}^*$ to the optimization problem (2), we can then perform $\mathcal{A}(\boldsymbol{c}^*, \{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}})$, which returns an ML model M^*, our desired ML model that has a low out-of-sample error (In particular, a minimum prediction error on the validation dataset $\{(\boldsymbol{X}_i^{\text{va}}, y_i^{\text{va}})\}_{i=1}^{N^{\text{va}}}$).

Unfortunately, the optimization problem (2) is usually intractable. Indeed, the search space is typically large. In addition, the function OOSV is typically not convex nor monotonic in $\boldsymbol{c}$. Moreover, the evaluation of OOSV could be computationally expensive. For example, in the case of MLP, the evaluation requires training deep neural networks on the

where $\boldsymbol{\theta}$ is appended by a y-intercept θ_0, and each feature vector $\boldsymbol{X}$ is appended by 1.

training data, which could be time consuming when the number of layers is large or when the training dataset is large. Consequently, it is infeasible to conduct a brute force search, and to evaluate OOSV on every HP-config $\boldsymbol{c}$ for minimizing OOSV. This motivates the design of Bayesian optimization algorithms, which converge to a near-optimal HP-config $\boldsymbol{c}$ with a small number of evaluations of OOSV.

To illustrate the idea of Bayesian optimization, we first think about how a human agent tunes the HP-config for constructing an ML model. Typically, the human agent starts with a few randomly chosen HP configs to get a sense of how different HP-configs perform. After that, the human agent is able to gain some intuition about the effects of varying different hyper-parameters. He/she is then able to narrow down the search space C in his/her subsequent search. For example, in training MLPs, the human agent could discover certain characteristics about learning rates. For example, it could be taht the training process always diverges when the learning rate is higher than 0.3, but always converges when the learning rate is lower than 0.1. Based on this insight, he/she can focus on trying various HP-configs with learning rates at most 0.1. While such an effort often leads to a competent choice of HP-config, the process of extracting such insights is often laborious, and requires domain knowledge. Is it possible to automate such a HP-config optimization process, and save the laborious effort by humans?

Bayesian optimization algorithms provide an answer to the question above. A Bayesian optimization algorithm is an online algorithm, which involves evaluating different HP-configs in iterations. At iteration t, a Bayesian optimization determines the HP-config $\boldsymbol{c}_t \in C$ for evaluation, based on the previously tested HP-configs $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{t-1}$ and their respective evaluations $\mathrm{OOSV}(\boldsymbol{c}_1), \ldots, \mathrm{OOSV}(\boldsymbol{c}_{t-1})$. Such a online decision procedure mirrors the sequential nature of the human agent, who tries to optimize the choice of HP-config by a series of trial-and-errors on different HP-configs, as previously discussed.

We provide the pseudo-codes for a typical Bayesian optimization algorithm in Algorithm 1. Essentially, the online selection of HP-configs $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_T \in C$ is guided by the functions $\widetilde{\mathrm{OOSV}}_1, \ldots, \widetilde{\mathrm{OOSV}}_T$, which serve to approximate the intractable function OOSV. In the end, a Bayesian algorithm returns a HP-config $\boldsymbol{c}_T^*$, which has the best evaluated function value of OOSV among the evaluated HP-configs $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_T$. To fully specify a Bayesian optimization algorithm, we need to specify the construction of $\widetilde{\mathrm{OOSV}}_t : C \to \mathbf{R}$, which is an approximation function to the function $\mathrm{OOSV} : C \to \mathbf{R}$. The construction is based on $\{(\boldsymbol{c}_s, \mathrm{OOSV}(\boldsymbol{c}_s)\}_{s=1}^{t-1}$. The approximation function $\widetilde{\mathrm{OOSV}}_t$ serves to crystallize the insights gained in experimenting $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_t$, similar to how a human agent extracts insights based on his/her experience on different HP-configs in his/her experimentation, as previously discussed. By putting forth a Bayesian optimization algorithm, we automate the HP-config optimization procedure. The construction of $\widetilde{\mathrm{OOSV}}_t$ is based on certain statistical techniques, hence the name Bayesian optimization algorithms.

Algorithm 1 Algorithmic framework of Bayesian optimization, with T evaluations of OOSV

1: Let $\widetilde{\mathrm{OOSV}}_1 : C \to \mathbf{R}_{\geq 0}$ be a random function.
2: **for** $t = 2, \ldots, T$ **do**
3: Construct $\widetilde{\mathrm{OOSV}}_t$ based on $\{(\boldsymbol{c}_s, \mathrm{OOSV}(\boldsymbol{c}_s)\}_{s=1}^{t-1}$.
4: Compute HP-config $\boldsymbol{c}_t$, which solves

$$\min_{\boldsymbol{c} \in C} \widetilde{\mathrm{OOSV}}_t(\boldsymbol{c}).$$

5: Evaluate OOSV at $\boldsymbol{c}_t$, which returns $\mathrm{OOSV}(\boldsymbol{c}_t)$.
6: **end for**
7: Return $\boldsymbol{c}_T^*$, where $\boldsymbol{c}_T^* \in \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_T\} \subset C$, and $\mathrm{OOSV}(c_T^*)$ is the smallest among the evaluations $\{\mathrm{OOSV}(\boldsymbol{c}_1), \ldots, \mathrm{OOSV}(\boldsymbol{c}_T)\}$.

Algorithm 2 Random Search algorithm with T evaluations of OOSV

1: **for** $t = 1, 2, \ldots, T$ **do**
2: Sample HP-config $\boldsymbol{c}_t$ uniformly at random from C.
3: Evaluate OOSV at $\boldsymbol{c}_t$, which returns $\mathrm{OOSV}(\boldsymbol{c}_t)$.
4: **end for**
5: Return $\boldsymbol{c}_T^*$, where $\boldsymbol{c}_T^* \in \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_T\} \subset C$, and $\mathrm{OOSV}(\boldsymbol{c}_T^*)$ is the smallest among the evaluations $\{\mathrm{OOSV}(\boldsymbol{c}_1), \ldots, \mathrm{OOSV}(\boldsymbol{c}_T)\}$.

There are two prominent ways to construct an approximation function $\widetilde{\mathrm{OOSV}}_t$, giving rise to the following two prominent Bayesian optimization algorithms:

1. SMAC, proposed by (Hutter et al., 2011), which constructs the approximation function $\widetilde{\mathrm{OOSV}}_t$ by random forest regression on $\{(\boldsymbol{c}_s, \mathrm{OOSV}(\boldsymbol{c}_s))\}_{s=1}^{t}$,
2. GP, proposed by (Srinivas et al., 2010), which constructs the approximation function $\widetilde{\mathrm{OOSV}}_t$ by combining Gaussian Processes regression on $\{(\boldsymbol{c}_s, \mathrm{OOSV}(\boldsymbol{c}_s))\}_{s=1}^{t}$ with optimistic exploration.

More details about SMAC and GP could be found in the survey (Shahriari et al., 2016). Finally, apart from Bayesian optimization algorithms, the Random Search algorithm is also a way to compute an efficient HP-config under the intractability of the hyper-parameter optimization problem (2), cf. reference (Bergstra & Bengio, 2012). Essentially the Random Search algorithm with T evaluations on OOSV simply involves evaluating OOSV on T randomly chosen HP-configs. Then, the algorithm returns the HP-config with the smallest value evaluated under OOSV. The Random Search algorithm is stated in Algorithm 2. While the Random Search algorithm is conceptually simple, it is often inferior compared to Bayesian optimization algorithms in identifying a competent

HP-config, as illustrated in the numerical experiments in the next Section.

4. HYPER-PARAMETER TUNING FOR MACHINE HEALTH PROGNOSTIC TASKS WITH PHM 2016 DATASETS

In this Section, we demonstrate the effectiveness of Bayesian optimization for automating the HP-config optimization of the machine health prognostic task for the PHM 2016 dataset (PHM, 2016). We first describe the ML task and the dataset involved in Section 4.1. Then we describe in Section 4.2 our construction procedure $\mathcal{A}_{\text{PHM}}$ for constructing an ML model M_{PHM} for the PHM 2016 Data Challenge task. We also highlight the HP-config $\boldsymbol{c}_{\text{PHM}}$ involved in the construction. Then, in Section 4.3, we define the hyper-parameter optimization problem for the task, and also provide the search space C_{PHM} for the optimization. Finally, we present the numerical results in Section 4.4.

4.1. Prognostic Task in the PHM 2016 Data Challenge

A brief description of the task. The PHM 2016 data challenge task involves the investigation of a wafer Chemical-Mechanical Planarization (CMP) tool that removes material from the surface of the wafer through a polishing process. The goal of the task is to predict the average polishing removal rate, based on the sensor data collected during the polishing process. The challenge task (PHM, 2016) provides a collection of time-series-label pairs, where each pair records the sensor data recorded during a polish process.

Data description. The time-series-label pair for the ith polishing process is denoted as $(\boldsymbol{X}_i, y_i)$. The time series data is expressed by the matrix $\boldsymbol{X}_i \in \mathbf{R}^{26\times T_i}$, where T_i is the number of time steps, and there are 26 sensor reading at a time step.[2] Different processes could have different time lengths, i.e. T_i could vary with different i. The 26 entries include sensor readings such as chamber pressure, usage measure of the dresser in the wafer CMP tool, etc. For more details, please consult (PHM, 2016). Finally, the label $y_i \in \mathbf{R}_{\geq 0}$ is the average polishing removal rate in the ith polishing process.

The collection of time-series-label pairs is organized in three datasets: the training dataset $\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}$, the test dataset $\{(\boldsymbol{X}_i^{\text{te}}, y_i^{\text{te}})\}_{i=1}^{N^{\text{te}}}$, and the validation dataset $\{(\boldsymbol{X}_i^{\text{va}}, y_i^{\text{va}})\}_{i=1}^{N^{\text{va}}}$, which contain $N^{\text{tr}} = 1977$, $N^{\text{te}} = 424$, $N^{\text{va}} = 424$ time-series-label pairs respectively. Typically, we have T_i roughly equal to 300 in each dataset, but the quantity T_i could vary from pair to pair. The label y_i in a time-series-label pair is typically lies in the range $[40, 200]$.

Objective. The objective of the PHM data challenge task is to construct a machine learning model M_{PHM} , so that the mean squared error

$$\frac{1}{N^{\text{te}}}\sum_{i=1}^{N^{\text{te}}}(M_{\text{PHM}}(\boldsymbol{X}_i^{\text{te}}) - y_i)^2 \tag{3}$$

on the test dataset $\{(\boldsymbol{X}_i^{\text{te}}, y_i^{\text{te}})\}_{i=1}^{N^{\text{te}}}$ is minimized.

4.2. Description of Our ML Model Construction Procedure $\mathcal{A}_{\text{PHM}}$

In this subsection, we define our construction $\mathcal{A}_{\text{PHM}}$ of the desired machine learning model M_{PHM}, and also explain the HP-config c_{PHM} involved in the construction. By the definition of an ML model construction, we have

$$\mathcal{A}_{\text{PHM}}(\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}, \boldsymbol{c}_{\text{PHM}}) = M_{\text{PHM}}. \tag{4}$$

Before describing $\mathcal{A}_{\text{PHM}}$ we first provide a high level view on the HP-config $\boldsymbol{c}_{\text{PHM}}$. The HP-config $\boldsymbol{c}_{\text{PHM}}$ consists of hyper-parameters in pre-processing the time-series-label-pairs, as well as the hyper-parameters in tuning the random forest regression model[3] used in the construction. In the Appendix, we replace the random forest regression in $\mathcal{A}_{\text{PHM}}$ by a multi-layer perceptron model.

Now, the construction procedure $\mathcal{A}_{\text{PHM}}$ is described in the pseudo-codes in Algorithm 3. The procedure $\mathcal{A}_{\text{PHM}}$ mainly consists of two steps. First, it involves a pre-processing step called *block decomposition*, which transform each time-series-label pair $(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})$ into a number of *block-augmented-label* pairs $\{(\tilde{\boldsymbol{X}}_{i,j}^{\text{tr}}, \tilde{y}_{i,j}^{\text{tr}})\}_{j=1}^{\boldsymbol{c}_{\text{num-block}}}$. The block decomposition step serves to extract succinct feature vectors from the time-series-label pairs. Second, it involves building a random forest regression model M_{RF}, where this ML model inputs a block, and output a prediction on the augmented label. The procedure $\mathcal{A}_{\text{PHM}}$ finally outputs M_{RF}.

The pre-processing step BLOCKDEC is detailed in Algorithm 4. Essentially, BLOCKDEC extracts a number of blocks $\{\tilde{\boldsymbol{X}}_{i,j}^{\text{tr}}\}_{j=1}^{c_{\text{num-block}}}$ from the time series data $\boldsymbol{X}_i^{\text{tr}}$, and attaches an augmented label $\tilde{y}_{i,j}^{\text{tr}}$ to block $\tilde{\boldsymbol{X}}_{i,j}^{\text{tr}}$. In the algorithm, the notation $\boldsymbol{X}[:, a : b]$ denotes the sub-matrix of $\boldsymbol{X}$ formed by the columns $a, a+1, \ldots, b$ of $\boldsymbol{X}$. Essentially, the blocks $\{\tilde{\boldsymbol{X}}_{i,j}^{\text{tr}}\}_{j=1}^{c_{\text{num-block}}}$ are $26\times c_{\text{num-block}}$ sub-matrices, where each two consecutive blocks are t_{sep} time steps apart. The quantity t_{sep} is chosen such that the blocks are as spread out as possible. Each block serves as a "snapshot" on the time series of sensor data, and each augmented label serves as a proxy for the removal rate during the snapshot. We define the augmented label as $\tilde{y}_{i,j}^{\text{tr}} = y_i^{\text{tr}}$, that is, the removal rates across different blocks are constant. We believe that, with more information about the wafer CMP tool, we can enrich the augmented labels.

[2] The time unit for a time step is not given in the task.

[3] For a brief introduction to the random forest regression model, please consult (Pedregosa et al., 2011).

Algorithm 3 Procedure $\mathcal{A}_{\text{PHM}}$ for PHM 2016 Data Challenge

1: Input data: training data $\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}$, where $\boldsymbol{X}_i^{\text{tr}} \in \mathbf{R}^{26\times T_i}, y_i^{\text{tr}} \in \mathbf{R}$.
2: Input HP-config: HP-config $\boldsymbol{c} = (\boldsymbol{c}_{\text{pre}}, \boldsymbol{c}_{\text{RF}})$.
3: **for** $i \in \{1, \ldots, N^{\text{tr}}\}$ **do**
4: Compute the Block Decomposition on $(\tilde{\boldsymbol{X}}_{i,j}^{\text{tr}}, \tilde{y}_{i,j}^{\text{tr}})$:

$$\{(\tilde{\boldsymbol{X}}_{i,j}^{\text{tr}}, \tilde{y}_{i,j}^{\text{tr}})\}_{j=1}^{c_{\text{num-block}}} = \text{BLOCKDEC}((\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}}), \boldsymbol{c}_{\text{pre}}).$$

5: **end for**
6: Collate the blocks into one grand dataset

$$D = \{(\tilde{\boldsymbol{X}}_{i,j}^{\text{tr}}, \tilde{y}_{i,j}^{\text{tr}})\}_{j=1,i=1}^{c_{\text{num-block}}, N^{\text{tr}}}.$$

7: Construct a random forest regression model M_{RF} on D:

$$M_{\text{RF}} = \mathcal{A}_{\text{RF}}(D, \boldsymbol{c}_{\text{RF}})$$

8: Return the random forest regression model M_{RF}.

Finally, we note that the ML model M_{RF} output by procedure $\mathcal{A}_{\text{PHM}}$ only provide prediction for a block, but not for a time-series sensor data $\boldsymbol{X}$ in general. Nevertheless, we can readily use BLOCKDEC in conjunction with M_{RF} for predicting the removal rate for a time-series sensor data, as illustrated in Algorithm 5.

Altogether, we have defined the construction procedure $\mathcal{A}_{\text{PHM}}$ in Algorithm 3. It inputs the training dataset $\{(\boldsymbol{X}_i^{\text{tr}}, y_i^{\text{tr}})\}_{i=1}^{N^{\text{tr}}}$ and an HP-config c_{PHM}, and returns a random forest regression model M_{RF}, which can be used to predict the average removal rate by Algorithm 5. The out-of-sample validation error $\text{OOSV}_{\text{PHM}}(\boldsymbol{c}_{\text{PHM}})$, which is on the validation dataset $\{(\boldsymbol{X}_i^{\text{va}}, y_i^{\text{va}})\}_{i=1}^{N^{\text{va}}}$, is defined as follows. First, we compute the random forest regression model M_{RF} according to equation (4). Then, for each time series data $\boldsymbol{X}_i^{\text{va}}$, we compute the prediction $\hat{y}_i^{\text{va}}$ according to Algorithm 5. Finally, we have

$$\text{OOSV}_{\text{PHM}}(\boldsymbol{c}_{\text{PHM}}) = \frac{1}{N^{\text{va}}} \sum_{i=1}^{N^{\text{va}}} (\hat{y}_i^{\text{va}} - y_i^{\text{va}})^2, \tag{5}$$

which is the out-of-sample mean squared error on the validation dataset. This serves to indicate the mean squared error on the test dataset, which is

$$\text{Test}_{\text{PHM}}(\boldsymbol{c}_{\text{PHM}}) = \frac{1}{N^{\text{te}}} \sum_{i=1}^{N^{\text{te}}} (\hat{y}_i^{\text{te}} - y_i^{\text{te}})^2.$$

4.3. Hyper-parameter optimization for the PHM 2016 Data Challenge

After describing $\mathcal{A}_{\text{PHM}}$ and explaining the HP-config c_{PHM}, we now define the hyper-parameter optimization involved in the Data Challenge task. We aim to solve the optimization

Algorithm 4 Pre-processing Routine BLOCKDEC

1: Input: a time-series-label pair $(\boldsymbol{X}, y)$, where $\boldsymbol{X} \in \mathbf{R}^{26\times T}, y \in \mathbf{R}$.
2: Input HP: $\boldsymbol{c}_{\text{pre}} = (c_{\text{num-block}}, c_{\text{len-block}})$.
3: Compute the number of time steps t_{sep} between consecutive blocks:

$$t_{\text{sep}} = \lfloor \frac{T - c_{\text{num-block}} * c_{\text{len-block}}}{c_{\text{num-block}} - 1} \rfloor.$$

4: Extract $c_{\text{num-block}}$ blocks, denoted $\tilde{\boldsymbol{X}}_1, \ldots, \tilde{\boldsymbol{X}}_{c_{\text{num-block}}}$, which are $\mathbf{R}^{26\times c_{\text{len-block}}}$ sub-matrices of $\boldsymbol{X}$. Denote stride $= c_{\text{len-block}} + t_{\text{sep}}$. For each $j = 1, \ldots, c_{\text{num-block}}$, the block $\tilde{\boldsymbol{X}}_j$ is defined as follows:

$$\tilde{\boldsymbol{X}}_j = \boldsymbol{X}[:, (j-1)*\text{stride}+1 : (j-1)*\text{stride}+c_{\text{len-block}}].$$

Note that $\tilde{\boldsymbol{X}}_j \in \mathbf{R}^{26\times c_{\text{len-block}}}$.
5: For each $j \in \{1, \ldots, c_{\text{num-block}}\}$, define $\tilde{y}_j = y$.
6: Output: $\{(\tilde{\boldsymbol{X}}_j, \tilde{y}_j)\}_{j=1}^{c_{\text{num-block}}}$.

Algorithm 5 Using M_{RF} to predict removal rates

1: Input: a time series $\boldsymbol{X} \in \mathbf{R}^{26\times T}$.
2: Decompose $\boldsymbol{X}$ into blocks $\tilde{\boldsymbol{X}}_1, \ldots, \tilde{\boldsymbol{X}}_{c_{\text{num-block}}}$, according to Algorithm 4.
3: **for** $j \in \{1, \ldots, c_{\text{num-block}}\}$ **do**
4: Compute $\hat{y}_j = M_{\text{RF}}(\tilde{\boldsymbol{X}}_j)$.
5: **end for**
6: Return the average $\hat{y} = \frac{1}{c_{\text{num-block}}} \sum_{j=1}^{c_{\text{num-block}}} \hat{y}_j$ as the prediction for the true removal rate y.

problem

$$\min_{\boldsymbol{c}_{\text{PHM}} \in C_{\text{PHM}}} \text{OOSV}_{\text{PHM}}(\boldsymbol{c}_{\text{PHM}}), \tag{6}$$

where the function OOSV_{PHM} is defined in equation (5). The HP-config search space C_{PHM} is displayed in Table 1. The search space C_{PHM} is the Cartesian product of the search ranges shown in Table 1. An HP-config can be retrieved by taking one element in the search range in each row. Note that the first two HPs are for the pre-processing procedure BLOCKDEC. The last four HPs are for the random forest regression model M_{RF}. For more details about these HPs, please consult the documentation from Scikit-Learn (Pedregosa et al., 2011), a Python package for ML which is used for our experimentation. It is interesting to note that there are altogether

$$7 \times 5 \times 51 \times 9 \times 10 \times 9 \approx 1.446 \times 10^6$$

many HP-configs in the search space C_{PHM}. Such a large space precludes any possibility of a brute-force search for the HP-config that minimizes OOSV_{PHM}. In the next Section, we solve the HP-config optimization problem (6) to near optimality by the Bayesian optimization algorithms SMAC and GP, as well as the Random Search Algorithm. For each of these algorithms, we perform 200 function evaluations. We

Table 1. Table of HP-config search space C_{PHM} for $\mathcal{A}_{\text{PHM}}$.

HP	Search range
$c_{\text{num-block}}$	$\{2, 3, 4, 5, 6, 8, 10\}$
$c_{\text{len-block}}$	$\{12, 24, 36, 48, 60\}$
No. trees	$\{10, 11, \ldots, 60\}$
Max features	$\{$"auto", "sqrt", "log2", $0.3, 0.4, \ldots, 0.8\}$
Max depth	$\{$None, $2, 3, \ldots, 10\}$
Min split	$\{2, 3, \ldots, 10\}$

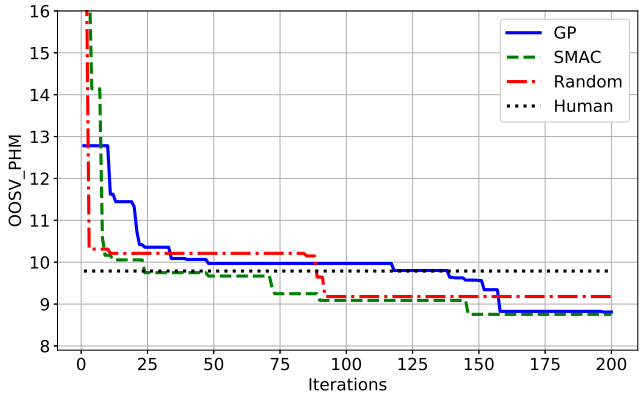

Figure 1. OOSV_{PHM} with the best HP-config identified so far.

compare the performance of these algorithms with the default HP-config defined by a human agent.

4.4. Numerical Results

In this Section, we provide the experiment results on hyper-parameter optimization on the PHM 2016 Data Challenge task. From the results, we see that Bayesian optimization algorithms are able to identify better HP-configs than the default HP-config set by a human agent, which can be found under the "Human" column in Table 2. Here, the default hyper-parameters for pre-processing the data are chosen based on a few trial and errors. The default hyper-parameters for the random forest regression model follows (Pedregosa et al., 2011).

First, we consider Figure 1, which provides a macroscopic view on these algorithms by comparing their out-of-sample validation errors, and also compares these errors with the validation error under the default HP-config. For each of these algorithms, we plot the best-so-far out-of-sample validation errors, which are defined as follows. First, recall that the sequence of HP-configs experimented by an algorithm is denoted as $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_T$. Now, for each $t \in \{1, \ldots, 200\}$, we first identify $\boldsymbol{c}_t^* \in \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_t\}$, for which $\text{OOSV}_{\text{PHM}}(\boldsymbol{c}_t^*) = \min_{s \in \{1,\ldots,t\}} \text{OOSV}_{\text{PHM}}(\boldsymbol{c}_s)$. Thus, $\text{OOSV}_{\text{PHM}}(\boldsymbol{c}_t^*)$ is the best out-of-sample validation error achieved by the HP-configs in $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_t\}$. Altogether, the sequence of best-so-far out-of-sample validation errors under an algorithm is $\text{OOSV}_{\text{PHM}}(\boldsymbol{c}_1^*), \text{OOSV}_{\text{PHM}}(\boldsymbol{c}_2^*), \ldots, \text{OOSV}_{\text{PHM}}(\boldsymbol{c}_{200}^*)$.

Clearly, for each algorithm, the sequence is a non-increasing sequence, and the corresponding plotted curve of the best-so-far out-of-sample errors in Figure 1 drops once the algorithm identifies a better HP-config than those experimented in previous iterations.

Figure 1 shows that algorithms GP, SMAC and Random Search are able to identify better HP-configs (in terms of out-of-sample errors) as each of these algorithms experiment with more HP-configs. We see that all three algorithms identify better HP-configs than the default (in terms of out-of-sample errors) at the end of 200 function evaluations. Moreover, Bayesian algorithms GP and SMAC, which conduct experimentation on HP-config in a more principled manner than the Random Search Algorithm, are able to identify better HP-config than the Random Search Algorithm.

To shed light on the process of hyper-parameter optimization, we then proceed to Figure 2. For each of the algorithms, both the sequence of best-so-far out-of-sample validation errors $\{\text{OOSV}_{\text{PHM}}(\boldsymbol{c}_t^*)\}_{t=1}^{200}$ and the sequence of out-of-sample validation errors $\{\text{OOSV}_{\text{PHM}}(\boldsymbol{c}_t)\}_{t=1}^{200}$ are plotted in solid lines and dots respectively.

We observe that Algorithm GP is the most stable, in the sense that the sequence of out-of-sample validation errors (plotted in dots) is close to the sequence best-so-far out-of-sample validation errors (plotted in a line). Algorithm SMAC also manifests such a stability. Nevertheless, the Random Search algorithm is far less stable. The sequence of out-of-sample validation errors is very far away (from above) compared to the sequence best-so-far out-of-sample validation errors. This signifies that Random Search could converge to a competent HP-config slowly, different from Bayesian optimization algorithms.

Next, in Figure 3, we compare the out-of-sample errors on the validation set (computed by evaluating OOSV_{PHM}) with the errors on the testing set. For each of algorithms GP, SMAC, Random Search, we plot the sequence $\{\text{OOSV}_{\text{PHM}}(\boldsymbol{c}_t^*)\}_{t=1}^{200}$ for showing the validation errors and the sequence $\{\text{Test}_{\text{PHM}}(\boldsymbol{c}_t^*)\}_{t=1}^{200}$ for showing the testing errors. The plots in Figure 3 shows that the constructed ML models generalize well, in the sense that the trend of the testing errors follows the trend of the validation errors.

In Figure 4, we provide a comparison of testing errors between different HP-config optimization algorithms, and compare these errors with the baseline by the default HP-config. It is demonstrated that all 3 algorithms achieve performance superior to the human baseline, signifying the value in HP-config optimization, which automates the process for finding a competent set of HP-config.

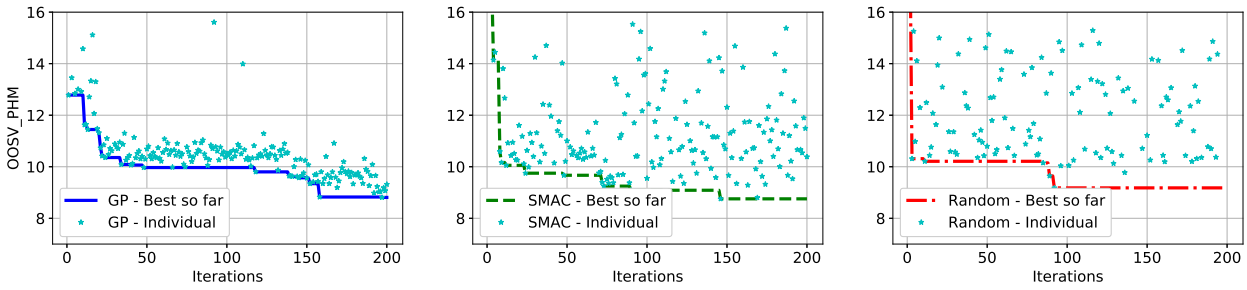

Figure 2. HP-config optimization process on OOSV_{PHM}.

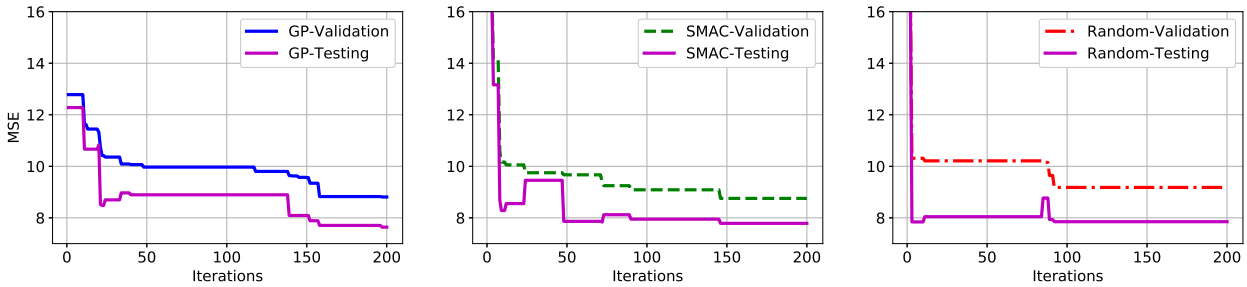

Figure 3. Performance of HP-configs on the validation (evaluated by OOSV_{PHM}) and testing datasets.

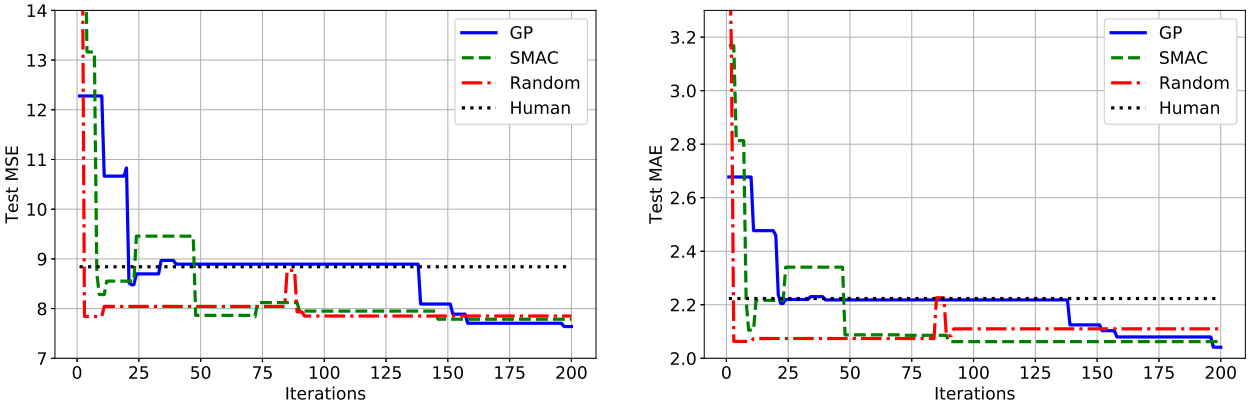

Figure 4. Mean squared error and mean absolute error on the testing dataset.

Table 2. Table of HP-configs $\boldsymbol{c}_T^*$ identified through automated HP-config tuning, and a comparison with a HP-config chosen by a human agent.

	GP	SMAC	Random	Human
$c_{\text{num-block}}$	3	5	5	2
$c_{\text{len-block}}$	48	60	60	12
No. trees	60	24	40	10
Max features	"log 2"	"log 2"	"log 2"	"auto"
Max depth	None	None	None	None
Min split	2	6	6	2
Train CV-MSE	8.810	8.755	9.180	9.789
Test MSE	7.639	7.786	7.786	8.842
Test MAE	2.041	2.062	2.062	2.223

Finally, Table 2 provides the competent HP-configs identified by GP, SMAC, Random Search, as well as the default HP-config used by a human agent. The default HPs for the random forest regression model follow (Pedregosa et al., 2011).

5. CONCLUDING REMARKS

In conclusion, we have introduced hyper-parameter optimization and its applications to automatically find a good hyper-parameter configuration in machine learning tasks for machine health prognostics. We evaluate hyper-parameter optimization algorithms on the PHM 2016 Data Challenge, which demonstrate promising results. A future direction is to seek a way to automate the process of feature selection and feature engineering by a similar idea.

REFERENCES

Babu, G. S., Zhao, P., & Li, X.-L. (2016). Deep convolutional neural network based regression approach for estimation of remaining useful life. In *International conference on database systems for advanced applications* (pp. 214–228).

Baptista, M., de Medeiros, I. P., Malere, J. P., Prendinger, H., Nascimento Jr, C. L., & Henriques, E. (2016). Improved time-based maintenance in aeronautics with regressive support vector machines. In *Annual conference of the prognostics and healthmanagement society.*

Bergstra, J., & Bengio, Y. (2012, February). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, *13*, 281–305.

Chen, Y., Lucas, C., Lee, J., & Buehner, M. (2015). Neural network-based gear failure prediction in a brushless dc actuation system. In *Proceedings of the annual conference of the prognostics and health management society.*

Chollet, F., et al. (2015). *Keras.* `https://keras.io`.

Deutsch, J., & He, D. (2016). Using deep learning based approaches for bearing remaining useful life prediction. In *Annual conference of the prognostics and health management society.*

Gugulothu, N., TV, V., Malhotra, P., Vig, L., Agarwal, P., & Shroff, G. (2017). Predicting remaining useful life using time series embeddings based on recurrent neural networks. *arXiv preprint arXiv:1709.01073*.

He, M., He, D., & Bechhoefer, E. (2016). Using deep learning based approaches for bearing fault diagnosis with ae sensors. In *Annual conference of the prognostics and health management society.*

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Proc. of lion-5* (p. 507523).

Liu, Z., Zuo, M. J., & Qin, Y. (2016). Remaining useful life prediction of rolling element bearings based on health state assessment. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, *230*(2), 314–330.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

PHM, S. (2016). *PHM 2016 Data Challenge.* `https://www.phmsociety.org/events/conference/phm/16/data-challenge`.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & de Freitas, N. (2016, Jan). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, *104*(1), 148-175.

Srinivas, N., Krause, A., Kakade, S., & Seeger, M. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th international conference on international conference on machine learning* (pp. 1015–1022).

Yang, F., Habibullah, M. S., Zhang, T., Xu, Z., Lim, P., & Nadarajan, S. (2016). Health index-based prognostics for remaining useful life predictions in electrical machines. *IEEE Transactions on Industrial Electronics*, *63*(4), 2633–2644.

APPENDIX

6. HYPER-PARAMETER OPTIMIZATION FOR THE PHM 2016 CHALLENGE TASK WITH MULTI-LAYER PERCEPTRONS

In this Section, we conduct hyper-parameter optimization by Algorithms GP, SMAC and Random Search in a similar way to Section 4. The only difference is that we use multi-layer perceptrons (MLPs) instead of random forest. Since the experimental set-up and the results are similar to those in Section 4, we only elaborate on the main difference, which is on the HP-config space.

Here, we consider a two-layer perceptron model. First, we provide the descriptions for hyper-parameters concerning the network architecture. The internal layers are layers 1, 2, while layer 3 is the output layer. For each $i = 1, 2$, we use Hidden i to denote the number of nodes in layer i, Activation i to denote the type of activation function used in the layer, and Dropout i to denote the dropout rate in the layer. Second, we provide the description for the hyper-parameters for training the MLP by the Backward Propagation (BP) algorithm. Batch size is the number of samples to feed into the BP algorithm every iteration. Learning rate, Decay and Momentum concern the rate at which the BP algorithm absorbs the information carried by each mini-batch. For more information about training neural networks, please consult (Chollet et al., 2015). The search space for the HP-config optimization task is provided in Table 3.

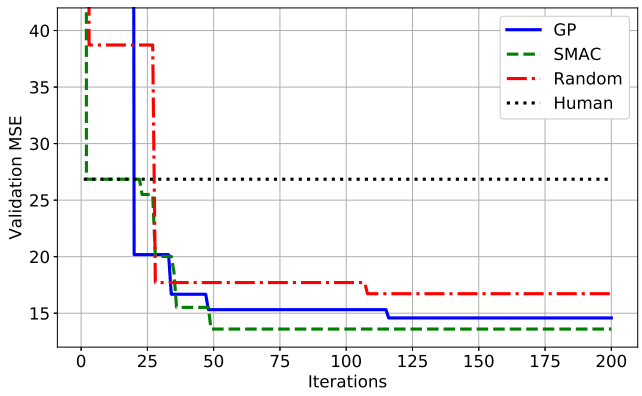

Figure 5. Validation errors with the best HP-config so far.

Finally, the numerical results for hyper-parameter optimization for improving MLP models can be found in Figures 5, 6, 7, 8. In general, the discussions for Figures 5, 6, 7, 8 are similar to the discussions for Figures 1, 2, 3, 4. In addition, the HP-configs identified by algorithms GP, SMAC, Random Search as well as the default HP-config by a human agent are provided in Table 4. The defualt HP-config is identified through trial and error by a human agent. Interestingly, the dropout rates should always be kept at 1.

Table 3. Table of Hyper-parameter search space for predicting the average removal rate with MLP models.

HP	Search range
$c_{\text{num-block}}$	$\{2, 3, \ldots, 10\}$
$c_{\text{len-block}}$	$\{2, 4, 8, 16, 24, 32, 64\}$
Hidden 1	$\{16, 32, 64, 128, 256, 512\}$
Activation 1	{"selu", "relu", "tanh", "sigmoid"}
Dropout 1	$\{1., 0.9, 0.8\}$
Hidden 2	$\{16, 32, 64, 128, 256, 512\}$
Activation 2	{"selu", "relu", "tanh", "sigmoid"}
Dropout 2	$\{1., 0.9, 0.8\}$
Activation 3	{"selu", "relu", "tanh", "sigmoid"}
Batch size	$\{4, 8, 16, 24, 32\}$
Learning rate	$\{0.05, 0.01, 0.008, 0.005, 0.001\}$
Decay	$\{0., 1e-6, 1e-5, 1e-4\}$
Momentum	$\{0.9, 0.8, 0.7, 0.6, 0.\}$

Table 4. Table of HP-configs identified through automated HP-config tuning, and a comparison with a HP-config chosen by a human agent. (sigm = sigmoid)

	GP	SMAC	Random	Human
$c_{\text{num-block}}$	10	7	6	9
$c_{\text{len-block}}$	16	24	24	16
Hidden 1	128	128	512	512
Activation 1	"tanh"	"relu"	"relu"	"sigm"
Dropout 1	1.	1.	1.	1.
Hidden 2	512	256	64	512
Activation 2	"relu"	"sigm"	"tanh"	"sigm"
Dropout 2	1.	1.	1.	1.
Activation 3	"selu"	"selu"	"selu"	"selu"
Batch size	4	8	8	24
Learning rate	0.05	0.05	0.01	0.008
Decay	1e-5	1e-4	1e-6	1e-6
Momentum	0.7	0.9	0.8	0.
Train CV-MSE	14.582	13.599	16.725	26.851
Test MSE	13.412	14.708	17.993	28.512
Test MAE	2.837	2.942	3.318	4.138

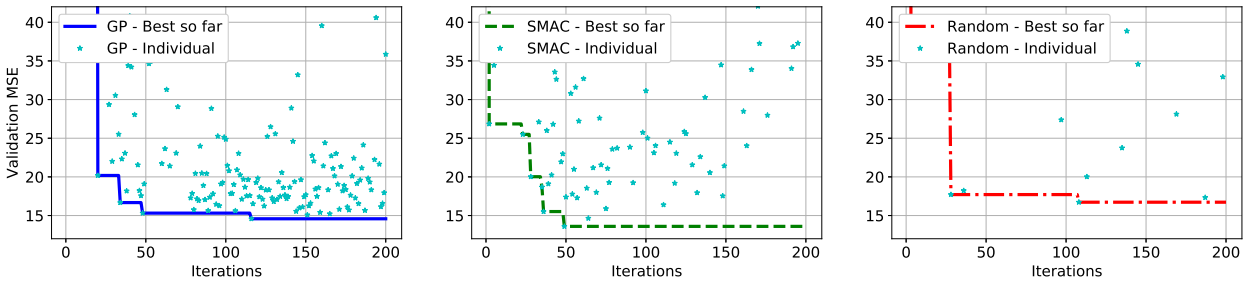

Figure 6. HP-config optimization process for MLPs.

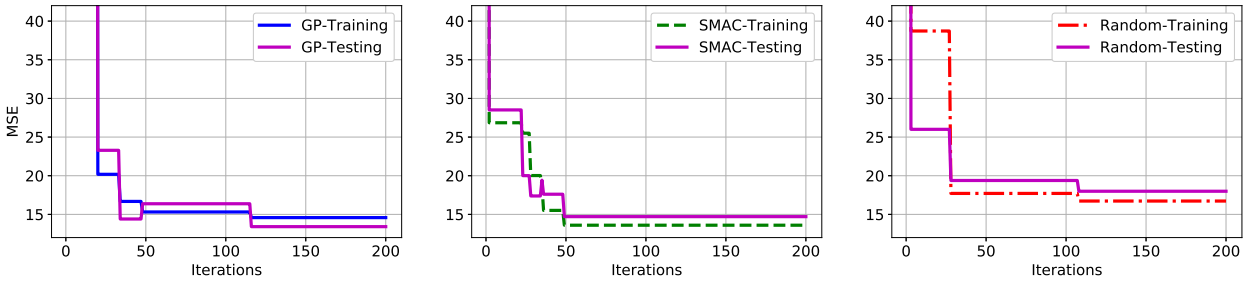

Figure 7. Performance of HP-configs on the validation and testing datasets under MLP.

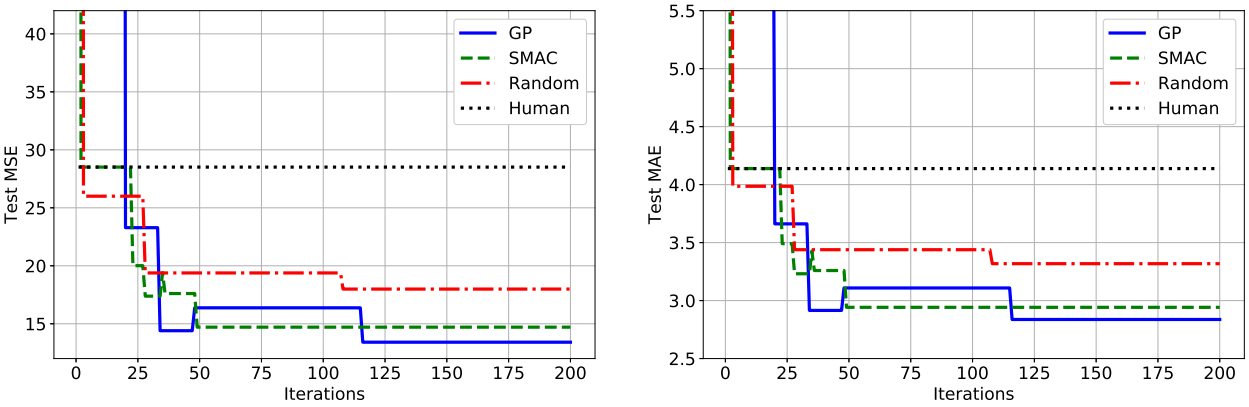

Figure 8. Mean squared error and mean absolute error on the testing dataset under MLP.