2016 23rd International Conference on Pattern Recognition (ICPR)
Cancún Center, Cancún, México, December 4-8, 2016

Hyperparameter Tuning for Big Data using Bayesian Optimisation

Tinu Theckel Joy, Santu Rana, Sunil Gupta, Svetha Venkatesh
Centre for Pattern Recognition and Data Analytics, Deakin University, Australia 3216
{ttheckel, santu.rana, sunil.gupta, svetha.venkatesh}@deakin.edu.au

***Abstract*—Hyperparameters play a crucial role in the model selection of machine learning algorithms. Tuning these hyperparameters can be exhaustive when the data is large. Bayesian optimisation has emerged as an efficient tool for hyperparameter tuning of machine learning algorithms. In this paper, we propose a novel framework for tuning the hyperparameters for big data using Bayesian optimisation. We divide the big data into chunks and generate hyperparameter configurations for the chunks using the standard Bayesian optimisation. We utilise the information from the chunks for the hyperparameter tuning for the big data using a transfer learning setting. We evaluate the performance of the proposed method on the task of tuning hyperparameters of two machine learning algorithms. We show that our method achieves the best available hyperparameter configuration within less computational time compared to the state-of-art hyperparameter tuning methods.**

I. Introduction

Machine learning has become an essential driving force for the developments in science and technology in today's world. The successful deployment of the machine algorithms highly depend on the model parameters which are extremely difficult to tune. Bayesian optimisation offers powerful solutions in the hyperparameter tuning space [1]. Bayesian optimisation has captured the attention of the machine learning scientists as an efficient tool for hyperparameter tuning, especially for complex models like deep neural networks [1], [2], [3]. It offers an efficient framework for optimising the highly expensive black-box functions without knowing its form [4]. Bayesian optimisation has been successfully applied in many different fields including learning optimal robot mechanics [5], sequential experimental design [6], synthetic gene design [7], etc.

The standard Bayesian optimisation suffers from a “cold start” problem when a new tuning problem is tackled. It may recommend bad hyperparameter settings in the initial trials. Bayesian optimisation can be costly especially when the model is learnt over a large volume of data. Many training iterations need to be performed before a good set of hyperparameters is found.

A plausible solution is to leverage the information from the past models for tuning models on the big the data. Yogatama and Mann [8] use a Bayesian optimisation setting to transfer the knowledge from one dataset to the next by using a common function to model deviations from the per dataset mean. Yogatama and Mann [8] assumes strong similarity in the deviations from the respective means between the two tasks. However, this is not practical in all applications. In one of our recent work [9], we have proposed a novel transfer learning based Bayesian optimisation algorithm that can handle even when two tasks are semi-related. We achieved this by modeling the observations from the previous task as a noisy observations of the succeeding task. We compute the noise in a Bayesian method. Although our method showed promise, it still is not specifically developed for big data application and parallel computing. In big data application, it may be useful to exploit parallel computation during tuning. Recently, Klein et al. [10] try to address this problem by considering dataset size as an additional factor for selection while performing the optimisation. However, they find that their Entropy Search variant of the acquisition function has shortcomings while finding the minimum, which leads the model to perform poorly [10]. Hence the task of tuning the hyperparameters on big data is still an open problem.

Addressing this problem, we propose an efficient framework for hyperparameter tuning on big data using knowledge acquired from the small chunks of data. We divide the whole big data into small chunks of fixed size. We then perform the hyperparameter tuning on these chunks using Bayesian optimisation in parallel. We then average the response surface of the chunks to get a smooth response of the different hyperparameter settings across the chunks. We leverage the observations from the chunks by modeling them as noisy observations on the big data. We assume that the response surface of the chunks and the big data lie within some envelop of each other. The closeness of these two surfaces is determined by the width of the envelop. We estimate the width of the envelop in a Bayesian method.

We prove the efficiency of the proposed method empirically by on two bench-marked real-world datasets and two classification algorithms, Deep neural Networks and Support Vector Machines(SVM) with Radial Basis Kernel(RBF). We divide the big training data into small chunks of data of same size. We generate hyperparameter settings on the chunks using Bayesian optimisation in parallel. The model performance is evaluated on a separate held out dataset. Since the cost of conducting Bayesian optimisation on the small chunks is small and they can be run in parallel, we efficiently leverage the combined knowledge from the chunks to conduct Bayesian optimisation on the big training data. Our experiment clearly demonstrates the improved efficiency of our algorithm over the state of the art methods. We empirically evaluate the com-

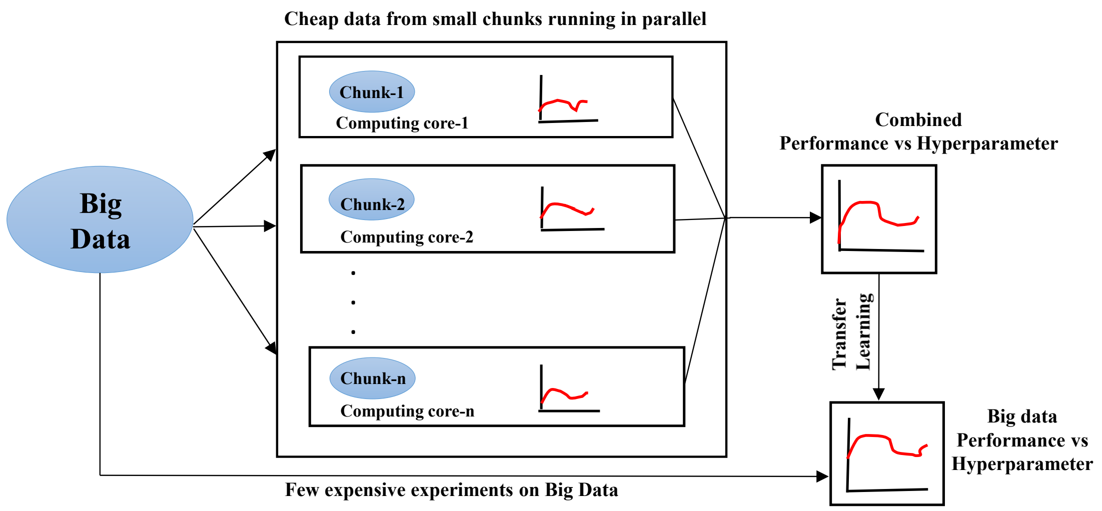
Cheap data from small chunks running in parallel
Big Data
Chunk-1
Computing core-1
Chunk-2
Computing core-2
Chunk-n
Computing core-n
Combined Performance vs Hyperparameter
Transfer Learning
Big data Performance vs Hyperparameter
Few expensive experiments on Big Data

Figure 1. Illustration of the proposed hyperparameter tuning framework for Big Data. The idea is to generate cheaper data from chunks using paralleilsation and then using the combined performance vs hyperparameter data to learn the actual performance vs hyperparameter for the big data in a few experiments utilising transfer learning.

putational effectiveness of our algorithm in all the experiments (time consumed to achieve the best setting).

In short our contributions are:

- Proposal of a new framework for hyperparameter tuning on big data. We propose an efficient Bayesian optimisation framework which uses posterior distribution of a combined GP, obtained by averaging the GPs fitted at the chunks of big data.
- Proposal of a Novel setting, where we divide the big data into small chunks, generate hyperparameter settings for these chunks in parallel and leverage this information to boost the search for a good set of hyperparameters on the big data.
- Evaluation of the proposed algorithm for the best hyperparameter search for two machine learning algorithms on two bench-marked real-world classification datasets. We prove the efficiency of our algorithm on the big data theoretically and empirically. We also compare the efficiency of the method by comparing it with the two transfer learning frameworks [8], [9] and the standard Bayesian optimisation.

II. Preliminaries

A. Gaussian Process

A Gaussian process is a collection of infinite number of random variables, for which any finite number of combination has a joint Gaussian distribution [11] . Gaussian processes (GPs) offer a way of specifying prior distributions over the space of smooth functions. It is specified by its mean function, $\mu(\mathbf{x})$ and covariance function, $k(\mathbf{x},\mathbf{x}^{'})$. Using the property of Gaussian distributions, the predictive mean and variance can be computed in the closed form.A sample from a Gaussian process is a function as,

$$f(\mathbf{x}) \sim \mathcal{GP}(\mu(\mathbf{x}), k(\mathbf{x},\mathbf{x}^{'})). \tag{1}$$

where the value $f(\mathbf{x})$ at an arbitrary $\mathbf{x}$ is a Gaussian distributed random variable specified by a mean and a variance. The prior mean function can be assumed to be zero in Gaussian process without any loss of generality. This leaves the Gaussian process to be fully defined by the covariance functions. A popular choice of covariance function is squared exponential function,

$$k(\mathbf{x},\mathbf{x}') = exp(-\frac{1}{2}||\mathbf{x}-\mathbf{x}'||^2) \tag{2}$$

Matérn kernel is another popular choice of the kernel.

Let us start with data points $\mathbf{x}_{1:p}$ and the function values corresponding to those data points are sampled from the prior Gaussian process with mean zero and the covariance function $k(\mathbf{x}_i,\mathbf{x}_j)$. Let us denote $\mathbf{y}_{1:p} = f(\mathbf{x}_{1:p})$ as the function values corresponding to the data points $\mathbf{x}_{1:p}$. The function values $\mathbf{y}_{1:p}$ jointly follow a multivariate Gaussian distribution $\mathbf{y}_{1:p} \sim \mathcal{N}(\mathbf{0},\mathbf{K})$, where

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & \dots & k(\mathbf{x}_1,\mathbf{x}_t) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_p,\mathbf{x}_1) & \dots & k(\mathbf{x}_p,\mathbf{x}_p) \end{bmatrix} \tag{3}$$

is the positive definite gram matrix. For a new data point $\mathbf{x}_{p+1}$, let the function value be $\mathbf{y}_{p+1} = f(\mathbf{x}_{p+1})$. Then, by the properties of Gaussian process, $\mathbf{y}_{1:p}$ and $\mathbf{y}_{p+1}$ are jointly Gaussian as,

$$\begin{bmatrix} \mathbf{y}_{1:p} \\ \mathbf{y}_{p+1} \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^T & k(\mathbf{x}_{p+1},\mathbf{x}_{p+1}) \end{bmatrix} \tag{4}$$

where $\mathbf{k} = [k(\mathbf{x}_1,\mathbf{x}_{p+1})\ k(\mathbf{x}_2,\mathbf{x}_{p+1})\ \dots\ k(\mathbf{x}_p,\mathbf{x}_{p+1})]$. Using Sherman-Morrison-Woodburry [11] formula, the predictive distribution of the function value at a new location ($\mathbf{x}_{p+1}$) can be written as,

$$P(y_{p+1}|\mathbf{x}_{1:p},\mathbf{y}_{1:p}) \sim \mathcal{N}(\mu_p(\mathbf{x}_{p+1}),\sigma_p^2(\mathbf{x}_{p+1})) \tag{5}$$

where the predicted mean and the variance is given by $\mu_p(\mathbf{x}_{p+1}) = \mathbf{k}^T\mathbf{K}^{-1}\mathbf{y}_{1:p}$ and $\sigma(\mathbf{x}_{p+1}) = k(\mathbf{x}_{p+1},\mathbf{x}_{p+1}) - \mathbf{k}^T\mathbf{K}^{-1}\mathbf{y}_{1:p}$ respectively. If the observation is a noisy estimate of the actual function value i.e. $y = f(x)+\eta$, where $\eta \sim \mathcal{N}(0,\sigma^2_{noise})$ the modified predictive distribution becomes

$$P(y_{p+1}|\mathbf{x}_{1:p},\mathbf{y}_{1:p}) \sim \mathcal{N}(\mu_p(\mathbf{x}_{p+1}),\sigma_p^2(\mathbf{x}_{p+1})+\sigma^2_{noise}) \tag{6}$$

where mean $\mu_p$and variance σ_p^2 is given as,

$$\mu_p(\mathbf{x}_{p+1}) = \mathbf{k}^T[\mathbf{K}+\sigma^2_{noise}\mathbf{I}]^{-1}\mathbf{y}_{1:p} \tag{7}$$

$$\sigma_p^2(\mathbf{x}_{p+1}) = k(\mathbf{x}_{p+1},\mathbf{x}_{p+1}) - \mathbf{k}^T[\mathbf{K}+\sigma^2_{noise}\mathbf{I}]^{-1}\mathbf{y}_{1:p} \tag{8}$$

B. *Bayesian Optimisation*

Bayesian optimisation is an efficient method for the global optimisation of objective functions that are expensive to evaluate [4]. It is useful when the user doesn't have access to the function form. Bayesian optimisation is efficient when the user has access only to the noisy evaluations of the objective function. Applications of Bayesian optimisation include the hyperparameter tuning of machine learning models. The hyperparameter tuning can be really complex since the choice of hyperparameters highly affects the model performance.

The unknown function space is modeled using a Gaussian process. Bayesian optimisation uses a surrogate function, which are cheap to evaluate, to optimise the expensive functions. These surrogate utility functions are called acquisition functions. The role of the acquisition function is to guide us to reach the optimum of the underlying function. Acquisition functions are defined such that a high value of the acquisition function corresponds to an high value of the underlying function. The acquisition function is optimised to get the next point for evaluation.

1) Acquisition Functions: Acquisition function can be defined either using improvement based criteria or using confidence based criteria. Improvement based criteria such as Probability of Improvement (PI) [12] or Expected Improvement (EI) [13] results in maximizing the probability of improvement over the current best or the improvement in the expected sense. Another acquisition function, GP-UCB [14] uses the upper confidence bound of the GP predictive distribution. A combination of these acquisition functions are also used [6]. We use EI as the acquisition function in this paper because of its simplicity. A brief description of EI is provided below.

a) Expected Improvement (EI): Let us assume that our optimisation problem is optimizing $\arg\max_{\mathbf{x}} f(\mathbf{x})$ and the current best is at $\mathbf{x}^+ = \arg\max_{\mathbf{x}_i\in\mathbf{x}_{1:p}} f(\mathbf{x}_i)$. The improvement function is defined as,

$$I(\mathbf{x}) = \max\{0, f(\mathbf{x}) - f(\mathbf{x}^+)\} \tag{9}$$

The acquisition function is then defined on the expected value of $I(\mathbf{x})$ [4] as,

$$\arg\max_{\mathbf{x}} \mathbb{E}(I(\mathbf{x})|D_{1:p}) \tag{10}$$

where $D_{1:p} \equiv \{\mathbf{x}_{1:p},\mathbf{y}_{1:p}\}$. The analytic form of $E(I(\mathbf{x}))$ can be obtained as [13],

$$\mathbb{E}(I(\mathbf{x})) = \begin{cases}(\mu(\mathbf{x}) - f(\mathbf{x}^+))\Phi(z) + \sigma(\mathbf{x})\phi(z) & \text{if } \sigma(\mathbf{x}) > 0\\ 0 & \text{if } \sigma(\mathbf{x}) = 0\end{cases} \tag{11}$$

where $z = (\mu(\mathbf{x}) - f(\mathbf{x}^+))/\sigma(\mathbf{x})$. $\Phi(.)$ and $\phi(.)$ are the CDF and PDF of a standard normal distribution respectively.

The standard Bayesian optimisation algorithm is presented in 1.

Algorithm 1 The Generic Bayesian Optimisation Algorithm

```
1: Input: The initial observation $D \equiv \{\mathbf{x}_{1:p},\mathbf{y}_{1:p}\}$.
2: Output: $\{\mathbf{x}^*_n,\mathbf{y}^*_n\}_{n=1}^T$
3: for $n = 1,2,..T$
4:        Find $\mathbf{x}^*_n = \arg\max_{\mathbf{x}} \mathbb{E}(I(\mathbf{x})|GP(D))$ of (11).
5:        Evaluate the objective function: $\mathbf{y}^*_n = f(\mathbf{x}^*_n)$.
6:        Augment the observation set $D = D \cup (\mathbf{x}^*_n,\mathbf{y}^*_n)$.
7: end for
```

III. Proposed Method

The standard Bayesian optimisation can be costly when tuning the hyperparameters of complex models on big data. To improve this, we propose an efficient Bayesian optimisation framework that exploits parallelisation.

Let us consider a scenario where we have to tune the hyperparameters of the model on big data. In our setting, we denote the hyperparameter settings as $\mathbf{x}$ and the generalisation performance on the held out dataset as $\mathbf{y}$. In our setting, we have the performance of the model as function of hyperparameters on the big data, which is denoted as $f^b(.)$, where the superscript b denotes the big data. We start by dividing the whole training big data into chunks of data having the same size. The cost of tuning the hyperparameters on the chunks is small and can be performed in parallel. We leverage this information for tuning the hyperparameters on the big data. The performance of the model as a function of hyperparameters on these chunks of training data are denoted as $f^c(.)$, where the superscript c denotes the chunks. We tune the hyperparameters for m different chunks using Bayesian optimisation in parallel. The GPs fitted on the chunks are denoted as $GP^c_{1:m}$. We would like to leverage the information from the GPs fitted at the chunks.

We acquire the best possible hyperparameter settings on the chunks by performing Bayesian optimisation, which is computationally cheap. We believe that by dividing the big data into chunks, help us to get the behavior of hyperparameters across the whole data. The performance of the models on these chunks can be highly noisy. However, the averaged response surface of these chunks will be less noisy and the best performance in the averaged response surface of the hyperparameter settings across the chunks correspond to that of the big data.

A. Combined GP

We achieve the overall model performance across the data, by averaging the GPs fitted on the chunks, $GP^c_{1:m}$. Using the properties of GP, we define the combined GP as $GP_{combined} \sim \mathcal{N}(\mu_{combined}, \sigma^2_{combined})$ where mean and variance are given as follows,

$$\mu_{combined} = 1/m \sum_{i=1}^{m} (\mu_c)_i \tag{12}$$

$$\sigma^2_{combined} = 1/m^2 \sum_{i=1}^{m} (\sigma^2_c)_i \tag{13}$$

The Bayesian optimisation on the big data uses the combined GP $GP_{combined}$ as the posterior model for tuning the hyperparameters on the big data. We start the tuning task with initial observations $\{\mathbf{x}^b, \mathbf{y}^b\}$ on the big data. We then induce the information from the chunks to tune the hyperparameters on big data, $f^b(.)$. We model the observations from the chunks as a noisy observations on the big data, i.e, as a noisy measurement of $f^b(.)$,

$$\mathbf{y}^c_i = f^b(\mathbf{x}^c_i) + \alpha^c_i, \forall i = 1, \ldots, N^c \tag{14}$$

where α^c is a random variable which is defined as, $\alpha^c \sim \mathcal{N}(0, \sigma^2_c)$. We estimate α^c between the response surfaces of the hyperparameters on the big data and chunks in a Bayesian setting.

The kernel matrix for each of the chunk GPs, $\mathbf{K_c}$, is computed after combining the observations from chunks and the big data. The kernel matrix $\mathbf{K_c}$ is then updated by adding different noise variance levels as,

$$\mathbf{K_c} = \mathbf{K_c} + \begin{bmatrix} \sigma^2_c I_{N^c \times N^c} & \mathbf{0} \\ \mathbf{0}^T & \sigma^2_b I_{N^b \times N^b} \end{bmatrix} \tag{15}$$

where σ^2_c models the closeness between observations on the big data and each of the chunks and σ^2_b is the measurement noise variance for big data. N^c and N^b are the number of observations in the chunks GPs and the big data respectively. We update the response function for each of the chunk GPs every time a recommended hyperparameter setting is evaluated on the big data. The response surface of the chunks are updated such a way that the observations from the chunks are added with higher noise variance. The predictive mean μ_c and variance σ_c of each of the chunk GPs, GP^c, is obtained using their respective modified kernel matrices. We then combine the predictions on the chunks to form the $GP^{combined}$ and uses this as the posterior model for performing Bayesian optimisation on big data.

Estimation of Noise Variance (σ^2_c): We estimate the variance σ^2_c of α^cusing a Bayesian framework. We place an inverse gamma distribution with parameters a_0 and b_0 as the prior distribution, given as,

$$\sigma^2_c \sim InvGamma(a_0, b_0) \tag{16}$$

We start with a wide prior and then update the posterior from the observation of output value of the big data (y^b) and the data chunks (y^c) for the same hyperparameter setting. Since the inverse gamma is a conjugate prior to the variance, the posterior is also an inverse gamma distribution with updated parameters a_n and b_n as,

$$P(\sigma^2_c / \{\mathbf{y}^b_{ij} - \mathbf{y}^c_{ij}\}^N_{i=1}) \sim InvGamma(a_n, b_n) \tag{17}$$

Assuming the mean of the difference to be zero, the parameters $a_n$and $b_n$is updated as follows,

$$(a^i_n)_{i=1:m} = a_0 + n/2 \tag{18}$$

In 20, n refers to the number of trials and i denotes the particular chunk.

$$(b^i_n)_{i=1:m} = b_0 + \frac{\sum_{j=1}^{N} (y^b_{ij} - y^c_{ij})^2}{2} \tag{19}$$

We use the mode of the posterior distribution as the value of noise variance and it is given by,

$$(\sigma^2_c)_{i=1:m} = \frac{(b^i_n)_{i=1:m}}{(a^i_n)_{i=1:m} + 1} \tag{20}$$

The proposed algorithm is illustrated in Alg. 2.

Algorithm 2 Proposed Algorithm.

```
1: Input:
      Initial Observations:$\{\mathbf{x}^b, \mathbf{y}^b\}$,
2: Initial Settings:
      Generate hyperparameter settings for Chunks : $GP^c_{1:m}$
      Build the combined GP $GP^{combined}$ using (12) and (13).
3: Output:
      $\{\mathbf{x}^*_n, \mathbf{y}^*_n\}^T_{n=1}$.
4: for $n = 1, 2, ..T$
5:     Find $\mathbf{x}^*_n = \arg\max_{\mathbf{x}} \mathbb{E}(I(\mathbf{x}) | GP^{combined})$ of (11).
6:     Evaluate $\mathbf{y}^*_n = f^b(\mathbf{x}^*_n)$
8:     Compute $\mathbf{y}^c_n = f^c(\mathbf{x}^*_n)$ for different data chunks.
9:     Update $a^{1:m}_n$,$b^{1:m}_n$ and $\alpha_{1:m}$ using (18), (19) and 20)
10:    Compute $\mathbf{K}_c$ for each $GP^c$ by augmenting$\{\mathbf{x}^*_n, \mathbf{y}^*_n\}$.
11:    Update each of the chunk GPs $GP^c$ using (15).
11:    Update the combined GP $GP^{combined}$.
12: end for
```

IV. Experiments

We conduct experiments on two bench-marked real-world classification datasets. Experiments are performed to evaluate our algorithm with respect to the baselines on the efficiency of tuning hyperparameters for two classification algorithms.

We compare the proposed method with the following baselines:

- **Transfer-BO** [9]: In this transfer learning approach, Bayesian optimisation is carried out in the target task by leveraging the information from the source task. A small percentage of the training data acts as the source and the whole as the target task .
- **Efficient-BO** [8]: In this transfer learning approach, they learn a common function for source and target, where the common function is represented as deviations from their respective means.
- **Generic-BO**: Algorithm 1 is used only on task of tuning the hyperparameters for big data. This is the standard Bayesian optimisation method with out any transfer knowledge from previous experiments.

We tune the hyperparameters of two machine learning algorithms: Deep Neural Networks and Support Vector Machines(SVM) with radial basis function(RBF). We evaluate the results by comparing the performance achieved by different methods in a given computational time. The task of all the baselines is to tune the hyperparameters of two machine learning models on two real-world data sets.

For Transfer-BO [9] and Efficient-BO [8], we use 30% of the whole training data as the source and whole as the big data data. For the proposed method, we divide the big data into chunks and perform Bayesian optimisation on those. The hyperparameter settings for the chunks are then leveraged for the tuning task on big data. We report the test error of the model evaluated on a held out data set. All timings reported in the experiments, are computed for programs running on a 3.46 GHz dual processor with 96 GB RAM and NVIDIA Tesla M2070 GPU.

The two bench-marked real-world classification datasets used in the experiments are collected from [15], [16]. A brief description of the datasets are provided in Table I. Of the two datasets we have used in our paper, MNIST [15] is the bench-marked real-world multi-class classification dataset of handwritten digit recognition. This dataset contains a training set of 60000 and test set of 10000 instances. Another dataset, a8a [16] is the bench-marked real-world binary classification dataset. For this dataset, we have used a training set of 6000 and test set of 2400 instances.

Table I
Datasets used in the experiments.

Dataset	Number of Data points	Number of Features
MNIST	70000	784
a8a	8400	123

Table II
Search Space Configuration for the Hyperparameters.

Algorithm	Hyperparameter	Bound
SVM with RBF	**Cost C**	10^{-3}-10^{3}
SVM with RBF	**Kernel Length**	10^{-5}-10^{0}
Deep Neural Network	**Hidden Layers**	$1-3$
Deep Neural Network	**Neurons**	100 -800
Deep Neural Network	**Learning Rate**	10^{-3}-10^{0}
Deep Neural Network	**Batch Size**	100 -1000
Deep Neural Network	**Momentum**	10^{-3}-10^{0}
Deep Neural Network	**Drop-out Factor**	$10\% - 50\%$

The search space configuration of the hyperparameters for the two machine learning models are listed in Table II. We perform Bayesian optimisation on the logarithmic of the values of those hyperparameters where ranges are high. We conducted the experiments for SVM using the publicly available software, LIBSVM [16]. For deep neural networks, we used a package which is publicly available[1]. We perform Bayesian optimisation on the chunks in parallel by allotting dedicated workers. We achieve this by using the "Parallel Computing Toolbox" in matlab. The matlab implementation for the proposed algorithm on tuning the hyperparameters of Deep Neural Networks is publicly available[2].

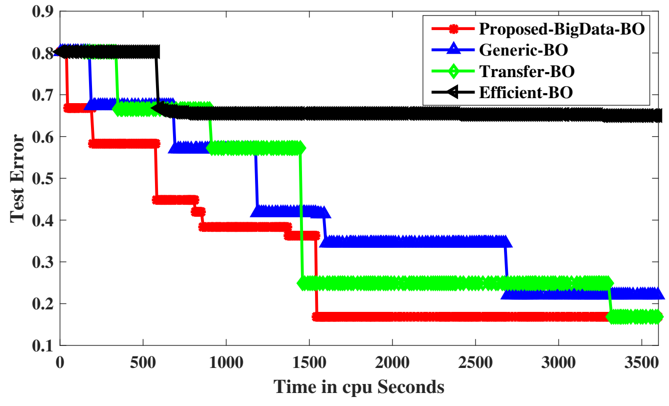

Figure 2. Tuning the hyperparameters of Deep Neural Networks on MNIST dataset .

The result of hyperparameter tuning for deep neural networks on the MNIST dataset is shown in Figure 2. The time spent for performing Bayesian optimisation on chunks for the proposed method and time spent for building the source for the two transfer learning methods are also taken into account while plotting the results. The results show that the proposed method achieves the best performance in small amount of time. The proposed method performs better than all the other baselines. The other methods take considerably long time to reach a good

[1]https://github.com/skaae/DeepLearnToolbox
[2]http://bit.ly/2bPpGdZ

hyperparameter configuration. Our previous method, Transfer-BO [9] performs better than the Generic-BO and the Efficient-BO [8]. The Efficient-BO[8] performs badly in this task since the deviation from the mean of two tasks is considerably different.

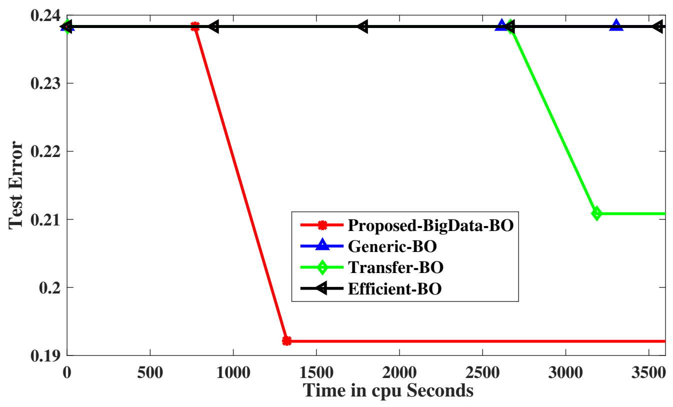

Figure 3. Tuning the Hyperparameters of SVM with rbf Kernel on a8a dataset: The Efficient-BO and Generic-BO did not move from the initial result and overlapped each other. .

The performance of the four methods on tuning the hyperparameters of support vector machine with radial basis function kernel on the a8a dataset is shown in Figure 3. The results show that the proposed method achieves the best configuration for the hyperparameters within less time. The proposed method performs better than our Transfer-BO [9]. All the other methods take significantly more time to achieve a good performance.

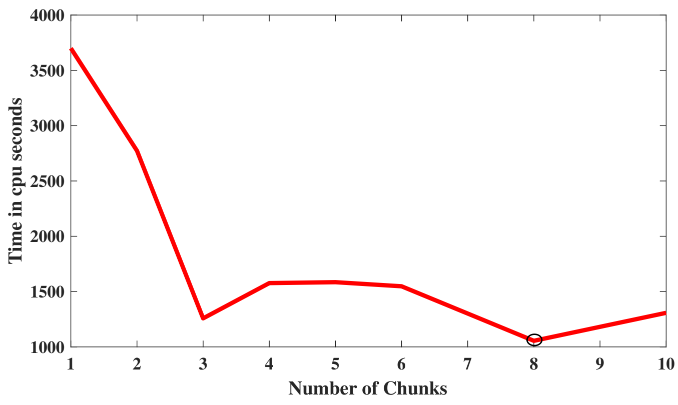

Figure 4. Tuning the Hyperparameters of Deep Neural Networks: Number of chunks against time taken in cpu seconds to achieve the best hyperparameter setting

We have also experimented on how the performance of the proposed method varies, when the number of chunks are changed. We plot the number of chunks against the time taken in cpu seconds to achieve the best hyperparameter settings in Figure 4. The Figure 4 shows that a moderate number of chunks between 3 to 10 are always a good option.

V. Conclusion

In this paper, we proposed a framework for hyperparameter tuning on big data using Bayesian optimisation. We divide the whole big data into chunks of data and generated the hyperparameter tuning on these chunks in parallel. We leverage this information for tuning the hyperparameter on the big data using transfer learning setting. We evaluate the performance of the proposed framework on the task of tuning the hyperparameters of two machine learning algorithms, deep neural network and svm with rbf kernel, on two real-world datasets. In all the experiments, we outperform the state-of-art hyperparameter tuning methods.

VI. Acknowledgement

This work is partially supported by the Telstra-Deakin Centre of Excellence in Big Data and Machine Learning.

References

[1] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[2] J. Bergstra, R. Bardenet, B. Kégl, and Y. Bengio, “Implementations of algorithms for hyper-parameter optimization,” in *NIPS Workshop on Bayesian optimization*, 2011, p. 29.

[3] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Auto-weka: Combined selection and hyperparameter optimization of classification algorithms,” in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 847–855.

[4] J. Mockus, “Application of bayesian approach to numerical methods of global and stochastic optimization,” *Journal of Global Optimization*, vol. 4, no. 4, pp. 347–365, 1994.

[5] D. J. Lizotte, T. Wang, M. H. Bowling, and D. Schuurmans, “Automatic gait optimization with gaussian process regression.” in *IJCAI*, vol. 7, 2007, pp. 944–949.

[6] 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.

[7] J. González, J. Longworth, D. C. James, and N. D. Lawrence, “Bayesian optimization for synthetic gene design,” *arXiv preprint arXiv:1505.01627*, 2015.

[8] D. Yogatama and G. Mann, “Efficient transfer learning method for automatic hyperparameter tuning,” *Transfer*, vol. 1, p. 1, 2014.

[9] T. T. Joy, S. Rana, S. K. Gupta, and S. Venkatesh, “Flexible transfer learning framework for bayesian optimisation,” in *Advances in Knowledge Discovery and Data Mining*. Springer, 2016, pp. 102–114.

[10] A. Klein, S. Bartels, S. Falkner, P. Hennig, and F. Hutter, “Towards efficient bayesian optimization for big data,” in *NIPS 2015 workshop on Bayesian Optimization (BayesOpt 2015)*, 2015.

[11] C. K. Williams and C. E. Rasmussen, “Gaussian processes for machine learning,” *the MIT Press*, vol. 2, no. 3, p. 4, 2006.

[12] H. J. Kushner, “A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise,” *Journal of Fluids Engineering*, vol. 86, no. 1, pp. 97–106, 1964.

[13] J. Mockus, V. Tiesis, and A. Zilinskas, “The application of bayesian methods for seeking the extremum,” *Towards Global Optimization*, vol. 2, no. 117-129, p. 2, 1978.

[14] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, “Gaussian process optimization in the bandit setting: No regret and experimental design,” in *Proc. International Conference on Machine Learning (ICML)*, 2010.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] C.-C. Chang and C.-J. Lin, “Libsvm: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.