

A Data Parallel Approach for Large-Scale Gaussian Process Modeling

*Arindam Choudhury, Prasanth B. Nair, and Andy J. Keane**

Abstract

This paper proposes an enabling data parallel local learning methodology for handling large data regression through the Gaussian Process (GP) modeling paradigm. The proposed model achieves parallelism by employing a specialized compactly supported covariance function defined over spatially localized clusters. The associated load balancing constraints arising from data parallelism are satisfied using a novel greedy clustering algorithm, GeoClust producing balanced clusters localized in space. Further, the use of the proposed covariance function as a building block for GP models is shown to decompose the maximum likelihood estimation problem into smaller decoupled subproblems. The attendant benefits which include a significant reduction in training complexity, as well as sparse predictive models for the posterior mean and variance make the present scheme extremely attractive. Experimental investigations on real and synthetic data demonstrate that the current approach can consistently outperform the state-of-the-art Bayesian Committee Machine (BCM) which employs a random data partitioning strategy. Finally, extensive evaluations over a grid-based computational infrastructure using the NetSolve distributed computing system show that the present approach scales well with data and could potentially be used in large-scale data mining applications.

Keywords

Gaussian Processes, Regression, Kernel Machines, Parallel Architectures, Grid Computing.

*Arindam Choudhury, Prasanth B. Nair and Andy J. Keane {A.Choudhury, P.B.Nair, Andy.Keane}@soton.ac.uk, are with the Computational Engineering and Design Center, School of Engineering Sciences, University of Southampton

1 Introduction

The torrential amounts of data in today's world poses new challenges to the statistics and the machine learning communities. Indeed, the past few years have seen a fury of research at all levels for finding ways and means of tackling with the immensity of data. This is being adequately supported by the explosive growth in hardware technology for serial and parallel computing architectures.

Over the last decade, kernel machines such as Gaussian Processes (GPs) and Support Vector Machines (SVMs) have emerged as some of the most popular machine learning tools. This is perhaps because of their impressive generalization capabilities over a range of applications. The focus of the present work is on GP regression [1, 2, 3] which is a powerful kernel machine with demonstrated effectiveness in a wide range of application areas. GP regression arises from a natural Bayesian treatment of the regression problem using priors over the class of functions intended to be modeled. The fully Bayesian treatment makes the error bars for prediction readily available. As an aside, unlike neural networks or even SVMs, GP models can be used off-the-shelf since they have no free parameters to be fiddled with. Finally, GP models naturally compute an automatic relevance determination (ARD) of the input variables which can be exploited in exploratory data analysis applications.

However, a common problem with most kernel machines is their poor scaling with data size. For example, typical training times for such machines may scale as $\mathcal{O}(n^3)$, where n is the size of the training data, along with a storage requirement which scales as $\mathcal{O}(n^2)$. Clearly, training such machines for large data rapidly becomes computationally intractable. Recent research has intensively focussed on these problems and the current state-of-the-art algorithms especially for SVMs have managed to bring down training times closer to $\mathcal{O}(n^2)$ [11, 13, 14] through innovative means and clever heuristics. In the context of GP regression, researchers have attempted to tackle the computational cost issue through Skilling's approximations[1], online learning techniques based on sequential model updates [6], and model sparsification strategies[8, 9].

A popular approach to scaling machine learning algorithms is that of Divide And Conquer (DAC). In general, the DAC approach is used in the context of *meta-learning*, which involves the generation of an ensemble of global models over random partitions of data. The basic rationale behind this approach is that computational and storage complexity can be reduced by partitioning the original data into smaller manageable chunks and then applying a base learner to each of the partitions in sequential or parallel mode. Conceptually, this approach is related to the idea of a Mixture of Experts(ME) [10], the only difference being that the experts are trained on partitions and not the whole data. Prediction at a new point is done by suitably aggregating the outputs from all the experts. Recent examples of such meta-learning approaches include parallel mixtures of SVMs[12], combining local experts through density estimation[15], Bayesian Committee Machines[16] and growing decision trees on random data partitions[24].

For the various reasons mentioned above, we have chosen to follow a DAC philosophy towards building a scalable GP model. However, we abandon the meta-learning approach in favor of a local learning strategy. In contrast to meta-learning,

local learning strategies construct models for spatially localized partitions of the data. The motivation for this arises from the fact that local learning techniques are known to perform better than meta-learning techniques which suffer from high variance and poor generalization. Vapnik (1996) proposed a local learning technique which creates the model at runtime after the testing point is known. This is in fact a lazy learning strategy analogous to the k-nearest neighbor algorithm which selects a subset of points from the training data closest to the testing point to make a prediction. Even though, this approach has provided breakthrough performance in some application domains, the associated computational cost precludes its application in practice.

In what follows, we approach the local learning problem explicitly from the point of view of the covariance function, which is the fundamental building block of the Gaussian Stochastic Process. Specifically, we show that a compactly supported covariance function defined over spatially localized clusters naturally leads to a data parallel scheme for modeling large-scale data. It is shown that using this new covariance function, the maximum likelihood estimation (MLE) problem can be decomposed into smaller decoupled subproblems. This leads to a significant speedup in the training phase of GP models. Further, the present approach also leads to sparse¹ yet effective predictive models which consistently outperform Bayesian Committee machines (BCM) – a state-of-the-art approach for model aggregation. Since equally sized or balanced partitioning of data is crucial to the efficiency of such data parallel approaches, we next develop a novel clustering scheme GeoClust, which outputs clusters of (nearly) equal sizes. Finally although GP models serve as a good testbed for testing our algorithms, we stress that our data parallel local learning strategy may also be readily applied to other kernel machines such as SVMs.

Detailed experimental simulations presented later in the text support our intuitions about the algorithms presented and show conclusively that our data parallel approach scales favorably with data size both in sequential mode where the models are trained sequentially and in parallel where they maximally leverage the presence of multiple processors. The parallel scalability studies have been conducted using NetSolve [21], a computational platform which facilitates grid-based heterogeneous computing in a transparent and efficient manner. The Southampton Computational Grid was utilized in part to conduct these studies. It is generally recognized that data mining in the future would shift to such grid based computing platforms and architectures[22].

The rest of the paper is organized as follows: In section 2, the basic GP regression formulation is discussed briefly along with the associated computational hurdles. Section 3 present the essential ingredients of the proposed data parallel local learning strategy. Section 4 elaborates on load balancing requirements arising out of the data parallel paradigm and proposes a new clustering algorithm tailored to satisfy such requirements. Section 5 discusses aggregating predictions from multiple models and shows how it can be done efficiently in the present case without resorting to a model aggregation strategy. Section 6 presents a discussion on the training

¹The term *sparse* is used here loosely to denote models which only use a subset of the training dataset to make predictions.

and runtime computational aspects of the schemes presented. This is followed by detailed experimental investigations in section 7, where the effectiveness of the our approach is demonstrated. The paper concludes with section 8, where the contributions of the current work is summarized and related future work discussed.

2 Gaussian Process Regression

Let $D = \{\mathbf{x}_i, t_i\}, i = 1 \dots n$ represent a collection of observational data, where $\mathbf{x}_i \in \mathbb{R}^p$ denotes the input vector and $t_i \in \mathbb{R}$ is the corresponding target value. The standard starting point for a Bayesian regression model assumes the presence of an unknown true modeling function $y(\mathbf{x})$ and an additive noise term ν to account for anomalies in the observed data. Thus :

$$t(\mathbf{x}) = y(\mathbf{x}) + \nu \quad (1)$$

The standard analysis requires the specification of prior probabilities on the modeling function and the noise model. From a stochastic process viewpoint, the collection $\mathbf{t} = \{t_1, t_2, \dots, t_n\}$ is called a Gaussian process if every subset of \mathbf{t} has a joint Gaussian distribution. More specifically,

$$P(\mathbf{t}|\mathbf{C}, \{x_n\}) = \frac{1}{Z} \exp\left(-\frac{1}{2}(\mathbf{t} - \boldsymbol{\mu})^T \mathbf{C}^{-1}(\mathbf{t} - \boldsymbol{\mu})\right) \quad (2)$$

where \mathbf{C} is a covariance matrix parameterized in terms of hyperparameters $\boldsymbol{\theta}$, i.e., $\mathbf{C}_{ij} = k(x_i, x_j; \boldsymbol{\theta})$ and $\boldsymbol{\mu}$ is the process mean. The Gaussian process is characterized by this covariance structure since it incorporates prior beliefs both about the true underlying function as well as the noise model. A detailed exposition of the criteria involved in selecting an appropriate covariance function can be found in the literature; see, for example, [1, 4]. In the present research, we use the exponential covariance model

$$k(x_i, x_j) = e^{-(\mathbf{x}_i - \mathbf{x}_j)^T \Theta (\mathbf{x}_i - \mathbf{x}_j)} + \theta_{p+1}, \quad (3)$$

where $\Theta = \text{diag}\{\theta_1, \theta_2, \dots, \theta_p\} \in \mathbb{R}^{p \times p}$ is a diagonal matrix of undetermined hyperparameters, and $\theta_{p+1} \in \mathbb{R}$ is an additional hyperparameter arising from the assumption that noise is output dependent. In practice, the undetermined hyperparameters are tuned to the data using the evidence maximization framework. The computational issues involved in MLE are discussed in the forthcoming subsection.

Once the hyperparameters have been estimated from the data, predictions can be readily made for a new testing point. To illustrate this, assume that \mathbf{t}_n represents the set of n targets, \mathbf{C}_n the corresponding covariance matrix and that the process to be modeled has zero mean, i.e., $\boldsymbol{\mu} = 0$. Given a new point \mathbf{x}_{n+1} , it can be shown that the prediction t_{n+1} has a conditional probability distribution given by :

$$P(t_{n+1}|D, C, \mathbf{x}_{n+1}) = \frac{1}{Z} \exp\left(-\frac{(t_{n+1} - \hat{t}_{n+1})^2}{2\hat{\sigma}^2}\right) \quad (4)$$

where,

$$\hat{t}_{n+1} = \mathbf{k}_{n+1}^T(\mathbf{x})\mathbf{C}_n^{-1}\mathbf{t}_n \quad (5)$$

$$\sigma^2 = \mathbf{k}(\mathbf{x}_{n+1}, \mathbf{x}_{n+1}; \boldsymbol{\theta}) - \mathbf{k}_{n+1}^T(\mathbf{x})\mathbf{C}_n^{-1}\mathbf{k}_{n+1} \quad (6)$$

where, \hat{t}_{n+1} and σ^2 is the prediction for the posterior mean and the variance, respectively, and $\mathbf{k}_{n+1} = \{k(x_{n+1}, x_1), k(x_{n+1}, x_2), \dots, k(x_{n+1}, x_n)\} \in \mathbb{R}^n$.

2.1 Determining the Optimal Hyperparameters

From a computational perspective, the search for an optimal GP regressor under the evidence maximization framework [4] involves solving the following nonlinear maximum likelihood estimation (MLE) problem to determine the most probable hyperparameters $\boldsymbol{\theta}_{MP}$ for the given data.

$$\text{Maximize}_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = -\frac{1}{2} \log \det \mathbf{C}_n - \frac{1}{2} \mathbf{t}_n^T \mathbf{C}_n^{-1} \mathbf{t}_n - \frac{n}{2} \log 2\pi \quad (7)$$

where $L(\boldsymbol{\theta})$ denotes the log likelihood for a Gaussian process.

Since computing $L(\boldsymbol{\theta})$ and its gradient generally involves computing and inverting a dense $n \times n$ covariance matrix (requiring $\mathcal{O}(n^3)$ resources) at each iteration, training the GP model can be prohibitively expensive even for moderately sized data (e.g., say a few thousand data points). This is amply demonstrated in Figure 1 where the training time is shown for a typical 10 variable dataset. Unfortunately, other schemes such as Markov chain Monte Carlo sampling for approximating the predictive distribution [3, 2, 5] are also plagued by similar problems involving escalating training costs with data size. In any case, it has been shown that evidence maximization is faster and performs better than its counterparts on large datasets[3].

Since the main obstacle to efficient maximization of (7) is the presence of the dense covariance matrix, it would seem natural to choose a covariance function which leads to a sparse or banded covariance matrix. Similar concerns have been raised and addressed elsewhere in the literature [17] in the context of radial basis function networks. In particular, the use of compactly supported radial basis functions has been shown to bring about appreciable speedups in the training as well as prediction phases. In what follows, we pursue a similar line of argument but for the Gaussian process model. It is easily shown that such a line of approach naturally leads to a data parallel local learning scheme for GP regression.

3 Data Parallel Local Learning

Since a Gaussian stochastic process is completely specified by its covariance function, training a GP involves considering a parameterized covariance function and determining its hyperparameters $\boldsymbol{\theta}$ such that the log likelihood $L(\boldsymbol{\theta})$ of the data is maximized (see Eqn. 7). In this section, we propose a compactly supported covariance function to facilitate data parallel local learning.

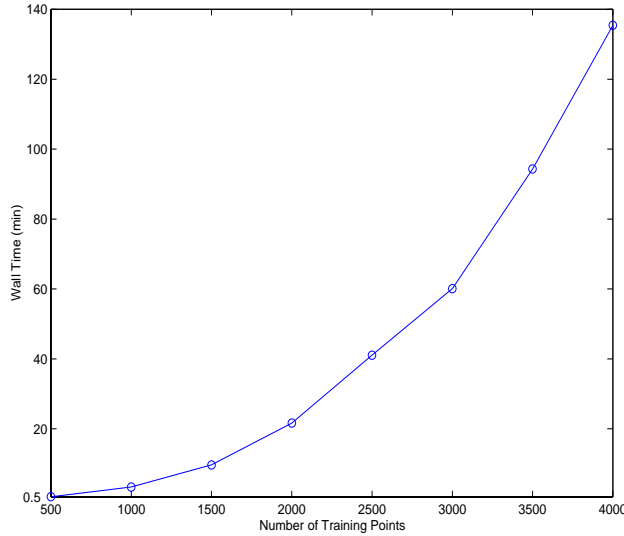


Figure 1. Computational cost of training the GP regression model as a function of data size

To illustrate our approach, let us assume the existence of m disjoint and spatially localized subsets of the training data say $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_m$. Given such a partitioning, we propose the use of the following covariance model :

$$\tilde{k}(\mathbf{x}_i, \mathbf{x}_j; c(\mathbf{x}_i), c(\mathbf{x}_j), \boldsymbol{\theta}) = \delta_{c(\mathbf{x}_i)c(\mathbf{x}_j)} k(\mathbf{x}_i, \mathbf{x}_j; \boldsymbol{\theta}) \quad (8)$$

where $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^p$ are input vectors, δ_{ij} is the Kronecker delta function, $\boldsymbol{\theta}$ is a set of hyperparameters and $c : c(\mathbf{x}) \mapsto \{1, 2, \dots, m\}$ is the assignment function which maps the input point \mathbf{x} to one of m available clusters.² Then the covariance function in (8) can be immediately written for cluster i as :

$$\begin{aligned} \tilde{k}(\mathbf{x}_1, \mathbf{x}_2; c(\cdot), \boldsymbol{\theta}) &= k(\mathbf{x}_1, \mathbf{x}_2; \boldsymbol{\theta}_i), & c(\mathbf{x}_1) = c(\mathbf{x}_2) = i \\ &= 0 & \text{otherwise} \end{aligned} \quad (9)$$

where $\boldsymbol{\theta}_i$ denote the set of hyperparameters for the local model trained on the i th cluster. This is readily identified as an example of a compactly supported covariance function popularly used in the radial basis function literature [17]. Consider the case when $m = 2$, i.e, when the data has been partitioned into 2 disjoint spatially localized subsets. Then using (9), the covariance matrix can be written as follows:

$$\mathbf{K} = \begin{pmatrix} \mathbf{K}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{K}_{22} \end{pmatrix} \quad (10)$$

²For example, $c(\mathbf{x})$ could be the nearest neighbor operator, where the neighbors considered are the cluster centers.

where $\mathbf{K}_{ii} \in \mathbb{R}^{n_i \times n_i}$ contains correlation terms explicitly from the i th cluster which consists of n_i points. Since in this case the determinant of the covariance matrix \mathbf{K} can be written as the product of determinants of the blocks \mathbf{K}_{11} and \mathbf{K}_{22} , the log likelihood can be split into individual log likelihoods for the two partitions, i.e.,

$$L(\boldsymbol{\theta}) = L(\boldsymbol{\theta}_1) + L(\boldsymbol{\theta}_2) \quad (11)$$

It is important to note at this point that while the conventional GP regression model uses a single set of hyperparameters to model the full data, the data parallel version uses different sets of hyperparameters to model the covariance structures of different regions of the data. Hence, maximizing the overall log likelihood for the data is equivalent to maximizing the log likelihood for each partition, since the individual objective functions depend on separate sets of hyperparameters. It may seem at first sight that the proposed compactly supported covariance function would result in a less accurate oversimplified global model. However, the results in the later sections do not seem to bear out this observation. Indeed, it can be argued that given enough flexibility in the choice of the local covariance models, the performance degradation can be somewhat circumvented.

From the preceding discussion, it is clear that the use of a compactly supported covariance function naturally leads to a data parallel local learning approach to GP regression and hence provides a means to handle large datasets. Having made this connection, we are now confronted with a problem: that of partitioning or clustering the data into subsets. From a data parallel point of view the subset sizes chosen need to be (nearly) equal.³ What we really need is an algorithm which creates clusters which are localized in the data space. Local models can then be built on each of these clusters. Finally the ensemble of local models can be used in place of the global model. As argued in Vapnik [23], for unevenly distributed data, such local learning may lead to effective capacity control which can significantly improve the performance for some cases.

Even though, we discussed the data parallel approach in the context of GP regression, our approach can also be applied to other kernel-based machine learning algorithms such as SVMs. The basic approach is elucidated as a pseudo code in Algorithm 1, where $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_m$ denote the m clusters or partitions of data and M_1, M_2, \dots, M_m denote the corresponding local prediction models. As is clear from Algorithm 1, a complete description of the data parallel approach requires the specification of three functions viz., Partition(), CreateLocalModel() and Aggregate().

³This is primarily because of efficiency reasons and also to reduce the waiting times when operating in a multiprocessor scenario.

Algorithm 1: The basic data parallel approach

Input	: Training Data \mathcal{D}
Input	: Number of partitions desired m
Begin Modeling	
Step 1	: $[\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_m] := \mathbf{Partition}(\mathcal{D}, m)$
Step 2	: For $i = 1, \dots, m$
	$M_i := \mathbf{CreateLocalModel}(\mathcal{C}_i)$
	end For
	$\mathbf{M} = \bigcup_{i=1}^m M_i$
End Modeling	
Prediction	: Given a new point \mathbf{x}
	$y(\mathbf{x}) = \mathbf{Aggregate}(\mathbf{M}, \mathbf{x})$

4 Data Partitioning

The preceding discussion clearly indicates that to facilitate local model building under a data parallel paradigm, one needs to generate spatially localized clusters which contain (nearly) equal number of points. Although clustering techniques abound in the literature, (EM and K-Means being two of the more popular ones), not many of them seem to offer a handle on the number of points to be held in each cluster.

One way to generate balanced spatially localized clusters would be to solve the nonlinear programming problem

$$\mathbf{Minimize}_{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m} : \sum_{i=1}^m (W_i - W_{des})^2, \quad (12)$$

where \mathbf{c}_i and \mathbf{W}_i denote the center and number of points in the i th cluster, respectively, and \mathbf{W}_{des} is the desired number of points per cluster.

Note that since W_i can only be expressed in terms of indicator functions, the resulting objective function is non-differentiable. As a result, one has to resort to less efficient non-gradient search techniques. In this section, we present a geometry motivated clustering algorithm GeoClust which attempts to solve (12) greedily.

In the GeoClust technique, we use a greedy strategy to incrementally update the cluster centers. The pseudo code for the new algorithm is presented in Algorithm 2. Consider the case of 2-d data and three clusters $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$ with cluster centers $\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3 \in \mathbb{R}^2$ respectively. GeoClust is an iterative scheme which updates cluster centers until each cluster has (nearly) equal number of instances. During the course of clustering, each cluster center \mathbf{c}_i moves towards every other cluster \mathbf{c}_j according to the fraction $(\frac{W_j}{W_i} - 1)$, where W_i, W_j denote the strengths (number of points) of the i th and the j th clusters, respectively. Thus at any iteration, cluster center \mathbf{c}_1 moves towards cluster center \mathbf{c}_2 by the fraction $(\frac{W_2}{W_1} - 1)$ along the direction of the vector $(\mathbf{c}_2 - \mathbf{c}_1)$ and by a fraction $(\frac{W_3}{W_1} - 1)$ along the direction $(\mathbf{c}_3 - \mathbf{c}_1)$ towards

cluster center \mathbf{c}_3 . At the end of each iteration, the cluster strengths W_1, W_2, W_3 are reassessed. The process continues till the increments become zero or very small. The iterative process is assisted by a relaxation/learning factor α . Typically, α is set to a low value. In our experiments, we have found 10^{-2} to be reasonable choice when the data is normalized. Note that as the cluster strengths become close, the relative movement of the centers towards each other slows down and finally ceases. Hence, the algorithm converges when the clusters have equal strengths.⁴ Figure 2 illustrates how GeoClust partitions a toy 2-D dataset into 2, 3 and 4 clusters respectively.

Algorithm 2: GeoClust

Inputs : Data D
: Number of partitions desired m
: Maximum number of iterations $Maxiter$
: learning parameter α

Initialize m random centers, $\mathbf{c}_i^0, i = 1, 2, \dots, m$

For $t = 1 : Maxiter$

- Assign each data point to the cluster nearest to it
- Compute number of points in each cluster: W_1, W_2, \dots, W_m
- For each cluster C_i , Update the cluster center as follows:
 - Compute $\delta \mathbf{v}_i = \sum_{j=1, j \neq i}^m (\frac{W_j}{W_i} - 1)(\mathbf{c}_j^{t-1} - \mathbf{c}_i^{t-1})$
 - Update center $\mathbf{c}_i^t = \mathbf{c}_i^{t-1} + \alpha \delta \mathbf{v}_i$
- End

End

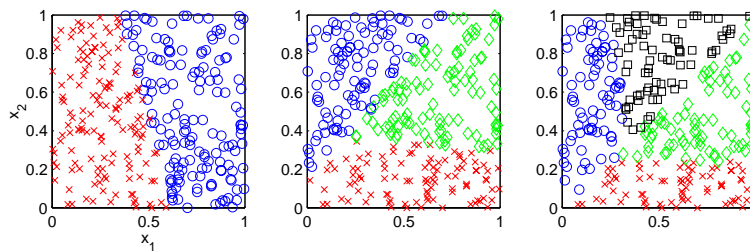


Figure 2. The partitions obtained by GeoClust for a toy dataset when the desired number of partitions are increased from 2 to 4 (left to right)

Remark : Ideally one would like to solve a graph partitioning problem where m disjoint partitions of a set of N nodes (points) are sought so as to minimize the total weight of the edges connecting nodes from different sets. In the present case, the weights of the edges are given by the correlation between two points. Unfortunately,

⁴When the number of instances is not exactly divisible by the number of clusters, the cluster strengths settle to nearly equal values.

the original problem is NP-hard and heuristics are often applied to obtain good quality partitions [20]. Interestingly, GeoClust actually solves an easier problem, that of unconstrained partitioning of a set of points into equally sized disjoint yet geographically localized clusters. This is obviously much cheaper to perform and results in the later sections show that local models built on GeoClust partitions fare well in terms of generalization performance compared to the conventional GP model.

5 Prediction with multiple models

Since the current computing paradigm is essentially data parallel, we generate multiple models for explaining the given data. Given a new input point, the task of predicting its output therefore must take into account the outcomes from the multiple models in some reasonable way. Intuitively, such an aggregation strategy should also depend on the clustering technique used.

Recently, Bayesian Committee Machines (BCM)[16] have been proposed to combine multiple models trained on different datasets under a Bayesian framework. Specifically, BCMs use the posterior variance predicted by GP models as weighting factors while aggregating model predictions. In this section, we show that for a compactly supported covariance structure, the aggregate prediction strategy turns out to be much simpler and inexpensive as compared to BCMs.

Consider GP Regression using the compactly supported covariance model proposed in Eqn. (8). Given a new point \mathbf{x}_{n+1} , the mean of the conditional distribution of the prediction t_{n+1} follows from (5) as :

$$\hat{t}_{n+1} = \mathbf{k}_{n+1}^T(\mathbf{x}) \mathbf{C}_n^{-1} \mathbf{t}_n \quad (13)$$

where \mathbf{k}_{n+1} is now given by :

$$\mathbf{k}_{n+1} = \left[\delta_{c(\mathbf{x}_{n+1}), c(\mathbf{x}_1)} k(\mathbf{x}_{n+1}, \mathbf{x}_1; \boldsymbol{\theta}), \dots, \delta_{c(\mathbf{x}_{n+1}), c(\mathbf{x}_n)} k(\mathbf{x}_{n+1}, \mathbf{x}_n; \boldsymbol{\theta}) \right]$$

It is easy to see that this prediction strategy really boils down to a hard assignment strategy viz. given a new point, determine the cluster center nearest to it and then use the corresponding local GP model for prediction. Mathematically, Equation (13) is now equivalent to the following two statements:

$$\text{Determine Cluster : } i = c(\mathbf{x}) ; \quad c(\cdot) \in \{1, \dots, m\} \quad (14)$$

$$\text{Predict : } y(\mathbf{x}) = k_{i, \boldsymbol{\theta}_i}^T \mathbf{C}_{i, \boldsymbol{\theta}_i}^{-1} \mathbf{t}_i \quad (15)$$

where the subscript i and $\boldsymbol{\theta}_i$ indicate that the data points from the i th cluster and the corresponding set of hyperparameters are to be used respectively, and $c(\cdot)$ is the cluster assignment function as discussed in Section 3.

6 Computational Aspects

As discussed earlier, the key computational bottlenecks involved in training the conventional GP for large data are related to storing the dense covariance matrix

($\mathcal{O}(n^2)$ memory) and computing its inverse ($\mathcal{O}(n^3)$ flops). The computational advantages of the present local learning strategy over the conventional GP regression approach are significant and are straightforward to evaluate. Specifically, for n data points and m clusters, a sequential data parallel version of our algorithm requires training time of $\mathcal{O}(m\frac{n^3}{m^3})$ only as compared to $\mathcal{O}(n^3)$ for the global GP regression model. Further, the attendant memory requirements are reduced from $\mathcal{O}(n^2)$ for the conventional case to $\mathcal{O}(\frac{n^2}{m^2})$. The parallel version of the algorithm offers even more advantages. Typically, in such a scenario, one has access to at least as many processors as the number of clusters specified.⁵ The overall training time now reduces to $\mathcal{O}(\frac{n^3}{m^3})$ and the memory requirements fall to $\mathcal{O}(\frac{n^2}{m^2})$ per processor involved.

The prediction model obtained using the compactly supported covariance function is extremely cheap in terms of both runtime computations and memory required as compared to BCM. Since a BCM prediction involves the additional cost of computing the variance of prediction for each model using equation (6), it effectively means explicit computation and storage of multiple matrix inverses. Hence runtime computation and memory requirements both scale as $\mathcal{O}(m\frac{n^2}{m^2})$. In contrast, the runtime complexity of the predictive model arising from the data parallel local learning approach scales only as $\mathcal{O}(mp + \frac{n}{m})$ flops, where p is the dimensionality of the input data. Additionally, as demonstrated in the next section, experimental investigations suggest that the current strategy can indeed outperform BCM consistently in terms of accuracy of prediction.

It is clear from the preceding discussion that the efficiency of the algorithm increases with increasing number of clusters m . Further, it has been shown elsewhere that the use of compactly supported kernels produces more stable solutions [18]. However, it is important to mention that increasing the number of clusters beyond a certain limit actually leads to a deterioration in the generalization capability of the resulting model [18]. This results from an ever reducing number of points available for modeling each cluster.⁶

7 Experimental Investigation

The aim of the experimental studies presented in this section is two-fold: (i) we seek a direct comparison of the performance of the proposed algorithm with respect to the state-of-the-art BCM algorithm suggested recently [16] for handling large data using data parallel GP models, and (ii) we carry out a full scale timing study of our technique on a large data set to see how it scales with the size of the training data. Simultaneously, we also compute training times for the full scale GP model trained on the whole data.

We shall use the following acronyms henceforth to refer to the competing schemes: *LL-GP* for the scheme which uses the proposed local learning approach and *RP-BCM* for the scheme which uses a Random Partitioning of data and a BCM

⁵Conversely, it is reasonable to assume that the number of clusters are prespecified by the user according to the number of processors available.

⁶From the point of view of compactly supported covariance functions, poor generalization arises when the support radius is reduced below a certain threshold.

style inverse variance weighting for prediction in the spirit of [16].

The algorithms presented in this paper were implemented as a library of FORTRAN 77 routines which make extensive use of a machine optimized BLAS library. The numerical experiments presented in this section have been carried out on a variety of machines. These range from single to multiple R10000 processors (SGI Power Challenge) on the one hand to a 30 membered cluster of 700 MHz Pentium III processors running Linux. The main memory was typically of the order of 4.6 GB on the SGI machine while it was of the order of 256 MB on each Linux machine.

A first set of experiments was carried out on a synthetic dataset generated according to [19]. Subsequently, a detailed comparison study was performed on the Boston Housing data set available from the UCI machine learning repository. For all experimental investigations, the exponential covariance function model presented earlier in Section 2 was used in GP regression.

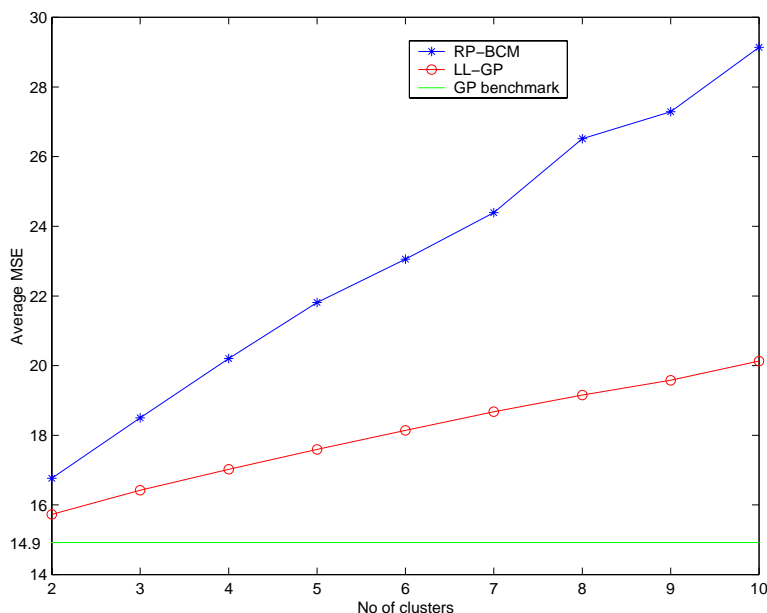


Figure 3. *MSE performance for 100 realizations of a synthetic function with 5 variables generated using F^* with varying cluster sizes.*

The synthetic dataset is generated using a linear combination of 20 basis functions [19], i.e. $F^*(x) = \sum_{l=1}^{20} a_l \mathbf{g}_l(x) + \mathcal{N}(0, \sigma^2)$, where $x \in \mathbb{R}^p$ and the scalar a_l s are drawn from a uniform distribution $[0, 15]$. Here $\mathbf{g}_l(x) = \exp\{-(x - \mu_l)^T \Sigma^{-1} (x - \mu_l)\}$, where $\mu_l \in \mathbb{R}^p$ is drawn from a uniform distribution $[0, 1]^p$ and $\Sigma \in \mathbb{R}^{p \times p}$ is a randomly generated SPD matrix.

Figures 3 and 4 compare the effect of varying cluster sizes on the average MSE behavior of LL-GP and RP-BCM. These are benchmarked against the superior mean MSE obtained using the full data to train the GP (the line at the bottom). Each run uses a training size of 1000 instances and the MSE is computed over a separate

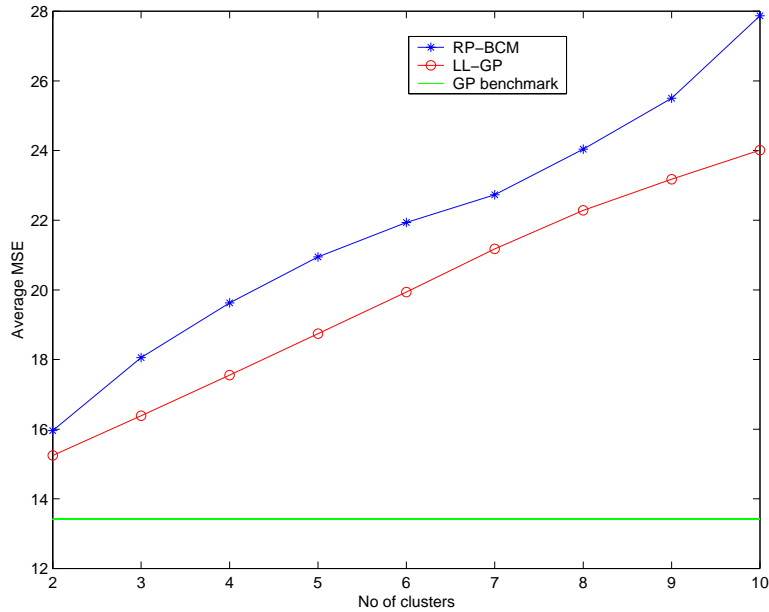


Figure 4. MSE performance for 100 realizations of a synthetic function with 10 variables generated using F^* with varying cluster sizes.

Methods	Measure	No. of clusters					
		1	2	4	6	8	10
LL-GP	MSE	8.04	8.98	9.33	10.38	10.67	10.72
	Std	6.2	8.55	7.20	8.12	8.41	7.5
	$-\log(L)(10^3)$	1.25	1.26	1.25	1.26	1.26	1.26
RP-BCM	MSE	-	11.03	15.17	17.13	20.33	40.67
	Std	-	9.5	12.73	12.84	21.56	189.22
	$-\log(L)(10^3)$	-	1.30	1.34	1.36	1.39	1.40

Table 1. Comparison of the statistics of MSE and negative log likelihood($-\log(L)$) values for the Boston dataset

test set of 2000 instances. From the figures, it is clear that the present local learning strategy is superior in terms of average MSE performance when compared to the RP-BCM. Also, this experiment clearly exhibits that the mean accuracy of prediction decreases with increasing number of clusters for both the algorithms although the training times are cut down significantly with increasing cluster sizes. This reflects the familiar speed-accuracy tradeoff, which limits the number of clusters one may use in practice and hence upper bounds the speedups one can obtain.

Table 1 summarizes the statistics for the Boston data set. The measures shown are averaged over 100 runs each of which includes training over 481 instances and testing over the remaining 25. Note that for a single cluster, the LL-GP scheme

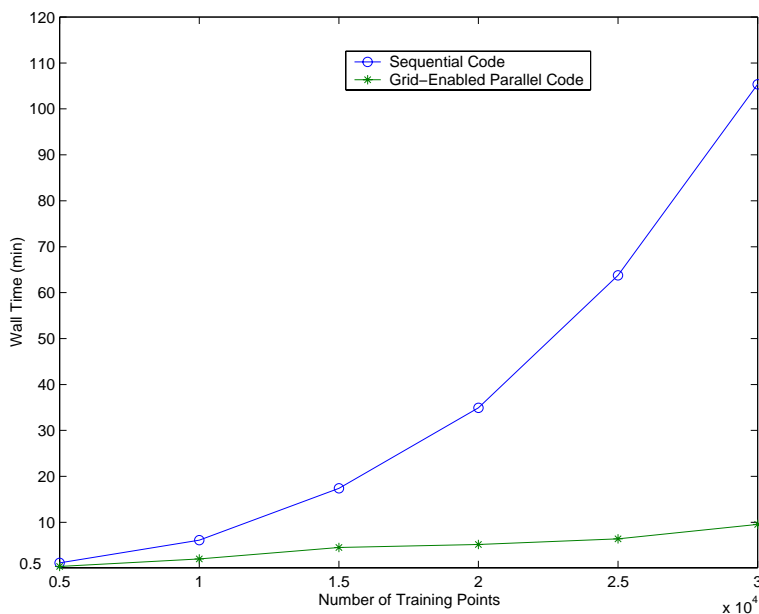


Figure 5. A comparison of computational speedups for the parallel and the sequential versions of the data parallel local learning approach.

reduces to the (superior but expensive) full scale GP and hence is treated as the target benchmark. Once again, it is observed that with the increase in the number of clusters, the mean performance for the LL-GP worsens more gracefully as compared to RP-BCM. Note that in this case, the performance of the later deteriorates rapidly from an average mse of 11.03 for 2 clusters all the way down to 40.67 for 10 clusters. In contrast, the LL-GP scheme deteriorates gracefully from 8.04 to 10.72 average mse with increase in the number of clusters. The extremely high value of standard deviation for RP-BCM for the case of 10 clusters is also worth noting. This clearly indicates that the scheme becomes unstable at this point. Finally, the negative log likelihood(error function) values obtained using LL-GP match closely with those of the GP model trained using the full data.

A final extensive set of experiments was carried out to study the scalability of the proposed algorithm when the size of the dataset is increased. A synthetic data set of 30,000 instances in 10 dimensions drawn from a realization of F^* was used for this scalability study. Figure 5 shows the timing comparison between a sequential and parallel grid-enabled version of the data parallel approach proposed in this work when the number of data points are increased. For each instance, we applied GeoClust in sequential mode to divide the data into 30 disjoint spatially localized clusters. The MLE problems for each cluster were then conducted in parallel on the Southampton grid computing facility.

It can be seen from Figure 5 that the speedups obtained using the grid-enabled version improve as the data size increases. As expected, the computational cost of

the parallel code increases approximately in a linear fashion when the number of training points are increased. For the case of 30,000 points, the grid-enabled version is nearly 12 times faster than a sequential version of our approach. Interestingly, the conventional GP rapidly became infeasible beyond a training set size of 4000 points, which itself took 140 minutes of training time. Hence, the proposed approach in both its sequential and grid-enabled form is clearly much faster than the conventional GP modeling and is an enabling approach for handling problems with large data.

It is also worth noting that meta-learning approaches which use random partitions of the data can be deployed on grid-based computational infrastructures. However, it is often the case that the model aggregation stage cannot be fully parallelized. As a consequence, the resulting speedups may not be very significant for some cases. For example, the parallelized version of the meta-learning strategy proposed recently for SVM mixtures [12] (running on 50 compute nodes) is only 3.3 times faster than the sequential code for a problem with 100,000 training points.

8 Concluding Remarks

Regression techniques based on Gaussian processes have recently been applied with great success to modeling observational data. However, the impressive generalization behavior of GPs is offset by the overwhelming training cost incurred even for modest datasets. This paper proposes a novel data parallel local learning paradigm as an enabling approach for applying GP models to large-scale regression tasks. This is achieved using a combination of a compactly supported covariance function and a novel data partitioning scheme which respects load balancing considerations as well as space localization. A simple yet effective prediction strategy follows naturally from the choices effected, which is shown to consistently outperform the state-of-the-art BCM framework in terms of predictive performance. Experimental studies provide empirical evidence for the attractive scaling behavior of the proposed methodology which indicates its potential as a data mining tool.

A number of interesting lines of research are worth pursuing. From a theoretical point of view, it is of interest to examine the relationship between the number of clusters to be chosen and the resulting generalization error of the predictive model. This could result in a more principled data dependent basis for the *a priori* selection of the number of clusters to be formed to achieve reasonable generalization, while still obtaining good training and runtime speedups. While Geoclust efficiently solves an unconstrained graph partitioning problem, it remains to be seen whether imposing edge constraints improves the generalization performance any further. Lastly, it is to be noted that our data parallel paradigm can also be applied to speed-up other kernel-based machine learning algorithms.

Bibliography

- [1] Gibbs, M. N., Bayesian Gaussian Processes for Regression and Classification, *PhD thesis*, University of Cambridge, 1997.
- [2] Williams, C.K.I., Rasmussen, C.E., Gaussian Processes for Regression, In *Advances in Neural Information Processing Systems, 8*, Touretzky, D.S., Mozer, M.C, Hasselmo, M.E.(Eds.), MIT Press, 1996
- [3] Rasmussen, C.E., *Evaluation of Gaussian Processes and Other Methods for Non-linear Regression*, PhD. Dissertation, University of Toronto, 1996
- [4] Mackay, D.J.C., *Bayesian Interpolation*, *Neural Computation*, Vol. 4(3), pp 415-447, 1992
- [5] Neal, R.M., Monte Carlo Implementation of Gaussian Process Models for Bayesian Regression and Classification, Technical Report, CRG-TR-97-2, Department of Computer Science, University of Toronto
- [6] Csató, L., Opper, M., Sparse Representation for Gaussian Process Models, *Advances in Neural Information Processing Systems, 13*, 2001
- [7] Yoshioka, T., Ishii, S., Fast Gaussian process regression using representative data, *International Joint Conference on Neural Networks*, 2001.
- [8] Smola, A. J., Bartlett, P. L., Sparse greedy Gaussian process regression, *Advances in Neural Information Processing Systems*, 13, 2001
- [9] Nair, P.B., Choudhury, A., Keane, A.J., Some Greedy Algorithms for Sparse Nonlinear Regression, *The Eighteenth International Conference on Machine Learning*, Morgan Kaufmann, 2001
- [10] Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E., Adaptive Mixture of Local Experts, *Neural Computation*, Vol. 3(1), pp 79-87, 1991
- [11] Collobert, R., Bengio, S., SVM-Torch: Support Vector Machines for Large-scale Regression Problems, *Journal of Machine Learning Research*, Vol. 1, pp. 143-160, 2001
- [12] Collobert, R., Bengio, S., Bengio, Y., A Parallel Mixture of SVMs for Very Large Scale Problems *Neural computation*, To Appear, 2002

- [13] Joachims, T., Making Large Scale Support Vector Machine Learning Possible, In *Advances in Kernel Methods*, Schölkopf, B., Burges, C., Smola, A.,(Eds.), MIT Press, 1999
- [14] Platt, J., Fast Training of Support Vector Machines using Sequential Minimal Optimization, In *Advances in Kernel Methods*, Schölkopf, B., Burges, C., Smola, A.,(Eds.), MIT Press, 1999
- [15] Ridda, A., Labbi, A., Pellegrini, C., Local Experts Combination Through Density Estimation, International Workshop on AI and Statistics, Uncertainty '99: Morgan Kaufmann, 1999
- [16] Tresp, V., A Bayesian Committee Machine, *Neural Computation*, Vol. 12, pp 2719-2741, 2000
- [17] Schaback, R., Creating surfaces from scattered data using radial basis functions, in *Mathematical Methods for Curves and Surfaces*, M. Daehlen, T. Lyche and L. Schumaker (eds.), pp 477-496 : Vanderbilt University Press, 1995
- [18] Schaback, R., Error estimates and condition numbers for radial basis function interpolation, *Advances in Computational Mathematics*, Vol. 3, pp. 251-264, 1995
- [19] Friedman, J.H., Greedy Function Approximation: A Gradient Boosting Machine, *Annals of Statistics*, To Appear
- [20] Kernighan, B.W., Lin, S., An Efficient Heuristic Procedure for Partitioning Graphs *Bell Systems Technical Journal*, Vol. 49, pp. 291-307, 1970
- [21] Casanova, H., Dongarra, J., NetSolve: A Network-enabled Server for Solving Computational Science Problems *The International Journal of Supercomputer Applications and High Performance Computing*, Vol. 11(3) pp. 212-223, 1997
- [22] Foster, I., Kesselman, C., (Eds), *The Grid: Blueprint for a New Computing Infrastructure*: Morgan Kaufmann, 1999.
- [23] Bottou L. and Vapnik V., Local learning algorithms, *Neural Computation*, Vol. 4(6), pp. 888-900, 1992.
- [24] Chan, P.K., Stolfo, S.J., Learning Arbiter and Combiner Trees from Partitioned Data for Scaling Machine Learning, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pp. 39-44 : AAAI Press, 1995