JMLR: Workshop and Conference Proceedings 13: 95-110
2nd Asian Conference on Machine Learning (ACML2010), Tokyo, Japan, Nov. 8–10, 2010.

Hierarchical Gaussian Process Regression

Sunho Park TITAN@POSTEH.AC.KR
Department of Computer Science
Pohang University of Science and Technology, Korea

Seungjin Choi SEUNGJIN@POSTECH.AC.KR
Department of Computer Science
Division of IT Convergence Engineering
Pohang University of Science and Technology, Korea

Editor: Masashi Sugiyama and Qiang Yang

Abstract

We address an approximation method for Gaussian process (GP) regression, where we approximate covariance by a block matrix such that diagonal blocks are calculated exactly while off-diagonal blocks are approximated. Partitioning input data points, we present a two-layer hierarchical model for GP regression, where prototypes of clusters in the upper layer are involved for coarse modeling by a GP and data points in each cluster in the lower layer are involved for fine modeling by an individual GP whose prior mean is given by the corresponding prototype and covariance is parameterized by data points in the partition. In this hierarchical model, integrating out latent variables in the upper layer leads to a block covariance matrix, where diagonal blocks contain similarities between data points in the same partition and off-diagonal blocks consist of approximate similarities calculated using prototypes. This particular structure of the covariance matrix divides the full GP into a pieces of manageable sub-problems whose complexity scales with the number of data points in a partition. In addition, our hierarchical GP regression (HGPR) is also useful for cases where partitions of data reveal different characteristics. Experiments on several benchmark datasets confirm the useful behavior of our method.

Keywords: Gaussian process regression, Kernel methods, Sparse approximations

1. Introduction

Gaussian process (GP) is a powerful non-parametric Bayesian method for supervised learning. It provides probabilistic predictions and allows standard Bayesian approaches to model selection. However, direct application of GP to regression problems is limited due to its unfavorable scaling: $\mathcal{O}(N^3)$ in time and $\mathcal{O}(N^2)$ in space, where N is the number of training samples. To overcome this limitation, various approximation methods have been proposed in the literature (Csató and Opper, 2002; Tresp, 2000; Williams and Seeger, 2001; Seeger et al., 2003; Smola and Bartlett, 2001; Lawrence et al., 2003; Snelson and Ghahramani, 2006; Walder et al., 2008; Lazaro-Gredilla and Figueiras-Vidal, 2009). Most of these methods fall within *sparse approximation* where a low-rank approximation is applied to the covariance matrix of the GP prior using a smaller subset of R ($\ll N$) inducing variables (Quiñonero-Candela and Rasmussen, 2005). In this case time complexity and memory space are reduced to $\mathcal{O}(R^2N)$ and $\mathcal{O}(RN)$, respectively.

The selection of input points corresponding to the inducing variables is a main issue in sparse approximation. Typical solution is to choose them among the training data by using specific criterions (Smola and Bartlett, 2001; Seeger et al., 2003; Lawrence et al., 2003). The restriction that the inducing inputs must be selected among the training data is relaxed in (Snelson and Ghahramani, 2006), where the inducing inputs are treated as auxiliary pseudo-inputs, and, are estimated in a joint continuous optimization with unknown hyperparameters. This idea can increase flexibility in fitting the approximation model to the data. Since tuning both pseudo-inputs and hyperparameters involves high-dimensional optimization, it can also lead to overfitting.

Instead of finding the inducing inputs, we directly approximate the covariance between the pairs of the latent variables based on the clustered structure in the input data. Partitioning input data points, the covariance evaluated at the pairs of training data can be exactly calculated if two points are in the same partition, otherwise it can be approximated by using the prototype vectors of the partitions. This idea leads us to propose a two-layer hierarchical model for GP regression. In the upper layer, prototype vectors of clusters are involved for coarse modeling by a GP. In the lower layer, the data points in each cluster are involved for fine modeling by an individual GP whose prior mean is given by the corresponding prototype and covariance is parameterized by data points in the partition. Integrating out latent variables in the upper layer leads to the block covariance matrix such that diagonal blocks are calculated exactly while off-diagonal blocks are approximated.

The block structure of the covariance matrix can divide the full GP into a pieces of manageable sub-problems whose complexity scales with the number of training inputs in a partition. This approach reduces both time complexity and memory space for training, i.e., if the number of training points in all partitions is roughly equal to $\widetilde{N}$, the time complexity and memory space are reduced to $\mathcal{O}(\widetilde{N}^2 N)$ and $\mathcal{O}(\widetilde{N}N)$, respectively. In contrast to the sparse approximation methods, our method does not need to optimize inducing points. In the case of high-dimensional input space, this fact makes our method robust to overfitting. Furthermore our method is related to the hierarchical (multilevel) regression models that have been proposed to analyze the datasets with a hierarchical or clustered structure (Raudenbush and Bryk, 2002). They can explain the variation between clusters by introducing latent variables, e.g., random intercepts or slope, that vary between clusters. Thus our method based on the hierarchical modeling can capture different characteristics among the partitions. Experimental results confirm the useful behavior of our method.

2. Gaussian Process Regression

In this section we briefly introduce GP regression. We are given a data set $\mathcal{D}$ consisting of N training input points $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^D$, and corresponding real valued targets $\boldsymbol{y} = [y_1, ..., y_N]^\top$.

GP defines a distribution over functions of the form $f : \mathcal{X} \mapsto \mathbb{R}$, which is completely described by its mean and covariance function (Rasmussen and Williams, 2006)

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), \kappa(\boldsymbol{x}, \boldsymbol{x}')), \quad \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}. \tag{1}$$

Usually the mean function $m(\cdot)$ is set to a zero function, and the covariance function $\kappa(\boldsymbol{x}, \boldsymbol{x}') \triangleq \langle f(\boldsymbol{x}), f(\boldsymbol{x}')\rangle$ is modeled as a squared exponential kernel such that

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') \quad = \quad \ell_f \exp\Big\{ -\frac{1}{2\rho}(\boldsymbol{x}-\boldsymbol{x}')^\top \boldsymbol{L}(\boldsymbol{x}-\boldsymbol{x}')\Big\}, \tag{2}$$

where $\boldsymbol{L} = \mathrm{diag}(\boldsymbol{\ell})$, $[\boldsymbol{\ell}]_i$ [1] is a hyper-parameter to determine a relevance of the ith input dimension.

In standard GP regression, the noisy observation is modeled as the noiseless latent function added independent noise, i.e., $y = f(\boldsymbol{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. When we evaluate the latent function at $\boldsymbol{X}$, we have a set of latent function values, $\boldsymbol{f} = [f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)]^\top$, which follows a multivariate Gaussian distribution

$$p(\boldsymbol{f}|\boldsymbol{X}, \theta) = \mathcal{N}(\boldsymbol{f}|0, \boldsymbol{K}), \tag{3}$$

where $[\boldsymbol{K}]_{i,j} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$. With the observational model, the likelihood is $p(\boldsymbol{y}|\boldsymbol{f}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}, \sigma^2\boldsymbol{I})$, where $\boldsymbol{I}$ is a $N \times N$ identity matrix.

The hyperparameters related to the covariance function and noise variance, $\theta = \{\boldsymbol{\ell}, \rho, \ell_f, \sigma^2\}$, can be estimated by maximizing the marginal-likelihood:

$$p(\boldsymbol{y}|\boldsymbol{X}, \theta) = \mathcal{N}(\boldsymbol{y}|0, \boldsymbol{\Sigma}_N), \tag{4}$$

where $\boldsymbol{\Sigma}_N \triangleq \boldsymbol{K} + \sigma^2\boldsymbol{I}$ denotes a noisy covariance matrix. Given the observations and the estimated hyperparameters $\hat{\theta}$, the predictive distribution of the target value at a test point $\boldsymbol{x}_*$ is given by

$$p(y_*|\boldsymbol{x}_*, \mathcal{D}, \hat{\theta}) = \mathcal{N}(y_*|\boldsymbol{k}^\top\boldsymbol{\Sigma}_N^{-1}\boldsymbol{y}, \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}^\top\boldsymbol{\Sigma}_N^{-1}\boldsymbol{k} + \sigma^2), \tag{5}$$

where $\boldsymbol{k}$ is a N-length vector with $[\boldsymbol{k}]_i = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_i)$.

3. Hierarchical Model for Gaussian Process Regression

This section presents the hierarchical model for GP regression (HGPR) based on the clustered structure in the input data. We assume that the input space $\mathcal{X}$ is divided into Q partitions $\mathcal{X} = \mathcal{X}_1 \cup ... \cup \mathcal{X}_Q$. It can be realized by clustering the training input data $\boldsymbol{X} = \{\{\boldsymbol{x}_i^{(1)}\}_{i=1}^{N_1}, ..., \{\boldsymbol{x}_i^{(Q)}\}_{i=1}^{N_Q}\}$, where N_j is a number of training data in the jth cluster. The partitions of input data can be provided by the information of the dataset or by using clustering methods, e.g., k-means algorithm. Let $\boldsymbol{c}_j$ be a prototype vector of the j partition, and $\boldsymbol{C} = \{\boldsymbol{c}_1, ..., \boldsymbol{c}_Q\}$ be a set of prototype vectors. When we use the k-means algorithm to partition the input data, the input space is divided into Q partitions:

$$\mathcal{X}_j = \{\boldsymbol{x} \in \mathcal{X} |\ d_j^2 < d_q^2,\ \text{for } q \neq j, q = 1, ..., Q\}, \tag{6}$$

where $d_j^2 = \|\boldsymbol{x} - \boldsymbol{c}_j\|^2$.

We model underlying latent functions for noisy observations in a hierarchical way: $g : \boldsymbol{C} \mapsto \mathbb{R}$ is a function in the upper layer (partition level), while $f_j : \mathcal{X}_j \mapsto \mathbb{R}$ is a function in

1. The bracket with subscripts denote the elements of matrices and vectors, and, colon subscript represents an entire column of the matrix.

the lower layer (data point level), defined on the jth partition. We now place a GP prior on the latent functions in such a way,

$$\begin{aligned} f_j(\boldsymbol{x}) \,|\, g &\sim \mathcal{GP}(g(\boldsymbol{c}_j), \kappa_j(\boldsymbol{x}, \boldsymbol{x}')), && (7)\\ g(\boldsymbol{c}) &\sim \mathcal{GP}(0, \kappa_g(\boldsymbol{c}, \boldsymbol{c}')), && (8)\end{aligned}$$

where k_j and k_g are covariance functions defined on each partition and the set of prototypes, respectively. Especially in the upper layer $g(\boldsymbol{c}_j)$ is used for a constant mean function of the GP prior in the lower layer function f_j. Based on the hierarchical GP priors, the noisy observation on the jth partition is defined as

$$y_i^{(j)} = f_j(\boldsymbol{x}_i^{(j)}) + \epsilon_i^{(j)}, i = 1, ...N_j, \tag{9}$$

where $\epsilon_i^{(j)} \sim \mathcal{N}(0, \sigma_j^2)$, and σ_j^2 is a noisy variance of the jth partition.

We therefore have two levels of multivariate Gaussian distributions associated with the functions g and $\{f_j\}$. In the upper layer, let $\boldsymbol{g} = [g_1, ..., g_Q]^\top$, where $g_j = g(\boldsymbol{c}_j)$, be a set of latent values evaluated at $\boldsymbol{C}$. The GP prior over $\boldsymbol{g}$ is given by

$$p(\boldsymbol{g} \,|\, \boldsymbol{C}) = \mathcal{N}(\boldsymbol{g}|0, \boldsymbol{K}_g), \tag{10}$$

where $\boldsymbol{K}_g$ is a $Q \times Q$ matrix with $[\boldsymbol{K}_g]_{ij} = \kappa_g(\boldsymbol{c}_i, \boldsymbol{c}_j)$.

In the lower layer, we can define a set of latent function values evaluated at all training data $\boldsymbol{X}$. By abuse of notation, let us define $\boldsymbol{f} = [\boldsymbol{f}_1^\top, ..., \boldsymbol{f}_Q^\top]^\top \in \mathbb{R}^N$, where $\boldsymbol{f}_j = [f_1^{(j)}, ..., f_{N_j}^{(j)}]^\top$ ($f_i^{(j)} = f_j(\boldsymbol{x}_i^{(j)})$). Note that given the upper layer latent values $\boldsymbol{g}$, $\boldsymbol{f}_j$ and $\boldsymbol{f}_k$, where $j \neq k$, are conditionally independent. Thus the training conditional is expressed as

$$p(\boldsymbol{f}|\boldsymbol{g}, \boldsymbol{X}) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{H}\boldsymbol{g}, \boldsymbol{D}), \tag{11}$$

where $\boldsymbol{H}$ is a $N \times Q$ binary matrix indicating the positions of the data points in the jth partition, so its jth column is defined as

$$[\boldsymbol{H}]_{:,j} = [0, ..., 0, \overbrace{1, ..., 1}^{N_j}, 0, ..., 0]^\top, \tag{12}$$

and, $\boldsymbol{D}$ is a $N \times N$ block diagonal matrix in which the jth block $\boldsymbol{D}_j$ is a $N_j \times N_j$ matrix of the covariances of all pairs of the training data in $j$the partition, i.e.,

$$[\boldsymbol{D}_j]_{lm} = \kappa_j(\boldsymbol{x}_l^{(j)}, \boldsymbol{x}_m^{(j)}).$$

The likelihood of the targets corresponding to the jth partition is given by

$$p(\boldsymbol{y}_j|\boldsymbol{f}_j) = \mathcal{N}(\boldsymbol{y}_j|\boldsymbol{f}_j, \sigma_j^2 \boldsymbol{I}_j), \tag{13}$$

where $\boldsymbol{y}_j = [y_1^{(j)}, ..., y_{N_j}^{(j)}]^\top$ and $\boldsymbol{I}_j$ is a $N_j \times N_j$ identity matrix. The likelihood of all targets, i.e., $\boldsymbol{y} = [\boldsymbol{y}_1^\top, ..., \boldsymbol{y}_Q^\top]^\top$, is then expressed by

$$p(\boldsymbol{y}|\boldsymbol{f}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{\Lambda}), \tag{14}$$

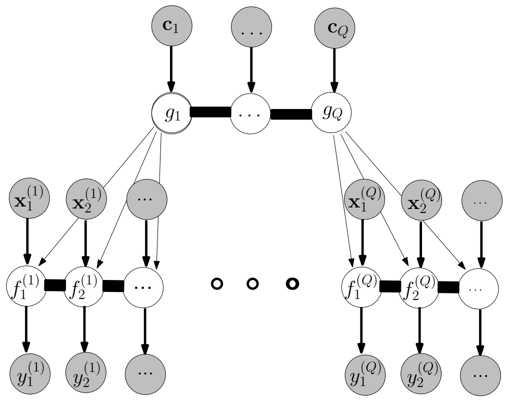

Figure 1: The graphical model for HGPR.

where $\boldsymbol{\Lambda} = \mathrm{diag}(\boldsymbol{\lambda}_1, ..., \boldsymbol{\lambda}_Q)$, and $\boldsymbol{\lambda}_j = [\sigma_j^2, ..., \sigma_j^2]$ is a N_j-dimensional row vector. Figure 1 describes the probabilistic graphical model for HGPR.

The hierarchical modeling provides an approximate covariance matrix, forming block matrix, for the GP prior. Integrating out the latent values in the upper level, the prior distribution over the latent values in the lower layer is given by

$$p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{C}) = \mathcal{N}\left(\boldsymbol{f}|0, \boldsymbol{K}_f\right), \tag{15}$$

where $\boldsymbol{K}_f = \boldsymbol{H}\boldsymbol{K}_g\boldsymbol{H}^\top + \boldsymbol{D}$. We obtain a block covariance matrix in which the elements of each off-diagonal block are approximated to a constant, i.e., the covariance between the corresponding prototype vectors. Due to the block structure of the covariance matrix, the memory space for the training data is $\mathcal{O}(\sum_{j=1}^{Q} N_j^2 + Q^2)$, much smaller than $\mathcal{O}(N^2)$. This scheme also provides the computational benefits for inference and learning hyperparameters which will be explained later.

4. Inference and Prediction

We describe inference and prediction in our HGPR, which involve calculating posterior distributions over latent functions and predictive distributions given test data points.

4.1 Calculation of Posterior Distribution

We first derive the posterior distribution over the latent values in the lower layer, $\boldsymbol{f}$, which is calculated by using Bayes rule on (14) and (15)

$$p(\boldsymbol{f}|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{C}) \;\; = \;\; \mathcal{N}(\boldsymbol{f}|\boldsymbol{\Sigma}_f\boldsymbol{\Lambda}^{-1}\boldsymbol{y}, \boldsymbol{\Sigma}_f), \tag{16}$$

where $\boldsymbol{\Sigma}_f = \boldsymbol{K}_f\boldsymbol{\Sigma}_N^{-1}\boldsymbol{\Lambda}$ and $\boldsymbol{\Sigma}_N$ is a noisy covariance matrix defined by

$$\boldsymbol{\Sigma}_N = \boldsymbol{H}\boldsymbol{K}_g\boldsymbol{H}^\top + \widetilde{\boldsymbol{D}}. \tag{17}$$

where $\widetilde{\boldsymbol{D}} = \boldsymbol{D} + \boldsymbol{\Lambda}$ is also a block diagonal matrix.

We then calculate the posterior distribution over the latent values in the upper layer, $\boldsymbol{g}$. It provides the intermediate quantities for further computations. The likelihood of all targets given $\boldsymbol{g}$ is given by

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{g},\boldsymbol{X}) &= \int p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{g})d\boldsymbol{f} \\ &= \mathcal{N}(\boldsymbol{y}|\boldsymbol{H}\boldsymbol{g},\widetilde{\boldsymbol{D}}). \end{aligned} \tag{18}$$

As a result, the posterior distribution over $\boldsymbol{g}$ is obtained by using Bayes rule on (14) and (10)

$$p(\boldsymbol{g}|\boldsymbol{y},\boldsymbol{C}) = \mathcal{N}(\boldsymbol{g}|\widetilde{\boldsymbol{\mu}},\boldsymbol{A}^{-1}), \tag{19}$$

where $\widetilde{\boldsymbol{\mu}} = \boldsymbol{A}^{-1}\boldsymbol{H}^\top\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{y}$ and $\boldsymbol{A} = \boldsymbol{K}_g^{-1} + \boldsymbol{H}^\top\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{H}$.

The structure of the binary matrix $\boldsymbol{H}$ yields simple computational forms for $\widetilde{\boldsymbol{\mu}}$ and $\boldsymbol{A}$. To show this, we define a Q dimensional vector $\boldsymbol{\mu} \triangleq \boldsymbol{H}^\top\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{y}$ and $Q \times Q$ diagonal matrix $\boldsymbol{\Delta} \triangleq \boldsymbol{H}^\top\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{H}$. Their elements are easily computed by

$$[\boldsymbol{\mu}]_j = \sum_{i=1}^{N_j}[\boldsymbol{\alpha}_j]_i, \quad \text{and} \quad [\boldsymbol{\Delta}]_{j,j} = \sum_{i}^{N_j}[\boldsymbol{d}_j]_i,$$

where $\boldsymbol{\alpha}_j = \widetilde{\boldsymbol{D}}_j^{-1}\boldsymbol{y}_j$ and and $[\boldsymbol{d}_j]_i = \sum_l^{N_j}[\widetilde{\boldsymbol{D}}_j^{-1}]_{i,l}$. In other words $[\boldsymbol{d}_j]_i$ and $[\boldsymbol{\Delta}]_{j,j}$ are the summations of the ith row elements and all elements of $\widetilde{\boldsymbol{D}}_j^{-1}$, respectively. The computations of $\widetilde{\boldsymbol{\mu}}$ and $\boldsymbol{A}$ are simplified as

$$\widetilde{\boldsymbol{\mu}} = \boldsymbol{A}^{-1}\boldsymbol{\mu}, \quad \text{and} \quad \boldsymbol{A} = \boldsymbol{K}_g^{-1} + \boldsymbol{\Delta}.$$

4.2 Calculation of Predictive Distribution

We wish to find the predictive distribution of the target at a new input point $\boldsymbol{x}_* \in \mathcal{X}_j$, where the partition of the new point is given by the dataset or is determined by a following rule $j = \arg\min_i \|\boldsymbol{x}_* - \boldsymbol{c}_i\|^2$. We first define the likelihood conditioned on the latent variable in the lower layer, $\boldsymbol{f}$:

$$p(y_*|\boldsymbol{f}) = \mathcal{N}(y_* \,|\, \boldsymbol{k}^\top\boldsymbol{K}_f^{-1}\boldsymbol{f},\, \kappa_j(\boldsymbol{x}_*,\boldsymbol{x}_*) - \boldsymbol{k}^\top\boldsymbol{K}_f^{-1}\boldsymbol{k} + \sigma_j^2), \tag{20}$$

where $\boldsymbol{k}$ is a vector of covariance between the test input point and all training input points. Here the evaluation of $\boldsymbol{k}$ also involves the approximation induced by the partitioned input space assumption: only the covariances between test point and the training input points in the jth partition are exactly calculated, i.e.,

$$\boldsymbol{k} = \underline{\boldsymbol{k}} + \boldsymbol{H}[\boldsymbol{K}_g]_{:,j}, \tag{21}$$

where $\underline{\boldsymbol{k}} = [0,...,\boldsymbol{k}_j,...,0]^\top$ and $\boldsymbol{k}_j$ is a N_j-dimensional vector, $[\boldsymbol{k}_j]_i = \kappa_j(\boldsymbol{x}_*,\boldsymbol{x}_i^{(j)})$.

Given the new input $\boldsymbol{x}_* \in \mathcal{X}_j$, the predictive distribution is then obtained by integrating the likelihood (20) with respect to the posterior (16)

$$\begin{aligned} p(y_*|\boldsymbol{y},\boldsymbol{X},\boldsymbol{C}) &= \int p(y_*|\boldsymbol{f})p(\boldsymbol{f}|\boldsymbol{y},\boldsymbol{X},\boldsymbol{C})d\boldsymbol{f} \\ &= \mathcal{N}(y_*|\bar{f}(\boldsymbol{x}_*),\bar{\sigma}(\boldsymbol{x}_*)), \end{aligned} \tag{22}$$

where

$$\begin{aligned}\bar{f}(\boldsymbol{x}_*) &= \boldsymbol{k}^\top \boldsymbol{\Sigma}_N^{-1}\boldsymbol{y},\\ \bar{\sigma}(\boldsymbol{x}_*) &= k_j(\boldsymbol{x}_*,\boldsymbol{x}_*) - \boldsymbol{k}^\top \boldsymbol{\Sigma}_N^{-1}\boldsymbol{k} + \sigma_j^2.\end{aligned}$$

The computational bottleneck in HGPR is to calculate the inverse of the noisy covariance matrix $\boldsymbol{\Sigma}_N$. Fortunately its block structure enables us to efficiently calculate the inverse matrix based on matrix inversion lemma

$$\boldsymbol{\Sigma}_N^{-1} = \widetilde{\boldsymbol{D}}^{-1} - \widetilde{\boldsymbol{D}}^{-1}\boldsymbol{H}\boldsymbol{A}^{-1}\boldsymbol{H}^\top\widetilde{\boldsymbol{D}}^{-1}. \tag{23}$$

Note that the inverse of $\boldsymbol{\Sigma}_N$ is given by the inverse of $\{\widetilde{\boldsymbol{D}}_j\}_{j=1}^Q$ and $\boldsymbol{K}_g$, which involve much smaller computational effort than full GP. With (23) and some manipulations, the predictive mean and the predictive variance in (22) are rewritten by

$$\begin{aligned}\bar{f}(\boldsymbol{x}_*) &= \boldsymbol{k}_j^\top\Big(\boldsymbol{\alpha}_j - [\widetilde{\boldsymbol{\mu}}]_j\boldsymbol{d}_j\Big) + ([\boldsymbol{K}_g]_{:,j})^\top\Big(\boldsymbol{\mu} - \boldsymbol{\Delta}\widetilde{\boldsymbol{\mu}}\Big), &(24)\\ \bar{\sigma}(\boldsymbol{x}_*) &= \sigma_j^2 + k_i(\boldsymbol{x}_*,\boldsymbol{x}_*) + \boldsymbol{k}_j^\top\widetilde{\boldsymbol{D}}_j^{-1}\boldsymbol{k}_j - v_j^2[\boldsymbol{A}^{-1}]_{j,j} + 2v_j[\boldsymbol{K}_g]_{j,j}\\ &- ([\boldsymbol{K}_g]_{:,j})^\top\Big\{2v_j\boldsymbol{\Delta}[\boldsymbol{A}^{-1}]_{:,j} - \boldsymbol{\Delta}[\boldsymbol{K}_g]_{:,j} + \boldsymbol{\Delta}\boldsymbol{A}^{-1}\boldsymbol{\Delta}[\boldsymbol{K}_g]_{:,j}\Big\}, &(25)\end{aligned}$$

where $v_j = \boldsymbol{k}_j^\top\boldsymbol{d}_j$. The time complexity for evaluating the predictive mean (24) and variance (25) are $\mathcal{O}(N_j)$ and $\mathcal{O}(N_j^2)$, respectively.

Note that only the training data in the partition including a test point are directly involved with the prediction, while other training data are indirectly used through the covariance function in the upper layer. This approach is useful when each partition of data shows a different property because only training points in the same partition are relevant to the prediction. If the dataset are well partitioned, HGPR naturally can capture these differences among the partitions.

5. Learning Hyperparameters

In our model assumption the covariance function for each partition in the lower layer can have its own parameterizations, i.e., we separately tune the hyperparameters for each partition. Since this flexibility can increase model complexity, we place the single squared exponential kernel in (2) on all partitions. In addition, we also assume that the noise variances of all partition are the same, that is $\sigma^2 = \sigma_1^2 = ... = \sigma_Q^2$. Thus the hyperparameters related to the lower layer are $\theta_f = \{\boldsymbol{\ell}, \rho, \ell_f, \sigma^2\}$. For the covariance function for the upper layer, we use 'rbf' kernel for $\boldsymbol{c}$ and $\boldsymbol{c}'$

$$\kappa_g(\boldsymbol{c},\boldsymbol{c}') = \ell_g \exp\Big(-\frac{1}{2\ell_c}\|\boldsymbol{c}-\boldsymbol{c}'\|^2\Big). \tag{26}$$

We denote hyperparameters related to the upper layer by $\theta_g = \{\ell_g, \ell_c\}$.

The optimal hyperparameters $\theta = \{\theta_g, \theta_f\}$ are estimated by maximizing the marginal likelihood

$$\begin{aligned}p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{C},\theta) &= \int p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{C})d\boldsymbol{f}\\ &= \mathcal{N}(\boldsymbol{y}|0,\boldsymbol{\Sigma}_N). &(27)\end{aligned}$$

Equivalently we can find the optimal solution by minimizing the negative log-marginal likelihood

$$\begin{aligned}\mathcal{L} = & \frac{1}{2}\log|\boldsymbol{\Sigma}_N| + \frac{1}{2}\boldsymbol{y}^\top\boldsymbol{\Sigma}_N^{-1}\boldsymbol{y} \\ = & \frac{1}{2}\left(\log|\widetilde{\boldsymbol{D}}| + \log|\boldsymbol{K}_g| + \log|\boldsymbol{A}| + \boldsymbol{y}^\top\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{y} - \boldsymbol{y}^\top\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{H}\boldsymbol{A}^{-1}\boldsymbol{H}^\top\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{y}\right) \\ = & \frac{1}{2}\left(\sum_{j=1}^{Q}\log|\widetilde{\boldsymbol{D}}_j| + \log|\boldsymbol{K}_g| + \log|\boldsymbol{A}| + \sum_{j=1}^{Q}\boldsymbol{\alpha}_j^\top\boldsymbol{y}_j - \boldsymbol{\mu}^\top\widetilde{\boldsymbol{\mu}}\right), \qquad (28)\end{aligned}$$

where an irrelevant constant is ignored.

The computational complexity of evaluating (28) is $\mathcal{O}(\sum_{j=1}^{Q} N_j^3)$ due to inverse of $\{\widetilde{\boldsymbol{D}}_j\}$. If the size of each partition is almost identical to $\widetilde{N}$, the training complexity of our model will be $\mathcal{O}(\widetilde{N}^2 N)$. It has roughly equal computational cost to sparse approximation methods with the same number of inducing variables. We can optimize the objective function (28) by using the gradient based methods, e.g., quasi-Newton. The gradients are provided in Appendix.

6. Related Work

Most existing methods for the scalable GP regression have been proposed in the frame of sparse approximation. A key assumption in sparse approximation is that the latent values $\boldsymbol{f}$ evaluated at the training inputs are conditionally independent with the latent values $\boldsymbol{f}_*$ evaluated at the test inputs, given R ($R \leqslant N$) inducing variables $\boldsymbol{u}$

$$\begin{aligned}p(\boldsymbol{f},\boldsymbol{f}_*|\boldsymbol{u}) &= p(\boldsymbol{f}|\boldsymbol{u})p(\boldsymbol{f}_*|\boldsymbol{u}) \\ &= \mathcal{N}(\boldsymbol{f}|\boldsymbol{K}_{f,u}\boldsymbol{K}_{u,u}^{-1}\boldsymbol{u}, \boldsymbol{K}_{f,f} - \boldsymbol{Q}_{f,f})\mathcal{N}(\boldsymbol{f}_*|\boldsymbol{K}_{*,u}\boldsymbol{K}_{u,u}^{-1}\boldsymbol{u}, \boldsymbol{K}_{*,*} - \boldsymbol{Q}_{*,*}), \quad (29)\end{aligned}$$

where $p(\boldsymbol{f}|\boldsymbol{u})$ is referred as a training conditional, $p(\boldsymbol{f}_*|\boldsymbol{u})$ as a test conditional. Here, $\boldsymbol{K}_{f,f}$, $\boldsymbol{K}_{*,*}$ and $\boldsymbol{K}_{u,u}$ are covariance matrices for prior distributions of $\boldsymbol{f}$, $\boldsymbol{f}_*$ and $\boldsymbol{u}$, respectively. Each element of these matrices is given by a single covariance function, e.g., $[\boldsymbol{K}_{f,f}]_{i,j} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $\boldsymbol{Q}$s are low-rank matrices, e.g., $\boldsymbol{Q}_{f,f} = \boldsymbol{K}_{f,u}\boldsymbol{K}_{u,u}^{-1}\boldsymbol{K}_{f,u}^\top$, where $\boldsymbol{K}_{f,u}$ is a $N \times R$ matrix of covariances between $\boldsymbol{f}$ and $\boldsymbol{u}$. We can apply further approximations to $p(\boldsymbol{f}|\boldsymbol{u})$ and $p(\boldsymbol{f}_*|\boldsymbol{u})$. By restring the form of covariances matrix in $p(\boldsymbol{f}|\boldsymbol{u})$, several different sparse approximation methods can be derived (Quiñonero-Candela and Rasmussen, 2005):

- deterministic training conditional (DTC) (Seeger et al., 2003)

$$q(\boldsymbol{f}|\boldsymbol{u}) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{K}_{f,u}\boldsymbol{K}_{u,u}^{-1}\boldsymbol{u}, 0),$$

- fully independent training conditional (FITC) (Snelson and Ghahramani, 2006)

$$q(\boldsymbol{f}|\boldsymbol{u}) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{K}_{f,u}\boldsymbol{K}_{u,u}^{-1}\boldsymbol{u}, \mathrm{diag}(\boldsymbol{K} - \boldsymbol{Q})),$$

- partially independent training conditional (PITC)

$$q(\boldsymbol{f}|\boldsymbol{u}) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{K}_{f,u}\boldsymbol{K}_{u,u}^{-1}\boldsymbol{u}, \text{block-diag}(\boldsymbol{K} - \boldsymbol{Q})).$$

Despite the similar form of the training conditionals in PITC and our HGPR (11), HGPR differs from the sparse approximation methods in the aspect that the test latent value $f(\boldsymbol{x}_*)$, where $\boldsymbol{x}_* \in \mathcal{X}_j$, directly communicates to the training latent values for jth partition, $\boldsymbol{f}_j$. This fact enables us to model the different characteristics among the partitions. In the case of sparse approximation, information from $\boldsymbol{f}$ can only be transmitted to $f(\boldsymbol{x}_*)$ through the inducing variables (see 29). Since, however, the inducing variables do not reflect any local property, the sparse approximation methods would fail to model the local characteristics.

Our method is related to partially independent conditional (PIC) (Snelson and Ghahramani, 2007), which is a variant of PITC and directly incorporates the partitioned structure of input data into inference. PIC assumes a more relaxed assumption than (29), where the test latent value $f(\boldsymbol{x}_*)$, for $\boldsymbol{x}_* \in \mathcal{X}_j$, is grouped with the jth partition. The approximate conditional is given by

$$p(\boldsymbol{f}, f(\boldsymbol{x}_*)|\boldsymbol{u}) = p(\boldsymbol{f}_j, f(\boldsymbol{x}_*)|\boldsymbol{u}) \prod_{i \neq j}^{Q} p(\boldsymbol{f}_i|\boldsymbol{u}), \tag{30}$$

where $\boldsymbol{f}_i$ is a set of latent values for the ith partition as described in Section 3. Here HGPR has also the same form of conditional if $\boldsymbol{u}$ is replaced with $\boldsymbol{g}$. As shown in (30), the test latent value can directly communicate $\boldsymbol{f}_j$, the training latent values for the jth partition, as in our HGPR model. In this point of view, PIC also can model the local characteristics among the partitions. It is worth for considering the covariance function to understand the difference between PIC and HGPR. When $\boldsymbol{x} \in \mathcal{X}_i$ and $\boldsymbol{x}' \in \mathcal{X}_j$, the covariance function in PIC is given by

$$\mathrm{cov}(\boldsymbol{x}, \boldsymbol{x}') = Q(\boldsymbol{x}, \boldsymbol{x}') + \delta(i,j)[k(\boldsymbol{x}, \boldsymbol{x}) - Q(\boldsymbol{x}, \boldsymbol{x})], \tag{31}$$

where $\delta(i,j)$ is a delta function such that it is equal to 1 if i=j, otherwise 0, and $Q(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{k}_{x,u}^\top \boldsymbol{K}_u^{-1} \boldsymbol{k}_{x,u}$, where $\boldsymbol{k}_{x,u}$ is a R dimensional vector of the covariances evaluated at all pairs of $\boldsymbol{x}$ and a set of inducing points that are input points corresponding to the inducing variables. In our HGPR model, the covariance function is defined by

$$\mathrm{cov}(\boldsymbol{x}, \boldsymbol{x}') = k_g(\boldsymbol{c}_i, \boldsymbol{c}_j) + \delta(i,j) k_f(\boldsymbol{x}, \boldsymbol{x}'). \tag{32}$$

Two methods are different in the way that how the covariance between two points from different partitions is evaluated. In PIC, the covariance is approximated by the low-rank covariance function Q, while in HGPR it is approximated by the covariance between two prototypes. Since in HGPR the covariances between all pairs of input points from two different partitions are set to a constant, i.e., the covariance between two prototypes, PIC provides more accurate covariances. However PIC should optimize the inducing inputs, which involves an optimization of $R \times D$ dimensional parameters. In the case of high-dimensional input space, PIC may suffer from overfitting. In this case, we expect that HGPR gives better results. We confirm this point in Section 7.1, where our HGPR is compared with PIC in the case of high-dimensional input data.

Our method is also related to a multistrategy approach that integrates two or more inferential strategies to solve more complex problems (Michalski and Tecuci, 1994). In our case, first strategy is clustering and second strategy is inference. Unfortunately since both strategies are completely separated, the results from the inference can not make clustering better. In that sense, our method is a naive version of mixture of GPs (Tresp, 2001; Rasmussen and Ghahramani, 2002) in which clustering and inference are done simultaneously. However due to unfavorable scaling of the mixture of GPs, our method still has attractions to handle with the large datasets.

7. Numerical Experiments

We evaluate the performance of our method, HGPR, on several large datasets, compared to four sparse approximation methods, DTC, FITC, PITC, and PIC described in Section 6. We denote R as the number of the inducing variables for these methods. To implement these sparse approximation methods, we use the GP software toolbox[2]. Additionally we consider a full GP as a baseline method in comparison. For all cases, the covariance function is the squared exponential kernel in (2) and the hyperparamters are trained separately for each model.

This section consists of two parts: the comparison of PIC and HGPR in the case of high-dimensional input data; the performance evaluation of the methods on the large datasets. As mentioned before, PIC need to optimize the inducing inputs, which involves the high-dimensional optimization. In the experiment with the high-dimensional input data, we show that PIC suffers from overfitting. We next compare the performance of all methods on the real-world datasets, some of which have the clustered structure in the input data.

In order to evaluate the predictive performance of each method, we use a normalized mean square error (NMSE) that is defined by

$$\mathrm{NMSE} = \big\langle (y^* - \bar{f}(\boldsymbol{x}_*))^2 \big\rangle / \big\langle (y^* - \bar{y})^2 \big\rangle, \tag{33}$$

where $\langle \cdot \rangle$ averages over the test data and $\bar{y}$ is the mean value of the test targets.

7.1 Comparison with PIC

We examine the performance of PIC and our method in the case where input points are lying on high-dimensional space, and are already clustered. We generate input points from 50 number of 100-dimensional Gaussian distributions, the mean vectors of which are well separated. We draw the same number of input points from each Gaussian distribution, i.e., $Q = 50$ and $N_1 = N_2, ..., = N_Q = \widetilde{N}$. The target variable for $\boldsymbol{x}$ is defined by

$$y(\boldsymbol{x}) = \exp\Big\{ -5e^{-5} \sum_{l=1}^{20} [\boldsymbol{x}]_l^2 \Big\} + 0.05\varepsilon \tag{34}$$

where $\varepsilon \sim \mathcal{N}(0, 1)$ and we assume that only first 20 elements of input point are relevant. The test data are generated by the same manner (1000 samples for each test data). The label of Gaussian distributions are used to partition the input points.

We conduct the experiments with $\widetilde{N}$ varying from 10 to 100, thus the number of training data N is varying from 500 to 5000. For each $\widetilde{N}$, experiments are repeated ten times, and we report the averages. To impose the same computational complexity, we set the number

2. see http://www.cs.manchester.ac.uk/∼neill/software.html

of inducing variables R of PIC to $\widetilde{N}$. As mentioned in Section 6, PIC should optimize the inducing points to complete inference. In this case, the size of parameters to be estimated is $|\theta| + RD = |\theta| + 100\widetilde{N}$, where $|\theta|$ is a number of the hyperparameters in PIC. This high-dimensional optimization often yields overfitting problem. On the other hands, HGPR does not need the optimization of the inducing inputs: the parameters to be estimated are only the hyperparameters related to the covariance functions, $\{\theta_g, \theta_f\}$. Although PIC provides more accurate covariance function, it may suffer from overfitting in the case of the high-dimensional input data. We can confirm this fact from Figure 2, which shows the predicative performance, in terms of NMSE, of PIC and HGPR.

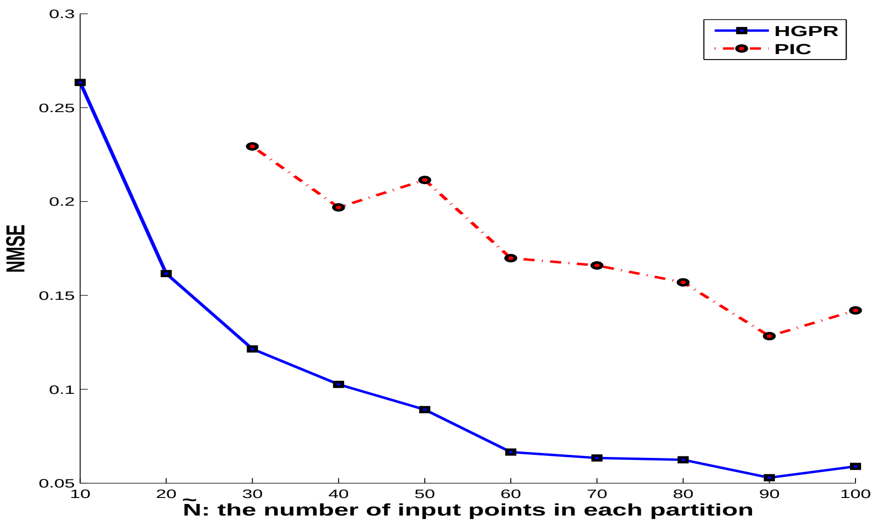

Figure 2: The predicative error (NMSE) of PIC and HGPR, when the number of input points in each partition, $\widetilde{N}$, varies from 10 to 100, in the case where the dimension of input data is 100 and the input points are clustered. Note that, PIC fails to work at $\widetilde{N} = 10, 20$.

7.2 Performance Evaluation

We use six datasets [3] whose descriptions are summarized in Table 1. Kin-40k and pumadyn-32nm are artificially generated datasets describing the dynamics of a robot arm. Elevators is a prediction problem related to control of the elevators of an F16 aircraft. California housing (Cal-housing) and house-price-16H are collected from 1990 US Census, and are concerned with predicting the median price of the house in a small survey region. In contrast to Cal-housing collected in California, house-price-16H includes census records of all states in US.

3. Kin-40k: see http://ida.first.fraunhofer.de/~anton/data.html. Pumadyn-32nm and house-price-16H: see http://www.cs.toronto.edu/delve/data/datasets.html. *Cal-housing*: see http://lib.stat.cmu.edu. Elevators: see http://www.liaad.up.pt/~ltorgo/Regression/DataSets.htm. School-exam: see http://multilevel.ioe.ac.uk/intro/datasets.html.

School-exam consists of examination records from 139 secondary schools in 1985, 1986 and 1987. Actually it includes four features for each student and four features for each school. In this paper we only use the features for students as the inputs, and consider a regression problem to predict exam score for each student from these values.

For preprocessing, we normalize each input and target to have zero mean and unit variance. Only the attributes related to currency, e.g., the median price of house in Cal-housing dataset, are firstly transformed by the logarithmic function. Experiments are repeated ten times, and we report their statistics, mean and standard deviation.

Table 1: Description of data sets.

Data set	input dim.	# training(test)
kin-40k	8	10000(30000)
pumadyn-32nm	32	7168(1024)
Elevators	18	16599(-)
Cal-housing	8	20640(-)
house-price-16H	16	22784(-)
school-exam	4	15362(-)

For HGPR and PIC, we need to divide the training input data into Q partitions. The partitions of input data can be provided by the information of the dataset: in the cases of house-price-16H and school-exam datasets, the examples are clustered according to the state and the school, respectively. Otherwise we can use the k-means algorithm in which the center of each cluster becomes the prototype vector. We separately consider two artificial datasets, Kin-40k and pumadyn-32nm, from other datasets because their training input data are uniformly distributed without the clustered structure.

We thus first consider the datasets, Kin-40k and pumadyn-32nm. In both cases the size of each partition generated by k-means algorithm is almost identical, i.e., $\{N_j\}_{j=1}^{Q} \approx \widetilde{N} = N/Q$, where N_j is the number of training data in the jth partition. We investigate the relation of the predictive accuracy of HGPR and the average number of training data in the partition, $\widetilde{N}$. Figure 3 shows the average NMSE on both datasets when $\widetilde{N}$ is increased ($\widetilde{N}$ can be determined by adjusting Q). HGPR gives reasonable predictive accuracy when each partition has enough training points. Since the time complexity of our method is proportion to the number of points in the partition, we should carefully choose the number of partitions in terms of both efficiency and accuracy. In the case where the dataset does not have the clustered structure in the input data, we can not obtain further benefits from our method, compared to other sparse approximation methods.

We now compare the predictive performance of each method on four real-world datasets. Especially in the case of school-exam dataset, the prediction of exam score for the student can be slightly different according to the school. In other word, the partitions of dataset have different characteristics. The house-price-16H dataset shows the similar phenomena. To test whether the prediction utilizing the clustered structure in the input data improves the predictive accuracy, we consider the case where the training input data are randomly divided into $Q = 30$ partitions, and HGPR is trained on this randomly partitioned dataset. This approach is denoted as r-HGPR.

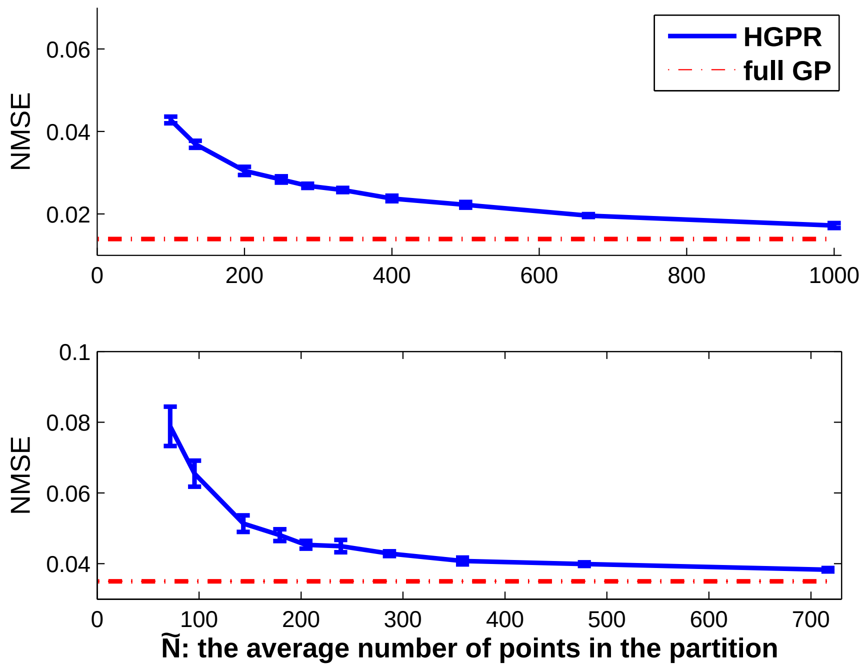

Figure 3: The predictive error (NMSE) of our method varying with respect to the average number of training data in the partitions, $\widetilde{N} = N/Q$, on: (top) kin-40K dataset; (bottom) pumadyn-32mn dataset. The horizontal line is NMSE for the full GP with the hyperarmaeters trained on a subset of data of size 2000 for kin-40k and 1024 for pumadyn-32.

Table 2: Comparison of the predictive performance, in terms of NMSE, for all methods. The result of PIC for 'Elevators' dataset is not available, since PIC shows unstable convergence to the given dataset.

Data set	f-GP	DTC	FITC	PITC	PIC	r-HGPR	HGPR
Cal-housing	**0.1779**	0.2881	0.2184	0.2150	0.2273	0.2935	0.2007
(std.)	(0.0099)	(0.0388)	(0.0082)	(0.0043)	(0.0046)	(0.0037)	(0.0051)
Elevators	**0.0867**	0.1044	0.1106	0.1083	-	0.1238	0.0933
(std.)	(0.0041)	(0.0029)	(0.0156)	(0.0034)	(-)	(0.0042)	(0.0023)
school-exam	0.6739	0.6645	0.7233	0.6973	0.6597	0.7182	**0.6541**
(std.)	(0.0067)	(0.0044)	(0.0751)	(0.0091)	(0.0097)	(0.0057)	(0.0119)
house-price-16H	0.2098	0.2647	0.2553	0.2446	0.1804	0.3534	**0.1760**
(std.)	(0.0063)	(0.0103)	(0.0071)	(0.0042)	(0.0046)	(0.0048)	(0.0029)

For Cal-housing and elevators datasets, we also use k-means algorithm. In these cases, the size of each partition can be variable and some partitions may have too small number of points. If N_j is smaller than given threshold (200), we merged the data points in the jth partition into the nearest partition, in terms of Euclidean distance between the data point and the center of the cluster. After this procedure the number of training points in the partitions for Cal-housing less than 800, and for elevators, 600. To impose equal computational cost to sparse approximation methods, we set R to 800 for Cal-housing and 600 for elevators.

For house-price-16H and School-exam datasets, we can partition the training data based on the additional variable indicating the state or the school of the given example. Then the mean vector of each cluster is used to the prototype vector. As the number of the training inputs in the partitions for house-price-16H less than 600, and for school-exam, 200, we set R to 600 and 200 for both datasets. For other compared methods, we add this indicate variable (state/school) into the input attributes. The dataset is randomly divided into the training data ($N = 10000$) and the test data (remainder), and the hyperparameters for full GP are trained on a subset of data of size 2000.

Table 2 presents the average NMSE of each method. For all cases our method shows good predictive performance. Especially as we claimed, our method gives best performance in the cases of house-price-16H and School-exam datasets, even the predictive error of our method is lower than those of full GP. We also see that our method always outperforms rand-HGPR. This results demonstrate that utilizing the clustered structure in the input data is useful to improve the prediction performance for the dataset which reveals the different characteristics among the partitions.

8. Conclusions

We have presented a two-layer hierarchical model for GP regression where the covariance matrix is approximated by a block matrix such that diagonal blocks are exactly calculated while off-diagonal blocks are approximated. Partitioning input data points using a clustering algorithm, the latent variables in the upper layer define the mean functions of GP priors in the lower layer, while the latent variables associated with individual GP priors in the lower layer represent the noiseless latent function values evaluated at data points in the corresponding partition. By integrating out the latent variables in the upper layer leads to the block covariance matrix that enables us to reduce both the time complexity and the memory space. With appropriate partitioning methods, our method showed the high predictive performance in the case where the partitions of data reveals the different characteristics. In the case where the information for partitioning the input data is not provided by the dataset, the result of clustering algorithm is crucial to the performance of our method. Since the clustering is completely separated from the inference, the feedback from inference can not be used to make clustering better. This limitation will be solved by adopting the idea of mixture of GPs (Tresp, 2001; Rasmussen and Ghahramani, 2002) into our model formulation. Furthermore we will extend our approximation scheme to the classification task, in which the full GP is intractable due to its limited scalability. Approximation inference such as variational Bayesian (VB) inference or expectation propagation (EP) can be incorporated into the hierarchical GP prior model. The assumption on the partitioned input space could be more appropriate to model classification datasets because they are naturally clustered according to the class.

Acknowledgments

This work was supported by National Research Foundation (NRF) of Korea (No. 2010-0014306), NIPA Program of Software Engineering Technologies Development and Experts Education, NIPA ITRC Support Program (NIPA-2010-C1090-1031-0009), and NRF WCU Program (Project No. R31-2008-000-10100-0).

Appendix

In this section we provide the gradient of the negative log-marginal likelihood (28), in Section 5, with respect to the hyperparameters $\phi \in \{\theta_f, \theta_g\}$. Objective function (28) can be rewritten as

$$\begin{aligned} \mathcal{L} &= \frac{1}{2}\Big(\log|\widetilde{\boldsymbol{D}}| + \log|\boldsymbol{K}_g| + \log|\boldsymbol{A}|\Big) \\ &+ \frac{1}{2}\text{tr}\left\{\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{y}\boldsymbol{y}^\top\right\} - \frac{1}{2}\text{tr}\left\{\boldsymbol{A}^{-1}\boldsymbol{H}^\top\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{y}\boldsymbol{y}^\top\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{H}\right\}, \end{aligned} \tag{35}$$

Then the gradient is calculated based on the notations for the matrix derivative in (Brookes, 2005). One can refer to the derivations in (Lawrence, 2007) which discusses the similar form of objective function. We only include the final results of the derivations.

We first define

$$\begin{aligned} \boldsymbol{M} &\triangleq \boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{H}^\top\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{y}\boldsymbol{y}^\top\widetilde{\boldsymbol{D}}^{-1}\boldsymbol{H}\boldsymbol{A}^{-1} \\ &= \boldsymbol{A}^{-1} + \widetilde{\boldsymbol{\mu}}\widetilde{\boldsymbol{\mu}}^\top. \end{aligned} \tag{36}$$

In the case of the upper layer, the derivative of (35) with respect to $\phi \in \theta_g$ is given by,

$$\frac{\partial \mathcal{L}}{\partial \phi} = \frac{1}{2}\text{tr}\left\{\Big(\boldsymbol{K}_g^{-1} - \boldsymbol{K}_g^{-1}\boldsymbol{M}\boldsymbol{K}_g^{-1}\Big)^\top \frac{\partial \boldsymbol{K}_g}{\partial \phi}\right\}. \tag{37}$$

In the case of the lower layer, the derivative of (35) with respect to $\phi \in \theta_f$ is given by

$$\frac{\partial \mathcal{L}}{\partial \phi} = \frac{1}{2}\sum_{j=1}^{Q}\text{tr}\left\{\boldsymbol{\Gamma}_j^\top \frac{\partial \widetilde{\boldsymbol{D}}_j}{\partial \phi}\right\} \tag{38}$$

where $\boldsymbol{\Gamma}_j = \widetilde{\boldsymbol{D}}_j^{-1} - \boldsymbol{\alpha}_j\boldsymbol{\alpha}_j^\top + \boldsymbol{d}_j\Big(2[\widetilde{\boldsymbol{\mu}}]_j\boldsymbol{\alpha}_j^\top - [\boldsymbol{M}]_{jj}\boldsymbol{d}_j^\top\Big)$.

References

M. Brookes. The matrix reference manual, 2005. [online] http://www.ee.ic.ac.uk/hp/staff/dmb/ matrix/intro.html.

L. Csató and M Opper. Sparse on-line Gaussian processes. *Neural Computation*, 14:641–668, 2002.

N. Lawrence. Learning for larger datasets with the Gaussian process latent variable model. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, San Juan, Puerto Rico, 2007.

N. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process models: The informative vector machine. In *Advances in Neural Information Processing Systems (NIPS)*, volume 15. MIT Press, 2003.

M. Lazaro-Gredilla and A. Figueiras-Vidal. Inter-domain Gaussian processes for sparse inference using inducing features. In *Advances in Neural Information Processing Systems (NIPS)*, volume 22. MIT Press, 2009.

R. S. Michalski and G. Tecuci. *Machine Learning: A Multistrategy Approach*. Morgan Kaufmann, San Francisco, CA, 1994.

J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

C. E. Rasmussen and Z. Ghahramani. Infite mixtures of Gaussian process experts. In *Advances in Neural Information Processing Systems (NIPS)*, volume 14. MIT Press, 2002.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

S. Raudenbush and A. Bryk. *Hierarchical Linear Models*. Thousand Oaks: Sage Publications, 2 edition, 2002.

M. Seeger, C. K. I. Williams, and N. D. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2003.

A. J. Smola and P. Bartlett. Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13. MIT Press, 2001.

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems (NIPS)*, volume 18. MIT Press, 2006.

E. Snelson and Z. Ghahramani. Local and global sparse Gaussian process approximataions. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, San Juan, Puerto Rico, 2007.

V. Tresp. A Bayesian committee machine. *Neural Computation*, 12(11):2719–2741, 2000.

V. Tresp. Mixture of Gaussian processes. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13. MIT Press, 2001.

C. Walder, K. I. Kim, and B. Schölkopf. Sparse multiscale Gaussian process regression. In *Proceedings of the International Conference on Machine Learning (ICML)*, Helsinki, Finland, 2008.

C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13. MIT Press, 2001.