Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence

Lazy Gaussian Process Committee for Real-Time Online Regression

Han Xiao and **Claudia Eckert**
Institute of Informatics, Technische Universität München
Garching, D-85748, Germany
{*xiaoh, claudia.eckert*}*@in.tum.de*

Abstract

A significant problem of Gaussian process (GP) is its unfavorable scaling with a large amount of data. To overcome this issue, we present a novel GP approximation scheme for online regression. Our model is based on a combination of multiple GPs with random hyperparameters. The model is trained by incrementally allocating new examples to a selected subset of GPs. The selection is carried out efficiently by optimizing a submodular function. Experiments on real-world data sets showed that our method outperforms existing online GP regression methods in both accuracy and efficiency. The applicability of the proposed method is demonstrated by the mouse-trajectory prediction in an Internet banking scenario.

Introduction

Gaussian process (GP) is a promising Bayesian method for non-linear regression and classification (Rasmussen and Williams 2006). It has been demonstrated to be applicable to a wide variety of real-world statistical learning problems. An important advantage of GP over other non-Bayesian models is the explicit probabilistic formulation of the model, allowing one to assess the uncertainty of predictions. In addition, since GP has a simple parameterization and the hyperparameters can be adjusted by maximizing the marginal likelihood, it is easy to implement and flexible to use.

Nevertheless, GP is not always the method of choice especially for large date sets. GP has inherently dense representations in the sense that all training data is required for the prediction. The training procedure requires computation, storage and inversion of the full covariance matrix, which can be time-consuming. Furthermore, the Bayesian posterior update to incorporate data is also computationally cumbersome. These drawbacks prevent GP from applications with large amounts of training data that require fast computation, such as learning motor dynamics in real-time.

Much research in recent years has focused on reducing the computational requirements of GP on large data sets. Many of these methods are based on a small set of training inputs, which summarizes the bulk of information provided by all training data (Smola and Bartlett 2001; Seeger, Williams, and Lawrence 2003; Snelson and Ghahramani 2006; Quiñonero-Candela and Rasmussen 2005). Other methods make structural assumptions about the covariance matrix so that a GP can be decomposed into a number of smaller GPs (Tresp 2000; Nguyen-tuong and Peters 2008; Chen and Ren 2009).

In this paper we present a novel approximation method called *lazy Gaussian process committee* (LGPC) for learning from a continuous data stream. As its name suggests, LGPC is based on a combination of multiple GPs, which is closely related to several previous work (Tresp 2000; Nguyen-tuong and Peters 2008; Chen and Ren 2009). Unlike previous work, our model is updated in a "lazy" fashion in the sense that new training examples are directly allocated to a subset of GPs without adapting their hyperparameters. The problem of selecting a near-optimal subset is formulated as submodular optimization, allowing the training procedure to be carried out efficiently. Experiments showed that LGPC has comparable accuracy to the standard GP regression and outperforms several GP online alternatives. The simplicity and the efficiency of LGPC make it more appealing in real-time online applications.

We applied LGPC to mouse-trajectory prediction in an Internet banking scenario. The intention was to predict user's hand movements in real-time, so as to offer support to security applications, such as recognizing identity theft or an abnormal funds transfer. Thus, the model's learning and prediction should be sufficiently fast. In the field of psychology and cognitive science, there has been abundant evidence that a motor dynamic of the hand can reveal the time course of mental process (Abrams and Balota 1991; Song and Nakayama 2008). Moreover, several studies have showed that computer mouse-trajectory can afford valuable information about the temporal dynamics of a variety of psychological process (Spivey, Grosjean, and Knoblich 2005; Freeman et al. 2010; Freeman and Ambady 2009). For instance, unintentional stress might manifest less smooth, more complex and fluctuating trajectories (Dale, Kehoe, and Spivey 2007). An online learning technique is necessary as it allows the adaption to changes in the trajectories. The effectiveness of the online mouse-trajectory learning confirms the applicability of our method.

Related Work

This section briefly reviews Gaussian process and previous attempts on reducing its computational complexity. The underlying problems that motivate this work are highlighted.

GP Regression

The problem of regression aims to find a function estimation from the given data, which is usually formulated as follows: given a training set $\mathcal{D} \coloneqq \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ of N pairs of input vectors $\mathbf{x}_n$ and noisy scalar outputs y_n, the goal is to learn a function f transforming an input into the output given by $y_n = f(\mathbf{x}_n) + \epsilon_n$, where $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ and σ^2 is the variance of the noise. A *Gaussian process* is a collection of random variables, any finite number of which have consistent joint Gaussian distribution. *Gaussian process regression* (GPR) is a Bayesian approach which assumes a GP prior over functions. As a result the observed outputs behave according to

$$p(\mathbf{y} \,|\, \mathbf{x}_1, \ldots, \mathbf{x}_N) = \mathcal{N}(\mathbf{0}, \mathbf{K}),$$

where $\mathbf{y} \coloneqq [y_1, \ldots, y_N]^\top$ is a vector of output values, and $\mathbf{K}$ is an $N \times N$ covariance matrix; the entries are given by a covariance function, i.e. $K_{ij} \coloneqq k(\mathbf{x}_i, \mathbf{x}_j)$. In this work, we consider a frequently used covariance function given by

$$k(\mathbf{x}_i, \mathbf{x}_j) \coloneqq \kappa^2 \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{W} (\mathbf{x}_i - \mathbf{x}_j)\right) + \sigma^2 \delta_{ij},$$

where κ denotes the signal variance and $\mathbf{W}$ are the widths of the Gaussian kernel. The last term represents an additive Gaussian noise, i.e. $\delta_{ij} \coloneqq 1$ if $i = j$, otherwise $\delta_{ij} \coloneqq 0$.

In the setting of probabilistic regression, the goal is to find a predictive distribution of the output y_* at a test point $\mathbf{x}_*$. Under GPR, the predictive distribution of y_* conditional on the training set $\mathcal{D}$ is also Gaussian

$$p(y_* \,|\, \mathcal{D}, \mathbf{x}_*) = \mathcal{N}\left(\mathbf{k}_*^\top \mathbf{K}^{-1} \mathbf{y}, k_* - \mathbf{k}_*^\top \mathbf{K}^{-1} \mathbf{k}_*\right), \quad (1)$$

where $\mathbf{k}_* \coloneqq [k(\mathbf{x}_*, \mathbf{x}_1), \ldots, k(\mathbf{x}_*, \mathbf{x}_N)]^\top$ and $k_* \coloneqq k(\mathbf{x}_*, \mathbf{x}_*)$. One can observe that the training data is explicitly required at the test time in order to construct the predictive distribution, which makes GP a non-parametric method. The hyperparameters are $[\kappa^2, \sigma^2, \{\mathbf{W}\}]^\top$, where $\{\mathbf{W}\}$ denotes parameters in the width matrix $\mathbf{W}$. The optimal hyperparameters for a particular data set can be derived by maximizing the marginal likelihood function using a gradient based optimizer. For a more detailed background on GP, readers are referred to the textbook (Rasmussen and Williams 2006).

It should be noted that in each iteration the computation of the likelihood and the derivatives involves inversion of a matrix of size $N \times N$, which requires time $\mathcal{O}(N^3)$. Once the inversion is done, inference on a new test point requires $\mathcal{O}(N)$ for the predictive mean and $\mathcal{O}(N^2)$ for the predictive variance. Thus, a simple implementation of GPR can handle problems with at most a few thousands training examples, which prevents it from real-time applications dealing with large amounts of data.

GP Approximations

The sparse representation of data has been studied exhaustively (Kimeldorf and Wahba 1971). It has been shown that the bulk of information provided by all training inputs can be summarized by a small set of inputs, which is often known as *inducing inputs* or support vectors. By assuming additional dependency about the training data given the inducing inputs, various sparse GP approximations were derived, such as the subset of regressors (Smola and Bartlett 2001), projected latent variables (Seeger, Williams, and Lawrence 2003) and sparse GP with pseudo-inputs (Snelson and Ghahramani 2006). A unifying view of these sparse GP methods was presented in (Quiñonero-Candela and Rasmussen 2005).

It should be noted that the selection of inducing inputs does leave an imprint on the final solution. Loosely speaking, the selection can be carried out either in a passive (e.g. random) or active fashion. An extensive range of proposals were suggested to find a near-optimal choice for inducing inputs, such as posterior maximization (Smola and Bartlett 2001), maximum information gain (Seeger, Williams, and Lawrence 2003), matching pursuit (Keerthi and Chu 2006) and pseudo-input via continuous optimization (Snelson and Ghahramani 2006). In particular, sparse online GP (Csató and Opper 2002) was developed by combining the idea of a sparse representation with an online algorithm, allowing the inducing inputs (basis vectors) to be constructed in a sequential fashion.

An alternative approach for speeding up GPR is Bayesian committee machine (BCM) introduced by (Tresp 2000). Loosely speaking, the original training data is partitioned into parts, where each part is maintained by a GP. The computational cost is thus significantly reduced due to much smaller number of training examples within each GP. BCM provides a principled approach to combining Bayesian estimators trained on different data sets for the prediction. Inspired by this idea, some other work have been focused on combining multiple GPs for regression (Nguyen-tuong and Peters 2008; Chen and Ren 2009).

In the online setting, a straightforward application of the approaches mentioned above is impeded by two obstacles. First, it is inefficient to optimize hyperparameters every time a new training example is presented. Second, adding new training examples to the model may cause non-smooth fluctuations in the marginal likelihood function and its gradients, meaning that a smooth convergence is not guaranteed (Seeger, Williams, and Lawrence 2003). Although several heuristics can be adapted to alleviate this problem, there is no reliable way of learning hyperparameters. Moreover, as training examples are presented sequentially rather than in batch, selecting inducing inputs from a data stream in a far-sighted fashion becomes extremely challenging. Furthermore, the inducing inputs selection and the hyperparameters estimation are somewhat undermined by each other (Snelson and Ghahramani 2006), which may adversely affect the quality and the efficiency of online regression.

LGPC for Online Regression

The basic idea of our approach is straightforward. Instead of training a single GP using all the training data, we partition the data and allocate it to a committee consisted of Q independent GPs with different hyperparameters. That is, each GP maintains a subset of the training data, which is denoted as $\mathcal{D}_1, \ldots, \mathcal{D}_Q$ respectively, where $\mathcal{D}_q \coloneqq \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ and T is the maximum number of training examples of each GP. Intuitively, each GP in the committee corresponds to an interpretation of the relationship between input and output. For predicting the output y_* of a query point $\mathbf{x}_*$, the outputs from all GPs are combined together. Assuming that any partition $\mathcal{D}_i$ is conditionally independent with all other partitions $\{D_j, \ldots\}, \forall j \neq i$ given the outputs at test points, which is reasonable when the number of test points is large, we have $p(\mathbf{y}_1, \ldots, \mathbf{y}_Q \,|\, \mathcal{D}) \approx \prod_{q=1}^{Q} p(\mathbf{y}_q \,|\, \mathcal{D}_q)$, the predictive distribution can be approximated as

$$\widehat{p}(y_* \,|\, \mathcal{D}, \mathbf{x}_*) \propto c \frac{\prod_{q=1}^{Q} p(y_* \,|\, \mathcal{D}_q, \mathbf{x}_*)}{[p(y_*)]^{Q-1}}, \tag{2}$$

where c is a normalization constant. The denominator of Eq. (2) serves as a prior distribution of y_*. For a single point, it is considered as zero-mean Gaussian with unit variance. For a set of query points, the prior is considered as $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_*)$, where $\mathbf{\Sigma}_*$ is the sample covariance matrix.

This section describes LGPC in three parts, namely the allocation of new training examples, the incremental update and the predictions of query points. Note that all hyperparameters of LGPC are constants during online learning.

Allocation of New Training Examples

Denote all GPs in the committee as $\mathcal{Q} \coloneqq \{1, \ldots, Q\}$. Given a new training example $(\mathbf{x}_{N+1}, y_{N+1})$, we select a subset $\mathcal{A} \subseteq \mathcal{Q}$ and allocate the new example to their data collection, respectively. Denote $\mathcal{D}_q^{\langle N \rangle}$ as the training examples allocated to the q^{th} GP at time N, the update rule is formalized as

$$\mathcal{D}_q^{\langle N+1 \,|\, \mathcal{A} \rangle} \coloneqq \begin{cases} \mathcal{D}_q^{\langle N \rangle} \cup \{(\mathbf{x}_{N+1}, y_{N+1})\} & \text{if } q \in \mathcal{A}; \\ \mathcal{D}_q^{\langle N \rangle} & \text{otherwise.} \end{cases}$$

On the one hand, if $\mathcal{A}$ contains only one element, meaning that only one GP is updated each time, then the information provided by the new training example may not be well utilized. On the other hand, if we let $\mathcal{A} \coloneqq \mathcal{Q}$, then all GPs must update their corresponding Gram matrix to include the new training example (no matter whether such inclusion will contribute to the prediction of the committee or not), which can degrade the efficiency and the quality of the prediction. Thus, we need an *active* selection policy to choose at most S GPs from the committee, such that their data inclusion can yield the maximal improvement for prediction.

Clearly it can make sense to select which GPs are taken into $\mathcal{A}$ by optimizing some criterion. The idea here is to maximize the likelihood of LGPC on a small subset of training examples. To see this we introduce a *reference set* $\mathcal{R}$, in which both inputs $\mathbf{X}_\mathcal{R}$ and outputs $\mathbf{y}_\mathcal{R}$ are observed. The reference set can be constructed, for instance, by subsampling all previous training data $\{\mathcal{D}_q^{\langle N \rangle}\}_{q=1}^{Q}$ with the addition of the current training example $(\mathbf{x}_{N+1}, y_{N+1})$. Note that the terms in the numerator and the denomination of Eq. (2) are all Gaussian distributions over y_*. Thus, the predictive distribution for $\mathbf{y}_\mathcal{R}$ at time N can be approximated by a Gaussian distribution with mean and covariance as follows

$$\mathbb{E}_{\widehat{p}}^{\langle N \rangle}(\mathbf{y}_\mathcal{R}) = \mathbb{C}_{\widehat{p}}^{\langle N \rangle}(\mathbf{y}_\mathcal{R}) \sum_{q \in \mathcal{Q}} \Big(\mathbb{C}(\mathbf{y}_\mathcal{R} \,|\, \mathcal{D}_q^{\langle N \rangle}, \mathbf{X}_\mathcal{R})^{-1} \mathbb{E}(\mathbf{y}_\mathcal{R} \,|\, \mathcal{D}_q^{\langle N \rangle}, \mathbf{X}_\mathcal{R}) \Big) \tag{3}$$

$$\mathbb{C}_{\widehat{p}}^{\langle N \rangle}(\mathbf{y}_\mathcal{R}) = \Big(-(Q-1)\mathbf{\Sigma}_{\mathcal{R}\mathcal{R}}^{-1} + \sum_{q \in \mathcal{Q}} \mathbb{C}(\mathbf{y}_\mathcal{R} \,|\, \mathcal{D}_q^{\langle N \rangle}, \mathbf{X}_\mathcal{R})^{-1} \Big)^{-1}, \tag{4}$$

where $\mathbf{\Sigma}_{\mathcal{R}\mathcal{R}}$ is the covariance matrix evaluated at $\mathbf{X}_\mathcal{R}$. The predictive mean and covariance of each GP, i.e. $\mathbb{E}(\mathbf{y}_\mathcal{R} \,|\, \mathcal{D}_q^{\langle N \rangle}, \mathbf{X}_\mathcal{R})$ and $\mathbb{C}(\mathbf{y}_\mathcal{R} \,|\, \mathcal{D}_q^{\langle N \rangle}, \mathbf{X}_\mathcal{R})$, can be obtained from Eq. (1). Note that as $\mathbf{y}_\mathcal{R}$ is known, the log probability of $\mathbf{y}_\mathcal{R}$ under the current model can be evaluated by substituting Eqs. (3) and (4) into the following

$$L_\mathcal{R}^{\langle N \rangle} \coloneqq -\frac{|\mathcal{R}|}{2} \log(2\pi) + \frac{1}{2} \log \left| \mathbb{C}_{\widehat{p}}^{\langle N \rangle}(\mathbf{y}_\mathcal{R})^{-1} \right| - \frac{1}{2} \left(\mathbf{y}_\mathcal{R} - \mathbb{E}_{\widehat{p}}^{\langle N \rangle}(\mathbf{y}_\mathcal{R}) \right)^\top \mathbb{C}_{\widehat{p}}^{\langle N \rangle}(\mathbf{y}_\mathcal{R})^{-1} \left(\mathbf{y}_\mathcal{R} - \mathbb{E}_{\widehat{p}}^{\langle N \rangle}(\mathbf{y}_\mathcal{R}) \right),$$

where $|\mathcal{R}|$ represents the number of references points in $\mathcal{R}$. When $\mathcal{R}$ is arbitrarily selected and sufficiently large, one can consider $L_\mathcal{R}^{\langle N \rangle}$ as a proxy for the likelihood of training examples for LGPC. Hence, we hereinafter call $L_\mathcal{R}$ the log *pseudo*-likelihood.

As a consequence, the problem of the optimal selection $\mathcal{A}^*$ at time $N+1$ can be formulated as

$$\mathcal{A}^* \coloneqq \arg\max_{\mathcal{A} \subseteq \mathcal{Q}} L_\mathcal{R}^{\langle N+1 \,|\, \mathcal{A} \rangle} - L_\mathcal{R}^{\langle N \rangle}, \text{ subject to } |\mathcal{A}| \leq S,$$

which is unfortunately a combinatorial problem and cannot be solved efficiently. However, it is worth to highlight that the increment of pseudo-likelihood $L_\mathcal{R}^{\langle N+1 \,|\, \mathcal{A} \rangle} - L_\mathcal{R}^{\langle N \rangle}$ satisfies the *diminishing returns* behavior. That is, adding a new GP to the selection $\mathcal{A}$ increases the pseudo-likelihood more, if we have selected few GPs; and less, if we have already selected many GPs. This formalism can be formalized using the combinatorial concept of submodularity (Nemhauser, Wolsey, and Fisher 1978). Specifically, let $F(\mathcal{A}) \coloneqq L_\mathcal{R}^{\langle N+1 \,|\, \mathcal{A} \rangle} - L_\mathcal{R}^{\langle N \rangle}$, the submodular characteristic of F indicates that for all $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{Q}$ and $q \in \mathcal{Q} \setminus \mathcal{B}$ it holds that $F(\mathcal{A} \cup \{q\}) - F(\mathcal{A}) \geq F(\mathcal{B} \cup \{q\}) - F(\mathcal{B})$.

Interest of optimizing a submodular function has grown in the machine learning community recently (Krause and Guestrin 2007; Krause, Singh, and Guestrin 2008; Krause et al. 2007). In practice, heuristics such as *greedy selection* are often used. The greedy algorithm starts with the empty set, and iteratively adds the element $q^* \coloneqq \arg\max_{q \in \mathcal{Q} \setminus \mathcal{A}} F(\mathcal{A} \cup \{q\})$, until S elements have been selected. A fundamental result by (Nemhauser, Wolsey, and Fisher 1978) stated that for submodular functions, the

greedy algorithm achieves at least a constant fraction $(1 - 1/e)$ of the objective value obtained by the optimal solution. Moreover, no polynomial time algorithm can provide a better approximation guarantee unless P = NP (Feige 1998).

Algorithm 1: Greedy subset selection for LGPC.

Input : desired size of selection S (≥ 2)
Output: greedy selection $\mathcal{A}$

Initialization $\mathcal{A} \leftarrow \emptyset$, $\mathcal{J} \leftarrow \{1, \ldots, Q\}$;
$\forall j \in \mathcal{J} : \Delta_j \leftarrow F(\mathcal{A} \cup \{j\}) - F(\mathcal{A})$;
$j^* \leftarrow \arg\max_{j \in \mathcal{J}} \Delta_j$;
$\mathcal{A} \leftarrow \mathcal{A} \cup \{j^*\}$, $\mathcal{J} \leftarrow \mathcal{J} \setminus \{j^*\}$;
for $s \leftarrow 2$ **to** S **do**
  repeat
    $j^* \leftarrow \arg\max_{j \in \mathcal{J}} \Delta_j$;
    $\Delta_{j^*} \leftarrow F(\mathcal{A} \cup \{j^*\}) - F(\mathcal{A})$;
    if $\forall j \in \mathcal{J} \setminus \{j^*\} : \Delta_{j^*} > \Delta_j$ **then**
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{j^*\}$, $\mathcal{J} \leftarrow \mathcal{J} \setminus \{j^*\}$;
  until $|\mathcal{A}| = s$;

Note that evaluating $F(\mathcal{A} \cup \{q\})$ can be expensive as it is required to re-compute Eqs. (3) and (4) for all $q \in \mathcal{Q}$ in each iteration. In fact, such computation is unnecessary due to the submodularity of F (Minoux 1978). Define the *marginal benefit* of q as $\Delta_q \coloneqq F(\mathcal{A} \cup \{q\}) - F(\mathcal{A})$, the submodularity indicates that Δ_q never increases over iterations. Based on this observation, our greedy selection scheme is given in Algorithm 1. The algorithm starts with the set $\mathcal{A}$ being empty, and $\mathcal{J}$ containing the indices of all GPs. In the first iteration, the GP with maximal Δ is selected from $\mathcal{J}$ and added to $\mathcal{A}$. This is achieved by evaluating the marginal benefit for all GPs in the committee. An ordered list of $\{\Delta_j\}_{j \in \mathcal{J}}$ is maintained. From the second iteration, only the top GP in this ordered list will be evaluated. If the new marginal benefit of that GP stays on top, then we add it to $\mathcal{A}$. Otherwise, the list of marginal benefits is re-sorted and, subsequently, the new top GP is evaluated.

Theoretically, the worst-case complexity for Algorithm 1 is $S^2\mathcal{O}(L)$, where $\mathcal{O}(L)$ denotes the complexity of one pseudo-likelihood calculation. Empirically, we found on average only 3 calculations of pseudo-likelihood was required at each iteration, which gave about $3S\mathcal{O}(L)$.

Incremental Update of LGPC

Once a set of GPs is selected by Algorithm 1, the problem turns to updating these GPs for the data inclusion in an incremental fashion. Although the update of inputs $\mathbf{X}$ and outputs $\mathbf{y}$ can be done straightforwardly, the update of a covariance matrix $\mathbf{K}$ and its inverse $\mathbf{J} \coloneqq \mathbf{K}^{-1}$ is more complicated. Specifically, the effect of a new point $\mathbf{x}_{N+1}$ on $\mathbf{K}$ and $\mathbf{J}$ can be expressed as

$$\mathbf{K}^{\langle N+1 \rangle} \coloneqq \begin{bmatrix} \mathbf{K}^{\langle N \rangle} & \mathbf{u}^\top \\ \mathbf{u} & v \end{bmatrix}, \mathbf{J}^{\langle N+1 \rangle} \coloneqq \begin{bmatrix} \mathbf{J}^{\langle N \rangle} + \frac{1}{\mu}\mathbf{g}\mathbf{g}^\top & \mathbf{g} \\ \mathbf{g}^\top & \mu \end{bmatrix},$$

with $\mathbf{u} \coloneqq [k(\mathbf{x}_{N+1}, \mathbf{x}_1), \ldots, k(\mathbf{x}_{N+1}, \mathbf{x}_N)]^\top$ and $v \coloneqq k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1})$. Following the partitioned inversion matrix equations we have

$$\mathbf{g} \coloneqq -\mu \mathbf{J}^{\langle N \rangle}\mathbf{u}, \quad \mu \coloneqq \left(v - \mathbf{u}^\top \mathbf{J}^{\langle N \rangle}\mathbf{u}\right)^{-1}.$$

In practice, Cholesky factorization is often used so that $\mathbf{L}^\top\mathbf{L} \coloneqq \mathbf{K}$, which leads to a more efficient and accurate solution for frequently occuring terms such as $\mathbf{x}^\top\mathbf{K}^{-1}\mathbf{x} = \|\mathbf{L}^{-1}\mathbf{x}\|^2$ and $\log|\mathbf{K}| = 2\left(\mathbf{1}^\top \log(\mathrm{diag}\,(\mathbf{L}))\right)$. The lower triangular matrix $\mathbf{L}$ can be also updated incrementally such that

$$\mathbf{L}^{\langle N+1 \rangle} \coloneqq \begin{bmatrix} \mathbf{L}^{\langle N \rangle} & \mathbf{0} \\ \boldsymbol{l}^\top & \eta \end{bmatrix},$$

where $\boldsymbol{l}$ can be solved by $\mathbf{L}^{\langle N \rangle}\boldsymbol{l} = \mathbf{u}$, and subuently, $\eta \coloneqq \sqrt{v - \|\boldsymbol{l}\|^2}$.

Further notice that the model may deal with an endless stream of data during online learning. Thus, we have to limit the number of training examples maintained by each GP and delete old training examples when necessary. Let m be the index of training example being deleted. Construct $\mathbf{P} \coloneqq \mathbf{I} - (\boldsymbol{\delta}_m - \boldsymbol{\delta}_{T+1})(\boldsymbol{\delta}_m - \boldsymbol{\delta}_{T+1})^\top$, as a $(T+1)$-dimensional permutation matrix, where $\boldsymbol{\delta}_m$ is a $(T+1)$-dimensional zero vector with one on the m^{th} dimension. When a new point comes in, it is first appended to $\mathbf{K}^{\langle N+1 \rangle}$. Then, the deletion of the m^{th} example can be performed efficiently using $\left[\mathbf{P}\mathbf{K}^{\langle N+1 \rangle}\mathbf{P}\right]_{\Uparrow}$, where $[\cdot]_{\Uparrow}$ represents shrinking a $(T+1)$-dimensional matrix or vector to a T-dimensional one by removing the last row and column of it. Thus, the non-increasing update equation of $\mathbf{J}$ is given by

$$\mathbf{J}^{\langle N+1 \rangle} \coloneqq \left[\mathbf{P}\mathbf{J}^{\langle N+1 \rangle}\mathbf{P}\right]_{\Uparrow} - \frac{1}{r}\mathbf{s}^\top\mathbf{s},$$

where $\mathbf{s} \coloneqq [[k(\mathbf{x}_m, \mathbf{x}_1), \ldots, k(\mathbf{x}_m, \mathbf{x}_{N+1})]\mathbf{P}]_{\Uparrow}$ and $r \coloneqq k(\mathbf{x}_m, \mathbf{x}_m)$.

So far we have not said which training example should be deleted. One simple method is to choose it randomly or to remove the oldest training example over time. Alternatively, one can remove the point that yields maximal mutual information with the new point. In this work, we follow the score measure introduced by (Csató and Opper 2002). Specifically, for each point i in set $\mathcal{D}_q$ the score is given by $\xi_t \coloneqq \frac{\alpha_t}{J_{tt}^{\langle N+1 \rangle}}$, where α_t is the t^{th} element of $\mathbf{J}^{\langle N+1 \rangle}\mathbf{y}$. If a deletion is needed for the q^{th} GP, then the training example with minimal ξ will be removed from $\mathcal{D}_q$. The scores are computationally cheap as they can be calculated in linear time. Although there may exist more sophisticated selection schemes, they usually consume more computational time and hence are not considered in this work.

Predictions of Query Points

Given a query point $\mathbf{x}_*$, the predictive mean and variance can be calculated by replacing $\mathcal{R}$ in Eqs. (3) and (4) with the test set. Namely, we first compute the predictive mean and variance of each GP member in the traditional way, and then substitute them into Eqs. (3) and (4) to get the committee's prediction. One can observe that the way of combining predictions in Eq. (3) automatically assigns less weight

(through the inverse predictive variance) to those GPs that are uncertain about their predictions. The time complexity is $\mathcal{O}(QT)$ for predicting the mean and variance of a test point.

For the sake of efficiency and accuracy, it is reasonable to only invoke nearby GPs in a neighborhood of $\mathbf{x}_*$ for the prediction. The key observation is that $k(\mathbf{x}_*, \mathbf{x})$ depends merely on the constant σ if $\mathbf{x}_*$ is far away from $\mathbf{x}$. As σ is randomly initialized in LGPC, a poor prediction could be given by the q^{th} GP when $\mathbf{x}_*$ is far away from all points in the set $\mathcal{D}_q$. The search of neighboring GPs can be performed by evaluating $\frac{1}{|\mathcal{D}_q|}\sum_{\mathbf{x}\in\mathcal{D}_q} k(\mathbf{x}_*, \mathbf{x})$ for all $\{\mathcal{D}_q\}_{q=1}^{Q}$ and, subsequently, selecting those having largest values.

Experimental Results

Two sets of experiments were carried out to validate our algorithm. First, we compared the accuracy and the efficiency of LGPC with the standard GPR and state-of-the-art online regression methods. Second, we investigated several factors that affect the performance of LGPC in order to gain more insights of it. An application to the mouse-trajectory prediction is demonstrated at the end.

The experiments were conducted on six large data sets downloaded from the Internet. On each data set, we linearly rescaled into the range of $[-1, 1]$ and randomly held out half of the data for training; the remaining was used as a test set. Each experiment was repeated 10 times.

Five baseline methods were employed for comparison. They were standard GP regression (GPR), sparse GP using pseudo-inputs (SGPP) (Snelson and Ghahramani 2006), local GP regression (LoGP) (Nguyen-tuong and Peters 2008), Bayesian committee machine (BCM) (Tresp 2000) and sparse online GP regression (SOGP) (Csató and Opper 2002). URLs of these MATLAB implementations are listed in the references. Note that GPR and SGPP are offline algorithms, they are presented here to show the performance when the whole training data is given beforehand. A Gaussian kernel function with white noise was used as the covariance function for all methods, which was obtained by setting $\mathbf{W}$ in the covariance function as the identity matrix. The maximum number of inducing inputs in SGPP and SOGP was 50. The threshold for creating a new local model in LoGP was 0.5. The size of the committee was 20 for BCM and LGPC, in which each GP maintained at most 100 training examples. For BCM and LGPC, the first 20 training examples were sequentially assigned to each member for initialization. The reference set of LGPC had a size of 3, which consisted of the current training example and two examples randomly sampled from the aforetime data. The number of selected GPs in LGPC for data inclusion was 5 (i.e. $S \coloneqq 5$ in Algorithm 1). Moreover, three variations of BCM were employed in the experiment. BCM_o denotes that each time only one GP is randomly selected for data inclusion; BCM_s represents randomly selecting a subset with a size of 5, which corresponds to a randomized counterpart of LGPC; and BCM_a represents selecting all GPs. For predicting test inputs, all GPs in the committee of BCM and LGPC were invoked. The hyperparameters were randomly initialized for all methods. In particular, we drew a set of hyperparameters from $\text{Uniform}(0, 10)$ for initializing BCM and LGPC. The conjugate gradient method (in Netlab toolbox (Nabney 2004)) was employed to optimize the hyperparameters for GPR, SGPP, SOGP and LoGP. For SOGP, an EM algorithm with 10 cycles was built for jointly optimizing the posterior process and the hyperparameters.

Comparison of Predictive Accuracy

The comparison of predictive accuracy between different GP methods is summarized in Table 1(a), where the root mean square error was used as the evaluation metric. This causes the trivial method of guessing the mean to have a SMSE of approximately 1 and a MSLL of zero. Small value of SMSE or negative value of MSLL indicates a better method. It is evident from the results that, LGPC gave a considerably lower test error than LoGP, BCM and SGPP. In particular, LGPC showed a significant improvement over all BCM variants on the majority data sets, which indicates the effectiveness of our subset selection strategy. Empirically, we found that good performance could have been achieved after learning from first few thousands examples. After that, the accuracy of LGPC did not change significantly. On `delta`, `cpuact` and `houses`, the performance of LGPC was comparable to SOGP. In fact, the performance of LGPC can be further improved by allowing each GP to maintain more training examples, as it is shown in the third experiment.

Comparison of Computation Speed

The comparison of training and prediction speed is shown in Fig. 1. Different online methods were trained (tested) on `houses` data set with increasing training (testing) examples, i.e. 500, $1,000$, $2,000$, $4,000$ and $8,000$ data points, respectively. As can be seen in Fig. 1, in both training and prediction phrases, LGPC showed a substantial reduction of time comparing to GPR and LoGP. In particular, LGPC took less time for learning $8,000$ points (210s) than SGPP (240s) and SOGP (340s). On the other hand, SGPP and SOGP were found to be extremely efficient in the prediction, as their time cost only increased at a very low pace with respect to the number of test points. This is due to the fact that the prediction of SGPP and SOGP involves only a small covariance matrix based on 50 inducing inputs, whereas LGPC invokes all GPs in the committee for prediction. Nonetheless, it is possible to speed up LGPC by invoking only the nearest GP for the prediction as aforementioned. In short, LGPC is a more appropriate choice than SOGP in a real-time application which requires fast training.

One may notice that LGPC took more training time than BCM_s and its speed advantages over BCM_a was not dramatic, which is slightly disappointing. This is attributed to the fact that, although LGPC saves the computational resources by limiting the number of GPs for data inclusion, it spends extra computational time on selecting a near-optimal subset with Algorithm 1. The prediction complexity of LGPC, BCM_o, BCM_s and BCM_a was virtually same. Nonetheless, it should be noted that LGPC outperformed all BCM variants significantly on `houses` in terms of predictive accuracy as detailed in Table 1(a), which makes LGPC overall more preferable than BCM.

Table 1: The root mean square error $\pm$ the standard deviation on different test sets. URLs of these data sets can be found in the references. `delta`: $7,129 \times 6$, `bank`: $8,192 \times 8$, `cpuact`: $8,192 \times 12$, `elevator`: $8,752 \times 17$, `houses`: $20,640 \times 8$, `sarcos`: $44,484 \times 21$. Results were averaged over ten runs. Small value for better methods.

(a) LGPC versus baseline methods.

Method	delt	bank	cpua	elev	hous	sarc
LGPC	0.041±0.001	0.084±0.011	0.072±0.010	0.053±0.005	0.157±0.004	0.032±0.002
LoGP	0.065±0.009	0.188±0.069	0.219±0.041	0.107±0.009	0.232±0.028	0.089±0.001
BCM_o	0.044±0.002	0.113±0.012	0.115±0.011	0.066±0.004	0.164±0.008	0.070±0.003
BCM_s	0.043±0.001	0.108±0.012	0.114±0.016	0.069±0.005	0.180±0.008	0.073±0.003
BCM_a	0.045±0.001	0.122±0.017	0.119±0.009	0.083±0.003	0.203±0.007	0.077±0.004
SOGP	0.040±0.001	0.047±0.001	0.074±0.007	0.038±0.001	0.143±0.001	0.023±0.001
GPR	0.039±0.001	0.041±0.001	0.030±0.001	0.031±0.001	0.115±0.002	0.016±0.002
SGPP	0.045±0.013	0.061±0.045	0.079±0.057	0.065±0.035	0.161±0.046	0.095±0.059

(b) Predictive error of LGPC w.r.t. different size of the committee.

Q	delt	bank	cpua	elev	hous	sarc
5	0.043±0.001	0.093±0.029	0.087±0.030	0.065±0.016	0.171±0.005	0.034±0.006
10	0.042±0.001	0.102±0.016	0.079±0.018	0.056±0.010	0.167±0.007	0.033±0.004
15	**0.041**±0.001	0.085±0.012	0.075±0.013	0.057±0.008	0.161±0.003	0.034±0.004
20	0.041±0.001	0.084±0.011	**0.072**±0.010	0.053±0.007	0.157±0.004	0.033±0.003
25	0.041±0.001	**0.076**±0.006	0.080±0.009	0.053±0.003	0.156±0.003	**0.032**±0.003
30	0.041±0.001	0.085±0.012	0.095±0.009	**0.052**±0.004	**0.155**±0.004	0.032±0.003

(c) Predictive error of LGPC w.r.t. different maximum number of examples.

T	delt	bank	cpua	elev	hous	sarc
50	0.042±0.001	0.114±0.012	0.098±0.016	0.072±0.014	0.172±0.007	0.040±0.004
100	0.041±0.001	0.084±0.011	0.072±0.010	0.053±0.007	0.157±0.004	0.032±0.002
150	**0.040**±0.001	0.068±0.011	0.057±0.005	0.040±0.003	0.152±0.003	0.028±0.003
200	0.040±0.001	**0.054**±0.004	**0.050**±0.007	**0.039**±0.002	**0.146**±0.002	**0.022**±0.002

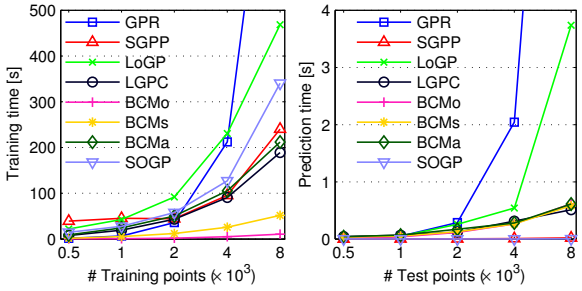


Figure 1: Time cost in second (averaged over 10 runs) required for training and predicting, respectively. In each run, a training set and a test set were randomly sampled from `houses` data set and the time cost was measured respectively. The training and prediction time of $8,000$ data points required for GPR was 1100s and 15s, respectively. Note that prediction time of LGPC can be reduced to 0.02s if only the nearest GP is invoked for predicting a test point.

Exploration of Model Parameters

In order for LGPC to be a practical tool in real-world applications, it is necessary to make decisions about the details of its specification. Fortunately, there are not many free parameters in LGPC, as all hyperparameters are randomly initialized and are fixed during the learning process[1]. Our exploration focused on three parameters that mainly govern the performance of LGPC. Namely, the committee size Q, the maximum number of maintained training points T and the size S of the selected subset for data inclusion.

The performance of LGPC with respect to different sizes of the committee is summarized in Table 1(b), where $S := 5$ and $T := 100$. On `delta`, `bank` and `cpuact`, the predictive accuracy reached its peak when the size of the committee was around 20. After that, the performance dropped slightly. On larger data sets such as `houses` and `sarcos` the test error had a steady decline as Q increasing. This indicates that it suffices to employ a small committee for learning a small amount of data. For a large data set increasing the size of the committee provides more capability to account for the complex pattern, which generally leads to higher predictive accuracy.

Table 1(c) summarizes the test results of LGPC with respect to different settings of T, where $S := 5$ and $Q := 20$. As expected, one can improve the predictive accuracy sig-

[1]We did try few heuristics for initializing the hyperparameters, such as initializing 20 Gaussian kernels with different widths (e.g. $2^{-9}, \ldots, 2^{9}, 2^{10}$) and the same noise level; sampling from a Gamma distribution; implementing a weak prior by optimizing on a small validation set. However, such attempts did not yield better predictive accuracy than the random initialization.

nificantly by allowing each GP to maintain more training examples. Moreover, when $T := 200$ it was observed that the high accuracy was achieved much earlier than $T := 50$; and the performance was often more robust afterwards.

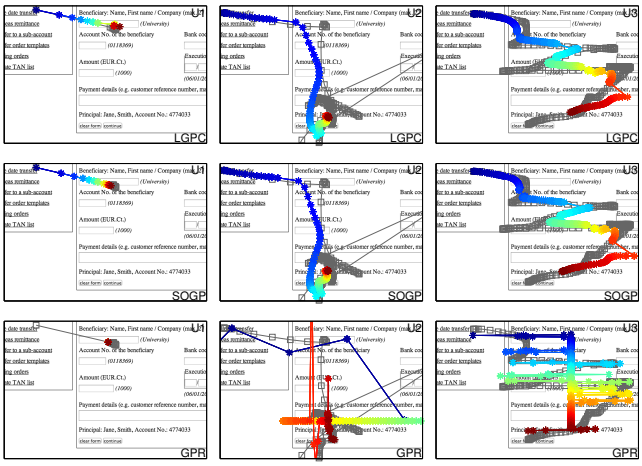
Finally, to study the performance with respect to different sizes of the selected subset, we fixed $T := 100, Q := 20$ and trained LGPC with $S := 2, 4, 6, 8$ and 10, respectively. It was found that on `delta` and `sarcos` the predictive accuracy was not sensitive to the size of selected subset. On `cpuact`, `elevator` and `houses`, selecting more than two GPs slightly degraded the performance. The optimal size for `bank` was 6. In short, it seems that the optimal value of S varies with data sets.

Mouse-Trajectory Prediction

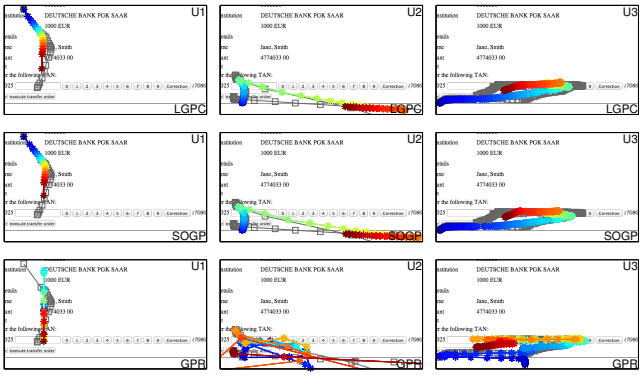
We applied LGPC for learning mouse-trajectory of different users in an Internet banking scenario. To collect the data, we developed a website for simulating an environment, in which participants were asked to transfer funds to a dummy account. A complete procedure was composed of five interfaces, i.e. login, account overview, transaction details, TAN authentication and confirmation. Ten participants were involved and each with three trials; the input information was *same* for all trials. A Javascript code was developed for tracking mouse coordinates on every `onmousemove` event. The trajectories of the first two trials (ca. 2700 points/user) were used for training models. The goal was to predict the trajectory of the last trial (ca. 1000 points/user).

The predicted trajectories of three users using LGPC, SOGP and offline GPR are visualized in Fig. 2. It was observed that users behaved differently even when they were performing the same task. For instance, the first user used the tab key moves the cursor and entered the TAN code with the numpad, resulting a short and simple mouse-trajectory. On the other hand, the third user entered the TAN code using a virtual keyboard on the web page, which made the trajectory sway horizontally. By learning from the first two trials, reasonable predictions of the third trial were obtained from all methods. However, when the interface contained many elements and the trajectory became more complicated, the offline GPR gave noisy predictions as depicted in Fig. 2(a). It can be seen that LGPC performed as good as the state-of-the-art SOGP in learning users' trajectories.

With a new training point arriving about every 10ms (less than one minute of running time will result in thousands of data points), LGPC is a more preferable method due to its fast learning speed. Efficient trajectory prediction would be beneficial in security applications, such as distinguishing between individuals and early warning of identity theft.



(a) Mouse-trajectories on "transaction details"



(b) Mouse-trajectories on "TAN authentication"

Figure 2: The predictions of three users' mouse-trajectories on two interfaces. Each column represents a user. The gray curve with □ denotes a user's trajectory in the third trial. The model's prediction is illustrated by the color curve with ⋆, whose head is blue and tail is red.

Conclusions

The efficiency problem of Gaussian process becomes a primary issue when it is applied to real-time online applications. This work has proposed a novel method for reducing the computational demand of GP regression. It consists of multiple GPs where each maintains only a small set of training examples. Each time a subset of GPs is selected for including newly arrived training examples. The selection is performed by optimizing a submodular function. Unlike previous work, our model removes the need for parameter-fitting and requires little tuning efforts. An improvement of accuracy and efficiency over existing online GP methods has been demonstrated in the experiment. In particular, as a modified version of Bayesian committee machine, we have showed that updating a chosen subset of GPs is more effective than updating the whole committee, which leads to better predictive accuracy. As demonstrated in the task of mouse-trajectory prediction, LGPC can be applied to a wide range of real-time applications, such as learning motor dynamics and inferring temporal dynamics of mental phenomena.

An important question for future studies is to determine the optimal size of the committee. One possible way is to expand or shrink the committee size during the online learning. In addition, more effective strategies for initializing the hyperparameters remain to be determined. Furthermore, it should be interesting to infringe the independence between GP members for a further improvement on the accuracy.

Acknowledgments

This work is funded by the HIVE project (FKZ: 16BY1200D) of the German Federal Ministry of Education and Research. We would like to thank Nan Li and Ping Luo for their comments on an earlier draft, Yurong Tao for help with experiments, and anonymous reviewers for helpful suggestions.

References

Abrams, R., and Balota, D. 1991. Mental chronometry: Beyond reaction time. *Psychological Science* 2(3):153–157.

Chen, T., and Ren, J. 2009. Bagging for gaussian process regression. *Neurocomputing* 72(7):1605–1610.

Csató, L., and Opper, M. 2002. Sparse on-line gaussian processes. *Neural Computation* 14(3):641–668.

`delta` data set. http://www.dcc.fc.up.pt/~ltorgo/Regression/.

`bank` data set. http://www.cs.toronto.edu/~delve/.

`cpuact` data set. http://www.cs.toronto.edu/~delve/.

`elevator` data set. http://www.dcc.fc.up.pt/~ltorgo/Regression/.

`houses` data set. http://lib.stat.cmu.edu/datasets/.

`sarcos` data set. http://www.gaussianprocess.org/gpml/data/.

Dale, R.; Kehoe, C.; and Spivey, M. 2007. Graded motor responses in the time course of categorizing atypical exemplars. *Memory & Cognition* 35(1):15–28.

Feige, U. 1998. A threshold of ln n for approximating set cover. *Journal of the ACM* 45(4):634–652.

Freeman, J., and Ambady, N. 2009. Motions of the hand expose the partial and parallel activation of stereotypes. *Psychological Science* 20(10):1183–1188.

Freeman, J.; Pauker, K.; Apfelbaum, E.; and Ambady, N. 2010. Continuous dynamics in the real-time perception of race. *Journal of Experimental Social Psychology* 46(1):179–185.

GPR and SGPP MATLAB implementation. http://www.cs.man.ac.uk/~neill/gp/.

LoGP MATLAB implementation. http://www.ias.informatik.tu-darmstadt.de/Member/DuyNguyen-Tuong.

SOGP MATLAB implementation. http://www.cs.ubbcluj.ro/~csatol/.

Keerthi, S., and Chu, W. 2006. A matching pursuit approach to sparse gaussian process regression. In *Advances in Neural Information Processing Systems*, volume 18, 643. MIT Press.

Kimeldorf, G., and Wahba, G. 1971. Some results on tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications* 33(1):82–95.

Krause, A., and Guestrin, C. 2007. Near-optimal observation selection using submodular functions. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, 1650. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Krause, A.; Mcmahan, H.; Guestrin, C.; and Gupta, A. 2007. Selecting observations against adversarial objectives. In *Advances in Neural Information Processing Systems*. MIT Press.

Krause, A.; Singh, A.; and Guestrin, C. 2008. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research* 9:235–284.

Minoux, M. 1978. Accelerated greedy algorithms for maximizing submodular set functions. *Optimization Techniques* 234–243.

Nabney, I. T. 2004. *NETLAB: algorithms for pattern recognition*. Springer.

Nemhauser, G.; Wolsey, L.; and Fisher, M. 1978. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming* 14(1):265–294.

Nguyen-tuong, D., and Peters, J. 2008. Local gaussian process regression for real time online model learning and control. In *Advances in Neural Information Processing Systems*. MIT Press.

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

Rasmussen, C., and Williams, C. 2006. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, MA.

Seeger, M.; Williams, C.; and Lawrence, N. 2003. Fast forward selection to speed up sparse gaussian process regression. In *Workshop on AI and Statistics*, volume 9, 2003.

Smola, A. J., and Bartlett, P. 2001. Sparse greedy gaussian process regression. In *Advances in Neural Information Processing Systems*, 619–625. MIT Press.

Snelson, E., and Ghahramani, Z. 2006. Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, 1257–1264. MIT press.

Song, J., and Nakayama, K. 2008. Target selection in visual search as revealed by movement trajectories. *Vision research* 48(7):853–861.

Spivey, M.; Grosjean, M.; and Knoblich, G. 2005. Continuous attraction toward phonological competitors. *Proceedings of the National Academy of Sciences of the United States of America* 102(29):10393–10398.

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