Predictive Approaches For Sparse Gaussian Process Regression

S. Sundararajan SSRAJAN@YAHOO-INC.COM
Yahoo! India R & D
Bangalore, India

S. Sathiya Keerthi SELVARAK@YAHOO-INC.COM
Yahoo! Research Labs
Santa Clara, CA, USA

Shirish Shevade SHIRISH@CSA.IISC.ERNET.IN
Computer Science and Automation
Indian Institute of Science
Bangalore, India

Editor:

Abstract

We propose and study algorithms using leave-one-out cross validation (LOO-CV) based predictive measures namely, LOO-CV error (LOO-CVE), Geisser's surrogate Predictive Probability (GPP) and Predictive Mean Squared Error (GPE) to select basis vectors for building sparse Gaussian process regression models. While the LOO-CVE measure uses only predictive mean information, the GPP and GPE measures use predictive variance information as well. All of these three LOO-CV based predictive measures can be used to find the number of basis vectors in the model automatically. The computational complexities of these measures are same as that of marginal likelihood and approximate log posterior probability maximization approaches. We also give an efficient cache implementation for all the algorithms which gives similar or better generalization performance with lesser number of basis vectors. The experimental results on several real world benchmark datasets show better or comparable generalization performance over existing approaches.

Keywords: Sparse models, Gaussian process regression, Cross validation

1. Introduction

Gaussian process (GP) regression models are flexible, powerful, and easy to implement probabilistic models that can be used to solve regression problems in many areas of application (Rasmussen and Williams, 2006). While GPs exhibit state of the art performance, they suffer from a high computational cost of $O(n^3)$ for learning from n samples; further, predictive mean and variance computation on each sample cost $O(n)$ and $O(n^2)$ respectively (Candela and Rasmussen, 2005a). The high computational cost limits direct implementation of GPs to problems with few thousands of samples. There have been several approaches proposed to address this concern and build sparse approximate GP models. Gibbs and MacKay (1997) and Williams and Seeger (2001) used matrix approximations to reduce the computational cost. Tresp (2000) introduced Bayesian committee machine. Csato and Op-

per (2002) developed an online algorithm to maintain a sparse representation of the GP model. Smola and Bartlett (2001) proposed a forward basis vector selection method that maximizes approximate log posterior probability for building sparse GP models. Other selection methods include entropy and information gain based score optimization (Seeger, 2003; Lawrence et al., 2003; Seeger et al., 2003), marginal likelihood maximization (Candela and Rasmussen, 2005a) and kernel matching pursuit (Keerthi and Chu, 2005). Snelson and Ghahramani (2006) proposed to use pseudo-inputs to build sparse GP models. It is also interesting to note that relevance vector machine and subset of regressors can be thought of as sparse linear approximations to GPs (Tipping, 2001; Wahba et al., 1999).

In general, all sparse GP methods aim at selecting an informative set of basis vectors for the predictive model. Due to memory and computational constraints, the number of basis vectors in the model is usually limited by a user defined parameter d_{max}. With $n \gg d_{max}$, the sparse GP models have reduced training computational complexity of $O(nd_{max}^2)$; the reduced prediction complexity is $O(d_{max})$ and $O(d_{max}^2)$ to compute the predictive mean and variance respectively. Considering the various sparse approximations that have been studied, Candela and Rasmussen (2005b) brought in a unifying view of sparse approximate GP regression that includes all existing proper probabilistic sparse approximations.

We note that none of the approaches mentioned above estimate predictive ability of the model directly. CV based predictive measures have been successfully used in model selection in various contexts (Cawley and Talbot, 2004; Geisser, 1975; Geisser and Eddy, 1979; Stone, 1974; Sundararajan and Keerthi, 2001; Wahba et al., 1999). Hong, Chen, and Harris (2006) combined orthogonal forward selection and a LOO-CV based measure to design sparse linear kernel classifiers with Gaussian kernels. Cawley and Talbot (2004) designed a LOO-CVE based sparse least squares support vector machine.

The main contributions of this article are as follows. Borrowing ideas from the LOO-CV and sparse GP literature, we propose and study algorithms using the LOO-CV based predictive measures to select basis vectors for building sparse GP regression (SGPR) models. In this context, we consider the LOO-CV error (LOO-CVE), Geisser's surrogate Predictive Probability (GPP) and Predictive Mean Squared Error (GPE) measures. These measures are quite generic and mesh well with the unifying view on SGPR presented by Candela and Rasmussen (2005b). The importance of these measures lies in the fact that they estimate the predictive ability of the model and, the GPP and GPE measures make use of the predictive variance information as well. The computational complexity is same as that of the marginal likelihood (ML) and approximate log posterior probability maximization approaches. Like the ML approach, the use of predictive measures has the advantage that the number of basis vectors in the model can be automatically determined. We also give an efficient cache implementation for all the algorithms. This implementation allows one to select the basis vectors from a larger set of candidate basis vectors and gives similar or better generalization performance with lesser number of basis vectors. In our algorithm, hyperparameters are selected by maximizing the marginal likelihood instead of minimizing one of the predictive measures. This is done to achieve computational efficiency in the presence of many hyperparameters and is found to be effective. The experimental results on several benchmark datasets show better or comparable generalization performance over the existing approaches.

This paper is organized as follows. We give a brief introduction to the SGPR and related work in Section 2. In Section 3, we define the LOO-CV based predictive measures and illustrate their use for a given predictive distribution. Various efficient implementation aspects and the main algorithm are given in Section 4. Section 5 discusses the behaviors of the proposed algorithms with the predictive measures. In Section 6, we present the experimental results and conclude with Section 7.

2. Sparse GP Regression

In regression problems, we are given a training data set composed of n input-output pairs $(\mathbf{x}_i, y_i)$ where $\mathbf{x}_i \in R^D$, $y_i \in R$, $i \in \tilde{I}$ and $\tilde{I} = \{1, 2, \ldots, n\}$. The true function value at $\mathbf{x}_i$ is represented as a latent variable $f(\mathbf{x}_i)$ and the target $y_i (= f(\mathbf{x}_i) + \epsilon_i)$ is a noisy measurement of $f(\mathbf{x}_i)$. The goal is to compute the predictive distribution of the function values f_* (or noisy y_*) at test location $\mathbf{x}_*$. In standard GPs for regression (Rasmussen and Williams, 2006), the latent variables $f(\mathbf{x}_i)$ are modelled as random variables in a zero mean GP indexed by $\{\mathbf{x}_i\}$. The prior distribution of $\{\mathbf{f}(\mathbf{X}_n)\}$ is a zero mean multivariate joint Gaussian, denoted as $p(\mathbf{f}) = N(\mathbf{0}, \mathbf{K}_{\mathbf{f},\mathbf{f}})$, where $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)]^T$, $\mathbf{X}_n = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ and $\mathbf{K}_{\mathbf{f},\mathbf{f}}$ is the $n \times n$ covariance matrix whose $(i,j)^{th}$ element is $k(\mathbf{x}_i, \mathbf{x}_j)$ and is often denoted as $\mathbf{K}_{i,j}$. One of the most commonly used covariance function is the squared exponential covariance function given by: $cov(f(\mathbf{x}_i), f(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j) = \beta_0 \, exp(-\frac{1}{2} \sum_{k=1}^{D} \frac{(x_{i,k} - x_{j,k})^2}{\beta_k})$. Here, β_0 represents signal variance and the β_k's represent width parameters across different input dimensions. These parameters are also known as *automatic relevance determination* (ARD) hyperparameters. We call this covariance function as the ARD Gaussian kernel function. Now, given the prior the likelihood is a model of additive measurement noise ϵ_i $i \in \tilde{I}$, which is modelled as $p(\mathbf{y}|\mathbf{f}) = N(\mathbf{f}, \sigma^2 \mathbf{I})$, where $\mathbf{y} = [y_1, \ldots, y_n]^T$ and σ^2 is the noise variance. These models with the hyperparameters $\boldsymbol{\theta} = [\beta_0, \beta_1, \ldots, \beta_D, \sigma^2]$ characterize the GP model. These hyperparameters can be either estimated from the dataset, or can be integrated out using Markov Chain Monte Carlo methods in full Bayesian solution. Using standard Bayesian rule, inference is made for $\mathbf{x}_*$ from the posterior predictive distribution: $p(f_*|\mathbf{y}) = N(\mathbf{K}_{*,\mathbf{f}}(\mathbf{K}_{\mathbf{f},\mathbf{f}} + \sigma^2 \mathbf{I})^{-1}\mathbf{y}, \mathbf{K}_{*,*} - \mathbf{K}_{*,f}(\mathbf{K}_{\mathbf{f},\mathbf{f}} + \sigma^2 \mathbf{I})^{-1}\mathbf{K}_{\mathbf{f},*})$.

Candela and Rasmussen (2005b) noted that by approximating the joint prior $p(\mathbf{f}, f_*)$ to $q(\mathbf{f}, f_*) = \int q(f_*|\mathbf{u}) q(\mathbf{f}|\mathbf{u}) p(\mathbf{u}) d\mathbf{u}$ with *additional assumptions* about the two approximate *inducing* conditionals $q(\mathbf{f}|\mathbf{u})$ and $q(f_*|\mathbf{u})$ almost all probabilistic sparse GP approximations are obtained with exact inference. Here $\mathbf{u}$ denotes an additional set of m latent variables $\mathbf{u} = [u_1, \ldots, u_m]^T$ which are called *inducing variables*; these latent variables are values of GP like $\mathbf{f}$, corresponding to a set of input locations $\mathbf{X}_\mathbf{u}$, referred to as *inducing inputs* and are commonly known as basis vectors or active set. In the context of predictive approaches we are mainly interested in the resultant posterior predictive distributions corresponding to these approximations and we give some of them that are useful for our discussion below.

A likelihood approximation proposed by Snelson and Ghahramani (2006) which is termed as the *Fully Independent Training Conditional* (FITC) approximation (Candela and Rasmussen, 2005b) results in the posterior predictive distribution:

$$q_{FITC}(f_*|\mathbf{y}, \mathbf{X}_\mathbf{u}, \boldsymbol{\theta}) = N(\hat{f}(x_*), \sigma_*^2) \tag{1}$$

where the predictive mean and variance are given by $\hat{f}(x_*) = \mathbf{K}_{*,\mathbf{u}}\boldsymbol{\alpha}$ and $\sigma_*^2 = \mathbf{K}_{*,*} - \mathbf{Q}_{*,*} + \mathbf{K}_{*,\mathbf{u}}\boldsymbol{\Sigma}\mathbf{K}_{\mathbf{u},*}$. Here $\boldsymbol{\alpha} = \boldsymbol{\Sigma}\mathbf{K}_{\mathbf{u},\mathbf{f}}\boldsymbol{\Lambda}^{-1}\mathbf{y}$ and $\boldsymbol{\Sigma} = (\mathbf{K}_{\mathbf{u},\mathbf{f}}\boldsymbol{\Lambda}^{-1}\mathbf{K}_{\mathbf{f},\mathbf{u}} + K_{\mathbf{u},\mathbf{u}})^{-1}$ and $\boldsymbol{\Lambda} = diag[\mathbf{K}_{\mathbf{f},\mathbf{f}} - \mathbf{Q}_{\mathbf{f},\mathbf{f}} + \sigma^2\mathbf{I}]$; further $\mathbf{Q}_{*,*}$ and $\mathbf{Q}_{\mathbf{f},\mathbf{f}}$ are defined with the convention $\mathbf{Q}_{a,b} = \mathbf{K}_{a,\mathbf{u}}\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u},b}$. We note that the *Deterministic Training Conditional* (DTC) approximation corresponding to the likelihood approximation proposed by Seeger, Williams, and Lawrence (2003) and the subset of regressors (SoR) approximation (Silverman, 1985; Wahba et al., 1999) result in similar expressions except that in the cases of DTC and SoR approximations we have $\boldsymbol{\Lambda} = \sigma^2\mathbf{I}$; further in the case of SoR approximation the predictive variance is only $\mathbf{K}_{*,\mathbf{u}}\boldsymbol{\Sigma}\mathbf{K}_{\mathbf{u},*}$. Note that the posterior predictive distribution is dependent on the *inducing inputs* $\mathbf{X}_\mathbf{u}$; therefore, *the choice of* $\mathbf{X}_\mathbf{u}$ *is very important in achieving good generalization performance.* Similar to the posterior predictive distributions, the marginal likelihood can be obtained for the different effective priors and its negative logarithmic form is given by: $q(\mathbf{y}|\mathbf{X}_\mathbf{u},\boldsymbol{\theta}) = \frac{1}{2}\mathbf{y}^T(\mathbf{Q}_{\mathbf{f},\mathbf{f}} + \boldsymbol{\Lambda})^{-1}\mathbf{y} + \frac{1}{2}log|\mathbf{Q}_{\mathbf{f},\mathbf{f}} + \boldsymbol{\Lambda}| + \frac{n}{2}log(2\pi)$. As earlier in the cases of SoR and DTC approximations $\boldsymbol{\Lambda} = \sigma^2\mathbf{I}$.

In sparse GPs the basis vectors are chosen from the training instances or test instances in a transduction setup (Schwaighofer and Tresp, 2003) or as pseudo-inputs in a continuous optimization setup (Snelson and Ghahramani, 2006). To reduce computational burden, the basis vectors are selected in a greedy fashion with suitably defined measure. For example, Smola and Bartlett (2001) proposed to select the basis vector that minimizes the negative logarithm of an approximate posterior probability (NLPP); Candela and Rasmussen (2005a) suggested to minimize negative logarithm of marginal likelihood (NLML) with the DTC approximation and made comparison with the NLPP algorithm. See also references given in Section 1 for other approaches. None of these approaches measure the predictive ability of the model directly. The GPP and GPE predictive measures take the predictive variance information into account. Though entropy and information gain score criteria use predictive variance (Seeger et al., 2003; Lawrence et al., 2003), they make approximations which result in $O(1)$ score computation per sample. Such approximations may affect the generalization performance for a given number of basis vectors, as was observed in Keerthi and Chu (2005). Further, they may result in more number of basis vectors for a given generalization performance. In our approaches, we do not make any approximation in the computation of predictive measures and this comes with additional cost. However, the computational complexity is same as that of the ML and approximate log posterior probability maximization approaches.

We note that the LOO-CV based predictive measures are quite generic in the sense that they can be used to select the basis vectors irrespective of whether they are selected from the training instances and/or the test instances in the transduction setup or optimized as pseudo-inputs in the continuous optimization setup mentioned earlier. Nevertheless to illustrate our approaches here we restrict our discussion to the selection of basis vectors from the training inputs.

In the following sections, we give the LOO-CV based predictive measures and the algorithms that we propose to build sparse GP regression models. For the purpose of comparison we restrict to NLML and NLPP minimization algorithms only. This is because they have similar computational complexity as that of the proposed algorithms.

3. LOO-CV based predictive measures

Let $q(y_i|\mathbf{y}_{-i}, \mathbf{X_u}, \boldsymbol{\theta})$ be the Gaussian posterior predictive distribution with mean $\hat{f}_{-i}(\mathbf{x}_i)$ and variance $\sigma^2_{-i}(\mathbf{x}_i)$. Here, y_i denotes the i^{th} noisy measurement of $f(\mathbf{x}_i)$ and $\mathbf{y}_{-i}$ denote the training set outputs with i^{th} sample removed. Note that we use y_i for both the variable and observed noisy sample leaving the context to explain its usage. Then, the LOO-CV based predictive measures are defined as follows.

3.1 LOO-CV Error

The LOO-CV error (LOO-CVE) is defined as the average squared error of the predictive mean of the i^{th} sample with the predictive distribution $q(y_i|\mathbf{y}_{-i}, \mathbf{X_u}, \boldsymbol{\theta})$ obtained from leaving out i^{th} sample. To be specific,

$$LOO - CVE(\mathbf{X_u}, \boldsymbol{\theta}) \;=\; \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}_{-i}(\mathbf{x}_i))^2 \tag{2}$$

Note that though $\hat{f}_{-i}(\mathbf{x}_i)$ is dependent on the inducing inputs $\mathbf{X_u}$ and hyperparameters $\boldsymbol{\theta}$, we have suppressed them for notational convenience.

3.2 Geisser's surrogate measures

The negative logarithm of Geisser's surrogate predictive probability (NLGPP) measure is defined (Geisser, 1975; Sundararajan and Keerthi, 2001) as

$$NLGPP(\mathbf{X_u}, \boldsymbol{\theta}) \;=\; -\frac{1}{n}\sum_{i=1}^{n}\log\, q(y_i|\mathbf{y}_{-i}, \mathbf{X_u}, \boldsymbol{\theta})$$

and it can be written within some constant as

$$NLGPP(\mathbf{X_u}, \boldsymbol{\theta}) \;=\; \frac{1}{n}\sum_{i=1}^{n}\frac{(y_i - \hat{f}_{-i}(\mathbf{x}_i))^2}{\sigma^2_{-i}(\mathbf{x}_i)} \;+\; log(\sigma^2_{-i}(\mathbf{x}_i)) \tag{3}$$

On comparing (2) and (3), we see that the key difference is while the LOO-CVE takes only the predictive mean into account, NLGPP takes the predictive variance also into account. Geisser's surrogate predictive mean squared error (GPE) is defined (Sundararajan and Keerthi, 2001) as $GPE(\mathbf{X_u}, \boldsymbol{\theta}) \;=\; \frac{1}{n}\sum_{i=1}^{n} E((y_i - t_i)^2)$ where y_i is the observed output and t_i is a random variable and the expectation operation is defined with respect to $q(t_i|\mathbf{y}_{-i}, \mathbf{X_u}, \boldsymbol{\theta})$. Then the GPE measure can be written as

$$GPE(\mathbf{X_u}, \boldsymbol{\theta}) \;=\; \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}_{-i}(\mathbf{x}_i))^2 + \sigma^2_{-i}(\mathbf{x}_i) \tag{4}$$

Note that the first term is nothing but LOO-CVE and the second term comes from uncertainty associated with the predictions. On comparing (3) and (4), we see that the predictive variance in GPE is additive in nature compared to the NLGPP measure where the predictive variance interacts in a nonlinear fashion.

3.3 Predictive measures with training inputs

The choice of training inputs as the inducing inputs is motivated by the fact that they are often representative of the input distribution. While using the subset of training inputs as the inducing inputs, a subtle point is that we *may not* be strictly leaving the $(\mathbf{X_u}, \mathbf{y_u})$ pairs in the LOO-CV measures given above; this is because the summation is defined over all the samples. However, this is not a limitation since we are only using $\mathbf{X_u}$ for the inducing inputs and *not using* $\mathbf{y_u}$ while predicting those outputs; also, we can afford to leave those samples in the summation as we are working with a large number of samples and $n \gg d_{max}$.

Let us consider the *FITC* posterior predictive distribution (1). Then, the LOO predictive mean $\hat{f}_{-i}(\mathbf{x}_i)$ and variance $\sigma^2_{-i}(\mathbf{x}_i)$ are given by: $\hat{f}_{-i}(\mathbf{x}_i) = \mathbf{K}_{i,\mathbf{u}}\mathbf{\Sigma}_{-i}\mathbf{K}_{\mathbf{u},-i}\mathbf{\Lambda}_{-i}^{-1}\mathbf{y}_{-i}$ and $\sigma^2_{-i}(\mathbf{x}_i) = \mathbf{K}_{i,i} - \mathbf{Q}_{i,i} + \mathbf{K}_{i,\mathbf{u}}\mathbf{\Sigma}_{-i}\mathbf{K}_{\mathbf{u},i}$ where $\mathbf{\Sigma}_{-i} = (\mathbf{K}_{\mathbf{u},-i}\mathbf{\Lambda}_{-i}^{-1}\mathbf{K}_{-i,\mathbf{u}} + \mathbf{K}_{\mathbf{u},\mathbf{u}})^{-1}$ and $\mathbf{Q}_{i,i} = \mathbf{K}_{i,\mathbf{u}}\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u},i}$. Here, $\mathbf{K}_{\mathbf{u},-i}$ is nothing but $\mathbf{K}_{\mathbf{u},\mathbf{f}}$ with the i^{th} column removed. Similarly, $\mathbf{\Lambda}_{-i}$ denotes $\mathbf{\Lambda}$ with i^{th} column and row removed. Note that in the case of noisy sample, the predictive variance contains σ^2 additionally. In the following sections we consider (1) with $\mathbf{\Lambda} = \sigma^2\mathbf{I}$ (that is, the DTC approximation); note that the FITC predictive distribution (1) case can be extended in a straightforward way by considering transformed set of matrices like $\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{y}$ and $\mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{\Lambda}^{-\frac{1}{2}}$.

4. Implementation Aspects

For calculating the LOO-CV based predictive measures we need to find efficient ways of computing $\hat{f}_{-i}(\mathbf{x}_i)$ and $\sigma^2_{-i}(\mathbf{x}_i)$ for all $i \in \tilde{I}$ given $\mathbf{u}$; we also need to efficiently evaluate the measure as we add a new basis vector to $\mathbf{u}$. To do that, we take advantage of rank one update and single basis vector addition to the matrices $\mathbf{\Sigma}$ and $\mathbf{K}_{\mathbf{u},\mathbf{u}}$. In practice, working with Cholesky decomposition of these matrices provides both numerical stability and computational advantages. We present here only the basic ideas; see appendix for detailed derivations.

We first illustrate finding $\hat{f}_{-i}(\mathbf{x}_i)$ for a given $\mathbf{u}$. Using rank one update of $\mathbf{\Sigma}$ we have $y_i - \hat{f}_{-i}(\mathbf{x}_i;\mathbf{u}) = \frac{y_i - \hat{f}(\mathbf{x}_i;\mathbf{u})}{1-\eta_i(\mathbf{u})}$ where $\hat{f}(\mathbf{x}_i;\mathbf{u}) = \mathbf{K}_{i,\mathbf{u}}\boldsymbol{\alpha}$, $\boldsymbol{\alpha} = \sigma^{-2}\mathbf{\Sigma}\mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{y}$ and $\eta_i(\mathbf{u}) = \sigma^{-2}\mathbf{K}_{i,\mathbf{u}}\mathbf{\Sigma}\mathbf{K}_{\mathbf{u},i}$ (Cawley & Talbot, 2004; Hong et al., 2006). Therefore, given $\hat{f}(\mathbf{x}_i;\mathbf{u})$ and $\eta_i(\mathbf{u})$ for all $i \in \tilde{I}$ the LOO-CV error computation has $O(n)$ complexity. Note that with some abuse of notation we have indicated explicit dependence on the inducing inputs $\mathbf{X_u}$. Now to select a basis vector based on the various predictive measures we need to compute the LOO-CV error as we add a new basis vector. Let u_j be an additional element added to the set $\mathbf{u}$ and $\bar{\mathbf{u}}_j = (\mathbf{u} \quad u_j)$. The key is to efficiently compute $\hat{\mathbf{f}}(\bar{\mathbf{u}}_j)$ and $\boldsymbol{\eta}(\bar{\mathbf{u}}_j)$ from $\hat{\mathbf{f}}(\mathbf{u})$ and $\boldsymbol{\eta}(\mathbf{u})$ with incremental cost; as shown in the appendix these computations have $O(nm)$ complexity where m represents the number of basis vectors selected so far. Note that $\hat{\mathbf{f}}(\bar{\mathbf{u}}_j)$ and $\boldsymbol{\eta}(\bar{\mathbf{u}}_j)$ represent the vectorized forms.

Next we discuss computation of the predictive variance. In the case of DTC approximation it can be shown that $\sigma^2_{-i}(\mathbf{x}_i,\mathbf{u})_{DTC} = \mathbf{K}_{i,i} - \mathbf{Q}_{i,i}(\mathbf{u}) + \frac{\sigma^2}{1-\eta_i(\mathbf{u})}$. Since $\eta_i(\mathbf{u})$ is already computed we need to compute $\mathbf{Q}_{i,i}(\mathbf{u}) = \mathbf{K}_{i,\mathbf{u}}\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u},i}$ additionally and this is needed only for the NLGPP measure. Again the key is to efficiently compute $\mathbf{Q}_{i,i}(\bar{\mathbf{u}}_j)$ from $\mathbf{Q}_{i,i}(\mathbf{u})$; this computation has $O(nm)$ complexity. In the case of GPE, we need not explicitly compute

$\mathbf{Q}_{i,i}(\bar{\mathbf{u}}_j)$; this is because it is enough to find $\sum_{i=1}^{n} \mathbf{Q}_{i,i}(\bar{\mathbf{u}}_j)$ and can be computed efficiently in $O(m^2)$ time.

The generic greedy selection algorithm used in the SGPR model learning is well-known (Rasmussen and Williams, 2006). Note that the goal is to find the optimal set of basis vectors and hyperparameters; the algorithm interleaves basis vector set selection and hyperparameter optimization and continues until a stopping criterion is met.

ALGORITHM

1. Initialize the hyperparameters $\boldsymbol{\theta}$.
2. Initialize $\mathbf{X_u} = \phi$, $A = \phi$, and $R = \{1, 2, \ldots, n\}$.
3. Create a working set $J \subseteq R$. Compute $M(\mathbf{X}_{\bar{\mathbf{u}}_j}, \boldsymbol{\theta})$ (one of the predictive measures given in (2), (3) and (4)) for all $j \in J$. $l = \underset{j \in J}{argmin}\ M(\mathbf{X}_{\bar{\mathbf{u}}_j}, \boldsymbol{\theta})$.
4. Set $\mathbf{X_u} \leftarrow \mathbf{X_u} \cup \{\mathbf{x}_l\}$, $A \leftarrow A \cup \{l\}$ and $R \leftarrow R \setminus \{l\}$. Initialize relevant variables in the first iteration and update them in subsequent iterations.
5. Terminate the basis vector addition and go to step 6 if the measure starts increasing or stops decreasing significantly or d_{max} basis vectors are added. Otherwise, go to step 3.
6. Re-estimate the hyperparameters $\boldsymbol{\theta}$.
7. Terminate the algorithm if the stopping criterion (Seeger, 2003) is met. Otherwise, go to step 2.

In step 3 of the algorithm due to resource constraints like memory and computational cost requirements for choosing a basis vector, the algorithm maintains a set of candidate basis vectors J of fixed size κ; we refer to this set as *working set*. Smola and Bartlett (2001) suggested to construct this working set in each iteration by randomly choosing κ elements from the remaining set of training inputs R and set κ to 59. In our *cache implementation* apart from randomly chosen 59 candidate basis vectors we propose to retain some of the members of the current working set in the cache. After sorting the working set members according to the chosen measure the top basis vector is added to $\mathbf{X_u}$ (step 4) and the next d_{cache} basis vectors are kept in the cache. The cache implementation has the advantage that we can choose basis vector from a larger working set subsequently. Then as shown in the appendix, for each member in the cache necessary computations (that is, computing mainly $\eta_i(\bar{\mathbf{u}}_j)$, $\mathbf{Q}_{i,i}(\bar{\mathbf{u}}_j)$ for all $i \in \tilde{I}$) can be done in an incremental fashion efficiently in $O(n)$ cost instead of $O(nm)$.

In step 5 of the algorithm, we observed in our experiments that the predictive measures start increasing beyond the addition of a certain number of basis vectors; we refer this number as d_{opt}. This happens as these measures estimate the predictive ability of different models when the basis vectors are added sequentially and the predictive ability falls-off when the model becomes more complex and starts fitting noise. Thus, *the number of basis vectors needed can be automatically determined.* Since d_{opt} is not known *apriori* the user

defined d_{max} can still be used if there are computational constraints and the algorithm can be terminated if d_{max} basis vectors are added and $d_{max} \leq d_{opt}$.

In step 6 of the algorithm, ideally the complete LOO-CV based predictive approach would select the hyperparameters (Sundararajan and Keerthi, 2001). However gradient evaluation is expensive ($O(nDm^2)$) in this approach when the ARD type hyperparameters are used. Therefore we re-estimate the hyperparameters by maximizing the marginal likelihood (Candela and Rasmussen, 2005a). Though this approach is not optimal it is found to be very effective in practice.

The worst case storage and computational complexities for the proposed algorithms are $O(nd_{max})$ and $O(\kappa n d_{max}^2)$ respectively; additional memory of size $O(nd_{cache})$ and computational cost of $O(nd_{cache}d_{max})$ are needed for cache implementation. Note that the NLML and NLPP algorithms have the same computational and storage complexities.

Next, we make following additional remarks.

1. In step 3 of the algorithm, the NLML and NLPP algorithms can also benefit from the cache implementation with some additional memory as shown in the appendix. Next note that it is not necessary to select d_{cache} basis vectors from the top; the reason is, if some of these basis vectors are very close to the best chosen basis vector in the current working set, then they will have measure values similar to that of the chosen basis vector. Here, closeness is defined with reference to the kernel in the inducing input space. However, as we include the chosen basis vector, these *close* basis vectors are *less likely* to be at the top next time. This can possibly be avoided by introducing an additional check (based on some distance measure) to select d_{cache} basis vectors; but this comes with additional cost. In our experiments we did not use any additional step; nevertheless, the simple approach mentioned above is very effective.

2. In step 6 of the algorithm, since the marginal likelihood function and gradient evaluation cost $O(nm^2 + nDm)$ (Candela and Rasmussen, 2005a) only short optimization of hyperparameters is done in the interleaving optimization process. In cases where the ARD type hyperparameters may not be needed it would be interesting to study the performances with LOO-CV based predictive approaches for the hyperparameter optimization as well.

3. The actual training time for the different algorithms depends on several other factors. This is because d_{opt} may not be same for the different algorithms. Further, even though we use ML based optimization for the hyperparameters the different algorithms need not converge at same hyperparameters and as a consequence depending on the function surface in that region of hyperparameters, the number of function and gradient evaluations needed in the hyperparameter optimization algorithm can be different unless it is fixed.

4. The stopping criterion of the interleaving optimization iteration (step 7) is typically based on changes in the measure values over the iterations; therefore, the number of iterations needed for convergence can vary depending on the surface nature of the different measures.

5. Though the proposed algorithms have the same computational and storage complexities as the NLML and NLPP algorithms, additional quantities like $\eta_i(\mathbf{u})$, $\mathbf{Q}_{i,i}(\mathbf{u})$ and

some associated quantities have to be computed and stored. See the appendix for more details.

5. Discussion

In this section we discuss the behaviors of algorithms using different measures as we add the basis vectors in a greedy fashion. The behaviors on the Friedman3 dataset (Friedman, 1991) are given in Figure 1. The multiple lines indicate the behaviors for different values of hyperparameters - as the optimization proceeds. Notice that after a certain number of basis vectors are added, the measures start increasing. Like ML measure, the number of basis vectors can be automatically selected with all the predictive measures. As pointed out by Candela and Rasmussen (2005a), this may not be the case with the NLPP measure. The point at which the chosen measure starts increasing is a useful point to stop the addition of basis vector. As a measure may sometime have multiple minima it is useful to observe it for a few more iterations and terminate only when it has increased significantly. The multi-minima nature of the measure is mainly due to the greedy selection process and randomly chosen elements with constrained working set size. The optimal number of basis vectors shows some variation as we optimize over hyperparameters - but in most cases we observed that the variation is only about 10%.

Among the LOO-CV based measures, the GPE measure minimization results in the highest number of basis vectors. This can be explained with reference to the variance term which is reduced as we add more basis vectors. Note that initially both the first term in (4) (which is essentially LOO-CVE) and the second term (variance part) will start decreasing. However, beyond a certain point the LOO-CVE term will start increasing; but, the variance term will continue decreasing. Then, there can be an interval where the reduction in the variance term is more compared to the increase in LOO-CVE. This results in an overall decrease in GPE. But, beyond that point the increase in LOO-CVE dominates, resulting in an overall increase in GPE. This explains why the use of the GPE measure results in more number of basis vectors than the use of the LOO-CVE measure. If we compare it with the NLGPP measure, the main difference comes from the way the variance term is utilized. It is known that predictive density loss function (which is essentially NLGPP) penalizes both over-confident and under-confident estimates. However, it penalizes over-confident estimates more for a given error; therefore, it would go with under-confident estimates. On the other hand, the GPE measure encourages over-confident estimates because of its additive nature. Therefore, as the estimates become more and more over-confident with increase in the number of basis vectors, the NLGPP measure will start penalizing them more. This results in lesser number of basis vectors than those given by the GPE measure. It is difficult to compare the NLML measure with the LOO-CV based measures in a similar fashion.

6. Numerical Experiments

In this section, we study and compare the performances of the algorithms using predictive measures presented in Section 3 with the algorithms using the NLML and NLPP measures discussed in Section 2; we refer to these algorithms as LOO-CVE, NLGPP, GPE, NLML

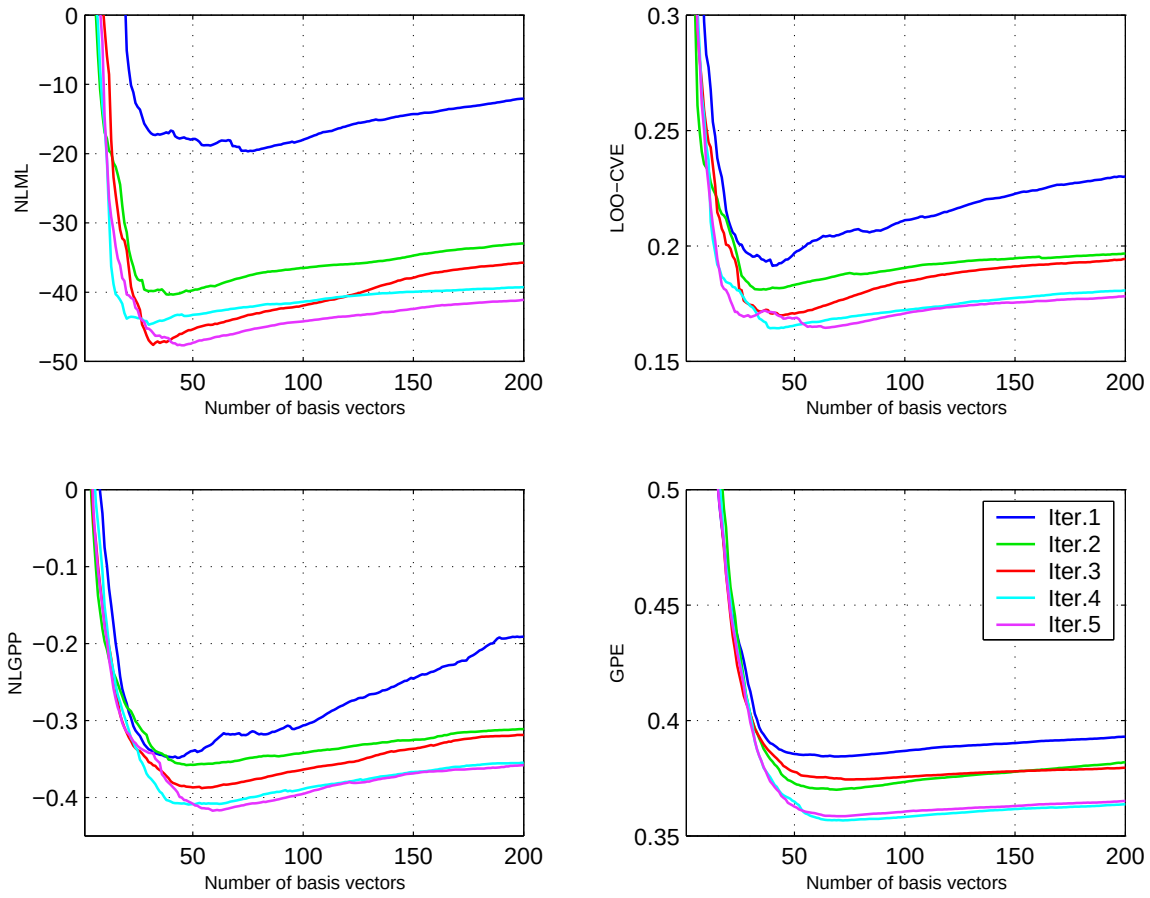

Figure 1: Friedman3 dataset: The behaviors of algorithms using different measures as we add basis vector and optimize hyperparameters in an interleaving fashion are shown. The legend in the bottom-right plot holds for all the plots.

and NLPP algorithms. In the case of the NLPP algorithm we fixed the number of basis vectors. To evaluate the generalization performance we use normalized mean squared error (NMSE) (normalized with respect to the data variance) and the negative predictive density loss (NPDL). We also report results from paired-t statistical significance test at a significance level of 0.05. This test was conducted by comparing the algorithm that gives the best average performance with the rest of the algorithms. If the test reveals that the results are statistically significant, then the algorithm under comparison is highlighted with symbol $*$ and this implies that the average performance of this algorithm is inferior in statistical sense. The experiments were conducted using eight datasets. The description of the datasets is given in Table 1; here n, D, p and nr represent number of training samples, input dimension, number of test samples and number of runs with different training/test sets respectively. The *Boston housing* dataset was obtained from the *StatLib* archive[1]. For the *Abalone* dataset[2] gender encoding (male/female/infant) was mapped into {(1,0,0), (0,1,0), (0,0,1)}.

1. *http://lib.stat.cmu.edu/datasets/boston*
2. *ftp://ftp.ics.uci.edu/pub/machine-learning-databases/abalone/*

Table 1: Data sets description

Dataset	n	D	p	nr
Boston housing	481	13	25	100
Abalone	3000	10	1177	20
bank8FM	4500	8	3692	20
cpusmall	4500	12	3692	20
cpuact	4500	21	3692	20
bank32NH	4500	32	3692	20
Friedman2	200	4	5000	100
Friedman3	200	4	5000	100

The last four datasets were obtained from another site[3]. The last two datasets were generated, as described by Friedman (1991). For all the experiments, we used the ARD Gaussian kernel as defined in Section 2. We conducted three experiments. In the first experiment, we fixed the hyperparameters by maximizing the marginal likelihood of full GP with only a subset of the training samples and skipped steps 6 and 7 of the algorithm. In the cases of *Boston housing* and *Abalone* datasets, we used 200 and 750 random samples respectively. In the cases of *bank8FM, cpusmall, cpuact* and *bank32NH* datasets we used 1000 samples. In the cases of *Friedman2* and *Friedman3* datasets we used 100 samples. In the second experiment we included cache implementation and the working set size was set to 300 for all the datasets except for the *Friedman2* and *Friedman3* datasets; for these datasets the working set size was set to 100. In the third experiment, we included hyperparameter optimization with ML function and gradient evaluations of the sparse GP model.

The results from the first and second experiments with fixed hyperparameters are given in Table 2 and Table 3. The average performances without cache over $20, 10$ and 5 runs for *Boston housing, abalone and bank8FM* datasets respectively are given in Figure 2; similarly average over 5 runs for *cpusmall, cpuact and bank32NH* datasets and average over 20 runs for *Friedman2 and Friedman3* datasets are given in Figure 3 and Figure 4 respectively. We observed that the plots reflect the behavior in individual run as well.

From Figures 2-4 we see that the NLGPP algorithm gives excellent NPDL performance and the NLGPP measure is highly correlated with the NPDL generalization measure; in general, the GPE algorithm closely follows the NLGPP algorithm in the initial iterations but the NLGPP algorithm outperforms as the iterations progress. In terms of NMSE performance the NLGPP algorithm gives good performance only in the *Abalone* dataset and it gives inferior performance on the *Boston housing* and *Friedman2* datasets. In contrast, the NLML, NLPP and LOO-CVE algorithms progress well in terms of NMSE performances and they closely follow each other. It is interesting to see that the NPDL performance of the LOO-CVE algorithm closely follows the NLML and NLPP algorithms in the beginning and slowly makes transition to the GPE/NLGPP algorithm. The NMSE performance of the GPE algorithm is better than the NLGPP algorithm in the *Boston housing* and *Friedman2* datasets.

We see from the *left half* of Tables 2-3 that the NLGPP algorithm gives the best average NPDL performance on six datasets; in five cases the results are statistically significant. When the results are not statistically significant it achieves similar performance with lesser number of basis vectors. In terms of NMSE performance it gives best average performance

3. *http://www.liacc.up.pt/ ltorgo/Regression/DataSets.html*

Table 2: Test results on the *Boston housing*, *Abalone*, *bank8FM* and *cpusmall* datasets (in that order) - fixed hyperparameters: Mean and Standard deviation of performance measures and number of basis vectors. In the case of Abalone dataset the NPDL measure has a multiplication factor of 10^{-2}.

	Without cache			*With cache*		
ALGORITHM	NMSE x 10^{-2}	NPDL	# BV	NMSE x 10^{-2}	NPDL	# BV
NLPP	**11.65 ± 6.74**	2.73 ±0.70*	200	**11.69 ± 6.80**	2.74±0.72*	200
NLML	11.73 ± 6.73	2.76±0.72*	156.2 ±17.2	12.02 ± 6.81	2.79±0.69*	131.1 ±17.3
LOO-CVE	12.34 ± 6.86	2.64±0.54	117.1±19.2	13.33 ± 8.63	2.59±0.50	107.4±15.3
NLGPP	16.85 ± 11.29*	**2.53 ± 0.32**	**107.4 ± 15.4**	16.88 ± 11.14*	**2.54 ± 0.34**	101.8 ± 15.4
GPE	12.30 ± 7.41	2.71 ± 0.67*	160.0 ± 12.8	12.20 ± 7.42	2.69 ± 0.66*	150.1 ± 13.2
NLPP	42.35 ± 1.81	100.91 ±1.34	200	42.36 ± 1.82	100.93±1.33	200
NLML	42.44 ± 1.78	104.54±3.10*	**47.9 ±11.0**	42.38 ± 1.91	107.47±4.23*	**53.6 ±16.9**
LOO-CVE	42.28 ± 1.87	101.71±2.16*	70.8±34.3	42.21 ± 1.94	102.10±1.53*	65.4±19.1
NLGPP	**42.27 ± 1.91**	**100.33 ± 1.31**	78.4 ± 26.4	**42.17 ± 1.90**	**100.17 ± 1.21**	79.1 ± 24.5
GPE	42.36 ± 1.80	100.87 ± 1.35	142.8 ± 37.2	42.36 ± 1.82	100.86 ± 1.36	130.9 ± 32.0
NLPP	3.62 ± 0.2	-2.14 ±0.01*	500	3.61 ± 0.14	-2.14±0.02*	500
NLML	3.60 ± 0.12	-2.15±0.01*	194.1 ±58.7	3.67 ± 0.17*	-2.15±0.01*	140.6 ±20.1
LOO-CVE	3.69 ± 0.18*	-2.15±0.01*	153.1±34.8	**3.56 ± 0.09**	-2.16±0.01*	151.8±34.2
NLGPP	3.70 ± 0.18*	**-2.17 ± 0.01**	**104.6 ± 20.0**	3.66 ± 0.20*	**-2.17 ± 0.02**	**99.9 ± 19.2**
GPE	**3.56 ± 0.14**	-2.15 ± 0.01*	259.5 ± 44.5	3.60 ± 0.11	-2.15 ± 0.01*	238.2 ± 54.3
NLPP	2.46 ± 0.13	2.45 ±0.02	500	2.49 ± 0.16	2.46±0.03*	500
NLML	**2.46 ± 0.12**	2.45±0.02	359.7 ±111.2	**2.45 ± 0.14**	2.45±0.02*	318.8 ±88.18
LOO-CVE	2.56 ± 0.25	2.45±0.02	265.3±68.2	2.55 ± 0.16	2.46±0.03*	249.9±66.5
NLGPP	2.57 ± 0.13*	**2.44 ± 0.03**	**206.1 ± 58.2**	2.57 ± 0.17*	**2.43 ± 0.02**	**202.1 ± 59.8**
GPE	2.55 ± 0.17	2.45 ± 0.03	363.6 ± 71.0	2.52 ± 0.14	2.45 ± 0.02*	330.2 ± 67.2

only in the case of *Abalone* dataset. The NMSE performances of NLML, NLPP, LOO-CVE and GPE algorithms are similar and the results are not statistically significant in five cases. However, the LOO-CVE algorithm gives better average NPDL performance compared to the NLML algorithm for *Boston housing, Abalone, Friedman2 and bank32NH* datasets; further, it requires lesser number of basis vectors on *bank8FM, cpusmall and cpuact* datasets where similar average performances are seen. Use of additional basis vectors in the NLPP algorithm does not provide any clear advantage over other algorithms. Though the GPE algorithm uses predictive variance information, it results in more number of basis vectors (Section 5); but in general it does not provide any advantage over other algorithms.

Next we see from the *right half* of the Tables 2-3 that the NLGPP algorithm again gives excellent average NPDL performance over all the datasets and the results are statistically significant in six cases. Overall, the performances of all the algorithms with and without cache are very similar on all the datasets *but* the important observation with all algorithms (except the NLPP algorithm) and all the datasets is, the average number of basis vectors used by all the algorithms is *less* compared to their respective results *without cache* implementation except in only one case. Thus the experimental results demonstrate the effectiveness of the cache implementation.

The results from the third experiment with hyperparameter optimization are given in Tables 4-5. On comparing the results on *Boston housing and Abalone* datasets in the *left half* of Table 2 and Table 4 we see improved generalization performances of all algorithms except for minor variations in the NMSE performances on *Abalone* dataset. In some cases increase

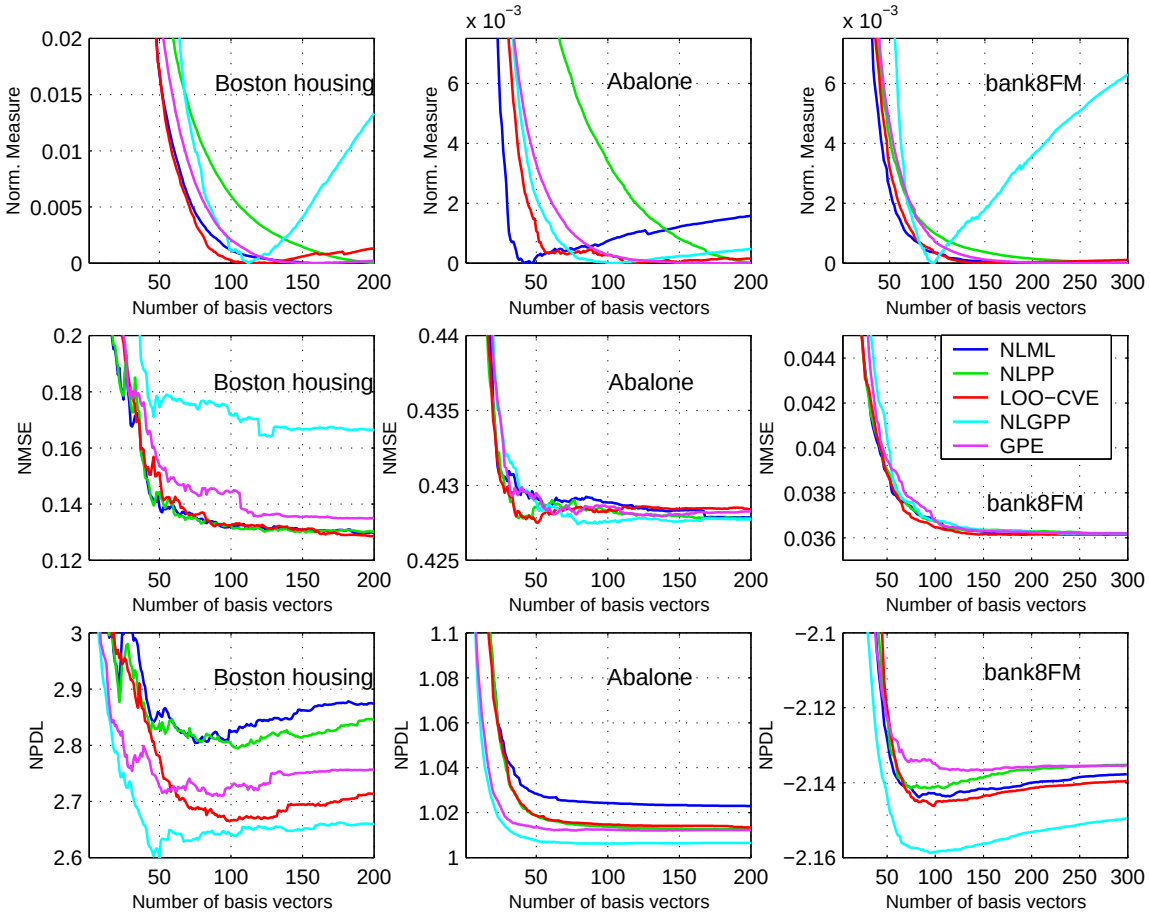

Figure 2: The normalized measure values and generalization performances of different measures on *Boston housing, Abalone and bank8FM datasets* as we add basis vector are shown. The legend in the middle-right plot holds for all the plots.

in the number of basis vectors is needed to improve the generalization performance. Due to excess computational time needed with hyperparameter optimization we reduced n to 2048 from 4500 and increased p to 6144 samples for *bank8FM* and *cpusmall* datasets. Hence these results are not directly comparable with those in Table 2. In general the NMSE performances of all the algorithms are similar for large datasets and the results are not statistically significant; this is not the case with the NPDL performance measure. As earlier the NLGPP algorithm gives excellent NPDL performance. Also except for *bank8FM* dataset its NMSE performance has improved significantly with hyperparameter optimization. On comparing the results on *Friedman2 and Friedman3* datasets in the *left half* of Table 3 and Table 5 we see slight degradation in the performances of NLPP and NLML algorithms; however, in the case of NLML algorithm this seems to be due to underfitting as the number of basis vectors has come down. On the other hand, the performances of all LOO-CV based algorithms have improved significantly with hyperparameter optimization and the results are statistically significant. In general, overall improved performances of all LOO-CV based algorithms on the different datasets demonstrate the effectiveness of hyperparameter

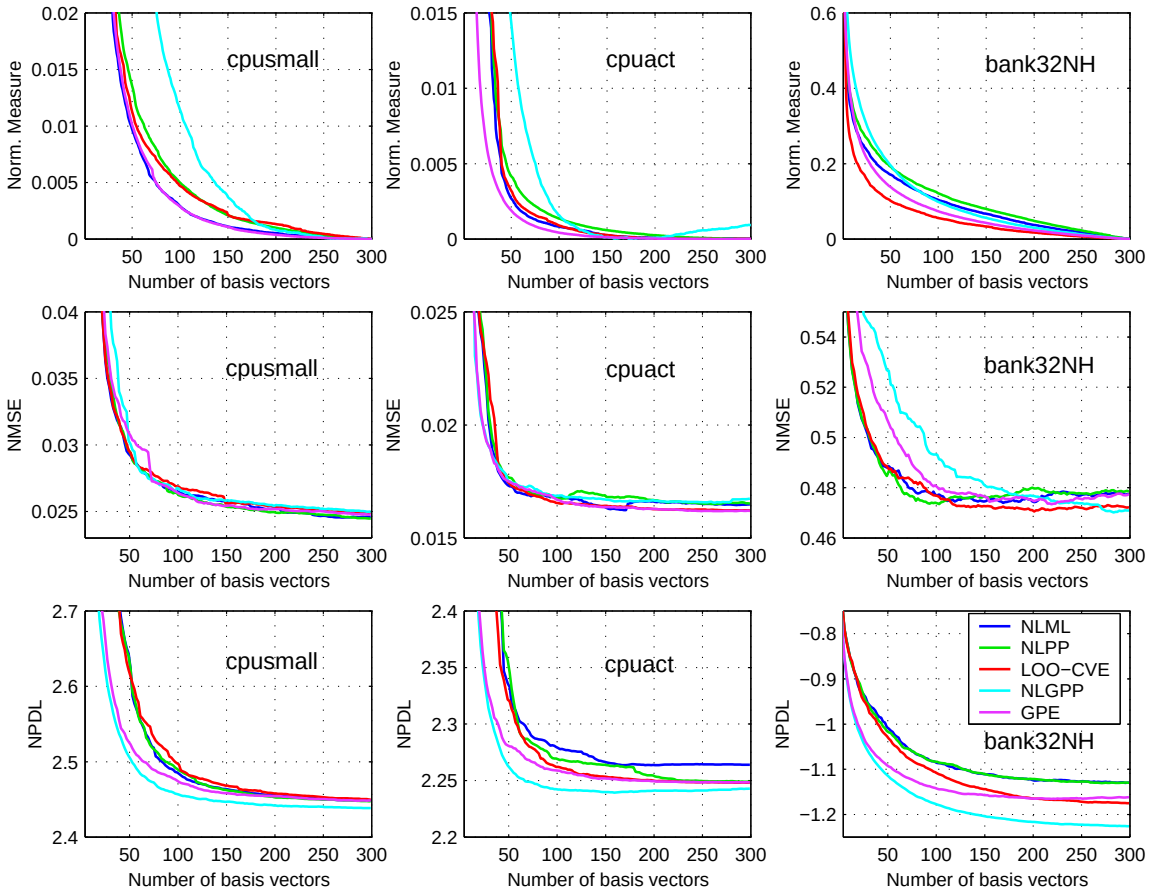

Figure 3: The normalized measure values and generalization performances of different measures on *cpusmall, cpuact and bank32NH datasets* as we add basis vector are shown. The legend in the bottom-right plot holds for all the plots.

estimation using marginal likelihood maximization interleaved with basis vector selection using predictive approaches.

Finally we note that though the computational and storage complexities of all the algorithms are similar the actual computational time and storage requirements of the proposed algorithms are about 2 to 3 times higher than those of the NLML and NLPP algorithms.

7. Conclusion

In this paper we proposed LOO-CV predictive measures based basis vector selection algorithms in building SGPR models. The computational complexities of these measures are same as that of the NLML and NLPP algorithms. These measures estimate the predictive ability of the model and they have the advantage that the number of basis vectors can be found automatically. The GPP and GPE measures also take the predictive variance information into account. We also gave an efficient cache implementation and the experiments demonstrate its effectiveness by achieving similar performance with *lesser* number of basis vectors. The NLGPP algorithm is very successful in achieving excellent NPDL performance

Table 3: Test results on the *Friedman2*, *Friedman3*, *bank32NH* and *cpuact* datasets (in that order) - fixed hyperparameters: Mean and Standard deviation of performance measures and number of basis vectors.

	Without cache			*With cache*		
ALGORITHM	NMSE x 10^{-2}	NPDL	# BV	NMSE x 10^{-2}	NPDL	# BV
NLPP	**11.44 ± 3.52**	6.57 ±0.16	75	**11.45 ± 3.58**	6.57±0.16	75
NLML	11.85 ± 3.76	6.79±0.23*	**44.1 ±13.6**	11.95 ± 3.74	6.82±0.23*	**42.0 ±10.5**
LOO-CVE	12.31 ± 4.02	6.67±0.17*	45.2±9.2	12.20 ± 4.12	6.70±0.19*	44.3±8.5
NLGPP	14.13 ± 7.21*	6.57 ± 0.18	56.2 ± 13.3	14.54 ± 7.74*	6.58 ± 0.17	54.4 ± 13.9
GPE	11.67 ± 3.63	**6.56 ± 0.16**	65.3 ± 13.5	11.80 ± 3.75	**6.56 ± 0.16**	62.5 ± 12.2
NLPP	**12.14 ± 1.39**	0.71 ±0.06	75	**12.13 ± 1.39**	0.71±0.06	75
NLML	12.59 ± 1.35	0.77±0.07*	**47.9 ±13.4**	12.64 ± 1.41*	0.78±0.07*	**46.6 ±13.7**
LOO-CVE	12.62 ± 1.43*	0.75±0.06*	49.6±10.9	12.63 ± 1.37*	0.75±0.06*	44.3±8.5
NLGPP	12.77 ± 1.44*	0.72 ± 0.05*	53.1 ± 11.5	12.81 ± 1.40*	0.73 ± 0.05*	52.1 ± 10.5
GPE	12.20 ± 1.37	**0.70 ± 0.06**	70.1 ± 13.0	12.20 ± 1.39	**0.70 ± 0.06**	68.4 ± 11.9
NLPP	50.3 ± 4.68	-1.11±0.03*	500	49.87 ± 3.27	-1.09±0.02*	500
NLML	49.1 ± 3.82	-1.11±0.02*	486.4 ±56.3	50.14 ± 3.68	-1.08±0.02*	481.5 ±66.8
LOO-CVE	**47.69** ± 3.89	-1.19±0.03*	**427.6±89.1**	**48.75** ± 3.79	-1.19±0.03*	**424.8±87.1**
NLGPP	48.47 ± 3.95	**-1.25 ± 0.04**	479.4 ± 47.65	49.18 ± 3.74	**-1.25 ± 0.04**	443.6 ± 124.6
GPE	48.03 ± 3.73	-1.16 ± 0.02*	488.2 ± 48.7	49.61 ± 3.37	-1.14 ± 0.03*	481.0 ± 68.5
NLPP	1.59 ± 0.11	2.23±0.02	500	1.60 ± 0.16	2.23±0.03	500
NLML	**1.56 ± 0.07**	2.24±0.02*	425.1 ±83.2	1.60 ± 0.09	2.25±0.02*	315.9 ±121.0
LOO-CVE	1.62 ± 0.08*	2.24±0.03	248.6± 49.5	**1.57 ± 0.07**	2.25±0.02*	206.0± 47.2
NLGPP	1.60 ± 0.09	**2.22 ± 0.03**	**182.5 ± 52.2**	1.63 ± 0.09*	**2.22 ± 0.02**	**160.0 ± 28.7**
GPE	1.60 ± 0.10	2.23 ± 0.03	351.9 ± 59.1	1.59 ± 0.08	2.24 ± 0.03*	313.4 ± 44.8

Table 4: The left half of the table contains test results on the *Boston housing and Abalone datasets* and the right half contains test results on the *bank8FM and cpusmall* datasets (in that order) - with hyperparameters optimization: Mean and Standard deviation of performance measures and number of basis vectors. In the case of Abalone dataset the NPDL measure has a multiplication factor of 10^{-2}.

	NMSE x 10^{-2}	NPDL	# BV	NMSE x 10^{-2}	NPDL	# BV
NLPP	10.06 ± 5.35	2.47±0.22	200	4.02 ± 0.20	-2.13±0.01*	500
NLML	10.94 ± 6.03	2.68±0.21*	**116.7 ±12.8**	3.96 ± 0.16	-2.14±0.01*	**112.6 ±20.0**
LOO-CVE	**9.87 ± 5.69**	2.48±0.173	143.2±18.0	**3.92 ± 0.16**	-2.14±0.01*	117.6±21.8
NLGPP	10.15 ± 6.02	**2.42 ± 0.23**	168.8 ± 16.8	4.10 ± 0.30*	**-2.15 ± 0.01**	177.8 ± 71.7
GPE	10.16 ± 5.78	2.42 ± 0.26	199.9 ± 0.33	4.06 ± 0.30	-2.12 ± 0.02*	274.5 ± 83.4
NLPP	42.87 ± 1.77	99.60±4.42*	200	2.92 ± 0.32	**2.45±0.02**	500
NLML	42.50 ± 1.69	100.60±2.42*	76.2 ±19.2	2.91 ± 0.29	2.47±0.01*	230.3 ±55.0
LOO-CVE	**42.29 ± 2.01**	97.70±1.87*	80.2±38.4	2.81 ± 0.19	2.48±0.01*	**198.8±46.5**
NLGPP	42.41 ± 2.28	**94.69 ± 1.51**	**35.6 ± 9.0**	2.91 ± 0.33	2.46 ± 0.01*	226.0 ± 36.2
GPE	42.48 ± 1.88	97.86 ± 2.23 *	164.5 ± 34.4	**2.76 ± 0.24**	2.46 ± 0.01	286.4 ± 51.7

and the LOO-CVE algorithm is very useful in achieving good NMSE performance. Both the algorithms result in sparse solutions and are excellent alternatives to existing methods in achieving good generalization performance.

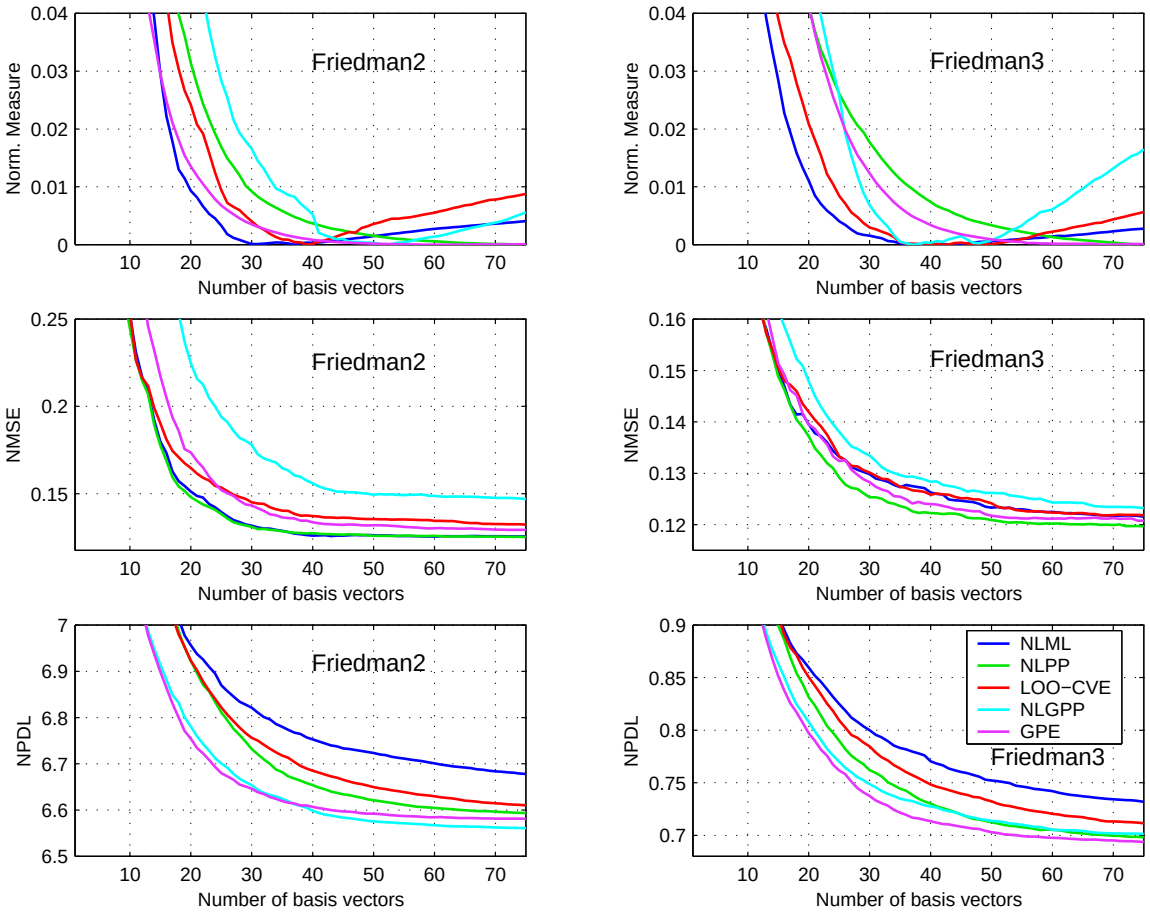

Figure 4: The normalized measure values and generalization performances of different measures on *Friedman2 and Friedman3 datasets* as we add basis vector are shown. The legend in the bottom-right plot holds for all the plots.

Table 5: The left half of the table contains test results on the *Friedman2* dataset and the right half contains test results on the *Friedman3* dataset - with hyperparameters optimization: Mean and Standard deviation of performance measures and number of basis vectors.

	NMSE x 10^{-2}	NPDL	# BV	NMSE x 10^{-2}	NPDL	# BV
NLPP	13.26 ± 8.78*	6.63±0.26*	75	12.71 ± 2.39*	0.77±0.11*	75
NLML	14.28 ± 7.19*	7.30±0.26*	**28.4 ±6.9**	13.41 ± 1.80*	0.97±0.08*	**34.8 ±9.4**
LOO-CVE	11.12 ± 4.41*	6.87±0.21*	34 ±6.7	12.36 ± 1.38*	0.84±0.06*	44.4 ±10.9
NLGPP	**10.00 ± 3.24**	6.51 ± 0.06	37.5 ± 6.3	11.93 ± 1.27	0.73 ± 0.05*	51.6 ± 10.2
GPE	10.09 ± 3.76	**6.50 ± 0.06**	52.2 ± 9.9	**11.63 ± 1.29**	**0.70 ± 0.04**	74.7 ± 11.6

References

J. Q. Candela and C. E. Rasmussen. Analysis of some methods for reduced rank Gaussian process regression. In R. Murray-Smith and R. Shorten, editors, *Switching and Learning*

in Feedback Systems, volume 3355 of *Lecture Notes in Computer Science*, pages 98–127. Springer, Heidelberg, Germany, 2005a.

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

G. C. Cawley and N. L. C. Talbot. Fast exact leave-one-out cross-validation of sparse least squares support vector machines. *Neural Networks*, 17(10):1467–1475, 2004.

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

J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19:1–141, 1991.

S. Geisser. The predictive sample reuse methods with applications. *Journal of American Statistical Association*, 70(35):320–328, 1975.

S. Geisser and W.F. Eddy. A predictive approach to model selection. *Journal of American Statistical Association*, 74(365):153–160, 1979.

M. Gibbs and D. J. C. MacKay. Efficient implementation of Gaussian processes. Technical report, Cavendish Laboratory, Cambridge university, Cambridge, London, UK, 1997.

X. Hong, S. Chen, and C. J. Harris. Fast kernel classifier construction using orthogonal forward selection to minimise the leave-one-out misclassification rate. volume 4113 of *Lecture Notes in Computer Science*, pages 106–114. Springer, 2006.

S.S. Keerthi and W. Chu. A matching pursuit approach to sparse Gaussian process regression. In *Advances in Neural Information Processing Systems*, volume 17. The MIT Press, 2005.

N. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems*, volume 15, pages 609–616. The MIT Press, 2003.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. The MIT press, 2006.

A. Schwaighofer and V. Tresp. Transductive and inductive methods for approximate Gaussian process regression. In *Advances in Neural Information Processing Systems*, volume 15. The MIT Press, 2003.

M. Seeger. *Bayesian Gaussian process models: PAC-Bayesian generalisation error bounds and sparse approximations*. PhD thesis, University of Edinburgh, 2003.

M. Seeger, C. K. I. Williams, and N. D. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In C. M. Bishop and B. J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, San Francisco, USA, 2003. Morgan Kaufmann.

B. W. Silverman. Some aspects of the spline smoothing approach to non-parametric regression curve fitting (with discussion). *Journal of Royal Statistical Society (Series B)*, 47 (1):1–52, 1985.

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

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

M. Stone. Cross-validatory choice and assessment of statistical predictions (with discussion). *Journal of Royal Statistical Society (Series B)*, 36:111–147, 1974.

S. Sundararajan and S. S. Keerthi. Predictive approaches for choosing hyperparameters in Gaussian processes. *Neural Computation*, 13(5):1103–1118, 2001.

M. Tipping. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

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

G. Wahba, X. Gao, F. Xiang, R. Klein, and B. Klein. The bias-variance trade-off and the randomized GACV. In *Advances in Neural Information Processing Systems*, volume 11. The MIT Press, 1999.

C. K. I. Williams and M. Seeger. Using the Nystrom method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, volume 13, pages 682–688. The MIT Press, 2001.

Appendix

In this appendix, we give the necessary details concerning Section 4. In the following, with some abuse of notations we indicate dependence on the basis vectors $\mathbf{X_u}$ and $\mathbf{x}_j$ through the variables $\mathbf{u}$ and u_j respectively. Let $\mathbf{u}$ denote the set of indices of the basis vectors in the present model and $\bar{\mathbf{u}}_j$ denote the set when u_j is added to the set $\mathbf{u}$. $\bar{\mathbf{u}}_j = (\mathbf{u}\ u_j)$. Let m denote the cardinality of the set $\mathbf{u}$. In order to calculate the predictive measures described in Section 3, we need to find the predictive mean and variance using the basis vectors in $\mathbf{u}$. Moreover, to observe the effect of adding a new basis vector u_j to the current basis vector set on different predictive measures, we need to compute the predictive mean and variance using the basis vectors in $\bar{\mathbf{u}}_j$ efficiently. The key idea here is to update the relevant matrices using simple matrix update formulas.

Recall from Section 3 that the predictive mean and variance using the *DTC* approximation are given by $\hat{f}(x_i;\mathbf{u}) = \sigma^{-2}\mathbf{K}_{i,\mathbf{u}}\mathbf{\Sigma}\mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{y}$ and $\hat{\sigma}^2(x_i;\mathbf{u}) = \mathbf{K}_{i,i} - \mathbf{Q}_{i,i}(\mathbf{u}) + \mathbf{K}_{i,\mathbf{u}}\mathbf{\Sigma}\mathbf{K}_{\mathbf{u},i}$ where $\mathbf{\Sigma} = \left(\mathbf{K}_{\mathbf{u},\mathbf{u}} + \sigma^{-2}\mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{K}_{\mathbf{f},\mathbf{u}}\right)^{-1}$ and $\mathbf{Q}_{i,i}(\mathbf{u}) = \mathbf{K}_{i,\mathbf{u}}\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u},i}$. Using the same approximation, the LOO predictive mean and variance are given by $\hat{f}_{-i}(x_i;\mathbf{u}) = \sigma^{-2}\mathbf{K}_{i,\mathbf{u}}\mathbf{\Sigma}_{-i}\mathbf{K}_{\mathbf{u},-i}\mathbf{y}_{-i}$ and $\hat{\sigma}^2_{-i}(x_i;\mathbf{u}) = \mathbf{K}_{i,i} - \mathbf{Q}_{i,i}(\mathbf{u}) + \mathbf{K}_{i,\mathbf{u}}\mathbf{\Sigma}_{-i}\mathbf{K}_{\mathbf{u},i}$. As discussed in Section 4, $y_i - \hat{f}_{-i}(x_i;\mathbf{u}) = \frac{y_i - \hat{f}(x_i;\mathbf{u})}{1-\eta_i(\mathbf{u})}$ and $\hat{\sigma}^2_{-i}(x_i;\mathbf{u}) = \mathbf{K}_{i,i} - \mathbf{Q}_{i,i}(\mathbf{u}) + \frac{\sigma^2}{1-\eta_i(\mathbf{u})}$ where $\eta_i(\mathbf{u}) = \sigma^{-2}\mathbf{K}_{i,\mathbf{u}}\mathbf{\Sigma}\mathbf{K}_{\mathbf{u},i}$. Thus, *it is*

easy to calculate different predictive measures in (2)-(4) if σ^{-2}, $\mathbf{\Sigma}$ *and* $\mathbf{K}_{i,\mathbf{u}}$, $\hat{f}(x_i;\mathbf{u})$, $\eta_i(\mathbf{u})$ *and* $\mathbf{Q}_{i,i}(\mathbf{u})$ *for every* i *are available for the basis vector set* $\mathbf{u}$. Initially, when m is one, all these quantities are easy to compute as the matrices involved are of size, 1×1. Further, as we will explain below, these quantities are easy to update if a new basis vector u_j is added to the set $\mathbf{u}$. Then, the computation of different predictive measures for the basis vector set $\bar{\mathbf{u}}_j$ becomes straightforward, resulting in an easy way to select the basis vector.

Let $\mathbf{\Sigma}^{-1} = A_{\mathbf{u}} = \mathbf{K}_{\mathbf{u},\mathbf{u}} + \sigma^{-2}\mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{K}_{\mathbf{f},\mathbf{u}} = \mathbf{L}_{\mathbf{u}}\mathbf{L}_{\mathbf{u}}^T$. We now see how to update $\mathbf{L}_{\mathbf{u}}$ to get the Cholesky decomposition of $A_{\bar{\mathbf{u}}_j}$ if a new basis vector, u_j, is added to the set $\mathbf{u}$. We know that

$$A_{\bar{\mathbf{u}}_j} = \begin{pmatrix} A_{\mathbf{u}} & \mathbf{b}_j \\ \mathbf{b}_j^T & c_j \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{\mathbf{u}} & \mathbf{0} \\ \mathbf{z}_j^T & d_j \end{pmatrix} \begin{pmatrix} \mathbf{L}_{\mathbf{u}}^T & \mathbf{z}_j \\ \mathbf{0}^T & d_j \end{pmatrix}$$

where $\mathbf{b}_j = \mathbf{K}_{\mathbf{u},j} + \sigma^{-2}\mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{K}_{\mathbf{f},j}$ and $c_j = \mathbf{K}_{j,j} + \sigma^{-2}\mathbf{K}_{j,\mathbf{f}}\mathbf{K}_{\mathbf{f},j}$. Thus, $\mathbf{z}_j$ can be obtained by forward substitution in the lower triangular system of equations, $\mathbf{L}_{\mathbf{u}}\mathbf{z}_j = \mathbf{b}_j$. Then d_j is calculated as $d_j = \sqrt{c_j - \mathbf{z}_j^T\mathbf{z}_j}$. Finding the Cholesky decomposition of $A_{\bar{\mathbf{u}}_j}$ thus requires $O(m^2)$ effort if the Cholesky decomposition of $A_{\mathbf{u}}$ is available.

We now describe a way to calculate $\eta_i(\bar{\mathbf{u}}_j)$ efficiently. Note that $\eta_i(\mathbf{u}) = \sigma^{-2}\mathbf{K}_{i,\mathbf{u}}A_{\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u},i} = \sigma^{-2}\mathbf{K}_{i,\mathbf{u}}\mathbf{L}_{\mathbf{u}}^{-T}\mathbf{L}_{\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u},i}$. If we obtain $\boldsymbol{\zeta}_i(\mathbf{u})$ by using the lower triangular system of equations, $\mathbf{L}_{\mathbf{u}}\boldsymbol{\zeta}_i(\mathbf{u}) = \mathbf{K}_{\mathbf{u},i}$, then we have $\eta_i(\mathbf{u}) = \sigma^{-2}\boldsymbol{\zeta}_i^T(\mathbf{u})\boldsymbol{\zeta}_i(\mathbf{u})$. Now, $\eta_i(\bar{\mathbf{u}}_j)$ can be expressed as,

$$\sigma^{-2}\left(\mathbf{K}_{i,\mathbf{u}}\ \mathbf{K}_{i,j}\right)\begin{pmatrix} \mathbf{L}_{\mathbf{u}}^T & \mathbf{z}_j \\ \mathbf{0}^T & d_j \end{pmatrix}^{-1}\begin{pmatrix} \mathbf{L}_{\mathbf{u}} & \mathbf{0} \\ \mathbf{z}_j^T & d_j \end{pmatrix}^{-1}\begin{pmatrix} \mathbf{K}_{i,\mathbf{u}} \\ \mathbf{K}_{i,j} \end{pmatrix}$$

which can be rewritten as $\eta_i(\bar{\mathbf{u}}_j) = \sigma^{-2}\boldsymbol{\zeta}_i^T(\bar{\mathbf{u}}_j)\boldsymbol{\zeta}_i(\bar{\mathbf{u}}_j)$ where,

$$\boldsymbol{\zeta}_i(\bar{\mathbf{u}}_j) = \begin{pmatrix} \boldsymbol{\zeta}_i(\mathbf{u}) \\ \zeta_{i,j} \end{pmatrix}$$

and $\zeta_{i,j} = \frac{\mathbf{K}_{i,j} - \mathbf{z}_j^T\boldsymbol{\zeta}_i(\mathbf{u})}{d_j}$. Therefore, we can write $\eta_i(\bar{\mathbf{u}}_j) = \eta_i(\mathbf{u}) + \sigma^{-2}\zeta_{i,j}^2$ and $\zeta_{i,j}$ can be evaluated in $O(m)$ time for every training set sample i.

Computation of $\hat{f}(x_i;\bar{\mathbf{u}}_j)$ can be done efficiently if $\hat{f}(x_i;\mathbf{u})$ is available for every i. We have $\hat{f}(x_i;\mathbf{u}) = \sigma^{-2}\mathbf{K}_{i,\mathbf{u}}\mathbf{L}_{\mathbf{u}}^{-T}\mathbf{L}_{\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{y}$. So, $\hat{f}(x_i;\bar{\mathbf{u}}_j)$ is

$$\left(\mathbf{K}_{i,\mathbf{u}}\ \mathbf{K}_{i,j}\right)\begin{pmatrix} \mathbf{L}_{\mathbf{u}}^T & \mathbf{z}_j \\ \mathbf{0}^T & d_j \end{pmatrix}^{-1}\begin{pmatrix} \mathbf{L}_{\mathbf{u}} & \mathbf{0} \\ \mathbf{z}_j^T & d_j \end{pmatrix}^{-1}\mathbf{v}_{\bar{\mathbf{u}}_j}$$

where

$$v_{\bar{\mathbf{u}}_j} = \begin{pmatrix} v_{\mathbf{u}} \\ v_j \end{pmatrix} = \begin{pmatrix} \sigma^{-2}\mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{y} \\ \sigma^{-2}\mathbf{K}_{j,\mathbf{f}}\mathbf{y} \end{pmatrix}.$$

Let

$$\begin{pmatrix} \mathbf{L}_{\mathbf{u}} & 0 \\ \mathbf{z}_j^T & d_j \end{pmatrix}\begin{pmatrix} \mathbf{w}_{\mathbf{u}} \\ w_j \end{pmatrix} = \begin{pmatrix} \mathbf{v}_{\mathbf{u}} \\ v_j \end{pmatrix}.$$

$\mathbf{w}_{\mathbf{u}}$ is obtained by solving the lower triangular system of equations, $\mathbf{L}_{\mathbf{u}}\mathbf{w}_{\mathbf{u}} = \mathbf{v}_{\mathbf{u}}$ and w_j is then calculated as $w_j = \frac{v_j - \mathbf{z}_j^T\mathbf{w}_{\mathbf{u}}}{d_j}$. It is easy to see that $\hat{f}(x_i;\bar{\mathbf{u}}_j) = \hat{f}(x_i;\mathbf{u}) + w_j\zeta_{i,j}$. Computation of $\mathbf{w}_{\mathbf{u}}$ requires $O(m^2)$ effort. w_j can then be evaluated in $O(n)$ time as

computation of v_j needs $O(n)$ effort. Note that $\zeta_{i,j}$ was evaluated during the computation of $\eta_i(\bar{\mathbf{u}}_j)$. Therefore, $\hat{f}(x_i;\bar{\mathbf{u}}_j)$ can be obtained in $O(n)$ time for all training samples.

Efficient computation of $\hat{\sigma}^2_{-i}(x_i;\bar{\mathbf{u}}_j)$ requires an easy way to compute $\mathbf{Q}_{i,i}(\bar{\mathbf{u}}_j)$ from $\mathbf{Q}_{i,i}(\mathbf{u})$. Let $\mathbf{K}_{\mathbf{u},\mathbf{u}} = \mathbf{G}_\mathbf{u}\mathbf{G}_\mathbf{u}^T$. Then, $\mathbf{K}_{\bar{\mathbf{u}}_j,\bar{\mathbf{u}}_j}$ is

$$\begin{pmatrix} \mathbf{G}_\mathbf{u} & \mathbf{0} \\ \boldsymbol{\nu}_j^T(\boldsymbol{\nu}) & e_j \end{pmatrix} \begin{pmatrix} \mathbf{G}_\mathbf{u}^T & \boldsymbol{\nu}_j(\mathbf{u}) \\ \mathbf{0}^T & e_j \end{pmatrix} = \begin{pmatrix} \mathbf{K}_{\mathbf{u},\mathbf{u}} & \mathbf{K}_{\mathbf{u},j} \\ \mathbf{K}_{j,\mathbf{u}} & \mathbf{K}_{j,j} \end{pmatrix}.$$

$\boldsymbol{\nu}_j(\mathbf{u})$ is obtained by solving the lower triangular system of equations $\mathbf{G}_\mathbf{u}\boldsymbol{\nu}_j(\mathbf{u}) = \mathbf{K}_{\mathbf{u},j}$ and e_j is set to $\sqrt{\mathbf{K}_{j,j} - \boldsymbol{\nu}_j^T(\mathbf{u})\boldsymbol{\nu}_j(\mathbf{u})}$. Therefore, $\mathbf{Q}_{i,i}(\mathbf{u}) = \boldsymbol{\nu}_i^T(\mathbf{u})\boldsymbol{\nu}_i(\mathbf{u})$ and $\mathbf{Q}_{i,i}(\bar{\mathbf{u}}_j) = \mathbf{Q}_{i,i}(\mathbf{u}) + \nu_{i,j}^2$ where $\nu_{i,j} = \frac{\mathbf{K}_{j,i} - \boldsymbol{\nu}_j^T(\mathbf{u})\boldsymbol{\nu}_i(\mathbf{u})}{e_j}$. Updating $\mathbf{Q}_{i,i}(\mathbf{u})$ to $\mathbf{Q}_{i,i}(\bar{\mathbf{u}}_j)$ can thus be done in $O(m)$ time for every training set sample i if $\boldsymbol{\nu}_j(\mathbf{u})$ is available.

In step 3 of the algorithm, $M(X_{\bar{\mathbf{u}}_j},\theta)$ can be calculated by using the following incremental calculations for every training sample i.

$$\begin{array}{lcl}
\zeta_{i,j} & = & \frac{\mathbf{K}_{i,j} - \mathbf{z}_j^T\boldsymbol{\zeta}_i(\mathbf{u})}{d_j} \\
\eta_i(\bar{\mathbf{u}}_j) & = & \eta_i(\mathbf{u}) + \sigma^{-2}\zeta_{i,j}^2 \\
w_j & = & \frac{v_j - \mathbf{z}_j^T\mathbf{w}_\mathbf{u}}{d_j} \\
\hat{f}(x_i;\bar{\mathbf{u}}_j) & = & \hat{f}(x_i;\mathbf{u}) + w_j\zeta_{i,j} \\
\nu_{i,j} & = & \frac{\mathbf{K}_{j,i} - \boldsymbol{\nu}_j^T(\mathbf{u})\boldsymbol{\nu}_i(\mathbf{u})}{e_j} \\
\mathbf{Q}_{i,i}(\bar{\mathbf{u}}_j) & = & \mathbf{Q}_{i,i}(\mathbf{u}) + \nu_{i,j}^2 \\
\hat{\sigma}^2_{-i}(x_i;\bar{\mathbf{u}}_j) & = & K_{i,i} - \mathbf{Q}_{i,i}(\bar{\mathbf{u}}_j) + \sigma^{-2}(1 - \eta_i(\bar{\mathbf{u}}_j))^{-1}.
\end{array}$$

The variables, $\eta_i(\mathbf{u})$, $\hat{f}(x_i,\mathbf{u})$ and $\mathbf{Q}_{i,i}(\mathbf{u})$ can be stored in the memory. Further, it is a good idea to store $\mathbf{w}_\mathbf{u}$,$\boldsymbol{\zeta}_i(\mathbf{u})$ and $\boldsymbol{\nu}_i(\mathbf{u})$. This will require an additional storage of $O(mn)$. But, this will result in doing the above calculations in $O(mn)$ time for all the training set samples. Note that the computation of $\zeta_{i,j}$ for a given $j \in J$ and for all training samples requires $O(mn + m^2)$ computational effort, which is $O(mn)$ if $n \gg m$. For a given $j \in J$, w_j can be computed in $O(n)$ time and $\nu_{i,j}$ for all the training set samples can be calculated in $O(mn)$ time.

Note that in the case of GPE measure (4), it is enough to compute $\sum_i \hat{\sigma}^2_{-i}(x_i;\mathbf{u})$ rather than the individual $\hat{\sigma}^2_{-i}(x_i;\mathbf{u})$. That is, we need to compute $\mathbf{Q}(\mathbf{u}) = \sum_i \mathbf{Q}_{i,i}(\mathbf{u}) = \sum_i \mathbf{K}_{i,\mathbf{u}}\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u},i}$ which is equivalent to $trace\left(\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}\mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{K}_{\mathbf{f},\mathbf{u}}\right)$. Recall that $A_\mathbf{u} = \mathbf{K}_{\mathbf{u},\mathbf{u}} + \sigma^{-2}\mathbf{K}_{\mathbf{u},\mathbf{f}}\mathbf{K}_{\mathbf{f},\mathbf{u}} = \mathbf{L}_\mathbf{u}\mathbf{L}_\mathbf{u}^T$; therefore, $\mathbf{Q}(\mathbf{u}) = \sigma^2 trace\left(\mathbf{K}_{\mathbf{u},\mathbf{u}}^{-1}A_\mathbf{u}\right) - \sigma^2 m$. Let us define $\mathbf{E}_\mathbf{u} = G_\mathbf{u}^{-1}\mathbf{L}_\mathbf{u}$. Then, $\mathbf{Q}(\mathbf{u}) = \sigma^2 trace\left(\mathbf{E}_\mathbf{u}\mathbf{E}_\mathbf{u}^T\right) - \sigma^2 m$. Next, to compute $\mathbf{Q}(\bar{\mathbf{u}}_j)$ from $\mathbf{Q}(\mathbf{u})$ incrementally we make use of the lower triangular structures of $\mathbf{G}_\mathbf{u}$ and $\mathbf{L}_\mathbf{u}$. More specifically, we have

$$\begin{pmatrix} \mathbf{G}_\mathbf{u} & \mathbf{0} \\ \boldsymbol{\nu}_j^T(\mathbf{u}) & e_j \end{pmatrix} \begin{pmatrix} \mathbf{E}_\mathbf{u} & \mathbf{0} \\ \mathbf{p}_j^T & f_j \end{pmatrix} = \begin{pmatrix} \mathbf{L}_\mathbf{u} & \mathbf{0} \\ \mathbf{z}_j^T & d_j \end{pmatrix}.$$

Hence, $f_j = \frac{d_j}{e_j}$ and $\mathbf{p}_j = \frac{\mathbf{z}_j - \mathbf{E}_\mathbf{u}^T\boldsymbol{\nu}_j(\mathbf{u})}{e_j}$. Note that $\mathbf{p}_j$ has computational complexity of $O(m^2)$. Therefore, $\mathbf{Q}(\bar{\mathbf{u}}_j)$ can be computed in $O(m^2)$ time.

In step 4 of the algorithm, the basis vector index set is updated to $(\mathbf{u}\ u_l)$. The necessary updates of the variables $\mathbf{K}_{\mathbf{u},\mathbf{u}}, \mathbf{L}_\mathbf{u}, \mathbf{G}_\mathbf{u}, \mathbf{E}_\mathbf{u}, \boldsymbol{\zeta}_i(\mathbf{u}), \boldsymbol{\nu}_i(\mathbf{u})$ and $\mathbf{w}_\mathbf{u}$ can be done easily if

the relevant variables like $z_l, d_l, e_l, w_l, \zeta_{i,l}$ and $\nu_{i,l}$ are available in memory. Note that the dimension of these variables increases by one after they are updated.

Next, we discuss cache implementation details. From the above-mentioned calculations, it is clear that we need to compute $\mathbf{z}_j^T \boldsymbol{\zeta}_i(\mathbf{u})$ for every $j \in J$ and every $i \in \tilde{I}$. Further, computation of $\mathbf{z}_j$ needs $O(m^2)$ effort. Once a basis vector l is selected in Step 3 of the algorithm, it is a good idea to store $\mathbf{z}_j$, $\zeta_{i,j}$ and d_j for some of the existing working set members and for every $i \in \tilde{I}$ if some additional memory is available. Computation of $\zeta_{i,j}$ in the next iteration will then require the computation of $z_{j,l}$ and d_j only as the necessary elements are already stored in the cache memory; as a consequence $\zeta_{i,j}$ can be computed in $O(n)$ time instead of $O(nm)$ time. Following the similar steps, by storing $\nu_{i,j}$ and e_j computation of $\mathbf{Q}_{i,i}(\bar{\mathbf{u}}_j)$ can be done efficiently. Similarly in the case of GPE, by storing $\mathbf{E}_{\mathbf{u}}^T \boldsymbol{\nu}_j(\mathbf{u})$ computation of $\mathbf{Q}(\bar{\mathbf{u}}_j)$ can be done efficiently. Finally, we note that the quantities like $\mathbf{z}_j$, $\zeta_{i,j}$, d_j and e_j are needed for the NLML/NLPP algorithms as well; therefore, the above implementation details are directly applicable.