

Scaling Multidimensional Inference for Structured Gaussian Processes

Elad Gilboa, Yunus Saatçi, and John P. Cunningham

Abstract—Exact Gaussian Process (GP) regression has $\mathcal{O}(N^3)$ runtime for data size N , making it intractable for large N . Many algorithms for improving GP scaling approximate the covariance with lower rank matrices. Other work has exploited structure inherent in particular covariance functions, including GPs with implied Markov structure, and equispaced inputs (both enable $\mathcal{O}(N)$ runtime). However, these GP advances have not been extended to the multidimensional input setting, despite the preponderance of multidimensional applications. This paper introduces and tests novel extensions of structured GPs to multidimensional inputs. We present new methods for additive GPs, showing a novel connection between the classic backfitting method and the Bayesian framework. To achieve optimal accuracy-complexity tradeoff, we extend this model with a novel variant of projection pursuit regression. Our primary result – projection pursuit Gaussian Process Regression – shows orders of magnitude speedup while preserving high accuracy. The natural second and third steps include non-Gaussian observations and higher dimensional equispaced grid methods. We introduce novel techniques to address both of these necessary directions. We thoroughly illustrate the power of these three advances on several datasets, achieving close performance to the naive Full GP at orders of magnitude less cost.

Index Terms—Gaussian Processes, Backfitting, Projection-Pursuit Regression, Kronecker matrices.

1 INTRODUCTION

Gaussian Processes (GP) have become a popular tool for nonparametric Bayesian regression. Naive GP regression has $\mathcal{O}(N^3)$ runtime (matrix inversions and determinants) and $\mathcal{O}(N^2)$ memory complexity, where N is the number of observations. At ten thousand or more, this problem is for all practical purposes intractable, given current hardware.

A significant amount of research has gone into sparse approximations (reducing run-time complexity to $\mathcal{O}(M^2N)$ for some $M \ll N$). For an excellent review of sparse GP approximations, see [1]. All sparse approximation methods are based on the assumption of conditional independence of the training and test sets, given an active set of inducing inputs. As emphasized in [1], the results of these algorithms can depend strongly on the properties of the data. Since different assumptions fit different datasets, and since sparsity has by no means solved all efficiency issues for GPs, it is imperative to explore alternative avenues for attaining scalability.

The central aim of this paper is to introduce *structured* GPs for multidimensional inputs. Specifically we present three novel advances which allow efficient and sometimes exact inference, or at least a superior runtime-accuracy tradeoff than existing methods. We say a GP is *structured* if its marginals $p(\mathbf{f}|\mathbf{X}, \theta)$ contain exploitable

structure that enables reduction in computational complexity. While these structured GP methods are known in the case of scalar inputs, many regression applications involve multivariate inputs. Our main contribution is three nontrivial extensions of these algorithms to deal with this case.

1.1 Gaussian Process Regression

In brief, GP regression is a Bayesian method for nonparametric regression, where a prior distribution over continuous functions is specified via a Gaussian process. (the use of GP in machine learning is well described in [2]).

A GP is a distribution on f over an input space X such that any finite selection of input locations $\mathbf{x}_1, \dots, \mathbf{x}_N \in X$ gives rise to a multivariate Gaussian density over the associated targets, i.e.,

$$p(f(\mathbf{x}_1), \dots, f(\mathbf{x}_N)) = \mathcal{N}(\mathbf{m}_N, \mathbf{K}_N), \quad (1)$$

where $\mathbf{m}_N = m(x_1, \dots, x_N)$ is the mean vector and $\mathbf{K}_N = \{k(x_i, x_j)\}_{i,j}$ is the covariance matrix for mean function m and covariance function k . In this paper we are specifically interested in the basic equations for GP regression, which involve two steps. First, for given data \mathbf{y} (making the standard assumption of zero-mean data, without loss of generality), we calculate the predictive mean and covariance at M unseen inputs as:

$$\boldsymbol{\mu}_* = \mathbf{K}_{MN} (\mathbf{K}_N + \sigma_n^2 \mathbf{I}_N)^{-1} \mathbf{y}, \quad (2)$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_M - \mathbf{K}_{MN} (\mathbf{K}_N + \sigma_n^2 \mathbf{I}_N)^{-1} \mathbf{K}_{NM}, \quad (3)$$

For model selection, since the function $k(\cdot, \cdot; \theta)$ is parameterized by hyperparameters such as amplitude and

- E. Gilboa is with the Preston M. Green Department of Electrical and System Engineering, Washington University in St. Louis. E-mail: gilboa@cse.wustl.edu
- Y. Saatçi is with the Department of Engineering University of Cambridge, UK. E-mail: yunus.saatci@gmail.com
- J. Cunningham is with the Department of Engineering University of Cambridge, UK. E-mail: jpc74@cam.ac.uk.

lengthscale (which we group into θ), we must consider the log marginal likelihood $Z(\theta)$:

$$\begin{aligned} \log Z(\theta) &= -\frac{1}{2} (\mathbf{y}^\top (\mathbf{K}_N + \sigma_n^2 \mathbf{I}_N)^{-1} \mathbf{y} \\ &+ \log(\det((\mathbf{K}_N + \sigma_n^2 \mathbf{I}_N))) + N \log(2\pi)) \end{aligned} \quad (4)$$

Here we use this marginal likelihood to optimize over the hyperparameters in the usual way [2]. The runtime of GP regression and hyperparameter learning is $\mathcal{O}(N^3)$ due to $(\mathbf{K}_N + \sigma_n^2 \mathbf{I}_N)^{-1}$, which is present in all equations.

1.2 Gauss-Markov Processes

We briefly review the use of Gauss-Markov Processes for efficient GP regression on scalar inputs, as a starting point for the multidimensional extensions in Section 2. Although the Gauss-Markov Process are well studied, their use for exact and efficient GP regression is underappreciated. A GP with a kernel corresponding to a state-space model can be viewed as a Gauss-Markov Process, enabling linear runtime. Gauss-Markov Processes can be viewed as the solution of an order- m linear, stationary stochastic differential equation (SDE), given by:

$$\frac{d^m f(x)}{dx^m} + a_{m-1} \frac{d^{m-1} f(x)}{dx^{m-1}} + \dots + a_1 \frac{df(x)}{dx} + a_0 f(x) = w(x), \quad (5)$$

where $w(x)$ is a zero-mean white noise process. Note that x can be any scalar input, including time. Because $w(x)$ is a GP and f is linear in its coefficients, f is also a GP. See [3] for an excellent introduction to SDEs. We can rewrite Eq. (5) as a vector Markov process:

$$\frac{d\mathbf{z}(x)}{dx} = \mathbf{A}\mathbf{z}(x) + \mathbf{L}w(x), \quad (6)$$

where

$$\mathbf{z}(x) = \left[f(x), \frac{df(x)}{dx}, \dots, \frac{d^{m-1}f(x)}{dx^{m-1}} \right]^\top, \quad (7)$$

and where $\mathbf{L} = [0, 0, \dots, 1]$, and \mathbf{A} is the usual suitable coefficient matrix. Eq. (6) shows that, given knowledge of $f(x)$ and its first m derivatives, we have Markov structure in the graph underlying GP inference, which will enable all efficiency gains in this section.

Earlier work [4], [5], derived the SDEs corresponding to several commonly used covariance functions including the Matérn family and spline kernels, and good approximate SDEs corresponding to the exponentiated-quadratic kernel. Once the SDE is known, the Kalman filtering [6] and Rauch-Tung-Striebel (RTS) [7] smoothing algorithms (which correspond to belief propagation) can be used to perform GP regression in $\mathcal{O}(N)$ time and memory, a noteworthy leap in efficiency. Note that the Gauss-Markov Process framework requires sorted input points. Thus, if a sorting step is required to preprocess the data, the runtime complexity will be $\mathcal{O}(N \log N)$. However, as this is not always relevant and certainly not the focus of the algorithms presented here, we assume the inputs are sorted in advance and refer to these

models as $\mathcal{O}(N)$ in runtime complexity. For this paper we will summarize the usual system equations as:

$$\text{Initial state : } p(\mathbf{z}(x_1)) = \mathcal{N}(\mathbf{z}(x_1); \boldsymbol{\mu}_1, \mathbf{V}_1), \quad (8)$$

$$\begin{aligned} \text{State update : } p(\mathbf{z}(x_i) | \mathbf{z}(x_{i-1})) \\ = \mathcal{N}(\mathbf{z}(x_i); \boldsymbol{\Phi}_{i-1} \mathbf{z}(x_{i-1}), \mathbf{Q}_{i-1}), \end{aligned} \quad (9)$$

$$\begin{aligned} \text{Emission : } p(y(x_i) | \mathbf{z}(x_i)) \\ = \mathcal{N}(y(x_i); \mathbf{h}^T \mathbf{z}(x_i), \sigma_n^2), \end{aligned} \quad (10)$$

where we assume that the inputs x_i are sorted in ascending order, and the system matrices $\boldsymbol{\Phi}$ and \mathbf{Q} are functions of the original GP's hyperparameters. Using these update and emission equations in the standard Kalman or RTS framework allows exact regression over the Gauss-Markov Process for a single-input dimension.

2 STRUCTURED GP ON MULTIPLE INPUT DIMENSIONS

Despite its importance for a variety of applications, tractable extension of the state-space models (Section 1.2) to higher-dimensional input spaces has not been addressed in the literature. Doing so involves a number of novel steps and is the primary contribution of this work. We introduce three new algorithms for structured GPs over multivariate inputs, namely: (Sec. 2.1) additive multidimensional regression (with extension to additive covariates), (Sec. 2.2) non-Gaussian likelihood extensions, and (Sec. 2.3) GPs over a multidimensional grid. Full details of the development of these algorithmic extensions, including important proofs, can be found in our supporting work [5].

2.1 GP Regression for Multidimensional State-Space Models

For the purposes of extending one-dimensional Gauss-Markov Processes (Sec. 1.2) to multiple dimensions, we use the assumption of additivity. The optimal accuracy-efficiency tradeoff will be presented in Section 2.1.3. Here we introduce the building blocks. The resulting model regresses a sum of D Gauss-Markov Processes (which are independent a priori), where $D > 1$ is the dimensionality of the input space. Additive GP regression can be described using the following generative model:

$$y_i = \sum_{d=1}^D \mathbf{f}_d(X_{i,d}) + \epsilon \quad i = 1, \dots, N, \quad (11)$$

$$\mathbf{f}_d(\cdot) \sim \mathcal{GP}(\mathbf{0}; k_d(\mathbf{x}_d, \mathbf{x}'_d; \theta_d)) \quad d = 1, \dots, D, \quad (12)$$

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2), \quad (13)$$

where $X_{i,d}$ is the d -th component of input i , $k_d(\cdot, \cdot)$ is the kernel of the scalar GP along dimension d , θ_d represent the dimension-specific hyperparameters, and σ_n^2 is the (global) noise hyperparameter. Although interactions between input dimensions are not modeled a priori, an additive model does offer interpretable results – one can

simply plot the posterior mean of the individual \mathbf{f}_d to visualize how each predictor relates to the target [8].

We introduce a novel multidimensional Gauss-Markov Process regression where the underlying algorithm can be viewed as a Bayesian interpretation of the classical backfitting method [9], [10]. As described in [11], a nonparametric regression technique (such as the spline smoother) which allows a scalable fit over a scalar input space can be used to fit an additive model over a D -dimensional space with the same overall asymptotic complexity, by means of the backfitting algorithm.

Surprisingly, the application of backfitting (Algorithm 1) can be proved to converge to the exact posterior mean. The easiest way to see this is by viewing (Algorithm 1) as a Gauss-Seidel iteration. We detail this in-depth proof in our supporting work [5]. As a reminder, Gauss-Seidel is an iterative technique to solve linear systems, in this case solving for the exact posterior mean (see [12] for a great Gauss-Seidel reference). It is precisely the additive Gauss-Markov Process structure that makes the backfitting update equivalent to a Gauss-Seidel step.

Algorithm 1: Efficient Computation of Additive GP Posterior Mean via Backfitting

inputs : Training data $\{\mathbf{X}, \mathbf{y}\}$. Suitable covariance function. Hypers $\theta = \bigcup_{d=1}^D \{\theta_d\} \cup \sigma_n^2$.
outputs: Posterior training means: $\sum_{d=1}^D \mu_d$, where $\mu_d \equiv \mathbb{E}(\mathbf{f}_d | \mathbf{y}, \mathbf{X}, \theta_d, \sigma_n^2)$.

```

1 Zero-mean the targets  $\mathbf{y}$ ;
2 Initialise the  $\mu_d$  (e.g. to 0);
3 while The change in  $\mu_d$  is above a threshold do
4   for  $d = 1, \dots, D$  do
5      $\mu_d \leftarrow \mathbb{E}(\mathbf{f}_d | \mathbf{y} - \sum_{j \neq d} \mu_j, \mathbf{X}_{:,d}, \theta_d, \sigma_n^2)$ ;  $\triangleright$  Use
      state-space model here.
6   end
7 end
```

To calculate posterior variances and learn hyperparameters, we must investigate further. We can express the underlying graphical model as in Fig. 1, where we have made the state-space representation of each scalar Gauss Markov process explicit. The observed variables are the targets \mathbf{y} , and the latent variables \mathbf{Z} consist of the D Markov chains:

$$\mathbf{Z} \equiv \left(\underbrace{\mathbf{z}_1^1, \dots, \mathbf{z}_1^N}_{\equiv \mathbf{Z}_1}, \underbrace{\mathbf{z}_2^1, \dots, \mathbf{z}_2^N}_{\equiv \mathbf{Z}_2}, \dots, \underbrace{\mathbf{z}_D^1, \dots, \mathbf{z}_D^N}_{\equiv \mathbf{Z}_D} \right). \quad (14)$$

Unfortunately, the true posterior $p(\mathbf{Z}_1, \dots, \mathbf{Z}_D | \mathbf{y}, \mathbf{X}, \theta)$ is hard to handle computationally because all variables \mathbf{Z}_i are coupled in the posterior. Although everything is still Gaussian, we are no longer able to use the efficient state-space methods of Section 1.2 returning us to the original computational intractability at large N . Thus, we require an approximate inference technique such as variational Bayesian expectation maximization (VBEM)

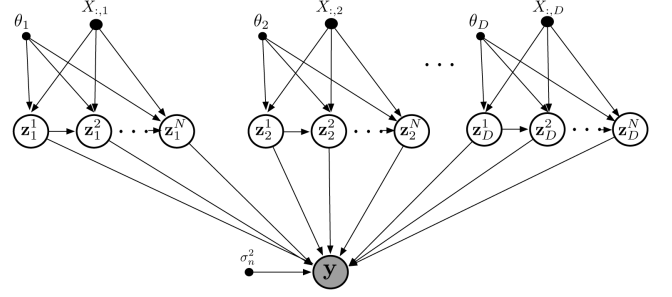


Fig. 1: Graphical model for efficient additive GP regression. Each dimension is written in its corresponding state-space model.

or Markov Chain Monte Carlo (MCMC) [13]. We now briefly introduce our use of these well-known technologies, as the details will demonstrate the important connection to the backfitting algorithm. Note, that the main benefits of using these algorithms comes from their scalability as they are able to inherit the linear time complexity of the state-space model.

2.1.1 Variational-Bayesian Expectation Maximization

E-Step: We use a variational-Bayesian (VB) approximation to the E-step by making the standard assumption of an approximate posterior that factorizes across the \mathbf{Z}_i , i.e.:

$$q(\mathbf{Z}) = \prod_{i=1}^D q(\mathbf{Z}_i). \quad (15)$$

Given such a factorized approximation, it can be shown that $\text{KL}(q(\mathbf{Z}) || p(\mathbf{Z} | \mathbf{y}, \theta))$ can be minimized in an iterative fashion, using the following central update rule [13]:

$$\log q(\mathbf{Z}_j) = \mathbb{E}_{i \neq j}(\log p(\mathbf{y}, \mathbf{Z} | \theta)) + \text{const}. \quad (16)$$

where $\mathbb{E}_{i \neq j}(\cdot)$ is an expectation with respect to $\prod_{i \neq j} q(\mathbf{Z}_i)$. Using Eqs. (15) and (16), we derive the iterative updates required for VBEM. We first write down the log joint over all variables, given by:

$$\begin{aligned} \log(p(\mathbf{y}, \mathbf{Z} | \theta)) = & \sum_{n=1}^N \log p \left(y_n | \mathbf{h}^T \sum_{d=1}^D \mathbf{z}_d^{t_d(n)}, \sigma_n^2 \right) + \\ & \sum_{d=1}^D \sum_{t=1}^N \log p(\mathbf{z}_d^t | \mathbf{z}_d^{t-1}, \theta_d), \end{aligned} \quad (17)$$

where we have defined $p(\mathbf{z}_d^t | \mathbf{z}_d^{t-1}, \theta_d) \equiv p(\mathbf{z}_d^1 | \theta_d)$, for $t = 1$, and $\mathbf{h}^T \mathbf{z}$ gives the first element of \mathbf{z} . Note that it is also necessary to define the mapping $t_d(\cdot)$ which gives, for each dimension d , the state-space model index associated with y_n . The index t iterates over the sorted input locations along axis d . Because the expectation of the right hand side of Eq. (17) does not depend on \mathbf{z}_j , we can consider that the mean of the Gaussian in the first term of Eq. (17), allowing us to write:

$$\begin{aligned}
\log q(\mathbf{Z}_j) &= \sum_{n=1}^N \log \mathcal{N} \left(\left(y_n - \mathbf{h}^T \sum_{i \neq j} \mathbb{E} [\mathbf{z}_i^{t_d(n)}] \right); \mathbf{h}^T \mathbf{z}_j^{t_j(n)}, \sigma_n^2 \right) \\
&+ \sum_{t=1}^N \log p(\mathbf{z}_j^t | \mathbf{z}_j^{t-1}, \theta_j) + \text{const.}
\end{aligned} \quad (18)$$

where

$$\mathbb{E} [\mathbf{z}_i^k] = \int \mathbf{z}_i^k q(\mathbf{Z}_i) d\mathbf{Z}_i. \quad (19)$$

A key and somewhat surprising outcome of Eq. (18) is that in order to update the factor $q(\mathbf{Z}_j)$ in the E step, it is sufficient to run the standard state-space model inference procedure using only the pseudo-observations: $(y_n - \mathbf{h}^T \sum_{i \neq j} \mathbb{E} [\mathbf{z}_i^{t_d(n)}])$.

A number of conclusions can be drawn from this connection. First, since VB iterations are guaranteed to converge, any moment computed using the factors $q(\mathbf{Z}_i)$ is also guaranteed to converge. Convergence of these moments is important because they are used to learn the hyperparameters. Second, since the true posterior $p(\mathbf{Z}_1, \dots, \mathbf{Z}_D | \mathbf{y}, \theta)$ is a large joint Gaussian over all the latent variables, $\mathbb{E}_q(\mathbf{Z})$ will be equal to the true posterior mean. This is because the true posterior is Gaussian (unimodal with the mean as its mode) and the VB approximation is mode-seeking [14]. Additionally, as is typical for variational methods, the posterior covariance will be underestimated because $\text{KL}(q(\mathbf{Z}) || p(\mathbf{Z} | \mathbf{y}, \theta))$ is an *exclusive* divergence measure [14].

The central VB update is precisely a backfitting update, thus illustrating a novel connection between approximate Bayesian inference for additive models and classical estimation techniques. Furthermore, this provides an alternative proof of why backfitting computes exact posterior means over latent function values.

M-Step: We must optimize $\mathbb{E}_q(\log p(\mathbf{y}, \mathbf{Z} | \theta))$ over θ . Using Eq. (17) it is easy to show that the expected sufficient statistics required to compute derivatives with respect to θ are the set of expected sufficient statistics for the state-space model associated with each individual dimension. This separability is another major advantage of using the factorized approximation to the posterior. Thus, for every dimension d , we use the Kalman filter and RTS smoother to compute $\{\mathbb{E}_q(\mathbf{Z}_d)(\mathbf{z}_d^n)\}_{n=1}^N$, $\{\mathbb{V}_q(\mathbf{Z}_d)(\mathbf{z}_d^n)\}_{n=1}^N$ and $\{\mathbb{E}_q(\mathbf{Z}_d)(\mathbf{z}_d^n \mathbf{z}_d^{n+1})\}_{n=1}^{N-1}$. We then use these expected statistics to compute derivatives of the expected complete data log-likelihood with respect to θ and use a standard minimizer (we use a conjugate gradient method) to complete the M step.

2.1.2 Markov Chain Monte Carlo (MCMC)

An important and customary comparison to VB is MCMC, which carries the usual benefits of approximate hyperparameter integration, but at a reduced efficiency

[15]. Here we briefly discuss our fairly standard MCMC implementation, noting only the important differences.

As in standard MCMC, we extended the model to include a prior over the hyperparameters. The hyperparameters for each univariate function \mathbf{f}_d are given a prior parameterized by $\{\mu_l, v_l, \alpha_\tau, \beta_\tau\}$, where $\{\mu_l, v_l\}$ correspond to the covariance function hyperparameter ℓ and $\{\alpha_\tau, \beta_\tau\}$ to τ_d . We also place a $\Gamma(\alpha_n, \beta_n)$ prior over the noise precision hyperparameter τ_n . We run Gibbs sampling where we block-sample the latent chains. The algorithm used to sample from the latent Markov chain in a state-space model has been called the forward-filtering, backward sampling algorithm, where forward filtering is followed by a backward sampling from the conditionals $p(\mathbf{z}_k | \mathbf{z}_{k+1}^{\text{sample}}; \mathbf{y}; \mathbf{X}_{:,d}; \theta_d)$ [16]. The sampling is initialized by sampling from $p(z_K | \mathbf{y}; \mathbf{X}_{:,d}; \theta_d)$, which is computed in the final step of the forward filtering run, to produce z_K^{sample} . The forward-filtering, backward sampling algorithm generates a sample of the entire state vector jointly (over training and test input locations).

2.1.3 Efficient Projected Additive GP Regression

So far, we have shown how the assumption of additivity can be exploited to derive non-sparse GP regression algorithms which scale as $\mathcal{O}(N)$. These considerable efficiency gains can however decrease accuracy and predictive power versus a full unstructured GP, due to the limited expressivity of the simple additive model. To address this, we now demonstrate a relaxation of the additivity assumption *without* sacrificing the $\mathcal{O}(N)$ scaling, by considering an additive GP regression model in a feature space linearly related to original space of covariates [17], [18]. We call this algorithm projection pursuit Gaussian Process regression (PPGPR).

We show that learning and inference for such a model can be performed by using *projection pursuit* GP regression, a novel fusion of the classical projection pursuit regression algorithm with GP regression, with no change to computational complexity. The graphical model illustrating this idea is given in Figure 2. We refer to the following *projected additive* GP prior:

$$y_i = \sum_{m=1}^M \mathbf{f}_m(\phi_m(i)) + \epsilon \quad i = 1, \dots, N, \quad (20)$$

$$\phi_m = \mathbf{X} \mathbf{w}_m, \quad (21)$$

$$\begin{aligned}
\mathbf{f}_m(\cdot) &\sim \mathcal{GP}(\mathbf{0}; k_m(\phi_m, \phi'_m; \theta_m)) \quad m = 1, \dots, M, \quad (22) \\
\epsilon &\sim \mathcal{N}(0, \sigma_n^2).
\end{aligned}$$

Notice that the number of projections, M , can be less or greater than D . Forming linear combinations of the inputs before feeding them into an additive GP model significantly enriches the flexibility of the functions supported by the prior above, including many terms which are formed by taking products of covariates, and thus can capture relationships where the covariates jointly affect the target variable. In fact, Eqs. (20) through to (22) are identical to the standard neural network model

where the nonlinear activation functions are modeled using GPs.

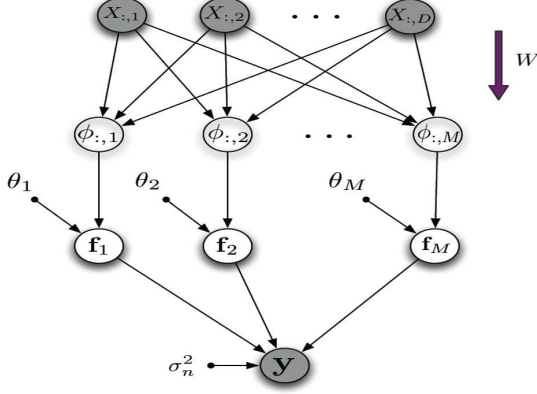


Fig. 2: Graphical model for Projected Additive GP Regression. In general, $M \neq D$. We present a greedy algorithm to select M , and jointly optimize W and $\{\theta_m\}_{m=1}^M$.

For inference and learning, we now derive a novel greedy algorithm which is similar to another classical nonparametric regression technique known as *projection pursuit* regression [19].

Consider the case where $M = 1$. In this case, the resulting projected additive GP regression model reduces to a scalar GP with inputs given by Xw . Recall from Section 1.2 that, for a kernel that can be represented as a state-space model, we can use the EM algorithm to optimize θ with respect to the marginal likelihood efficiently, for some fixed w . It is possible to extend this idea and jointly optimize w and θ with respect to the marginal likelihood, although we opt to optimize the marginal likelihood directly. Notice that every step of this optimization scales as $\mathcal{O}(N)$, since at every step we need to compute the marginal likelihood of a scalar GP (and its derivatives). These quantities are computed using the Kalman filter by differentiating the Kalman filtering process with respect to w and θ . All that is required is the derivatives of the state transition and process covariance matrices (Φ_t and Q_t , for $t = 1, \dots, N - 1$) with respect to w and θ .

We now handle the case where $M > 1$ using a greedy approach. At each iteration we find the optimal projection weights w . The greedy nature of the algorithm allows the learning of the dimensionality of the feature space, M , rather naturally – one keeps on adding new feature dimensions until there is no significant change in performance (e.g., normalized mean-squared error). One important issue which arises involves the initialization of w_m at step m . In our simulations we chose to initialize the weights as those obtained from a linear regression of X onto the target/residual vector y^m . This method acts as an educated guess expecting a faster convergence rate. We call this algorithm, which learns W and θ , projection pursuit GP regression (PPGPR). To be clear, note that PPGPR is used for the purposes of learning W and θ only. Once this is complete, one can simply run the VB E-step to compute predictions. The PPGPR algorithm offers

a bridge between the flexibility and elegance of the naive full GP, and the efficiency of its approximate additive counterpart. Its strength lies not only in the connection to backfitting, but also in its substantial improvement in performance and efficiency, as will be shown in Section 3.

2.1.4 Parallelization of State-Space Models

The above algorithms can be parallelized to achieve even more speed up. By transforming the full GP to a state-space model formulation, we replaced calculating an inverse of a large joint covariance matrix to that of manipulating much smaller evolution and emission matrices (Φ , Q), for all input locations. As these matrices are functions of the hyperparameters, they must be recalculated in every iteration during the hyperparameters learning stage. Notice, however, that for a fix set of hyperparameters, the values of Φ , and Q for all locations are independent, and hence can be precalculated in parallel. We used a very simple parallelization scheme across (up to) 8 worker threads. We will further discuss the gains this in Section 3.1. It is important to note that as the speed of the CPUs has come to a halt and the number of cores is on the rise, the ability to use parallel schemes will be a must for any efficient GP algorithm in the future.

2.2 Generalized Additive GP Regression

So far we have shown new methods for scaling multidimensional GP regression. Here, we extend the efficiency of these methods to include problems that need non-Gaussian likelihood functions, such as classification. Being able to efficiently handle large multidimensional datasets of non-Gaussian distributed targets is especially important in classification problems, which regularly have multiple input features, and a discrete label. Inference and learning with non-Gaussian emissions necessitates the use of approximations to the true posterior, e.g., expectation propagation (EP) [20], and Laplace approximation (LA) [2]. However, only a handful of works in the literature focus on tractable algorithms for big data where exact GP classification is intractable. These works are extensions of the sparse GP framework using various approximation methods [21], [22], [23]. Sparse GP classification, however, is not without problems: for example, pseudo input learning with EP and hyperparameter learning expand the parameter space making the problem more susceptible to overfitting. Other works which use a subset of data are more stable; however, this can affect accuracy as the active subset may not be optimal for the sparse conditional independence assumption. Additionally, there is the problem of model selection for the number of points to use in the sparse active set. Thus, extending our structured GP model to the non-Gaussian case is necessary and useful.

Here we derive, for additive structured GP kernels, an $\mathcal{O}(N)$ algorithm which performs MAP inference and

hyperparameter learning using Laplace’s approximation. The resulting LA algorithm is both stable and accurate. Note, that although it has been suggested in literature that the EP approximation has superior approximation qualities (for binary classification [24]), the choice of LA is due to its computational benefits as, to our knowledge, there is no good method for lowering the computational complexity of the EP updates from $\mathcal{O}(N^3)$. We now derive this LA algorithm, and then we end this section with another key insight showing that this algorithm is a Bayesian interpretation of another classical technique known as local scoring [25].

Given the likelihood $p(\mathbf{y}|\mathbf{f})$ is non-Gaussian, we use the standard Laplace approximation:

$$p(\mathbf{f}|\mathbf{y}, X, \theta) \approx \mathcal{N}(\hat{\mathbf{f}}, \Lambda^{-1}), \quad (23)$$

where $\hat{\mathbf{f}} \equiv \arg \max_{\mathbf{f}} p(\mathbf{f}|\mathbf{y}, X, \theta)$ and the approximated covariance matrix $\Lambda \equiv -\nabla \nabla \log p(\mathbf{f}|\mathbf{y}, X, \theta)|_{\mathbf{f}=\hat{\mathbf{f}}}$. We define the following objective:

$$\Omega(\mathbf{f}) \equiv \log p(\mathbf{y}|\mathbf{f}) + \log p(\mathbf{f}|X, \theta). \quad (24)$$

$\hat{\mathbf{f}}$ is found by applying Newton’s method to this objective. Newton’s method is guaranteed to converge to the global optimum, given the objective in Eq. (24) is convex with respect to \mathbf{f} . This is the case for many cases of generalized regression, as the likelihood term is usually log-concave, as well as the prior term.

If we assume that \mathbf{f} is drawn from an additive GP, then it follows that the required gradient and Hessian (for Newton iterations) are:

$$\nabla \Omega(\mathbf{f}) = \nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f}) - \mathbf{K}_{\text{add}}^{-1} \mathbf{f}, \quad (25)$$

$$\nabla \nabla \Omega(\mathbf{f}) = \underbrace{\nabla \nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})}_{\equiv -\mathbf{W}} - \mathbf{K}_{\text{add}}^{-1}. \quad (26)$$

This makes the Newton iteration:

$$\mathbf{f}^{(k+1)} \leftarrow \mathbf{f}^{(k)} + (\mathbf{K}_{\text{add}}^{-1} + \mathbf{W})^{-1} \cdot \left(\nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})|_{\mathbf{f}^{(k)}} - \mathbf{K}_{\text{add}}^{-1} \mathbf{f}^{(k)} \right), \quad (27)$$

$$= \mathbf{K}_{\text{add}} (\mathbf{K}_{\text{add}} + \mathbf{W}^{-1})^{-1} \cdot \left[\mathbf{f}^{(k)} + \mathbf{W}^{-1} \nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})|_{\mathbf{f}^{(k)}} \right]. \quad (28)$$

Since the generalized additive GP model assumes conditional independence of \mathbf{y} given \mathbf{f} , $p(\mathbf{y}|\mathbf{f}) = \prod_{i=1}^N p(y_i|f_i)$, and \mathbf{W} is a diagonal matrix and therefore easy to invert. Looking closer at Eq. (28), we see that it is *precisely* the same as the expression to compute the posterior mean of a GP, where the target vector is given by $\left[\mathbf{f}^{(k)} + \mathbf{W}^{-1} \nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})|_{\mathbf{f}^{(k)}} \right]$ and where the diagonal “noise” term is given by \mathbf{W}^{-1} . Given an additive kernel corresponding to a sum of scalar GPs that can be represented using state-space models, we can therefore use Algorithm 1 to implement a single iteration of Newton’s method. As a result, it is possible to compute $\hat{\mathbf{f}}$ in $\mathcal{O}(N)$ time, since in practice only a handful of Newton iterations are required for convergence. Wrapping backfitting

iterations inside a global Newton iteration is precisely how the local-scoring algorithm is run to fit a generalized additive model [25]. Thus, we can view the development in this section as a novel Bayesian interpretation of local scoring.

The above calculates the posterior Laplace approximation. To efficiently approximate the marginal likelihood, we use the Taylor expansion of the objective function $\Omega(\mathbf{F})$, although we will need to express it explicitly in terms of $\mathbf{F} \equiv [\mathbf{f}_1; \dots; \mathbf{f}_D]$, as opposed to the sum over \mathbf{f} .

$$\Omega(\mathbf{F}) = \log p(\mathbf{y}|\mathbf{F}) + \log p(\mathbf{F}|\mathbf{X}, \theta). \quad (29)$$

Once $\hat{\mathbf{F}}$ is known, it can be used to compute the approximation to the marginal likelihood. Using first-order Taylor expansion we obtain:

$$\begin{aligned} \log p(\mathbf{y}|\mathbf{X}) &\approx \Omega(\hat{\mathbf{F}}) - \frac{1}{2} \log \det \left(\tilde{\mathbf{W}} + \tilde{\mathbf{K}}^{-1} \right) + \frac{ND}{2} \log(2\pi) \end{aligned} \quad (30)$$

$$\begin{aligned} &= \log p(\mathbf{y}|\hat{\mathbf{F}}) - \frac{1}{2} \hat{\mathbf{F}}^\top \tilde{\mathbf{K}}^{-1} \hat{\mathbf{F}} \\ &\quad - \frac{1}{2} \log \det \left(\tilde{\mathbf{K}} + \tilde{\mathbf{W}}^{-1} \right) - \frac{1}{2} \log \det(\tilde{\mathbf{W}}), \end{aligned} \quad (31)$$

where $\tilde{\mathbf{K}}$ is a block diagonal tiling of $\mathbf{K}_1, \mathbf{K}_2, \dots, \mathbf{K}_D$, and $\tilde{\mathbf{W}}$ is a block diagonal tiling of the single \mathbf{W} matrix. We used the matrix determinant lemma to get from Eq. (30) to Eq. (31). Importantly, all the terms in Eq. (31) can be computed in $\mathcal{O}(N)$ runtime due to the fact that the latent function can be represented as a sum of state-space models (for more details see [5]).

In summary, we showed that by using the Laplace approximation, we are able to maintain low runtime complexity by combining a Newton method with additive regression update in Eq. (28) (local scoring), and by approximating the marginal likelihood using Eq. (30).

2.3 Gaussian Processes on Multidimensional Grids

The second GP kernel structure that can be exploited involves the assumption of equispaced inputs. This is commonly seen for GP regression in time and space (e.g., regular measurements at evenly spaced weather stations, or video captured by a CCD camera). Even though there are good Toeplitz methods for scalar equispaced inputs [26], the extension to a multidimensional grid has not been addressed in literature. In this section, we present a novel method to perform exact inference in $\mathcal{O}(N)$ time for any tensor product kernel (most commonly-used kernels are of this form), using properties of Kronecker products.

In this case, we can compute all the computationally troublesome quantities involved (such as $(\mathbf{K}_N + \sigma_n^2 \mathbf{I}_N)^{-1} \mathbf{y}$) using a few matrix-vector products (of size N). Importantly, these matrix-vector products have the form:

$$\boldsymbol{\alpha} = \left(\bigotimes_{d=1}^D \mathbf{A}_d \right) \mathbf{b}, \quad (32)$$

all of which can be computed in linear runtime.

Computing α using standard matrix-vector multiplication is an $\mathcal{O}(N^2)$ operation. However, with problems of this form, it is possible to attain linear runtime using tensor algebra [27]. For the relevant background in tensor algebra, see the appendix in our supporting work [5]. The product in Eq. (32) can be viewed as a tensor product between the tensor $\mathbf{T}_{i_1, j_1, \dots, i_D, j_D}^A$ representing the outer product over $[\mathbf{A}_1, \dots, \mathbf{A}_D]$, and $\mathbf{T}_{j_D, \dots, j_1}^B$. The term $\mathbf{T}_{j_D, \dots, j_1}^B$ represents the length- N vector \mathbf{b} . Conceptually, the aim is to compute a contraction over the indices j_1, \dots, j_D , namely:

$$\mathbf{T}^\alpha = \sum_{j_1=1}^{G_1} \dots \sum_{j_D=1}^{G_1} \mathbf{T}_{i_1, j_1, \dots, i_D, j_D}^A \mathbf{T}_{j_D, \dots, j_1}^B, \quad (33)$$

where \mathbf{T}^α is the tensor representing the solution vector α . As the sum in Eq. (33) is over $N = \prod_{d=1}^D G_d$ elements, it will run in $\mathcal{O}(N)$ time. Equivalently, we can express this operation as a sequence of matrix-tensor products and tensor transpose operations.

$$\alpha = \text{vec} \left(\left(\mathbf{A}_1 \dots \left(\mathbf{A}_{D-1} \left(\mathbf{A}_D \mathbf{T}^B \right)^\top \right)^\top \right)^\top \right), \quad (34)$$

where we define matrix-tensor products of the form $\mathbf{Z} = \mathbf{X}\mathbf{T}$ as:

$$\mathbf{Z}_{i_1 \dots i_D} = \sum_k^{\text{size}(\mathbf{T}, 1)} \mathbf{X}_{i_1 k} \mathbf{T}_{k i_2 \dots i_D}. \quad (35)$$

The operator \top is assumed to perform a cyclic permutation of the indices of a tensor, namely

$$\mathbf{Y}_{i_D i_1 \dots i_{D-1}}^\top = \mathbf{Y}_{i_1 \dots i_D}. \quad (36)$$

Furthermore, when implementing the expression in Eq. (34) it is possible to represent the tensors involved using matrices where the first dimension is retained and all other dimensions are collapsed into the second, resulting in a matrix \mathbf{B} which is a G_d -by- $\prod_{j \neq d} G_j$ matrix, where G_d is the number of elements in dimension d . Algorithm 2 gives pseudo-code illustrating these steps. In short, we use tensor algebra to create a linear-runtime method (Algorithm 2) for doing matrix-vector multiplications across a Kronecker product matrix, which arises quite naturally for most GP kernels on a grid of inputs.

The critical second step is to note that $(\mathbf{K} + \sigma_n^2 \mathbf{I}_N)^{-1}$ cannot itself be written as Kronecker product, due to the perturbation on the main diagonal. Nevertheless, it is possible to sidestep this problem using the eigendecomposition properties and the identity

$$(\mathbf{K} + \sigma_n^2 \mathbf{I}_N)^{-1} \mathbf{y} = \mathbf{Q} (\mathbf{\Lambda} + \sigma_n^2 \mathbf{I}_N)^{-1} \mathbf{Q}^\top \mathbf{y}. \quad (37)$$

Importantly, the eigenvector matrix \mathbf{Q} will also be a Kronecker product. Hence, to then efficiently solve Eq. (37), we first evaluate and perform eigendecompositions of covariances along individual dimensions to get $[\mathbf{Q}_d, \mathbf{\Lambda}_d]$. This has complexity $\mathcal{O}((\max_d G_d)^3)$, which is negligible

Algorithm 2: Efficient matrix-vector multiply for Kronecker matrices

inputs : D matrices $[\mathbf{A}_1 \dots \mathbf{A}_D]$, length- N vector \mathbf{b}
outputs: α , where $\alpha = \left(\bigotimes_{d=1}^D \mathbf{A}_d \right) \mathbf{b}$

```

1  $\mathbf{x} \leftarrow \mathbf{b}$ ;
2 for  $d \leftarrow D$  to 1 do
3    $G_d \leftarrow \text{size}(\mathbf{A}_d)$ ;
4    $\mathbf{X} \leftarrow \text{reshape}(\mathbf{x}, G_d, N/G_d)$ ;
5    $\mathbf{Z} \leftarrow \mathbf{A}_d \mathbf{X}$   $\triangleright$  Matrix-tensor product
6    $\mathbf{Z} \leftarrow \mathbf{Z}^\top$   $\triangleright$  Tensor rotation
7    $\mathbf{x} \leftarrow \text{vec}(\mathbf{Z})$ ;
8 end
9  $\alpha \leftarrow \mathbf{x}$ ;

```

compared to $N = \prod_{d=1}^D G_d$. Next, we calculate Eq. (37) in three steps:

$$\alpha \leftarrow \text{kron_mvprod}([\mathbf{Q}_1^\top, \dots, \mathbf{Q}_D^\top], \mathbf{y}), \quad (38)$$

$$\alpha \leftarrow (\mathbf{\Lambda} + \sigma_n^2 \mathbf{I}_N)^{-1} \alpha, \quad (39)$$

$$\alpha \leftarrow \text{kron_mvprod}([\mathbf{Q}_1, \dots, \mathbf{Q}_D], \alpha), \quad (40)$$

where we efficiently used `kron_mvprod` (Alg. 2) twice and noting that the matrix $\mathbf{\Lambda} + \sigma_n^2 \mathbf{I}_N$ is easy to invert as it is diagonal. Computation of the test set predictions $\mathbf{Q} (\mathbf{\Lambda} + \sigma_n^2 \mathbf{I}_N)^{-1} \mathbf{Q}^\top \mathbf{K}_{MN}$, can be done efficiently using the same approach. The result is the third main contribution of this paper: a linear-runtime¹ method for calculating the key regression equation $(\mathbf{K} + \sigma_n^2 \mathbf{I}_N)^{-1} \mathbf{y}$ using only the assumption that the inputs lie on a grid.

In summary, we exploited two important realizations: efficient eigendecomposition using properties of the Kronecker product, and tensor products enabling fast multiplication by matrices that can be written as a Kronecker product. This novel improvement for exact GP inference opens the door to a whole new set of applications, which would have never been considered otherwise, such as GP on images or videos.

3 RESULTS

Here we will compare the algorithms discussed in the paper to other commonly used algorithms both in the GP world, and other common machine learning techniques.

3.1 Multidimensional Regression

In this section we will compare methods for multidimensional regression on both simulated and real experimental data. For each experiment presented, we will compare both runtime and accuracy. If a particular algorithm has a stochastic component to it (e.g., if it involves MCMC) its performance will be averaged over 10 runs. Every

1. The complexity of the algorithm is $\mathcal{O}((\log_D N)^3 N)$, however it rapidly converges to $\mathcal{O}(N)$ as the number of dimensions grows.

experiment was composed of training (i.e., smoothing and hyperparameter learning given $\{\mathbf{X}, \mathbf{y}\}$) and testing phases. In each experiment, we used 1000 points for test sets.

In terms of accuracy, we use two standard performance measures: normalized mean square error (NMSE) and test-set Mean Negative Log Probability (MNLP).

$$\text{NMSE} = \frac{\sum_{i=1}^{N_*} (\mathbf{y}_*(i) - \boldsymbol{\mu}_*(i))^2}{\sum_{i=1}^{N_*} (\mathbf{y}_*(i) - \bar{y})^2},$$

$$\text{MNLP} = \frac{1}{2N_*} \sum_{i=1}^{N_*} \left[\frac{(\mathbf{y}_*(i) - \boldsymbol{\mu}_*(i))^2}{\mathbf{v}_*(i)} + \log \mathbf{v}_*(i) + \log 2\pi \right],$$

where $\boldsymbol{\mu}_* \equiv \mathbb{E}(\mathbf{f}_* | \mathbf{X}, \mathbf{y}, X_*, \theta)$, $\mathbf{v}_* \equiv \mathbb{V}(\mathbf{f}_* | \mathbf{X}, \mathbf{y}, X_*, \theta)$, and \bar{y} is the training-set average target value. These measures have been chosen to be consistent with those commonly used in the sparse GP regression literature. We compare runtime performance in seconds, taking into account both the learning and prediction phases.

We test the following algorithms (with the following names): the full naive GP implementation (Full GP), additive models (Sections 2.1.1 and 2.1.2) using VBEM inference (Additive-VB) and the MCMC inference (Additive-MCMC), projected additive models using greedy projection pursuit of Section 2.1.3 (PPGPR-Greedy) and a variation of MCMC (PPGPR-MCMC). Finally, for the sparse GP method we used the sparse pseudo-input Gaussian process (SPGP) [18]. However, to be conservative, we did not learn the pseudo inputs (which can potentially greatly increase the algorithm complexity and runtime) but rather used a random subset of the inputs as the active set. For both the SPGP and the Full-GP, we used the GPML Matlab Code version 3.1 [28]. Also note that, for Additive-VB and PPGPR-greedy we have set the number of outer loop iterations (the number of VBEM iterations for the former, and the number of projections for the latter) to be at maximum 10 for all N . Increasing this number increased the cost with no change to accuracy, so this is a reasonable choice. All algorithms were run both as a single thread and using a parallel multicore, but since SPGP and Full-GP do not offer efficient implementation of the parallel schemes, their results were the same for both cases².

3.1.1 Synthetic Data Experiments

First we used synthetic data generated by the following model:

$$y_i = \sum_{d=1}^D \mathbf{f}_d(x_{:,d}) + \epsilon \quad i = 1, \dots, N, \quad (41)$$

$$\mathbf{f}_d(\cdot) \sim \mathcal{GP}(\mathbf{0}; k_d(\mathbf{x}_d, \mathbf{x}'_d; [1, 1])) \quad d = 1, \dots, D, \quad (42)$$

$$\epsilon \sim \mathcal{N}(0, 0.01),$$

2. When discussing parallel schemes we refer to only the learning stage. As in all GP frameworks, parallelism can always be used for prediction, since we are only interested in the predictive marginals per test point. However, this does not have any noticeable effect on the runtime and is thus unimportant to the comparison.

where $k_d(\mathbf{x}_d, \mathbf{x}'_d; [1, 1])$ is given by the Matérn(7/2) kernel with unit lengthscale and amplitude. We used $D = 8$ dimensions, and collected runtimes for a set of values for N ranging from 1000 to 50000.

Figure 3 illustrates the significant computational savings attained by exploiting the structure of the additive kernel. To find the relationship between the number of inputs to the runtime, we calculated a linear slope of the data in the log-log scale. As expected, the slope of the Full-GP is close to three due to its cubic complexity, and all the approximation algorithms have runtimes that scale linearly with the input size. We can also see that parallel processing of the state-space model matrices offers further improvement in scaling. These results serve only as a rough estimate, because the performance can depend on the chosen algorithm parameters, such as: number of outer loop iterations in the Additive-VB, number of projections in PPGPR-greedy, or number of samples in the MCMC methods. This runtime/accuracy consideration should be used when comparing the efficiency of the algorithms.

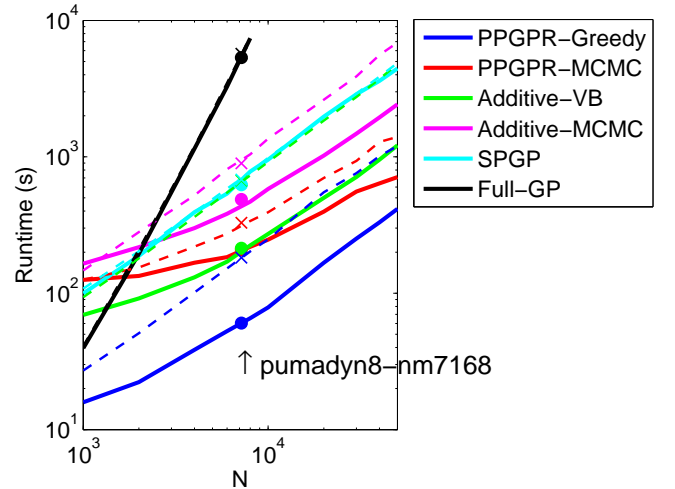


Fig. 3: A comparison of runtimes for efficient Bayesian additive GP regression, with $D = 8$, $N = [2; 4; 6; 8; 10; 20; 30; 40; 50] \times 10^3$, presented as a log-log plot. The algorithms ran on a Linux server, once as a single thread (dash lines) and once in a multicore parallel scheme using 8 processors (solid lines). At $N=7168$, we added an overlay of the runtime results for the pumadyn8-nm dataset (described in Section 3.1.2) for both single ('x') and multicore ('o') runs.

Additionally, runtime on a modern computer is by no means a perfect measure of algorithmic complexity. Nonetheless, we will see that the results of Fig. 3 agree with all the results from the real datasets. For example, in Fig. 3 we overlay the results of one of the real datasets, and one sees a close correspondence between synthetic and real data. Thus, these and subsequent results are highly representative and assert the primary point of this section: the runtime of our approximation algorithms do

indeed scale linearly with N , versus the cubic scaling of the naive GP implementation, not GP-grid.

Fig. 4 shows the effects of increased dimensionality on the approximate algorithms. In this figure we show the runtime speedup of the algorithms with respect to the runtime of the Full-GP on the synthetic data generated with dimensionality of either $D = 8$ or $D = 32$. In all the runs the number of inputs was set to $N = 8000$, and the algorithms were run once with a single thread (1 worker = 1W), and once using the parallel scheme (8 workers = 8W). In the multidimensional case, the projection pursuit algorithm exhibits the largest speedup, as it allows for a reduction in the number of effective dimensions (via the greedy selection). Notably, PPGPR-Greedy achieves consistently an order of magnitude improvement over SPGP.

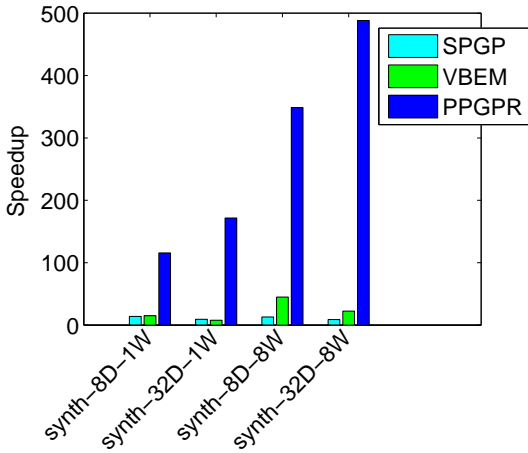


Fig. 4: A comparison of the speed up offered by the approximation algorithms compared with exact GP. The runtime was measured on the learning stage for three approximation algorithms: sparse GP, Additive VB, and greedy Projection-Pursuit. The comparison was done using synthetic results with different dimensions (8D and 32D), and running on both a single and multicore (8-core) computer. As can be seen from the figure, the greedy Projection-Pursuit offers the highest speedup, and is especially efficient in high dimensions.

3.1.2 Real Data Experiments

Next, we extend the comparison to real datasets, which will allow thorough accuracy comparisons. We test over seven well-known datasets. These data sets are: synth-8D ($N = 8000$ synthetic data from Section 3.1.1). Next, the pumadyn family is a robotic arm dataset from [2], and consist of three datasets: pumadyn8-fm1000 ($N = 1000$, fairly linear data with $D = 8$ dimensions), pumadyn8-fm7168 ($N = 7168$, fairly linear data with $D = 8$ dimensions), pumadyn32-nm ($N = 7168$, highly nonlinear data with $D = 32$). Elevators dataset consists of the current state of the f16 aircraft ($N = 8752$, 17-dimensional) [29], and kin40k is a highly nonlinear

dataset ($N = 10000$, 8-dimensional) as introduced in [30]. Fig. 5 demonstrates the central analysis of this section. In each subplot, we calculate speedup, MNLP, and NMSE across all seven datasets and six algorithmic options. To reiterate, the two additive and two PPGPR algorithms are our advances of Section 2.1, and our main comparison points are SPGP [18] and a naive full GP implementation. The top subplot in Fig. 5 indicates the substantial speedups offered by all algorithms over the full GP, with the exception only of the $N = 1000$ dataset (pumadyn8-fm1000; this is not surprising given small N). Further, as indicated in Figure 3, our PPGPR-Greedy achieves the largest speedup across all datasets, and in most cases the error (MNLP and NMSE) is the same as competing methods. The first four or five datasets tell a very similar accuracy story across PPGPR-Greedy, SPGP, and the full GP. We also see that the simple additive models almost always underperform in accuracy, which is as expected given their limited expressivity compared to PPGPR-Greedy. The one exception where Additive-VB outperforms PPGPR-Greedy is the synthetic data set. However, this is expected as we used an additive model to generate data and the greedy nature of PPGPR-Greedy causes it to underperform. In the final two datasets, we see that SPGP and the full GP have considerably better accuracy. This may be explained as both these datasets are highly nonlinear, making the additive assumption inaccurate.

Understanding the runtime-accuracy tradeoff based on problem requirements is essential. As we just described, PPGPR-greedy achieves the best runtimes but at times with an accuracy cost. Thus we want to quantify the notion of a runtime-accuracy tradeoff. To do so we plot all data sets and algorithms in a runtime vs. error plot (Fig. 6), and we use the economics concept of Pareto efficiency: *efficient* points in the runtime vs error plot represent algorithms with minimum runtime for a given error rate. Pareto inefficient algorithms are then those points that are unambiguously inferior. The efficient frontier is the convex hull of all {runtime,error} points (algorithms) for a given dataset. This will give us a clear picture of which algorithms are optimal choices across a range of datasets. Fig. 6 details this, with one efficient frontier for each dataset (a given color). Each algorithm has a given marker type. This immediately shows what one would expect: the full GP implementation is typically most accurate, but only if one is willing to invest substantial runtime. This choice is often Pareto efficient. Secondly, most often the PPGPR-greedy is the other efficient choice for a substantially reduced runtime, albeit higher error. Surprising to note is the relative weakness of SPGP over several datasets.

In Table 1 we count of the number of datasets where each algorithm is on the efficient frontier, which gives an idea of how often an algorithm is competitive with others, or optimal given a particular runtime or error budget. Three algorithms stand out in their overall efficiency: PPGPR-Greedy, SPGP, and full GP. The PPGPR-Greedy

is the *only* consistent efficient algorithm for *all* datasets as it achieves the fastest runtime. However, more interestingly, it also achieves very good accuracy results making most other algorithms inefficient. Of course, any trivial algorithm could achieve efficiency by having minimal runtime and arbitrary error, but the data demonstrates that this is not the case with our algorithms: the PPGPR-greedy error in almost all datasets is competitive or better than all alternatives. Thus the frequent efficiency of PPGPR-greedy is legitimate.

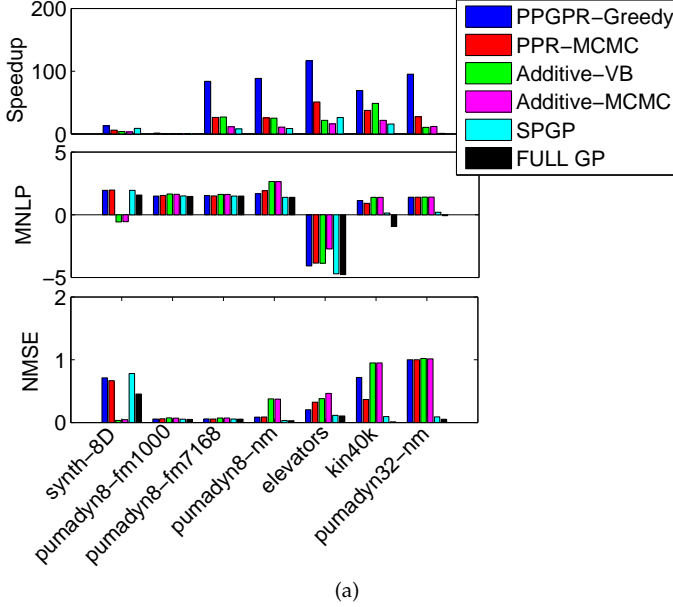


Fig. 5: These figures offer a comparison between the different GP methods discussed in the text, taking into account both speedup and accuracy. For comparison we used several known datasets from literature and ran the algorithms on a multicore (8-core) computer. The top figure illustrates the speedup of the approximation algorithms runtimes with respect to the full GP (exact inference) runtime. The bottom two figures show two metrics for calculating regression accuracy.

TABLE 1: Efficiency comparison, showing the number of datasets where the algorithm was on the Pareto efficient frontier. There were seven datasets tested.

Algorithm	Pareto Efficient Frontier Count
PPGPR-Greedy	7
PPGPR-MCMC	0
Additive-VB	1
Additive-MCMC	0
SPGP	4
Full-GP	6

3.2 Multidimensional Classification

In this section we will compare the generalized additive-GP from Section 2.2 to other kernel classifiers (both

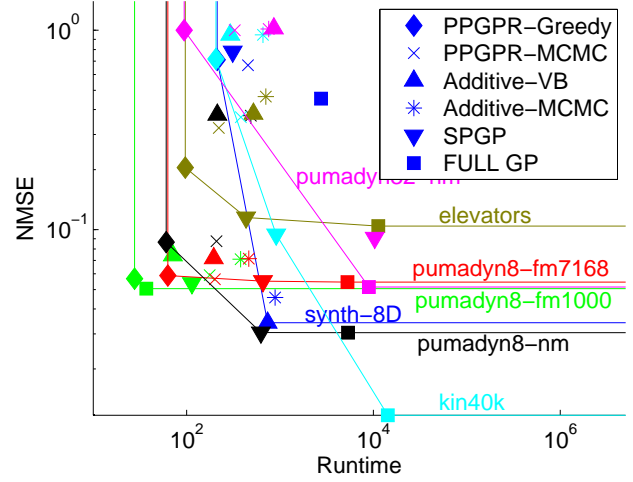


Fig. 6: The two fundamental desiderata of our algorithms are accuracy and speed. Here we plot error vs runtime to quantify the tradeoff between these two objectives using the notion of Pareto efficiency. Every algorithm is represented using a unique marker and with a color scheme chosen according to the datasets. For each dataset, the Pareto efficient frontier is shown as a color line passing through the efficient algorithms for that dataset.

Bayesian and non-Bayesian). We use common performance metrics from the sparse GP classification literature, enabling straightforward comparison with other experimental results. In this paper we will focus on the task of binary classification, however in principle, extensions to tasks such as multi-class classification and Poisson regression can be performed without affecting asymptotic complexity. For performance measures we use: algorithm runtime (in seconds), error rate, and the test-set mean negative log-likelihood (MNLL):

$$\text{Error Rate} = \frac{\#(\text{incorrect classifications})}{\#(\text{test cases})}, \quad (43)$$

$$\text{MNLL} = \frac{1}{N_*} \sum_{i=1}^{N_*} [y_i \log \hat{p}_i + (1 - y_i) \log(1 - \hat{p}_i)]. \quad (44)$$

For both the test error rate and MNLL measures lower values indicate better performance.

3.2.1 Synthetic Data Experiments

We used synthetic data generated by the following model:

$$y_i \sim \text{Bernoulli}(p_i) \quad i = 1, \dots, N, \quad (45)$$

$$p_i = g(f_i), \quad (45)$$

$$\mathbf{f}(\cdot) = \sum_d \mathbf{f}_d(\cdot), \quad (46)$$

$$\mathbf{f}_d(\cdot) \sim \mathcal{GP}(\mathbf{0}; k_d(\mathbf{x}_d, \mathbf{x}'_d; \theta_d)) \quad d = 1, \dots, D,$$

where $g(f_i)$ is the logistic link function.

Within the GP framework, we compared generalized additive GP Regression from Section 2.2 (Additive-LA), standard GP classification with Laplace's approximation

(Full-GP) [2], and sparse GP methods of informative vector machine (IVM) [23] and fully independent conditional (FIC) [1]. For completeness, we also include support vector machines (SVM) [31].³ For the Full-GP we used GPML Matlab Code version 3.1 [28]; for FIC we used the GPstuff Matlab package [33]; for SVM we used LIBSVM [34]; and for IVM we used the implementation given in [23]. We tested the algorithms on the synthetic data from the model above using 8 dimensions while varying the number of inputs $N = [2; 4; 6; 8; 10; 20; 30; 40; 50] \times 10^3$. We stopped running the Full-GP at 10000 as it took too long to finish. A comparison of the runtime results is shown in Fig. 7. To be consistent, we used exactly 25 iterations for all algorithms during the learning stage. As can be seen from the figure, Additive-GP offers excellent scaling for large input sizes. The only algorithm that offers faster runtime than the additive-GP is IVM. This can be expected as the IVM only uses the information in the active set and discards the rest. Our algorithm, on the other hand, makes use of all the data, and is thus able to achieve a more accurate estimation, as the results below demonstrate.

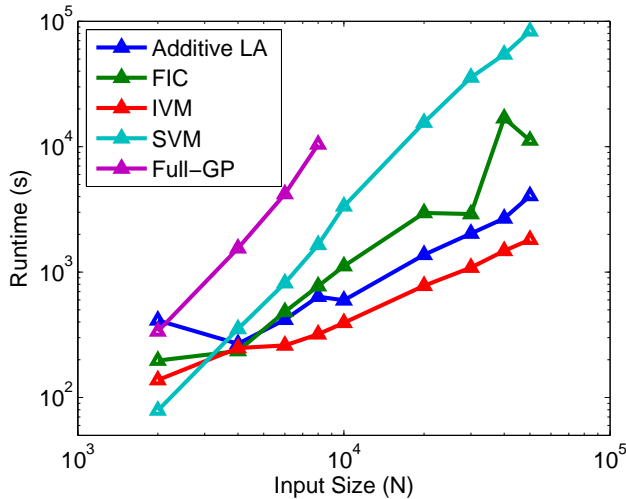


Fig. 7: This figure shows the runtime of the classification algorithms for the synthetic dataset with $D = 8$, $N = [2; 4; 6; 8; 10; 20; 30; 40; 50] \times 10^3$. For the learning stage we used 50 iterations, and we did prediction on 1000 points. The log-log slopes of the algorithms are: Full-GP = 2.75, Additive-GP = 1.07, FIC = 1.53, IVM = 0.80, SVM = 2.16.

3.2.2 Real Data Experiments

We tested the classification algorithms from the previous section on three additional popular datasets: Breast Cancer [35], Magic Gamma Telescope [36], and IJCNN [37]. We again only allowed 25 iterations in the learning stage. For sparse methods we tested two activeset sizes: 50, and $0.1N$. Table 2 summarizes the

classification results across all algorithms and datasets. Each column gives the classification error rate, MNLL, and runtime. First, we consider the runtime. We note that, as expected from Figure 7, the Full GP has the least attractive runtime in all but the Breast Cancer data (due to the small data size). The IVM has the best runtime performance across all datasets, after which our Additive-LA method is superior. In terms of performance, error rates are fairly consistent throughout the data sets, with the exception of notably high errors for IVM in the last two data sets. This is echoed by MNLL, where the IVM tends to have a significantly larger error. These results are not as compelling as in the regression case (as is often the case when comparing Bayesian methods to an SVM), but the Additive-LA is competitive overall. Thus, if a Bayesian method is needed for nonparametric classification, the Additive-LA approach is a viable and stable solution.

TABLE 2: Performance Comparison of efficient Bayesian additive GP classification algorithms with commonly-used classification techniques on larger datasets.

Algorithm	Error Rate	MNLL	Runtime (s)
<i>Synthetic Additive Data ($N = 4000, M = 1000, D = 8$)</i>			
Full-GP	0.6040	0.7402	2244.5111
Additive-LA	0.2800	0.5929	161.1185
FIC - 50	0.2800	1.0640	525.6684
FIC - 400	0.4550	1.3612	850.8427
IVM - 50	0.2800	0.6931	65.1951
SVM	0.2940	0.5838	345.7267
<i>Breast Cancer ($N = 359, M = 90, D = 9$)</i>			
Full-GP	0.0667	0.1436	6.0435
Additive-LA	0.0667	0.1215	89.2993
FIC - 50	0.0556	0.0999	27.9584
FIC - 36	0.0556	0.0999	55.9182
IVM - 50	0.0667	0.6382	12.4867
SVM	0.0556	3.1717	1.7062
<i>Magic Gamma Telescope ($N = 15216, M = 3804, D = 10$)</i>			
Full-GP	NA	NA	NA
Additive-LA	0.1393	0.3419	2345.6546
FIC - 50	0.1441	0.3656	3339.6185
FIC - 1522	0.1396	0.3654	7331.2780
IVM - 50	0.6583	0.6932	118.0407
SVM	0.1191	0.3026	8070.2400
<i>IJCNN ($N = 49990, M = 91701, D = 13$)</i>			
Full-GP	NA	NA	NA
Additive-LA	0.0516	0.1560	14505.5000
FIC - 50	0.0482	1.1859	4390.2498
FIC - 4999	0.0770	0.8171	16728.0911
IVM - 50	0.0950	0.6932	369.0000
SVM	0.0166	0.0509	22170.1000

3.3 Multidimensional Regression on a Grid

In this section we consider data with input on a multidimensional grid. We compare the exact GP-grid method

3. To calculate the MNLL, we used the probabilistic predictions from the SVM using cross-validation and cross-entropy metric [32]

from Section 2.3 to the naive Full-GP method and show an application for this method in image reconstruction.

We first start by comparing the runtime complexity of the GP-grid to Full-GP. For this we ran both algorithms on synthetic data where each synthetic dataset has input locations at the corners of the $\{-1, 1\}$ hypercube in D dimensions. The target value is set to noise. At each run we increased the dimension D by 1, thereby multiplying the number of input points by 2. Fig. 8 illustrates the time it takes for one iteration during the learning stage. As can be seen, the GP-grid scales linearly with the input size while the Full-GP is cubic.

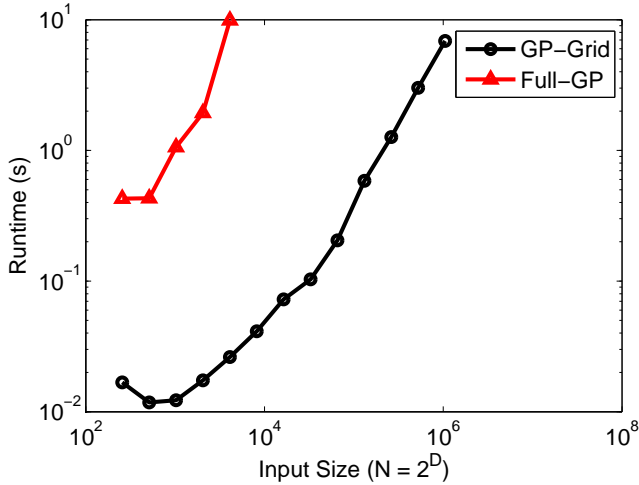


Fig. 8: Runtimes of naive Full-GP in red and GP regression on Grid using Kronecker product (GP-Grid from Sec. 2.3) in black, for $N = [2^8; 2^9; \dots; 2^{20}]$. The slope for the naive Full-GP is 2.6 and that of Kronecker product is 1.05 (based on last 7 points). This empirically verifies the improvement to linear scaling.

Improving the scalability of exact GP on a multi-dimensional grid may seem like an unusual special case, but it importantly enables new applications which were not tractable previously. One such application is picture reconstruction as pictures are an equispaced grid of pixels. Here we show how GP-grid can be used to reconstruct and interpolate a noisy image. We first apply GP-grid on a 200×200 pixel noisy image (Fig. 9a) and use the inferred GP posterior as a denoised reconstruction (Fig. 9b). Note that the GP nicely smooths out most of the compression artifacts that were present in the original image. Note also that this is a GP on $N = 40000$, which is hopelessly intractable for Full-GP. However, GP-grid was able to learn the parameters in 42.75 seconds, and denoise in 1490.3 seconds. Further, because the two methods are mathematically proven to be identical, any small numerical differences will be due to relative instability of the naive GP implementation.

Next, we down-sampled the original image by $\frac{1}{4}$ (Fig. 9c) and used GP-grid to interpolate the missing values (Fig. 9d). Here the learning stage was 4.57 seconds and

the reconstruction with interpolation was 186.61 seconds, and the overall quality of the interpolated reconstruction is still high. This image example is only a single example, but it is representative: due to the Kronecker method of Section 2.3, we have a provably exact GP method that scales linearly in the number of data points. Image and video analysis is a critical and common machine learning application, but the use of nonparametric Bayesian algorithms in this domain is infrequent. Our GP-grid algorithm importantly enables the use of GP technologies in this application area.

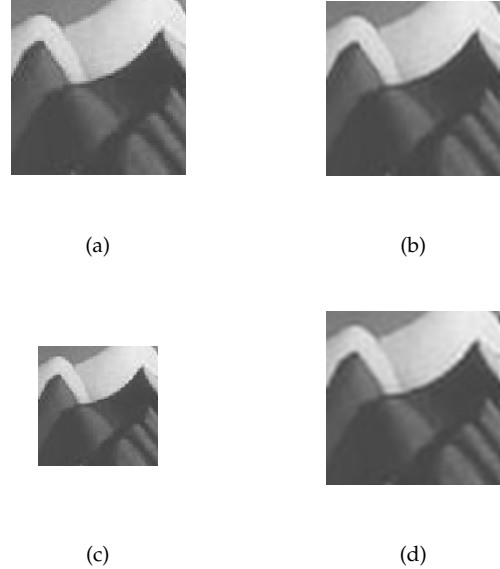


Fig. 9: Illustration of GP regression on a picture as an equidistance grid. Fig. 9a shows a 200×200 pixels of the original noisy picture. in Fig. 9b we used the GPR-grid method from Section 2.3 to learn the parameters and reconstruct the picture. Fig. 9c is a down-sampled the original picture (Fig. 9a) by 2 in both dimensions ($\frac{1}{4}$ of the data). Fig. 9d shows a reconstruction of the picture where the GPR-grid used the down-sampled data for learning, and then predicted the values at missing locations.

4 DISCUSSION AND CONCLUSION

Gaussian Processes are perhaps the most popular non-parametric Bayesian method in machine learning, but their adoption across other fields - and notably in application domains - has been limited by their burdensome scaling properties. Having fast, scalable methods for Gaussian Processes may mean the difference between a theoretically interesting approach and a method that is widely used in practice.

While important sparsification work has somewhat addressed this scalability issue, the problem is by no means closed. Our aim here has been to explore the use of structured GP models. We made nontrivial advances to existing state-space and equispaced GP methods in

order to extend structured GP techniques into the multi-dimensional input domain. Our results (Section 3) illustrate across a range of data and different algorithms that structured models are most often superior to the state of the art sparse methods (SPGP). Notably, we introduced projection pursuit Gaussian Process regression (PPGPR-greedy, Section 2.1.3), an $\mathcal{O}(N)$ runtime algorithm that combines the computational efficiency of additive GP models (Section 2.1.1) with the expressivity of a multidimensional coupled model. The result (Section 3.1.2, notably Table 1) is an algorithm that has a superior runtime-accuracy tradeoff than several other algorithms including the sparse SPGP. While its accuracy was often slightly lower than a full GP, the linear scaling properties of PPGPR mean that it can be efficiently used across a much broader range of data sizes and applications. The primary takeaway of this work is thus: while the naive GP implementation may often produce the highest accuracy, the PPGPR algorithm that we introduced offers the best runtime-accuracy tradeoff across many datasets and is able to scale well beyond the realm of a naive GP.

Of course, in some cases the researcher will prefer the SPGP method over PPGPR-greedy. Indeed, in many senses, these two approaches are orthogonal to each other. We see this as an inherent fact in approximation techniques: various methods will be more appropriate in different settings. Our results (Section 3) presented an in-depth investigation into this runtime-accuracy tradeoff, using both metrics on real datasets and meta-analyses of Pareto efficiency. Our well-founded and competitive alternatives for efficient GP regression and classification can thus enable the researcher to make an informed choice about a GP method for a given data size, data complexity, and available computational resource.

To the point of runtime-accuracy tradeoff, there are sometimes opportunities for great scaling advantages with no accuracy tradeoff whatsoever. We demonstrated such an example with equispaced inputs in Section 2.1.3. Though this method exploits structure differently than the main PPGPR-greedy algorithm, our novel use of tensor algebra to create an $\mathcal{O}(N)$ GP model belongs in this exposition of the computational advantages of careful structural consideration. Notably, this method also opens up an entirely new set of big-data applications, such as image and video processing, or financial engineering applications such as implied volatility surfaces. Our future work is pursuing these application domains.

As a last computational point, as growth in computational speed is increasingly driven by parallelism over raw processor speed, it will become increasingly important to use GP schemes that naturally incorporate parallel processing, to efficiently deal with the rapid growth of future datasets. Our PPGPR-greedy method stands out in this regard versus both the naive full GP and SPGP, and again the results of Section 3 reiterated this fact.

Finally, from an algorithmic perspective, another interesting byproduct of this work was a number of sur-

prising connections to classical statistical techniques. The additive model turned out to be a Bayesian interpretation of the backfitting algorithm, importantly yielding an alternative proof of that algorithm’s validity. We utilized another classical technique - projection pursuit - in the PPGPR model, which dramatically increased the expressivity of the additive model without sacrificing $\mathcal{O}(N)$ performance.

Understanding how our existing nonparametric models can scale and be used in real data, and how these models connect to other areas of statistics, will increase the utility of machine learning algorithms in general. This is perhaps most important with Gaussian Processes, which promise a wide range of useful applications.

REFERENCES

- [1] J. Quiñero-Candela and C. E. Rasmussen, “A unifying view of sparse approximate Gaussian process regression,” *Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, December 2005.
- [2] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006.
- [3] L. Arnold, *Stochastic Differential Equations: Theory and Practice*. Krieger Publishing Corporation, 1992.
- [4] J. Hartikainen and S. Särkkä, “Kalman filtering and smoothing solutions to temporal Gaussian process regression models,” in *Machine Learning for Signal Processing (MLSP)*. Kittilä, Finland: IEEE, August 2010, pp. 379–384.
- [5] Y. Saatchi, “Scalable inference for structured Gaussian Process models,” Ph.D. dissertation, University of Cambridge, 2011.
- [6] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Transactions of the ASME — Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.
- [7] H. E. Rauch, F. Tung, and C. T. Striebel, “Maximum likelihood estimates of linear dynamical systems,” *AIAA Journal*, vol. 3, no. 8, pp. 1445–1450, August 1965.
- [8] D. K. Duvenaud, H. Nickisch, and C. E. Rasmussen, “Additive gaussian processes,” in *NIPS*, 2011, pp. 226–234.
- [9] L. Breiman and J. H. Friedman, “Estimating optimal transformations for multiple regression,” in *Computer Science and Statistics: Proceedings of the 16th Symposium on the Interface*, 1985, pp. 121–134.
- [10] T. J. Hastie and R. J. Tibshirani, *Generalized additive models*. London: Chapman & Hall, 1990.
- [11] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*, Second ed. New York: Springer-Verlag, 2009.
- [12] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.
- [13] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2007.
- [14] T. Minka, “Divergence measures and message passing,” Microsoft Research, Tech. Rep., 2005. [Online]. Available: <http://research.microsoft.com/~minka/papers/message-passing/>
- [15] R. Neal, “Probabilistic inference using markov chain monte carlo methods,” University of Toronto, Tech. Rep., 1993.
- [16] R. Douc, A. Garivier, E. Moulines, and J. Olsson, “On the Forward Filtering Backward Smoothing Particle Approximations of the Smoothing Distribution in General State Space Models,” Apr. 2009. [Online]. Available: <http://arxiv.org/abs/0904.0316>
- [17] F. Vivarelli and C. K. I. Williams, “Discovering hidden features with Gaussian processes regression,” in *Advances in Neural Information Processing Systems*, vol. 11, 1998.
- [18] E. Snelson and Z. Ghahramani, “Sparse Gaussian processes using pseudo-inputs,” in *Advances in Neural Information Processing Systems 18*. Cambridge, MA, USA: The MIT Press, December 2006, pp. 1257–1264.
- [19] J. H. Friedman and W. Stuetzle, “Projection pursuit regression,” *Journal of the American Statistical Association*, vol. 76, pp. 817–823, 1981.

- [20] T. Minka, "A family of algorithms for approximate Bayesian inference," PhD thesis, MIT, 2001. [Online]. Available: <http://research.microsoft.com/~minka/papers/ep/minka-thesis.pdf>
- [21] A. Naish-Guzman and S. Holden, "The generalized FITC approximation," in *Advances in Neural Information Processing Systems* 21. Cambridge, MA, USA: The MIT Press, December 2007, pp. 534–542.
- [22] J. Vanhatalo and A. Vehtari, "Sparse log gaussian processes via mcmc for spatial epidemiology," *jmlr*, vol. 1, pp. 73–89, 2007.
- [23] N. D. Lawrence, M. Seeger, and R. Herbrich, "Fast sparse Gaussian process methods: the informative vector machine," in *Advances in Neural Information Processing Systems* 17. Cambridge, MA, USA: The MIT Press, December 2003, pp. 625–632.
- [24] H. Nickisch and C. E. Rasmussen, "Approximations for binary gaussian process classification," *Journal of Machine Learning Research*, vol. 9, pp. 2035–2078, December 2008.
- [25] T. Hastie and R. Tibshirani, "Generalized additive models (with discussion)," *Statistical Science*, 1986.
- [26] J. P. Cunningham, K. V. Shenoy, and M. Sahani, "Fast Gaussian Process methods for point process intensity estimation," in *Proceedings of the 25th international conference on Machine learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 192–199. [Online]. Available: <http://doi.acm.org/10.1145/1390156.1390181>
- [27] K. F. Riley, M. P. Hobson, and S. J. Bence, *Mathematical Methods for Physics and Engineering*, Third ed. Cambridge University Press, 2006.
- [28] C. E. Rasmussen and H. Nickisch, "Gaussian processes for machine learning (gpml) toolbox," *Journal of Machine Learning Research*, vol. 11, pp. 3011–3015, December 2010.
- [29] M. Lázaro-Gredilla, "Sparse Gaussian processes for large-scale machine learning," Ph.D. dissertation, Universidad Carlos III de Madrid, Madrid, Spain, March 2010.
- [30] M. Seeger, C. K. I. Williams, N. D. Lawrence, and S. S. Dp, "Fast forward selection to speed up sparse Gaussian Process Regression," in *Workshop on AI and Statistics 9*, 2003.
- [31] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [32] J. C. Platt, "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods," in *Advances in Large Margin Classifiers*. MIT Press, 1999, pp. 61–74.
- [33] J. Vanhatalo, J. Riihimäki, J. Hartikainen, P. Jylänki, V. Tolvanen, and A. Vehtari, "Bayesian Modeling with Gaussian rocesses using the MATLAB Toolbox GPstuff (v3.3)," *arXiv*, no. 1206.5754, 2012.
- [34] C. Chang and C. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [35] O. L. Mangasarian and W. H. Wolberg, "Cancer diagnosis via linear programming," *SIAM News*, vol. 23, no. 5, 1990.
- [36] K. Bock *et al.*, "Methods for multidimensional event classification: a case study using images from a Cherenkov gamma-ray telescope," *Nuclear Instruments and Methods in Physics Research A*, vol. 516, pp. 511–528, January 2004.
- [37] C. Chang and C.-J. Lin, "Ijcn 2001 challenge: Generalization ability and text decoding," in *In Proceedings of IJCNN. IEEE*, 2001, pp. 1031–1036.