Fast Gaussian Process Posteriors with Product Trees

David A. Moore
Computer Science Division
University of California, Berkeley
Berkeley, CA 94709
dmoore@cs.berkeley.edu

Stuart Russell
Computer Science Division
University of California, Berkeley
Berkeley, CA 94709
russell@cs.berkeley.edu

Abstract

Gaussian processes (GP) are a powerful tool for nonparametric regression; unfortunately, calculating the posterior variance in a standard GP model requires time $O(n^2)$ in the size of the training set. Previous work by Shen et al. (2006) used a k-d tree structure to approximate the posterior mean in certain GP models. We extend this approach to achieve efficient approximation of the posterior covariance using a tree clustering on pairs of training points, and demonstrate significant improvements in performance with negligible loss of accuracy.

1 INTRODUCTION

Complex Bayesian models often tie together many smaller components, each of which must provide its output in terms of probabilities rather than discrete predictions. Gaussian process (GP) regression (Rasmussen and Williams, 2006) is a natural fit for such systems, but its applications have been limited by computational concerns: training a GP model on n points requires $O(n^3)$ time, while computing the posterior distribution at a test point requires $O(n)$ and $O(n^2)$ operations for the mean and variance respectively.

This paper focuses on the focuses on the fast evaluation of GP posterior probabilities in a running inference system, for which a model has already been trained. Fast runtime performance is a common requirement for real-world systems; for example, a speech recognition system might be trained once in the cloud, then run many times on a smartphone under a tight computational budget. Our particular work is motivated by an application to nuclear test monitoring: after training on historical seismic events, we want to identify and localize new events in realtime by processing signals from a worldwide sensor network. In this application, as with many others, probabilities from a GP are computed in the inner loop of a message-passing or MCMC inference algorithm; this computation must be efficient if inference is to be feasible.

Previous work has explored the use of space-partitioning tree structures for efficient computation of GP posterior means in models where the covariance kernel has a short lengthscale or compact support (Shen et al., 2006). We extend this in several ways. First, we describe the *product tree* data structure, along with an algorithm that uses this structure to efficiently compute posterior covariances. This provides what is to our knowledge the first account of GP regression in which the major test-time operations (posterior mean and covariance) run in time sublinear in the training set size, given a suitably sparse kernel matrix. We give a novel cutoff rule, applicable to both mean and covariance calculations, that guarantees provably bounded error. We also extend the class of models to which tree-based methods can be efficiently applied, by showing how to include a low-rank global component modeled either by an explicit parametric representation or by an approximate GP with inducing points (Snelson and Ghahramani, 2006). Finally, we evaluate this work empirically, with results demonstrating significant speedups on synthetic and real-world data, and in the process identify a simple method that often provides competitive performance to the more complex product tree.

2 BACKGROUND

2.1 GP REGRESSION MODEL

We assume as training input a set of labeled points $\{(\mathbf{x}_i, y_i)|i = 1, \ldots, n\}$, where we suppose that

$$y_i = f(\mathbf{x}_i) + \epsilon_i \tag{1}$$

for some unknown function $f(\cdot)$ and i.i.d. Gaussian observation noise $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$. Treating the estimation of $f(\cdot)$ as a Bayesian inference problem, we consider a Gaussian process prior distribution $f(\cdot) \sim GP(0, k)$, parameterized by a positive-definite *covariance* or *kernel* function $k(x, x')$. Given a set X^* containing m test points, we de-

rive a Gaussian posterior distribution $f(X^*) \sim \mathcal{N}(\mu^*, \Sigma^*)$, where

$$\mu^* = K^{*T} K_y^{-1} \mathbf{y} \tag{2}$$

$$\Sigma^* = K^{**} - K^{*T} K_y^{-1} K^* \tag{3}$$

and $K_y = k(X, X) + \sigma_n^2 I$ is the covariance matrix of training set observations, $K^* = k(X, X^*)$ denotes the $n \times m$ matrix containing the kernel evaluated at each pair of training and test points, and similarly $K^{**} = k(X^*, X^*)$ gives the kernel evaluations at each pair of test points. Details of the derivations, along with general background on GP regression, can be found in Rasmussen and Williams (2006).

In this work, we make the additional assumption that the input points $\mathbf{x}_i$ and test points $\mathbf{x}_p^*$ lie in some metric space $(\mathcal{M}, d)$, and that the kernel is a monotonically decreasing function of the distance metric. Many common kernels fit into this framework, including squared-exponential, rational quadratic, piecewise-polynomial and Matérn kernel families; anisotropic kernels can be represented through choice of an appropriate metric.

2.2 RELATED WORK

Tree structures such as k-d trees (Friedman et al., 1977) form a hierarchical, multiresolution partitioning of a dataset, and are commonly used in machine learning for efficient nearest-neighbor queries. They have also been adapted to speed up nonparametric regression (Moore et al., 1997; Shen et al., 2006); the general approach is to view the regression computation of interest as a sum over some quantity associated with each training point, weighted by the kernel evaluation against a test point. If there are sets of training points having similar weight – for example, if the kernel is very wide, if the points are very close to each other, or if the points are all far enough from the query to have effectively zero weight – then the weighted sum over the set of points can be approximated by an unweighted sum (which does not depend on the query and may be precomputed) times an estimate of the typical weight for the group, saving the effort of examining each point individually. This is implemented as a recursion over a tree structure augmented at each node with the unweighted sum over all descendants, so that recursion can be cut off with an approximation whenever the weight function is shown to be suitably uniform over the current region.

This tree recursion can be thought of as an approximate matrix-vector multiplication (MVM) operation; a related method, the Improved Fast Gauss Transform (Morariu et al., 2008), implements fast MVM for the special case of the SE kernel. It is possible to accelerate GP training by combining MVM methods with a conjugate gradient solver, but models thus trained do not allow for the computation of predictive variances. One argument against MVM techniques (and, by extension, the approach of this paper) is that their efficiency requires shorter lengthscales than are common in machine learning applications (Murray, 2009); however, we have found them quite effective on datasets which do have genuinely sparse covariance structure (e.g., geospatial data), or in which the longer-scale variation can be represented by a low-rank component.

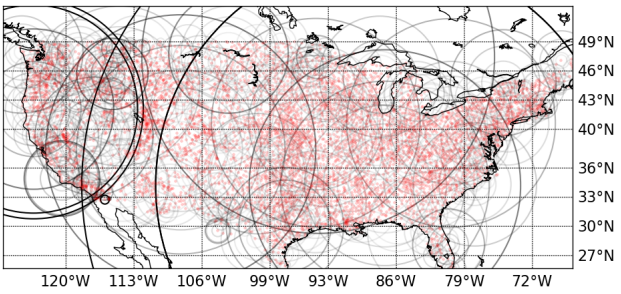


Figure 1: Cover tree decomposition of USA precipitation measurement stations (see Section 5.3).

Another related approach is the use of local approximations, in which different GPs are trained in different regions of the input space. There is some evidence that these can provide accurate predictions which are very fast to evaluate (Chalupka et al., 2013); however, they face boundary discontinuities and inaccurate uncertainty estimates if the data do not naturally form independent clusters.

2.3 k-d VERSUS COVER TREES

Although related work (Moore et al., 1997; Shen et al., 2006) has generally used k-d trees as the multiresolution structure, this paper instead uses cover trees (Beygelzimer et al., 2006) to allow for non-Euclidean metrics. A cover tree on n points can be constructed in $O(n \log n)$ time, and the construction and query times scale only with the *intrinsic* dimensionality of the data, allowing for efficient nearest-neighbor queries in higher-dimensional spaces (Beygelzimer et al., 2006). Figure 1 shows a cover-tree decomposition of one of our test datasets.

We do not depend specifically on the cover tree algorithm; any similar tree construction algorithm could be used, provided (a) there is a one-to-one correspondence between the leaves of the tree and the training points $x_i \in X$, and (b) each non-leaf node $\mathbf{n}$ is associated with some point $x_\mathbf{n} \in \mathcal{M}$, such that all descendants of $\mathbf{n}$ are contained within a ball of radius $r_\mathbf{n}$ centered at $x_\mathbf{n}$. For example, a ball tree (Uhlmann, 1991), or a tree created through agglomerative clustering on the training points, could also satisfy these criteria.

initialize globals sum $S \leftarrow 0$, error $\epsilon \leftarrow 0$, leaf count $\kappa \leftarrow 0$
function WEIGHTEDMETRICSUM(node $\mathbf{n}$, query points $(\mathbf{x}_i^*, \mathbf{x}_j^*)$, tolerance ϵ_{abs})
  $\delta_{\mathbf{n}} \leftarrow \delta((\mathbf{x}_i^*, \mathbf{x}_j^*), (\mathbf{n}_1, \mathbf{n}_2))$
  if $\mathbf{n}$ is a leaf **then**
    $S \leftarrow S + (K_y^{-1})_{\mathbf{n}} \cdot \left(k(d(\mathbf{x}_i^*, \mathbf{n}_1)) \cdot k(d(\mathbf{x}_j^*, \mathbf{n}_2))\right)$
    $\kappa \leftarrow \kappa + 1$
  else
    $w_{\text{min}} \leftarrow k_{\text{lower}}^{\text{prod}}(\delta_{\mathbf{n}} + r_{\mathbf{n}})$
    $w_{\text{max}} \leftarrow k_{\text{upper}}^{\text{prod}}(\max(\delta_{\mathbf{n}} - r_{\mathbf{n}}, 0))$
    $\epsilon_{\mathbf{n}} \leftarrow \frac{1}{2}(w_{\text{max}} - w_{\text{min}}) S_{\mathbf{n}}^{\text{Abs}}$
    if $\epsilon_{\mathbf{n}} \leq \kappa_{\mathbf{n}}/(n - \kappa) \cdot (\epsilon_{\text{abs}} - \epsilon)$ **then**
      $S \leftarrow S + \frac{1}{2}(w_{\text{max}} + w_{\text{min}}) \cdot S_{\mathbf{n}}^{\text{UW}}$
      $k \leftarrow \kappa + \kappa_{\mathbf{n}}$
      $\epsilon \leftarrow \epsilon + \epsilon_{\mathbf{n}}$
    else
      for each child $\mathbf{c}$ of $\mathbf{n}$
      sorted by descending $\delta((\mathbf{x}_i^*, \mathbf{x}_j^*), (\mathbf{c}_1, \mathbf{c}_2))$ **do**
        WEIGHTEDMETRICSUM $\left(\mathbf{c}, (\mathbf{x}_i^*, \mathbf{x}_j^*), \epsilon_{\text{abs}}\right)$
      end for
    end if
  end if
end function

Figure 2: Recursive algorithm to computing GP covariance entries using a product tree. Abusing notation, we use $\mathbf{n}$ to represent both a tree node and the pair of points $\mathbf{n} = (\mathbf{n}_1, \mathbf{n}_2)$ associated with that node.

3 EFFICIENT COVARIANCE USING PRODUCT TREES

We now consider efficient calculation of the GP covariance (3). The primary challenge is the multiplication $K^{*T} K_y^{-1} K^*$. For simplicity of exposition, we will focus on computing the (i, j)th entry of the resulting matrix, i.e., on the multiplication $\mathbf{k_i^*}^T K_y^{-1} \mathbf{k_j^*}$ where $\mathbf{k_i^*}$ denotes the vector of kernel evaluations between the training set and the ith test point, or equivalently the ith column of K^*. Note that a naïve implementation of this multiplication requires $O(n^2)$ time.

We might be tempted to apply the vector multiplication primitive of Shen et al. (2006) separately for each row of K_y^{-1} to compute $K_y^{-1} \mathbf{k_j^*}$, and then once more to multiply the resulting vector by $\mathbf{k_i^*}$. Unfortunately, this requires n vector multiplications and thus scales (at least) linearly in the size of the training set. Instead, we note that we can rewrite $\mathbf{k_i^*}^T K_y^{-1} \mathbf{k_j^*}$ as a weighted sum of the entries of K_y^{-1}, where the weight of the (p, q)th entry is given by $k(\mathbf{x_i^*}, \mathbf{x_p}) k(\mathbf{x_j^*}, \mathbf{x_q})$:

$$\mathbf{k_i^*}^T K_y^{-1} \mathbf{k_j^*} = \sum_{p=1}^{n} \sum_{q=1}^{n} (K_y^{-1})_{pq} k(\mathbf{x_i^*}, \mathbf{x_p}) k(\mathbf{x_j^*}, \mathbf{x_q}). \quad (4)$$

Our goal is to compute this weighted sum efficiently using a tree structure, similar to Shen et al. (2006), except that instead of clustering points with similar weights, we now want to cluster *pairs* of points having similar weights.

To do this, we consider the *product space* $\mathcal{M} \times \mathcal{M}$ consisting of all pairs of points from $\mathcal{M}$, and define a *product metric* δ on this space. The details of the product metric will depend on the choice of kernel function (section 3.2). For the moment, we will assume an SE kernel, of the form $k_{SE}(d) = \exp(-d^2)$, for which a natural choice is the 2-product metric:

$$\delta((\mathbf{x}_a, \mathbf{x}_b), (\mathbf{x}_c, \mathbf{x}_d)) = \sqrt{d(\mathbf{x}_a, \mathbf{x}_c)^2 + d(\mathbf{x}_b, \mathbf{x}_d)^2},$$

which has the fortunate property

$$k_{SE}(d(\mathbf{x}_a, \mathbf{x}_b)) k_{SE}(d(\mathbf{x}_c, \mathbf{x}_d)) = k_{SE}(\delta((\mathbf{x}_a, \mathbf{x}_b), (\mathbf{x}_c, \mathbf{x}_d))),$$

i.e., the property that evaluating the SE kernel in the product space (the right hand side) gives us the correct weight for our weighted sum (4) (the left hand side). Note that this property is convenient but not necessary; Section 3.2 describes how to choose a product metric for several common kernels.

Now we can run any metric tree construction algorithm (e.g., a cover tree) using the product metric to build a *product tree* on all *pairs* of training points. At each leaf node $\mathbf{L}$, representing a pair of training points, we store the entry $(K_y^{-1})_{\mathbf{L}}$ corresponding to those two training points, and at each higher-level node $\mathbf{n}$ we cache the unweighted sum $S_{\mathbf{n}}^{UW}$ of these entries over all of its descendant leaf nodes, as well as the sum of absolute values $S_{\mathbf{n}}^{Abs}$ (these cached sums will be used to determine when to cut off recursive calculations):

$$S_{\mathbf{n}}^{\text{UW}} = \sum_{\mathbf{L} \in \text{leaves}(\mathbf{n})} (K_y^{-1})_{\mathbf{L}} \quad (5)$$

$$S_{\mathbf{n}}^{\text{Abs}} = \sum_{\mathbf{L} \in \text{leaves}(\mathbf{n})} \left|(K_y^{-1})_{\mathbf{L}}\right|. \quad (6)$$

Given a product tree augmented in this way, the weighted-sum calculation (4) is approximated by the WEIGHTED-METRICSUM algorithm of Figure 2. It proceeds by a recursive descent down the tree, where at each non-leaf node $\mathbf{n}$ it computes upper and lower bounds on the weight of any descendant, and applies a cutoff rule (Section 3.1) to determine whether to continue the descent. Whenever the descent is halted, we approximate the contribution from leaves below $\mathbf{n}$ by $\frac{1}{2}(w_{\text{max}} + w_{\text{min}}) \cdot S_{\mathbf{n}}^{\text{UW}}$, i.e., by the average weight times the unweighted sum. Otherwise, the computation continues recursively over $\mathbf{n}$'s children.

3.1 CUTOFF RULE

The decision of when to cut off the tree recursion is crucial to correctness and performance. Many cutoff rules are possible. For predictive mean calculation, Moore et al. (1997) and Shen et al. (2006) maintain an accumulated lower bound on the total overall weight, and cut off whenever the difference between the upper and lower weight bounds at the current node is a small fraction of the lower bound on the overall weight. By contrast, we introduce a

rule (8) which takes into account the weights as well as the entries of K_y^{-1} being summed over (since we expect this matrix to be approximately sparse, some entries will contribute much more to the sum than others), and which provides a provable guarantee on approximation error not available in the earlier work. First, at each node $\mathbf{n}$ we compute an error bound

$$\epsilon_{\mathbf{n}} = \frac{1}{2}(w_{\max} - w_{\min})S_{\mathbf{n}}^{\text{Abs}}, \tag{7}$$

which we justify by the following lemma:

Lemma 1. *The error introduced in approximating the subtree at $\mathbf{n}$ by $\frac{1}{2}(w_{max} + w_{min})S_{\mathbf{n}}^{UW}$ is bounded by $\epsilon_{\mathbf{n}}$.*

Proof. Let $S_{\mathbf{n}}^{+}$ and $S_{\mathbf{n}}^{-}$ denote the sum of positive and negative leaf entries under $\mathbf{n}$, respectively, so $S_{\mathbf{n}}^{\text{UW}} = (S_{\mathbf{n}}^{+} + S_{\mathbf{n}}^{-})$ and $S_{\mathbf{n}}^{\text{Abs}} = (S_{\mathbf{n}}^{+} - S_{\mathbf{n}}^{-})$. The worst case for the approximation is if the true sum gives weight $w_{\max}$ to $S_{\mathbf{n}}^{+}$ and $w_{\min}$ to $S_{\mathbf{n}}^{-}$ (or vice versa), yielding an approximation error of

$$\left| \left(w_{\max} S_{\mathbf{n}}^{+} + w_{\min} S_{\mathbf{n}}^{-}\right) - \frac{w_{\max} + w_{\min}}{2} S_{\mathbf{n}}^{\text{UW}} \right| = \epsilon_{\mathbf{n}}.$$

□

Intuitively, we see that $\epsilon_{\mathbf{n}}$ is small whenever the leaves below $\mathbf{n}$ have nearly uniform weights, or when the total mass of K_y^{-1} entries under $\mathbf{n}$ is small. This motivates our cutoff rule

$$\epsilon_{\mathbf{n}} \leq \kappa_{\mathbf{n}}/(n - \kappa) \cdot (\epsilon_{\text{abs}} - \epsilon), \tag{8}$$

in which ϵ_{abs} is a user-specified bound on the absolute error of the overall computation, $\kappa_{\mathbf{n}}$ denotes the number of leaves below $\mathbf{n}$, n and κ denote respectively the total number of leaves in the tree and the leaves included thus far in the partially computed sum, and $\epsilon = \sum_{\mathbf{n} \in C} \epsilon_{\mathbf{n}}$, where C is the set of all intermediate nodes whose leaf sums we have previously approximated, is a running upper bound on the total error accumulated thus far in the sum. Intuitively, $(\epsilon_{\text{abs}} - \epsilon)$ gives the "error budget" remaining out of an initial budget of ϵ_{abs}; each cutoff is allowed to use a fraction of this budget proportional to the number of leaves being approximated. We show the following correctness result:

Theorem 1. *Let $\mathbf{T}$ be a product tree constructed on K_y^{-1}, where $\hat{\Sigma}_{ij}^{*} = K_{ij}^{**} - \text{WEIGHTEDMETRICSUM}(\boldsymbol{T}, (\boldsymbol{x}_i^{*}, \boldsymbol{x}_j^{*}), \epsilon_{abs})$ denotes the approximation returned by the tree recursion to the true posterior covariance Σ_{ij}^{*}. Then $\left|\Sigma_{i,j}^{*} - \hat{\Sigma}_{i,j}^{*}\right| \leq \epsilon_{abs}$.*

Proof. By our cutoff rule (8), we proceed with the approximation at $\mathbf{n}$ only if $\epsilon' := \epsilon_{\mathbf{n}} + \epsilon \leq \epsilon_{\text{abs}}$, i.e. only if the new error being introduced (bounded by Lemma 1), plus the error already accumulated, is still bounded by ϵ_{abs}.[1] Thus at every step we maintain the invariant $\epsilon \leq \epsilon_{\text{abs}}$, which establishes the result. □

[1] We have ignored the factor $\kappa_{\mathbf{n}}/(n - \kappa)$ here since it is always ≤ 1; this factor is included to help "pace" the computation and is not necessary for correctness.

Remark. *Although Theorem 1 bounds absolute error, we can also apply it to bound relative error in the case of computing a predictive variance that includes a noise component. Since the noise variance σ_n^2 is a lower bound on the predictive variance Σ_{ii}^{*}, setting $\epsilon_{abs} = \epsilon_{rel} \cdot \sigma_n^2$ is sufficient to ensure that the approximation error is smaller than $\epsilon_{rel} \cdot \Sigma_{ii}^{*}$.*

Note that this cutoff rule and correctness proof can be easily back-ported into the WEIGHTEDSUM algorithm of Shen et al. (2006), providing a bounded error guarantee for tree-based calculations of GP posterior means as well as covariances.

3.2 OTHER KERNEL FUNCTIONS

As noted above, the SE kernel has the lucky property that, if we choose product metric $\delta = \sqrt{d_1^2 + d_2^2}$, then the product of two SE kernels is equal to the kernel of the product metric δ:

$$k_{SE}(d_1)k_{SE}(d_2) = \exp\left(-d_1^2 - d_2^2\right) = k_{SE}(\delta).$$

In general, however, we are not so lucky: it is not the case that every kernel we might wish to use has a corresponding product metric such that a product of kernels can be expressed in terms of the product metric. In such cases, we may resort to upper and lower bounds in place of computing the exact kernel value. Note that such bounds are all we require to evaluate the error bound (7), and that when we reach a leaf node representing a specific pair of points we can always evaluate the exact product of kernels directly at that node.

For example, consider the kernel $k_{CS,0}(d) = (1-d)_+^j$ (taking $j = \lfloor \frac{D}{2} \rfloor + 1$, where D is the input dimension); this is a simple example of a more general class of piecewise-polynomial compactly supported kernels (Rasmussen and Williams, 2006) whose computational advantages are especially relevant to tree-based algorithms. Considering the product of two such kernels,

$$k_{\text{CS},0}(d_1)k_{\text{CS},0}(d_2) = (1 - (d_1 + d_2) + \Delta)_+^j$$
$$\text{where } \Delta = d_1 d_2 \textbf{ if } (d_1 < 1, d_2 < 1) \textbf{ else } 0$$

we notice that this is almost equivalent to $k_{\text{CS},0}(\delta)$ for the choice of $\delta = d_1 + d_2$, but with an additional pairwise term Δ. We bound this term by noting that it is maximized when $d_1 = d_2 = \delta/2$ (for $\delta < 2$) and minimized whenever either $d_1 = 0$ or $d_2 = 0$, so we have $(\delta/2)^2 \geq \Delta \geq 0$. This yields the bounds $k_{\text{lower}}^{\text{prod}}$ and $k_{\text{upper}}^{\text{prod}}$ as shown in Table 1. Bounds for other common kernels are obtained analogously in Table 1.

3.3 OPTIMIZATIONS

A naïve product tree on n points will have n^2 leaves, but we can reduce this and achieve substantial speedups by

Kernel	$k(d)$	$k(d_1)k(d_2)$	$\delta(d_1, d_2)$	$k^{\text{prod}}_{\text{lower}}(\delta)$	$k^{\text{prod}}_{\text{upper}}(\delta)$
SE	$\exp\left(-d^2\right)$	$\exp\left(-d_1^2-d_2^2\right)$	$\sqrt{d_1^2+d_2^2}$	$\exp\left(-(\delta)^2\right)$	$\exp\left(-(\delta)^2\right)$
γ-exponential	$\exp\left(-d^\gamma\right)$	$\exp\left(-d_1^\gamma-d_2^\gamma\right)$	$\left(d_1^\gamma+d_2^\gamma\right)^{1/\gamma}$	$\exp\left(-(\delta)^\gamma\right)$	$\exp\left(-(\delta)^\gamma\right)$
Piecewise polynomial $\text{CS}_{D,q=0}$, $j=\left\lfloor \frac{D}{2} \right\rfloor+1$	$(1-d)_+^j$	$(1-(d_1+d_2)+\Delta)_+^j$ where $\Delta = d_1 d_2$ **if** $(d_1<1, d_2<1)$ **else** 0	d_1+d_2	$(1-\delta)_+^j$	$\left(1-\delta+\frac{(\delta)^2}{4}\right)_+^j$
Rational Quadratic	$\left(1+\frac{d^2}{2\alpha}\right)^{-\alpha}$	$\left(1+\frac{d_1^2+d_2^2}{2\alpha}+\frac{d_1^2 d_2^2}{4\alpha^2}\right)^{-\alpha}$	$\sqrt{d_1^2+d_2^2}$	$\left(1+\frac{(\delta)^2}{2\alpha}+\frac{(\delta)^4}{16\alpha^2}\right)^{-\alpha}$	$\left(1+\frac{(\delta)^2}{2\alpha}\right)^{-\alpha}$
Matérn ($\nu = 3/2$)	$\left(1+\sqrt{3}d\right)$ $\cdot \exp\left(-\sqrt{3}d\right]\right)$	$\left(1+\sqrt{3}\left(d_1+d_2\right)+3d_1 d_2\right)$ $\cdot \exp\left(-\sqrt{3}(d_1+d_2)\right)$	d_1+d_2	$\left(1+\sqrt{3}\delta\right)\cdot\exp\left(-\sqrt{3}\delta\right)$	$\left(1+\sqrt{3}\delta+3(\delta/2)^2\right)\cdot\exp(-\sqrt{3}\delta)$

Table 1: Bounds for products of common kernel functions, all from from Rasmussen and Williams (2006).

exploiting the structure of K_y^{-1} and of the product space $\mathcal{M} \times \mathcal{M}$:

Sparsity. If K_y is sparse, as with compactly supported kernel functions, or can be well-approximated as sparse, as when using standard kernels with short lengthscales, then it can be shown that K_y^{-1} may also be approximated as sparse (Bickel and Lindner, 2012, sections 2 and 4.1). When this is the case, the tree need include only those pairs $(\mathbf{x}_p, \mathbf{x}_q)$ for which $(K_y^{-1})_{pq}$ is non-negligible.

Symmetry. Since K_y^{-1} is a symmetric matrix, it is redundant to include leaves for both $(\mathbf{x}_p, \mathbf{x}_q)$ and $(\mathbf{x}_q, \mathbf{x}_p)$ in our tree. Instead, we can build separate trees to compute the diagonal and upper-triangular components of the sum, then reuse the upper-triangle result for the lower triangle.

Factorization of product distances. In general, computing the product distance δ will usually involve two calls to the underlying distance metric d; these can often be reused. For example, when calculating both $\delta((\mathbf{x}_a, \mathbf{x}_b), (\mathbf{x}_c, \mathbf{x}_d))$ and $\delta((\mathbf{x}_a, \mathbf{x}_e), (\mathbf{x}_c, \mathbf{x}_d))$, we can reuse the value of $d(\mathbf{x}_a, \mathbf{x}_c)$ for both computations. This reduces the total number of calls to the distance function during tree construction from a worst-case n^4 (for all pairs of pairs of training points) to a maximum of n^2, and in general much fewer if other optimizations such as sparsity are implemented as well.

Leaf binning at short distances. If all leaves below a node $\mathbf{n}$ are within a kernel lengthscale of $\mathbf{n}$, we cut off the tree at $\mathbf{n}$ and just compute the exact weighted sum over those leaves, avoiding the tree recursion.

4 MIXED LOCAL/GLOBAL GP REPRESENTATIONS

In this section, we extend the GP model (1) to include both a local and a global component $g(\mathbf{x}_i)$, i.e.,

$$y_i = h(\mathbf{x}_i) + \epsilon_i = f(\mathbf{x}_i) + g(\mathbf{x}_i) + \epsilon_i, \tag{9}$$

where f is modeled by a short-lengthscale/compactly-supported GP, and g is a global component of constant rank (i.e., not directly dependent on n). We will show how to efficiently calculate posteriors from such models using a product tree.

Formally, we assume a GP of the form

$$h(X) \sim \mathcal{N}(\phi(X)^T \mathbf{b}, k_f(X) + \phi(X)^T B \phi(X)) \tag{10}$$

where $k_f(X)$ is a sparse matrix, B is an $m \times m$ matrix and $\mathbf{b}$ an m-dimensional vector, and $\phi(X)$ computes an $n \times m$ feature representation where $m \ll n$. This "sparse+low rank" formulation includes a wide range of models capturing global and local structure. For example, we can express the "explicit basis functions" model from section 2.7 of Rasmussen and Williams (2006) by letting $\phi(X)$ denote the basis functions $H(X)$ and letting $\mathbf{b}, B$ denote the mean and covariance of a Gaussian prior on their weights. Similarly, the CS+FIC model given by Vanhatalo and Vehtari (2008) may be represented[2] by taking $\phi(X) = K_{u,n}$, $B = K_{u,u}^{-1}$, $\mathbf{b} = \mathbf{0}$, and letting $k_f(X)$ absorb the diagonal term $\mathbf{\Lambda}$. Other approximate GP models for global variation (e.g., Quiñonero-Candela and Rasmussen, 2005; Snelson and Ghahramani, 2007; Rahimi and Recht, 2007; Vedaldi and Zisserman, 2010) can also be expressed in this form.

Given any model in the form of Eqn. (10), the posterior distribution $h(X^*) \sim \mathcal{N}(\mu'_*, \Sigma'_*)$ can be derived (Rasmussen and Williams, 2006) as

$$\mu'_* = \phi^{*T}\bar{\beta} + K^{*T} K_y^{-1}(\mathbf{y} - \phi^* \bar{\beta}) \tag{11}$$

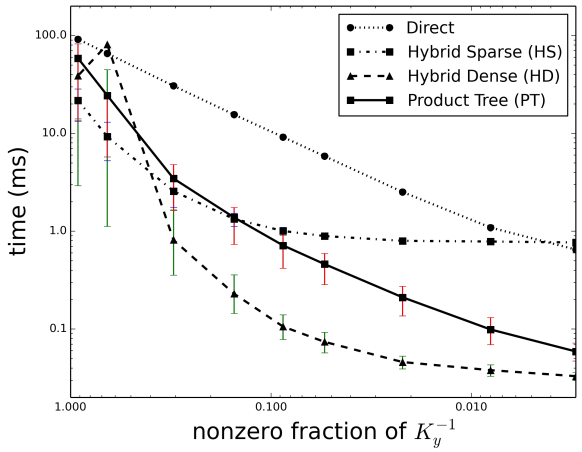
$$\Sigma'_* = K^{**} - K^{*T} K_y^{-1} K^* + R^T (B^{-1} + \phi K_y^{-1} \phi^T) R \tag{12}$$

where we let $\phi = \phi(X)$ and $\phi^* = \phi(X^*)$, and we have $\bar{\beta} = (B^{-1} + \phi K_y^{-1} \phi^T)^{-1}(\phi K_y^{-1} \mathbf{y} + B^{-1}\mathbf{b})$ and $R = \phi^* - \phi(X) K_y^{-1} K^*$. Section 2.7 of Rasmussen and Williams (2006) gives further details.

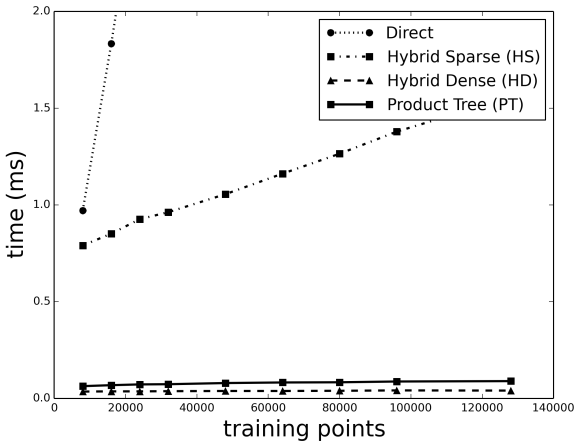
4.1 EFFICIENT OPERATIONS IN SPARSE+LOW RANK MODELS

Calculating the posterior given by (11, 12) is a straightforward extension of the standard case. The predictive

[2]Here the right side of each expression follows the notation of Vanhatalo and Vehtari.



(a) As a function of input density for a fixed-size (5000 points) training set, with error bars at the 10th to 90th percentiles.



(b) As a function of training set size with constant density $v = 5$.

Figure 3: Mean times to compute posterior variance on 2D synthetic data using a piecewise-polynomial $CS_{2,2}$ kernel.

mean (11) can be accommodated within the framework of Shen et al. (2006) using a tree representation of the vector $K_y^{-1}\left(\mathbf{y} - \phi^{*T}\bar{\beta}\right)$, then adding in the easily evaluated parametric component $\phi^{*T}\bar{\beta}$. In the covariance (12) we can use a product tree to approximate $K^{*T}K_y^{-1}K^*$ as described above; of the remaining terms, $\bar{\beta}$ and $B^{-1} + \phi K_y^{-1}\phi^T$ can be precomputed at training time, and ϕ^* and K^{**} don't depend on the training set. This leaves $\phi K_y^{-1}K^*$ as the one remaining challenge; we note that this quantity can be computed efficiently using m applications per test point of the vector multiplication primitive from Shen et al. (2006), reusing the same tree structure to multiply each column of K^* by each row of ϕK_y^{-1}. Thus, the full posterior distribution at a test point can be calculated efficiently with no explicit dependence on n (i.e., with no direct access to the training points except through space-partitioning tree structures).

5 EVALUATION

We compare the use of a product tree (**PT**) for predictive variance calculation with several alternatives:

Direct: sparse matrix multiplication, using a sparse representation of K_y^{-1} and dense representation of $\mathbf{k}_\mathbf{i}^*$.

Hybrid Sparse (HS): sparse matrix multiplication, using a sparse representation of $\mathbf{k}_\mathbf{i}^*$ constructed by querying a cover tree for all training points within distance r of the query point $\mathbf{x}_\mathbf{i}^*$, where r is chosen such that $k(r')$ is negligible for $r' > r$, and then filling in only those entries of $\mathbf{k}_\mathbf{i}^*$ determined to be non-negligible. Since all our experiments involve kernels with compact support, we simply set r equal to the kernel lengthscale.

Hybrid Dense (HD):[3] dense matrix multiplication, using only those entries of K_y^{-1} that correspond to training points within distance r of the query point. These training points are identified using a cover tree, as above, and entries of K_y^{-1} are retrieved from a hash table.

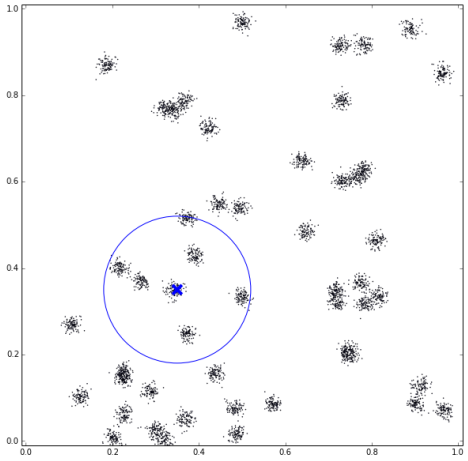
We do not show a comparison to the naïve dense matrix approach, since this is generally slower by orders of magnitude on the datasets we consider.

Our product tree implementation is a Python extension written in C++, based on the cover tree implementation of Beygelzimer et al. (2006) and implementing the optimizations from Section 3.3. In all experiments we set the approximation parameter ϵ_{rel} to ensure an approximation error of less than 0.1% of the exact variance. All sparse matrix multiplications are in CSR format using SciPy's sparse routines; we impose a sparsity threshold of 10^{-8} such that any entry less than the threshold is set to zero. Code to reproduce all experiments, along with the datasets, is included in the supplementary materials.

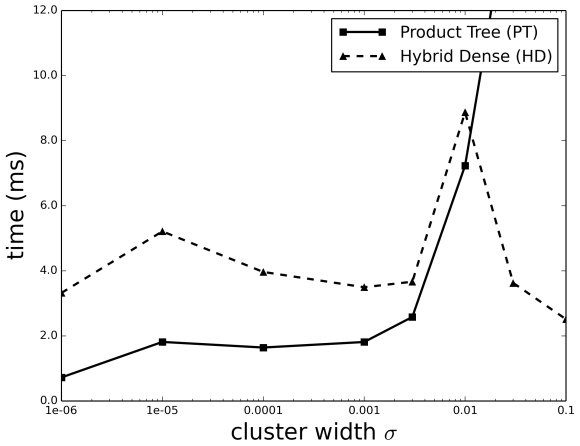
5.1 SYNTHETIC DATA

Figures 3a and 3b compare our methods on a simple two-dimensional synthetic data set, consisting of points sampled uniformly at random from the unit square. We train a GP on n such points and then measure the mean time per point to compute the predictive variance at 1000 random test points. The GP uses a piecewise-polynomial $CS_{2,2}$ kernel with observation noise $\sigma_n^2 = 1.0$ and lengthscale $\ell = \sqrt{v\pi/n}$, where v is a parameter indicating the average number of training points within a one-lengthscale ball of

[3] We are grateful to Iain Murray for suggesting this approach for comparison.



(a) Clumpy data with $\sigma = 0.01$, with a ball of covariance lengthscale $\frac{1}{\sqrt{10\pi}}$ overlayed.



(b) Mean times to compute posterior variance on data of varying clumpiness, using a piecewise-polynomial $CS_{2,2}$ kernel.

Figure 4: HD versus PT on clumpy data.

a random query point.

We see from Figure 3a that the hybrid dense method performs best when the kernel lengthscale is extremely short, followed by the product tree; in the most extreme cases these methods outperform the alternatives by an order of magnitude. However, performance degrades as the kernel lengthscale increases.

Figure 3b examines the scaling behavior of the algorithms in a relatively sparse setting, $v = 5$, chosen to allow the tractable inversion of large kernel matrices. Here we see that the direct calculation scales quite steeply (though linearly: the runtime at $n = 160000$ is approximately 15ms) with training size due to the need to explicitly compute all n entries of $\mathbf{k}_\mathbf{i}^*$. The hybrid sparse calculation avoids this bottleneck and is significantly more efficient, but its scaling is ultimately also linear with n, an inherent limitation of sparse matrix multiplication (Bank and Douglas, 1993) since it does not have access to the geometry of the data. By contrast, in this sparse setting the hybrid dense and product tree approaches remain efficient even for very large datasets, with a small constant-factor advantage for the hybrid dense method.

5.2 CLUMPINESS

The strong performance in the previous experiments of the hybrid dense method, relative to the product tree, is due to the uniformity of the training data. With no natural clusters, the only available optimization is to discard faraway points. The product tree would be expected to perform better when the data are 'clumpy', allowing it to merge kernel evaluations from nearby points. This is explored in Figure 4b, which compares the two methods on a synthetic dataset of 5000 points, sampled from a mixture of 50 Gaussians each with covariance $\sigma^2\mathcal{I}$ for varying clumpiness σ. As expected, the product tree is fastest when the data are tightly clustered and many points can be merged. Figure 4a shows an example of a dataset at σ=0.01, the approximate 'crossover' point at which the lower constant factor of the hybrid dense method begins to outweigh the advantages of the product tree.

5.3 REAL DATA

We evaluate the performance of our methods on the following datasets, shown in Table 2:

seismic: 20000 travel-time residuals between observed P-wave travel times to the seismic array in Alice Springs, Australia, and the times predicted by a one-dimensional IASPEI91 model (Kennett and Engdahl, 1991), indexed by latitude/longitude and depth of the source event.

snow: 20000 observations of water content of California snow pack recorded daily at 128 stations from November 1, 2011 to June 1, 2012, indexed by date, latitude/longitude and elevation. Collected from `http://cdec.water.ca.gov/queryCSV.html`.

precip: shown in Figure 1, total annual precipitation recorded in 1995 by each of 5775 stations in the continental US, indexed by latitude, longitude, and elevation (Vanhatalo and Vehtari, 2008).

tco: Total column ozone as recorded over the Earth's surface by the NIMBUS-7/TOMS satellite on October 1, 1998 (Park et al., 2011). Our experiments use a random sample of 20000 from the full 48331 measurements.

housing: Data from the 1990 California census, as used by Shen et al. (2006). We predict the median income of each block group as a function of median age and median house

value.

Table 3 compares, for each dataset, the performance of an SE kernel to that of a piecewise-polynomial, compactly-supported kernel $CS_{D,2}$ with a hand-selected number of FIC inducing points capturing global variation. The goal here is to show that the CS+FIC models, which are well-suited for fast tree-based calculations at test time, are a reasonable modeling choice even for purely predictive reasons. Model quality is measured by the Standardized Mean Squared Error (SMSE), i.e., the squared error divided by the squared error of the trivial predictor that just predicts the mean of the training set, and Mean Standardized Log Loss (MSLL), obtained by averaging $-\log p(y_i^*|x_i^*, X, \mathbf{y})$ over the test set and subtracting the same score for a trivial model which always predicts the mean and variance of the training set. Hyperparameters for each model were obtained by maximizing the marginal likelihood over a random subset of 5000 training points using a truncated Newton method; a prior was used to encourage short lengthscales for the CS components of the CS+FIC models. Note that, for each of the datasets considered in this paper, the CS+FIC model provides better posterior probability estimates (lower MSLL) than an SE model.[4]

In Table 4 we show, for each method and dataset, the mean and standard deviation of posterior variance computation time evaluated over the test set. Here the **HS**, **HD**, and **PT** methods use the tree-optimized FIC calculations from Section 4.1, while the **Direct** and $\mathbf{HS}_N$ methods use a naive FIC calculation; the latter is included explicitly for comparison with the tree-optimized version. Interestingly, the hybrid dense calculation is quickest in every case, sometimes tied by the product tree, suggesting that these real datasets do not possess the degree of clumpiness necessary for the product tree to dominate (Table 5 directly compares the number of terms used by the two method, showing that the product tree succeeds in merging a significant number of points only on the housing dataset). Both the product tree and the hybrid dense method are generally faster than the hybrid sparse method, with the exception of the US precipitation data in which the relative fullness of the inverse kernel matrix for that model (Table 3) greatly increases the size of the product tree. Comparing **HS** to $\mathbf{HS}_N$, we see that the tree-optimized FIC calculation provides significant speedups on all datasets except for the precipitation data, with an especially significant speedup for the `tco` data which uses 90 inducing points.

[4]The SE model may still be superior in many cases, of course. For example, we experimented with the well-known SARCOS inverse kinematics dataset but were unable to find a CS+FIC model that was competitive with the SE baseline.

	d	n_{train}	n_{test}	t_{build}
seismic	3	16000	4000	4.9s
snow	4	15000	5000	4.2s
precip	3	5000	775	23.0s
tco	2	15000	5000	2.2s
housing	2	18000	2000	3.3s

Table 2: Datasets, with product tree construction times.

	model	K_y^{-1} %	SMSE	MSLL
seismic	CS+FIC (20)	0.6%	**0.82**	**-0.20**
	SE	33.9%	0.84	-0.11
snow	CS+FIC (20)	0.8%	**0.0030**	**-2.91**
	SE	36.3%	0.0097	-2.31
precip	CS+FIC (20)	45.3%	0.129	**-1.17**
	SE	50.2%	**0.125**	-1.06
tco	CS+FIC (90)	0.4%	**0.041**	**-1.63**
	SE	34.6%	0.054	-1.45
housing	CS+FIC (20)	0.5%	0.83	**-0.18**
	SE	100%	**0.80**	-0.086

Table 3: Predictive performance of compactly-supported and squared-exponential kernels on the test datasets. Smaller is better for both SMSE and MSLL.

6 CONCLUSION AND FUTURE WORK

This paper introduces the *product tree*, a method for efficient adaptive calculation of GP covariances using a multiresolution clustering of pairs of training points. We empirically evaluate the performance of several such methods and find that a simple heuristic that discards faraway training points (the hybrid dense method described above) may yield the best performance on many real datasets. This follows Murray (2009), who found that tree-based methods are often unable to effectively merge points, though we do identify a regime of clustered data in which the product tree has an advantage. Other contributions of this paper include a cutoff rule with provable error bounds, applicable to both mean and covariance calculations on trees, and a description of efficient calculation in GP models incorporating both sparse and low-rank components, showing how such models can model global-scale variation while maintaining the efficiency of short-lengthscale GPs.

A limitation of all of the approaches considered in this paper is the need to invert the kernel matrix during training; this can be difficult for large problems. One avenue for future work could be an iterative factorization of K_y analogous to the CG training performed by MVM methods (Shen et al., 2006; Gray, 2004; Morariu et al., 2008).

Although our work has been focused primarily on low-dimensional applications, the use of cover trees instead of k-d trees ought to enable an extension to higher dimensions. We are not aware of previous work applying tree-based regression algorithms to high-dimensional data, but

	Direct (ms)	HS_N(ms)	HS (ms)	HD (ms)	PT (ms)
seismic	13.1 ± 1.1	4.5 ± 0.9	2.0 ± 1.6	**1.5** ± 2.7	1.6 ± 2.4
precip	45.0 ± 2.0	5.0 ± 1.5	5.7 ± 2.3	**3.0** ± 1.0	7.3 ± 3.4
snow	13.0 ± 1.0	4.3 ± 0.6	1.6 ± 0.4	**0.9** ± 0.5	1.0 ± 0.5
tco	19.4 ± 4.9	17.4 ± 6.4	2.7 ± 1.2	**1.7** ± 0.2	**1.7** ± 0.5
housing	12.2 ± 1.0	4.9 ± 0.8	1.5 ± 0.3	**0.8** ± 0.3	**0.8** ± 0.3

Table 4: Time to compute posterior variance of a CS+FIC model at points from the test set: mean $\pm$ standard deviation.

	HD	PT
seismic	6912	**6507**
precip	**21210**	22849
snow	1394	**1374**
tco	**169**	177
housing	1561	**670**

Table 5: Mean number of nonzero terms in the approximate weighted sums computed at test points.

as high-dimensional covariance matrices are often sparse, this may be a natural fit. For high-dimensional data that do not lie on a low-dimensional manifold, other nearest-neighbor techniques such as locality-sensitive hashing (Andoni and Indyk, 2008) may have superior properties to tree structures; the adaptation of such techniques to GP regression is an interesting open problem.

Acknowledgements

The authors are grateful to the anonymous reviewers for their helpful feedback, and to Iain Murray for suggesting the hybrid dense method. This work was supported by DTRA grant #HDTRA-11110026.

References

Andoni, A. and Indyk, P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122.

Bank, R. E. and Douglas, C. C. (1993). Sparse matrix multiplication package (SMMP). *Advances in Computational Mathematics*, 1(1):127–137.

Beygelzimer, A., Kakade, S., and Langford, J. (2006). Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 97–104.

Bickel, P. J. and Lindner, M. (2012). Approximating the inverse of banded matrices by banded matrices with applications to probability and statistics. *SIAM Journal on Probability Theory and Applications*, 56:1–20.

Chalupka, K., Williams, C. K., and Murray, I. (2013). A framework for evaluating approximation methods for Gaussian process regression. *Journal of Machine Learning Research*, 14:333–350.

Friedman, J. H., Bentley, J. L., and Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226.

Gray, A. (2004). Fast kernel matrix-vector multiplication with application to Gaussian process learning. Technical Report CMU-CS-04-110, School of Computer Science, Carnegie Mellon University.

Gray, A. G. and Moore, A. W. (2001). N-body problems in statistical learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 521–527.

Kennett, B. N. and Engdahl, E. (1991). Traveltimes for global earthquake location and phase identification. *Geophysical Journal International*, 105(2):429–465.

Moore, A. W., Schneider, J., and Deng, K. (1997). Efficient locally weighted polynomial regression predictions. In *Proceedings of the 14th International Conference on Machine Learning (ICML)*.

Morariu, V., Srinivasan, B. V., Raykar, V. C., Duraiswami, R., and Davis, L. (2008). Automatic online tuning for fast Gaussian summation. *Advances in Neural Information Processing Systems (NIPS)*, 21:1113–1120.

Murray, I. (2009). Gaussian processes and fast matrix-vector multiplies. In *Numerical Mathematics in Machine Learning workshop at the 26th International Conference on Machine Learning (ICML 2009)*.

Park, C., Huang, J. Z., and Ding, Y. (2011). Domain decomposition approach for fast gaussian process regression of large spatial data sets. *The Journal of Machine Learning Research*, 12:1697–1728.

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

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems (NIPS)*, 20:1177–1184.

Rasmussen, C. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Shen, Y., Ng, A., and Seeger, M. (2006). Fast Gaussian process regression using kd-trees. In *Advances in Neural Information Processing Systems (NIPS)*, volume 18, page 1225.

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

Snelson, E. and Ghahramani, Z. (2007). Local and global sparse Gaussian process approximations. In *Artificial Intelligence and Statistics (AISTATS)*, volume 11.

Uhlmann, J. K. (1991). Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175 – 179.

Vanhatalo, J. and Vehtari, A. (2008). Modelling local and global phenomena with sparse Gaussian processes. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*.

Vedaldi, A. and Zisserman, A. (2010). Efficient additive kernels via explicit feature maps. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3539–3546. IEEE.