Product Trees for Gaussian Process Covariance in Sublinear Time

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 process (GP) regression is a powerful technique for nonparametric regression; unfortunately, calculating the predictive variance in a standard GP model requires time $O(n^2)$ in the size of the training set. This is cost prohibitive when GP likelihood calculations must be done in the inner loop of the inference procedure for a larger model (e.g., MCMC). Previous work by Shen et al. (2006) used a k-d tree structure to approximate the predictive mean in certain GP models. We extend this approach to achieve efficient approximation of the predictive covariance using a tree clustering on pairs of training points. We show empirically that this significantly increases performance at minimal cost in accuracy. Additionally, we apply our method to "primal/dual" models having both parametric and nonparametric components and show that this enables efficient computations even while modeling longer-scale variation.

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. As a natively probabilistic technique, Gaussian process (GP) regression (Rasmussen and Williams, 2006) is a natural fit for such systems, but its applications in large-scale Bayesian models have been limited by computational concerns: training a GP model on n points requires $O(n^3)$ time, while computing the predictive distribution at a test point requires $O(n)$ and $O(n^2)$ operations for the mean and variance respectively.

This work focuses specifically on the fast evaluation of GP likelihoods, motivated by the desire for efficient inference in models that include a GP regression component. In particular, we focus on the predictive covariance, since this computation time generally dominates that of the predictive mean. In our setting, training time is a secondary concern: the model can always be trained offline, but the likelihood evaluation occurs in the inner loop of an ongoing inference procedure, and must be efficient if inference is to be feasible.

One approach to speeding up GP regression, common especially to spatial applications, is the use of covariance kernels with short lengthscales to induce sparsity or near-sparsity in the kernel matrix. This can be exploited directly using sparse linear algebra packages (Vanhatalo and Vehtari, 2008) or by more structured techniques such as space-partitioning trees (Shen et al., 2006; Gray, 2004); the latter approaches create a query-dependent clustering to avoid considering regions of the data not relevant to a particular query. However, previous work has focused on efficient calculation of the predictive mean, rather than the variance, and the restriction to short lengthscales also inhibits application to data that contain longer-scale variations.

In this paper, we develop a tree-based method to efficiently compute the predictive covariance in GP models. Our work extends the weighted sum algorithm of Shen et al. (2006), which computes the predictive mean. Instead of clustering points with similar weights, we cluster *pairs* of points having similar weights, where the weights are given by a kernel-dependent distance metric defined on the *product space* consisting of all pairs of training points. We show how to efficiently build and compute using a *product tree* constructed in this space, yielding an adaptive covariance computation that exploits the geometric structure of the training data to avoid the need to explicitly consider each pair of training points. This enables us to present what is to our knowledge the first account of GP regression in which the major test-time operations

(predictive mean, covariance, and likelihood) run in time sublinear in the training set size, given a suitably sparse kernel matrix. As an extension, we show how our approach can be applied to GP models that combine both parametric and nonparametric components, and argue that such models present a promising option for modeling global-scale structure while maintaining the efficiency of short-lengthscale GPs. Finally, we present empirical results that demonstrate significant speedups on synthetic data as well as a real-world seismic dataset.

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

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

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 derive a Gaussian posterior distribution $f(X^*) \sim \mathcal{N}(\mu_*, \Sigma_*)$, where

$$\mu_* = K^{*T} K_y^{-1} \mathbf{y} \quad (1)$$

$$\Sigma_* = K^{**} - K^{*T} K_y^{-1} K^* \quad (2)$$

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 the exponential, squared-exponential, rational quadratic, and Matérn kernel families; anisotropic kernels can be represented through choice of an appropriate metric.

2.2 k-d and Metric Trees

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.

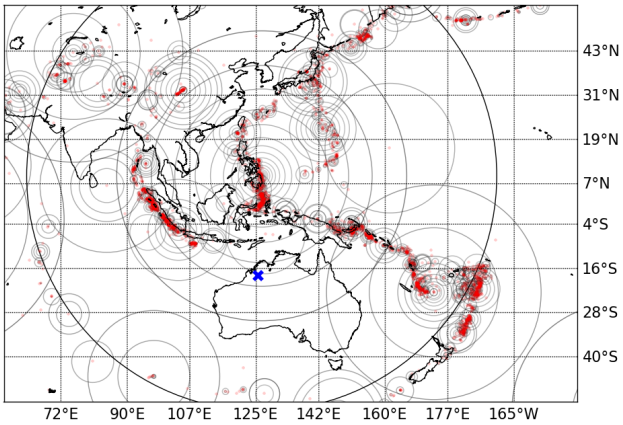


Figure 1: Cover tree decomposition of seismic event locations recorded at Fitzroy Crossing, Australia (with X marking the station location).

Major drawbacks of k-d trees include poor performance in high dimensions and a limitation to Euclidean spaces. By contrast, we are interested in non-Euclidean metrics both as a matter of practical application (e.g., in a geophysical setting we might consider points on the surface of the earth) and because some choices of kernel function require our algorithm to operate under a non-Euclidean metric even if the underlying space is Euclidean (see section 3.2). We therefore consider instead the class of trees having the following properties: (a) each 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 $x_\mathbf{n}$, and (b) for each leaf $\mathbf{L}$ we have $x_\mathbf{L} \in X$, with exactly one leaf node for each training point $\mathbf{x}_i \in X$. We call any tree satisfying these properties a *metric tree*.

function WEIGHTEDMETRICSUM(node $\mathbf{n}$, query points $(\mathbf{x}_i^*, \mathbf{x}_j^*)$, accumulated sum $\hat{S}$, tolerances ϵ_{rel}, 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**
    $\hat{S} \leftarrow \hat{S} + (K_y^{-1})_{\mathbf{n}} \cdot \left(k(d(\mathbf{x}_i^*, \mathbf{n}_1)) \cdot k(d(\mathbf{x}_j^*, \mathbf{n}_2))\right)$
  else
    $w_{\min} \leftarrow k_{\text{lower}}^{\text{prod}}(\delta_{\mathbf{n}} + r_{\mathbf{n}})$
    $w_{\max} \leftarrow k_{\text{upper}}^{\text{prod}}(\max(\delta_{\mathbf{n}} - r_{\mathbf{n}}, 0))$
    if $w_{\max} \cdot S_{\mathbf{n}}^{\text{Abs}} \leq \left(\epsilon_{\text{rel}} \left|\hat{S} + w_{\min} \cdot S_{\mathbf{n}}^{\text{UW}}\right| + \epsilon_{\text{abs}}\right)$ **then**
      $\hat{S} \leftarrow \hat{S} + \frac{1}{2}(w_{\max} + w_{\min}) \cdot S_{\mathbf{n}}^{\text{UW}}$
    else
      for each child $\mathbf{c}$ of $\mathbf{n}$
       sorted by ascending $\delta((\mathbf{x}_i^*, \mathbf{x}_j^*), (\mathbf{c}_1, \mathbf{c}_2))$ **do**
        $\hat{S} \leftarrow \hat{S} +$ WEIGHTEDMETRICSUM($\mathbf{c}$, $(\mathbf{x}_i^*, \mathbf{x}_j^*)$, $\hat{S}$, ϵ_{rel}, ϵ_{abs})
      end for
    end if
  end if
  return $\hat{S}$
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.

Examples of metric trees include many structures designed specifically for nearest-neighbor queries, such as ball trees (Uhlmann, 1991) and cover trees (Beygelzimer et al., 2006), but in principle any hierarchical clustering of the dataset, e.g., an agglomerative clustering, might be augmented with radius information to create a metric tree. Although our algorithms can operate on any metric tree structure, we use cover trees in our implementation and experiments. 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.

3 Efficient Covariance using Product Trees

We consider efficient calculation of the GP covariance (2). 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 $k(\mathbf{x_i^*}, \mathbf{x_p}) k(\mathbf{x_j^*}, \mathbf{x_q})$:

$$\mathbf{k_i^*}^T K_y^{-1} \mathbf{k_j^*} = \sum_{p=1}^{n} \sum_{q=1}^{n} (K_y^{-1})_{pq} k(\mathbf{x_i^*}, \mathbf{x_p}) k(\mathbf{x_j^*}, \mathbf{x_q}). \quad (3)$$

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, as discussed in section 3.2 below. For the moment, we will assume a SE kernel, of the form $k_{SE}(d) = \exp(-d^2)$, for which a natural choice is the 2-product metric:

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

Note that this metric, taken together with the SE kernel, 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 kernel in the product space (rhs) gives us the correct weight for our weighted sum (3) (lhs).

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. In principle, this tree contains n^2 leaves, one for each pair of training points. In practice it can often be made much smaller; see section 3.1 for details. At each leaf node $\mathbf{L}$, representing a pair of training points, we store the element $(K_y^{-1})_{\mathbf{L}}$ 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):

$$S_{\mathbf{n}}^{\text{UW}} = \sum_{\mathbf{L} \in \text{leaves}(\mathbf{n})} (K_y^{-1})_{\mathbf{L}} \quad (4)$$

$$S_{\mathbf{n}}^{\text{Abs}} = \sum_{\mathbf{L} \in \text{leaves}(\mathbf{n})} \left|(K_y^{-1})_{\mathbf{L}}\right|. \quad (5)$$

Given a product tree augmented in this way, the weighted-sum calculation (3) is performed by the WEIGHTEDMETRICSUM algorithm of Figure 2. This algorithm is similar to the WEIGHTEDSUM and WEIGHTEDXTXBELOW algorithms of Shen et al. (2006) and Moore et al. (1997) respectively, but

adapted to the non-Euclidean and non-binary tree setting, and further adapted to make use of bounds on the product kernel (see section 3.2). It proceeds by a recursive descent down the tree, where at each non-leaf node it computes upper and lower bounds on the weight of any descendant, and applies a cutoff rule to determine whether to continue the descent. 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. However, our setting differs from theirs: since we are computing a weighted sum over entries of K_y^{-1}, which we expect to be approximately sparse, we expect that some entries will contribute much more than others. Thus we want our cutoff rule to account for the weights of the sum *and* the entries of K_y^{-1} that are being summed over. We do this by defining a rule in terms of the current running weighted sum,

$$w_{\max} \cdot S_{\mathbf{n}}^{\mathrm{Abs}} \leq \left(\epsilon_{\mathrm{rel}} \left|\hat{S} + w_{\min} \cdot S_{\mathbf{n}}^{\mathrm{UW}}\right| + \epsilon_{\mathrm{abs}}\right), \quad (6)$$

which we have found to significantly improve performance in covariance calculations compared to the weight-based rule of Moore et al. (1997) and Shen et al. (2006). Here $\hat{S}$ is the weighted sum accumulated thus far, and ϵ_{abs} and ϵ_{rel} are tunable approximation parameters. We interpret the left-hand side of (6) as computing an upper bound on the contribution of node $\mathbf{n}$'s descendents to the final sum, while the absolute value on the right-hand side gives an estimated lower bound on the magnitude of the final sum (note that this is not a true bound, since the sum may contain both positive and negative terms, but it appears effective in practice). If the leaves below the current node $\mathbf{n}$ appear to contribute a negligible fraction of the total sum, we approximate the contribution from $\mathbf{n}$ by $\frac{1}{2}(w_{\max} + w_{\min}) \cdot S_{\mathbf{n}}^{\mathrm{UW}}$, i.e., by the average weight times the unweighted sum. Otherwise, the computation continues recursively over $\mathbf{n}$'s children. Following Shen et al. (2006), we recurse to child nodes in order of increasing distance from the query point, so as to accumulate large sums early on and increase the chance of cutting off later recursions.

3.1 Implementation

A naïve product tree on n points will have n^2 leaves, but we can reduce this and achieve substantial speedups by exploiting the structure of K_y^{-1} and of the product space $\mathcal{M} \times \mathcal{M}$:

Sparsity. If K_y is sparse, or can be well-approximated by a sparse matrix, then K_y^{-1} is often also sparse (or well-approximated as sparse) in practice. This occurs in the case of compactly supported kernel functions (Gneiting, 2002; Rasmussen and Williams, 2006), but also even when using standard kernels with short lengthscales. Note that although there is no guarantee that the inverse of a sparse matrix must itself be sparse (with the exception of specific structures, e.g., block diagonal matrices), it is often the case that when K_y is sparse many entries of K_y^{-1} will be very near to zero, since points with negligible covariance generally also have negligibly small correlations in the precision matrix, so K_y^{-1} can often be well-approximated as sparse. When this is the case, our product tree need include only those pairs $(\mathbf{x}_p, \mathbf{x}_q)$ for which $(K_y^{-1})_{pq}$ is non-negligible. This is often a substantial advantage.

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. We can decompose $K_y^{-1} = U + D + U^T$, where $D = \mathrm{diag}(K_y^{-1})$ is a diagonal matrix and $U = \mathrm{triu}(K_y^{-1})$ is a strictly upper triangular (zero diagonal) matrix. This allows us to rewrite

$$\mathbf{k_i^*}^T K_y^{-1} \mathbf{k_j^*} = \mathbf{k_i^*}^T U \mathbf{k_j^*} + \mathbf{k_i^*}^T D \mathbf{k_j^*} + \mathbf{k_i^*}^T U^T \mathbf{k_j^*},$$

in which the first and third terms can be implemented as calls to WeightedMetricSum on a product tree built from U; note that this tree will be half the size of a tree built for K_y^{-1} since we omit zero entries. The second (diagonal) term can be computed using a separate (very small) product tree built from the nonzero entries of D. The accumulated sum $\hat{S}$ can be carried over between these three computations, so we can speed up the later computations by accumulating large weights in the earlier computations.

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. This can dramatically speed up tree construction when the distance metric is slow to evaluate. It can also speed up test-time evaluation, if distances to the same point must be computed at multiple levels of the tree.

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

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	$\exp\left(-d^\gamma\right)$	$\exp\left(-d_1^\gamma-d_2^\gamma\right)$	$\left(d_1^\gamma+d_2^\gamma\right)^{1/\gamma}$	$\exp\left(-(\delta)^\gamma\right)$	$\exp\left(-(\delta)^\gamma\right)$
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 ($\nu=3/2$)	$\left(1+\sqrt{3}d\right)$ $\cdot\exp\left(-\sqrt{3}d\right]\Big)$	$\left(1+\sqrt{3}\left(d_1+d_2\right)+3d_1d_2\right)$ $\cdot\exp\left(-\sqrt{3}(d_1+d_2)\right)\Big)$	d_1+d_2	$\left(1+\sqrt{3}\delta\right)$ $\cdot\exp\left(-\sqrt{3}\delta\right)$	$\left(1+\sqrt{3}\delta+3(\delta/2)^2\right)$ $\cdot\exp(-\sqrt{3}\delta)$
Piecewise polynomial (compact support), $q=1$, dimension D, $j=\left\lfloor\frac{D}{2}\right\rfloor+2$	$(1-d)_+^{j+1}$ $\cdot((j+1)d+1)$	$\left((1-d_1)_+(1-d_2)_+\right)^{j+1}$ $\cdot\Big((j+1)^2d_1d_2$ $+(j+1)(d_1+d_2)+1\Big)$	d_1+d_2	$(1-\delta)_+^{j+1}$ $\cdot((j+1)\delta+1)$	$\left(1-\delta+\frac{(\delta)^2}{4}\right)_+^{j+1}$ $\cdot\left((j+1)^2\left(\frac{\delta}{2}\right)^2+(j+1)\delta+1\right)$

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

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 cutoff rule (6), 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. As an example, consider the Matérn kernel

$$k_{\text{M}}(d)=(1+\sqrt{3}d)\exp(-\sqrt{3}d)$$

(where we have taken $\nu=3/2$); this kernel is popular in geophysics because its sample paths are once-differentiable, as opposed to infinitely smooth as with the SE kernel. Considering the product of two Matérn kernels,

$$k_{\text{M}}(d_1)k_{\text{M}}(d_2)= (1+\sqrt{3}(d_1+d_2)+3d_1d_2)\exp(-\sqrt{3}(d_1+d_2))$$

we notice that this is almost equivalent to $k_{\text{M}}(\delta)$ for the choice of $\delta=d_1+d_2$, but with an additional pairwise term of $3d_1d_2$. We bound this term by noting that it is maximized when $d_1=d_2=\delta/2$ and minimized whenever either $d_1=0$ or $d_2=0$, so we have $3(\delta/2)^2\geq 3d_1d_2\geq 0$. This yields the bounds $k^{\text{prod}}_{\text{lower}}$ and $k^{\text{prod}}_{\text{upper}}$ as shown in Table 1. Bounds for other common kernels are obtained analogously in Table 1.

4 Primal / Dual and Mixed GP Representations

In this section, we extend the product tree approach to models combining a long-scale parametric component with a short-scale nonparametric component. We introduce these models, which we refer to as *mixed primal/dual* GPs, and demonstrate how they can mediate between the desire to model long-scale structure and the need to maintain a short lengthscale for efficiency. (Although this class of models is well known, we have not seen this particular use case described in the literature). We then show that the necessary computations in these models can be done efficiently using the techniques described above.

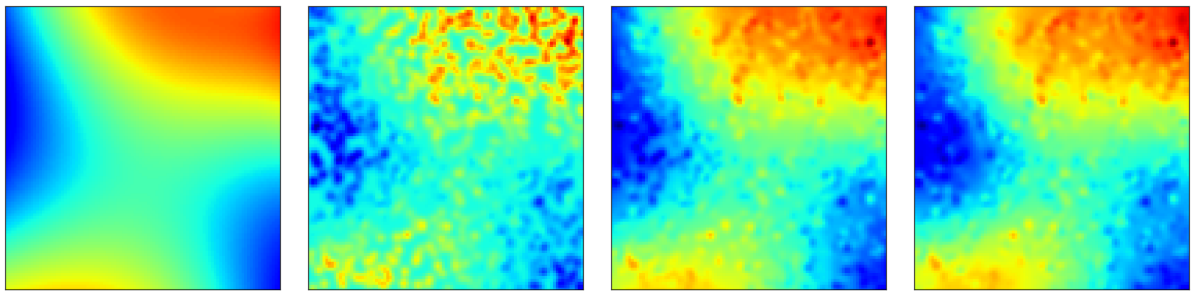
4.1 Mixed Primal/Dual GP Models

Although GP regression is commonly thought of as nonparametric, it is possible to implement parametric models within the GP framework. For example, a Bayesian linear regression model with Gaussian prior,

$$y=\mathbf{x}^T\beta+\epsilon,\ \ \beta\sim\ \mathcal{N}(\mathbf{0},I),\ \ \epsilon\sim\mathcal{N}(0,\sigma_n^2),$$

is equivalent to GP regression with a linear kernel $k(\mathbf{x},\mathbf{x}')=\langle\mathbf{x},\mathbf{x}'\rangle$, in the sense that both models yield the same (Gaussian) predictive distributions (Rasmussen and Williams, 2006). However, the two representations have very different *computational* properties: the primal (parametric) representation allows computation of the predictive mean and variance in $O(D)$ and $O(D^2)$ time respectively, where D is the input dimensionality, while the dual (nonparametric) representation requires time $O(n)$ and $O(n^2)$ respectively for the same calculations. When learning simple models on large, low-dimensional (e.g., spatial) data sets, the primal representation is obviously more attractive, since we can store and compute with model parameters directly, in constant time relative to n.

Of course, simple parametric models by themselves cannot capture the complex local structure that often appears in real-world datasets. Fortunately it is possible to combine a parametric model with a nonparametric GP model in a way that retains the advantages of both approaches. To define a combined model, we replace the standard zero-mean GP assumption with a



(a) *MAD*= 1.11
polynomial(5)

(b) *MAD*= 1.77
GP w/ $\ell = 0.02$

(c) *MAD*= 0.73
Mixed:
polynomial(5) +
GP w/ $\ell = 0.02$

(d) *MAD*= 0.65
True GP w/
$\ell_1 = 0.5$
$\ell_2 = 0.02$

Figure 3: A primal/dual mixture approximating a longer-scale GP.

parametric mean function $\mathbf{h}(\mathbf{x})^T\beta$, yielding the model

$$y = f(\mathbf{x}) + \mathbf{h}(\mathbf{x})^T\beta + \epsilon$$

where $\mathbf{h}(\mathbf{x})$ is a vector of feature values $[h_1(\mathbf{x}), \ldots, h_D(\mathbf{x})]$. The GP model is then learned jointly along with a posterior distribution on the coefficients β. Assuming a Gaussian prior $\beta \sim \mathcal{N}(b, B)$ on the coefficients, the predictive distribution $g(X^*) \sim \mathcal{N}(\mu'_*, \Sigma'_*)$ can be derived (Rasmussen and Williams, 2006) as

$$\mu'_* = H^{*T}\bar{\beta} + K^{*T}K_y^{-1}(\mathbf{y} - H^{*T}\bar{\beta}) \quad (7)$$

$$\Sigma'_* = K^{**} - K^{*T}K_y^{-1}K^* + R^T(B^{-1} + HK_y^{-1}H^T)R \quad (8)$$

where we define $H_{ij} = h_j(\mathbf{x}_i)$ for each training point x_i, similarly H^* for the test points, and we have $\bar{\beta} = (B^{-1} + HK_y^{-1}H^T)^{-1}(HK_y^{-1}\mathbf{y} + B^{-1}\mathbf{b})$ and $R = H^* - HK_y^{-1}K^*$. Section 2.7 of Rasmussen and Williams (2006) gives further details.

Note that linear regression in this framework corresponds to a choice of basis functions $h_1(x) = 1$ and $h_2(x) = x$; it is straightforward to extend this to polynomial regression and other models that are linear in their parameters. In general, any kernel which maps to a finite-dimensional feature space can be represented parametrically in that feature space, so this framework can efficiently handle kernels of the form $k(\mathbf{x}, \mathbf{x}') = \sum_i k_i(\mathbf{x}, \mathbf{x}') + k_S(\mathbf{x}, \mathbf{x}')$, where k_S is a short-lengthscale or compactly supported kernel, monotonically decreasing w.r.t. some distance metric as assumed above, and each k_i either has an exact finite-dimensional feature map or can be approximated using finite-dimensional features Rahimi and Recht (2007); Vedaldi and Zisserman (2010).

As an example, Figure 3 compares several approaches for inferring a function from a GP with long and short-lengthscale components. We drew training data from a GP with a mixture of two SE kernels at lengthscales $\ell_1 = 0.5$ and $\ell_2 = 0.02$, sampled at 1000 random points in the unit square. Figure 3 displays the posterior means of four models on a 100 by 100 point grid, reporting the mean absolute deviation (MAD) of the model predictions relative to the "true" values (drawn from the same GP) at 500 random test points. Note that although the short-scale GP (3b) cannot by itself represent the variation from the longer-scale kernel, when combined with a parametric polynomial component (3a) the resulting mixed model (3c) achieves accuracy approaching that of the true model (3d).

4.2 Efficient Operations in Primal/Dual Models

Likelihood calculation in primal/dual models is a straightforward extension of the standard case. The predictive mean (7) can be accommodated within the framework of Shen et al. (2006) using a tree representation of the vector $K_y^{-1}\left(\mathbf{y} - H^{*T}\bar{\beta}\right)$, then adding in the easily evaluated parametric component $H^{*T}\bar{\beta}$. In the covariance (8) we can use a product tree to approximate $K^{*T}K_y^{-1}K^*$ as described above; of the remaining terms, $\bar{\beta}$ and $B^{-1} + HK_y^{-1}H^T$ can be precomputed at training time, and H^* and K^{**} don't depend on the training set. This leaves $HK_y^{-1}K^*$ as the one remaining challenge; we note that this quantity can be computed efficiently using mD applications of the vector multiplication primitive from Shen et al. (2006), re-using the same tree structure to multiply each column of K^* by each row of HK_y^{-1}. Thus, all of the the operations required for likelihood computation can be implemented efficiently with no explicit dependence on n (i.e., with no direct access to the training set except through space-partitioning tree structures).

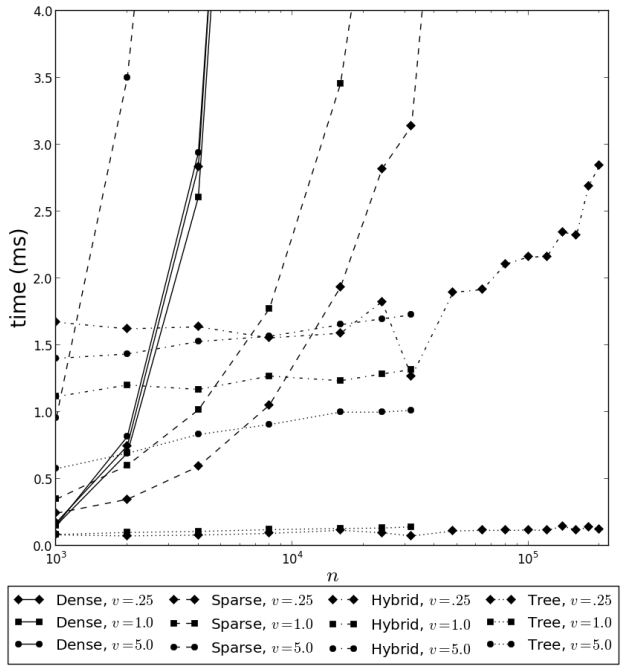


Figure 4: Mean runtimes for dense, sparse, hybrid, and product tree calculation of GP variance on a 2D synthetic dataset.

5 Evaluation

We compare calculation of the predictive variance using a product tree to several other approaches: a naïve implementation using dense matrices, a direct calculation using a sparse representation of K_y^{-1} and dense representation of $\mathbf{k_i^*}$, and a hybrid tree implementation that attempts to also construct a sparse $\mathbf{k_i^*}$ by querying a cover tree for all training points within distance r of the query point $\mathbf{x_i^*}$, where r is chosen such that $k(r')$ is negligible for $r' > r$, and then filling in only those entries of $\mathbf{k_i^*}$ determined to be non-negligible.

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 3.1. The approximation parameters ϵ_{rel} and ϵ_{abs} were set appropriately for each experiment so as to ensure that the mean approximation error is 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.

Figure 4 compares performance of these approaches 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 average time per point to compute the predictive variance at 1000 random test points. The GP uses an SE kernel with observation noise $\sigma_n^2 = 0.1$ 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 a random query point (thus, on average there will be $4v$ points within two lengthscales, $9v$ within three lengthscales, etc.).

The results of Figure 4 show a significant advantage for the tree-based approaches, which are able to take advantage of the geometric sparsity structure in the training data. The dense implementation is relatively fast on small data sets but quickly blows up, while the sparse calculation holds on longer (except in the relatively dense $v = 5.0$ setting) but soon succumbs to linear growth, since it must evaluate the kernel between the test point and each training point. The hybrid approach has higher overhead but scales very efficiently until about $n = 48000$, where the sparse matrix multiplication's $\Omega(n)$ runtime (Bank and Douglas, 1993) begins to dominate. Conversely, the product tree remains efficient even for very large, sparse datasets, with $v = 0.25$ runtimes growing from 0.08ms at $n = 1000$ to just 0.13ms at $n = 200000$. Due to memory limitations we were unable to evaluate $v = 1.0$ and $v = 5.0$ for values of n greater than 32000.

Our second experiment uses amplitude data from 3105 seismic events (earthquakes) detected by a station in Fitzroy Crossing, Australia; the event locations are shown in Figure 1. The amplitudes are normalized for event magnitude, and the task is to predict the recorded amplitude of a new event given that event's latitude, longitude, and depth. Here our distance metric is the great-circle distance, and we expect our data to contain both global trends and local structure, since events further away from the detecting station will generally have lower amplitudes, but this may vary locally as signals from a given source region generally travel along the same paths through the earth and are dampened or amplified in the same ways as they travel to the detecting station.

Table 2 considers several models for this data. A simple parametric model, the fifth-order polynomial in event-to-station distance shown in Figure 5, is not very accurate but does allow for very fast variance evaluations. The GP models are more accurate, but the most accurate GP model uses a relatively long lengthscale of 50km, with correspondingly slow variance calculations. Depending on application requirements, the most appealing tradeoff might be given by the mixed model combining a fifth-degree polynomial with a 10km SE GP: this model achieves accuracy close to that of the 50km models, but with significantly faster variance calculations due to the shorter lengthscale, especially when using a product tree.

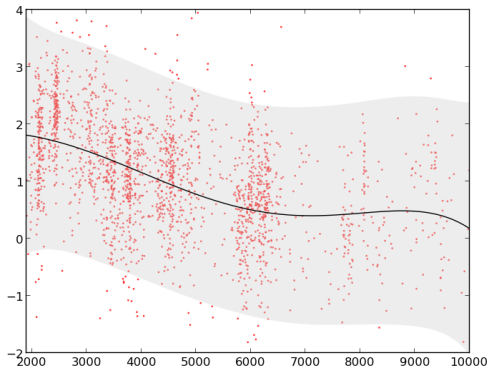


Figure 5: Normalized amplitude as a function of event-station distance, with a fifth-degree polynomial fit shading ±2std.

Model	Error	Sparse (ms)	Tree (ms)
Polynomial in distance (deg 5)	0.78	0.050	n/a
GP, SE, $\ell = 10$km	0.67	0.722 ± 0.032	0.216 ± 0.224
poly/GP, deg 5,SE, 10km	0.62	0.795 ± 0.033	0.413 ± 0.307
GP, Matérn, $\ell = 10$km	0.65	1.256 ± 0.592	0.337 ± 0.365
poly/GP, deg 5, Matérn, 10km	0.62	1.327 ± 0.602	0.654 ± 0.499
GP, SE, $\ell = 50$km	0.61	1.399 ± 0.661	1.168 ± 1.242
poly/GP, deg 5, SE, 50km	0.60	1.490 ± .677	1.551 ± 1.409

Table 2: Models for Fitzroy Crossing amplitude data, with mean absolute prediction error from five-fold cross validation and (mean ± std) time to compute the predictive variance via a direct sparse calculation versus a product tree.

6 Related Work

Previous approximations for GP mean prediction (Moore et al., 1997; Shen et al., 2006; Gray, 2004), which inspired this work, use tree structures to implement an efficient matrix-vector multiplication (MVM); the Improved Fast Gauss Transform (Morariu et al., 2008) also 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, our product tree approach) 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 parametric component.

Another set of approaches to speeding up GP regression, sparse approximations (Csató and Opper, 2002; Seeger et al., 2003; Snelson and Ghahramani, 2006; Quiñonero-Candela and Rasmussen, 2005), attempt to represent n training points using a smaller set of m points, allowing training in $O(nm^2)$ time and predictive covariance (thus likelihood) computation in $O(m^2)$ time. This is philosophically a different approach from that of this paper, where we generally want to retain all of our training points in order to represent local structure. However, there is no formal incompatibility: many sparse approaches, including all of those discussed by Quiñonero-Candela and Rasmussen (2005), yield predictive covariances of the form $\mathbf{k}_\mathbf{i}^{*T} Q \mathbf{k}_\mathbf{j}^*$ for some matrix Q (or a sum of terms of this form), where this product could be computed straightforwardly using a product tree. Several non-sparse approximations, e.g., the Nyström approximation (Williams and Seeger, 2001), also yield predictive covariances of this form.

More closely related to our setting are 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. Since training multiple local GPs is equivalent to training a single global GP with a block diagonal covariance matrix, it should be possible to enhance local GPs with global parametric components as in section 4, similarly to the combined local/global approximation of Snelson and Ghahramani (2007).

7 Conclusion and Future Work

We introduce the *product tree* structure for efficient adaptive calculation of GP covariances using a multiresolution clustering of pairs of training points. Specific contributions of this paper include product metrics and bounds for common kernels, the adaptation to metric trees, a novel cutoff rule incorporating both the weights and the quantity being summed over, and covariance-specific performance optimizations. Additionally, we describe efficient calculation in GP models incorporating both primal and dual components, and show how such models can model global-scale variation while maintaining the efficiency of short-lengthscale GPs.

A limitation of our approach is the need to explicitly invert the kernel matrix during training; this can be quite 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). Another topic would be a better understanding of cutoff rules for the weighted sum recursion, e.g., an empirical investigation of different rules or a theoretical analysis bounding the error and/or runtime of the overall computation.

Finally, 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 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.

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.

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.

Csató, L. and Opper, M. (2002). Sparse online Gaussian processes. *Neural Computation*, 14(3):641–668.

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.

Gneiting, T. (2002). Compactly supported correlation functions. *Journal of Multivariate Analysis*, 83(2):493–508.

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.

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)*.

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.

Seeger, M., Williams, C. K., and Lawrence, N. D. (2003). Fast forward selection to speed up sparse Gaussian process regression. In *Artificial Intelligence and Statistics (AISTATS)*, volume 9.

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.

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