Journal of Machine Learning Research 18 (2017) 1-42 Submitted 7/15; Revised 6/17; Published 8/17

Hierarchically Compositional Kernels for Scalable Nonparametric Learning

Jie Chen CHENJIE@US.IBM.COM
IBM Thomas J. Watson Research Center

Haim Avron HAIMAV@POST.TAU.AC.IL
Tel Aviv University

Vikas Sindhwani SINDHWANI@GOOGLE.COM
Google Brain, NYC

Editor: Koji Tsuda

Abstract

We propose a novel class of kernels to alleviate the high computational cost of large-scale nonparametric learning with kernel methods. The proposed kernel is defined based on a hierarchical partitioning of the underlying data domain, where the Nyström method (a globally low-rank approximation) is married with a locally lossless approximation in a hierarchical fashion. The kernel maintains (strict) positive-definiteness. The corresponding kernel matrix admits a recursively off-diagonal low-rank structure, which allows for fast linear algebra computations. Suppressing the factor of data dimension, the memory and arithmetic complexities for training a regression or a classifier are reduced from $O(n^2)$ and $O(n^3)$ to $O(nr)$ and $O(nr^2)$, respectively, where n is the number of training examples and r is the rank on each level of the hierarchy. Although other randomized approximate kernels entail a similar complexity, empirical results show that the proposed kernel achieves a matching performance with a smaller r. We demonstrate comprehensive experiments to show the effective use of the proposed kernel on data sizes up to the order of millions.

Keywords: Hierarchical kernels, Nonparametric Learning

1. Introduction

Kernel methods (Schölkopf and Smola, 2001; Hastie et al., 2009) constitute a principled framework that extends linear statistical techniques to nonparametric modeling and inference. Applications of kernel methods span the entire spectrum of statistical learning, including classification, regression, clustering, time-series analysis, sequence modeling (Song et al., 2013), dynamical systems (Boots et al., 2013), hypothesis testing (Harchaoui et al., 2013), and causal modeling (Zhang et al., 2011). Under a Bayesian treatment, kernel methods also admit a parallel view in Gaussian processes (GP) (Rasmussen and Williams, 2006; Stein, 1999) that find broad applications in statistics and computational sciences, including geostatistics (Chilès and Delfiner, 2012), design of experiments (Koehler and Owen, 1996), and uncertainty quantification (Smith, 2013).

This power and generality of kernel methods, however, are limited to moderate sized problems because of the high computational costs. The root cause of the bottleneck is the fact that kernel matrices generated by kernel functions are typically dense and unstruc-

©2017 Jie Chen, Haim Avron and Vikas Sindhwani.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v18/15-376.html.

tured. For n training examples, storing the matrix costs $O(n^2)$ memory and performing matrix factorizations requires $O(n^3)$ arithmetic operations. One remedy is to resort to compactly supported kernels (e.g., splines (Monaghan and Lattanzio, 1985) and Wendland functions (Wendland, 2004)) that potentially lead to a sparse matrix. In practice, however, the support of the kernel may not be sufficiently narrow for sparse linear algebra computations to be competitive. Moreover, prior work (Anitescu et al., 2012) revealed a subtle drawback of compactly supported kernels in the context of parameter estimation, where the likelihood surface is bumpy and the optimum is difficult to locate. For this reason, we focus on the dense setting in this paper and the goal is to exploit structures that can reduce the prohibitive cost of dense linear algebra computations.

1.1 Preliminary

Let $\mathcal{X}$ be a set and let $k(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a symmetric and strictly positive-definite function. Denote by $X = \{\boldsymbol{x}_i \in \mathcal{X}\}_{i=1,\ldots,n}$ a set of points. We write $K(X,X)$, or sometimes K for short when the context is clear, to denote the kernel matrix of elements $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. *Because of the confusing terminology on functions and their counterparts on matrices, here, we follow the convention that a strictly positive-definite function k corresponds to a positive-definite matrix K, whereas a positive-definite function corresponds to a positive semi-definite matrix.* For notational convenience, we write $k(\boldsymbol{x}, X)$ to denote the row vector of elements $k(\boldsymbol{x}, \boldsymbol{x}_j)$ and similarly $k(X, \boldsymbol{x})$ to denote the column vector. In the context of regression/classification, the set $\mathcal{X}$ is often the d-dimensional Euclidean space $\mathbb{R}^d$ or a domain $S \subset \mathbb{R}^d$. Some of the methods discussed in this paper naturally generalize to a more abstract space. Associated to each point $\boldsymbol{x}_i$ is a target value $y_i \in \mathbb{R}$. We write $\boldsymbol{y}$ for the vector of all target values.

The Reproducing Kernel Hilbert Space $\mathcal{H}_k$ associated to a kernel k is the completion of the function space

$$\left\{ \sum_{i=1}^{m} \alpha_i k(\boldsymbol{z}_i, \cdot) \mid \boldsymbol{z}_i \in \mathcal{X},\, \alpha_i \in \mathbb{R},\, m \in \mathbb{Z}_+ \right\}$$

equipped with the inner product

$$\left\langle \sum_i \alpha_i k(\boldsymbol{z}_i, \cdot),\ \sum_j \beta_j k(\boldsymbol{w}_j, \cdot) \right\rangle = \sum_{ij} \alpha_i \beta_j k(\boldsymbol{z}_i, \boldsymbol{w}_j).$$

Given training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1,\ldots,n}$, a typical kernel method finds a function $f \in \mathcal{H}_k$ that minimizes the following risk functional

$$\mathcal{L}(f) = \sum_{i=1}^{n} V(f(\boldsymbol{x_i}), y_i) + \lambda \|f\|^2_{\mathcal{H}_k}, \tag{1}$$

where V is a loss function and $\lambda > 0$ is a regularization. When V is the squared loss $V(t,y) = (t-y)^2$, the Representer Theorem (Schölkopf et al., 2001) implies that the minimizer is

$$f(\boldsymbol{x}) = k(\boldsymbol{x}, X)[K(X,X) + \lambda I]^{-1} \boldsymbol{y}, \tag{2}$$

which is nothing but the well-known kernel ridge regression. Similarly, when V is the hinge loss, the minimizer leads to support vector machines.

In the GP view, the kernel k serves as a covariance function. Assuming a zero-mean Gaussian prior with covariance k, for any separate set of points X_* and the associated vector of target values $\boldsymbol{y}_*$, the joint distribution of $\boldsymbol{y}$ and $\boldsymbol{y}_*$ is thus

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{y}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right).$$

Because of the Gaussian assumption, the posterior is the conditional $\boldsymbol{y}_*|X_*, X, \boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ where

$$\boldsymbol{\mu} = K(X_*, X)K(X,X)^{-1}\boldsymbol{y}, \tag{3}$$

and

$$\Sigma = K(X_*, X_*) - K(X_*, X)K(X,X)^{-1}K(X, X_*). \tag{4}$$

A white noise of variance λ may be injected to the observations $\boldsymbol{y}$ so that the mean prediction $\boldsymbol{\mu}$ in (3) is identical to (2). One may also impose a nonzero-mean model in the prior to capture the trend in the observations (see, e.g., the classic paper O'Hagan and Kingman (1978) and also Rasmussen and Williams (2006, Section 2.7)).

Equations (2)–(4) exemplify the demand for kernels that may simplify computations for a large $K(X,X)$. We discuss a few popular approaches in the following. To motivate the discussion, we consider stationary kernels whose function value depends on only the difference of the two input arguments; that is, we can write by abuse of notation $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{r})$, where $\boldsymbol{r} = \boldsymbol{x} - \boldsymbol{x}'$; for example, the Gaussian kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right) \tag{5}$$

parameterized by σ. The Fourier transform of $k(\boldsymbol{r})$, coined *spectral density*, in a sense characterizes the decay of eigenvalues of the finitely dimensional covariance matrix K (Stein, 1999; Chen, 2013). The decay is known to be the fastest among the Matérn class of kernels (Stein, 1999; Rasmussen and Williams, 2006; Chilès and Delfiner, 2012), where Gaussian being a special case is the smoothest. Additionally, the range parameter σ also affects the decay. When $\sigma \to \infty$, K tends to a rank-1 matrix; whereas when $\sigma \to 0$, K tends to the identity. The numerical rank of the matrix varies when σ moves between the two extremes. The decay of eigenvalues plays an important role on the effectiveness of the approximate kernels discussed below.

1.2 Approximate Kernels

The first approach is low-rank kernels. Examples are Nyström approximation (Williams and Seeger, 2000; Schölkopf and Smola, 2001; Drineas and Mahoney, 2005), random Fourier features (Rahimi and Recht, 2007), and variants (Yang et al., 2014). The Nyström approximation is based on a set of landmark points, $\underline{X}$, randomly sampled from the training data X. Then, the kernel can be written as

$$k_{\text{Nyström}}(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \underline{X})K(\underline{X}, \underline{X})^{-1}k(\underline{X}, \boldsymbol{x}'). \tag{6}$$

For the convenience of deriving approximation bounds, Drineas and Mahoney (2005) consider sampling with repetition, which makes $\underline{X}$ possibly a multiset and the matrix inverse

in (6) necessarily replaced by a pseudo inverse. Various approaches for choosing the landmark points were compared in Zhang and Kwok (2010). For random Fourier features, let $\hat{k}(\boldsymbol{\omega})$ be the Fourier transform of $k(\boldsymbol{r})$ and let $\hat{k}$ be normalized such that it integrates to unity; that is, $\hat{k}$ is the normalized spectral density of k. Then, the kernel is

$$k_{\text{Fourier}}(\boldsymbol{x}, \boldsymbol{x}') = \frac{2}{r} \sum_{i=1}^{r} \cos(\boldsymbol{\omega}_i^T \boldsymbol{x} + b_i) \cos(\boldsymbol{\omega}_i^T \boldsymbol{x}' + b_i), \tag{7}$$

where r is the rank, and b_i and $\boldsymbol{\omega}_i$ are iid samples of Uniform$(0, 2\pi)$ and of a distribution with density $\hat{k}$, respectively. Note that (6) applies to any kernel whereas (7) applies to only stationary ones.

The Nyström approximation admits a conditional interpretation in the context of GP. The covariance kernel (6) can be equivalently written as

$$k_{\text{Nyström}}(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') - k(\boldsymbol{x}, \boldsymbol{x}' | \underline{X}),$$

where

$$k(\boldsymbol{x}, \boldsymbol{x}' | \underline{X}) = k(\boldsymbol{x}, \boldsymbol{x}') - k(\boldsymbol{x}, \underline{X}) K(\underline{X}, \underline{X})^{-1} k(\underline{X}, \boldsymbol{x}')$$

is nothing but the covariance of $\boldsymbol{x}$ and $\boldsymbol{x}'$ conditioned on $\underline{X}$. In other words, the covariance kernel of Nyström approximation comes from a deduction of the original covariance by a conditional covariance. The conditional covariance for any $\boldsymbol{x}$ (or symmetrically, $\boldsymbol{x}'$) within $\underline{X}$ vanishes and hence the approximation is lossless. Intuitively speaking, the closer the sites are to $\underline{X}$, the smaller the loss is. This explains frequent observations that when the size of the set $\underline{X}$ is small, using the centers of a k-means clustering as the landmark points often improves the approximation (Zhang et al., 2008; Yang et al., 2012). A caveat is that the time cost of performing k-means clustering is often much higher than that of the Nyström calculation itself. Hence, the improvement gained from clustering may not be as significant as that from increasing the size of the conditioned set $\underline{X}$. When $\underline{X}$ is large, the improvement brought about by clustering is less significant (see, e.g., Rasmussen and Williams (2006, Section 8.3.7)).

A limitation of the low-rank kernels is that the size of the conditioned set, or equivalently the rank, needs to correlate with the decay of the spectrum of K in order to yield a good approximation. For a slow decay, it is not rare to see in practice that the rank r grows to thousands (Yen et al., 2014) or even several hundred thousands (Huang et al., 2014; Avron and Sindhwani, 2016) in order to yield comparable results with other methods, for a data set of size on the order of millions. See also the experimental results in Section 5.

The second approach is a cross-domain independent kernel. Simply speaking, the kernel matrix is approximated by keeping only the diagonal blocks of the matrix. In a GP language, we partition the domain S into m sub-domains S_j, $j = 1, \ldots, m$, and make an independence assumption across sub-domains. Then, the covariance between $\boldsymbol{x}$ and $\boldsymbol{x}'$ vanishes when the two sites come from different sub-domains. That is,

$$k_{\text{independent}}(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} k(\boldsymbol{x}, \boldsymbol{x}'), & \text{if } \boldsymbol{x}, \boldsymbol{x}' \in S_j \text{ for some } j, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Whereas such a kernel appears ad hoc and associated theory is possibly limited, the scenarios when it exhibits superiority over a low-rank kernel of comparable sizes are not rare (see the

likelihood comparison in Stein (2014) and also the experimental results in Section 5). An intuitive explanation exists in the context of classification. The cross-domain independent kernel works well when geographically nearby points possess a majority of the signal for classifying points within the domain. This happens more often when the kernel has a reasonably centralized bandwidth (outside of which the kernel value becomes marginal). In such a case, nearby points are the most influential.

The third approach is covariance tapering (Furrer et al., 2006; Kaufman et al., 2008). It amounts to defining a new kernel by multiplying the original kernel k with a compactly supported kernel k_{compact}:

$$k_{\text{taper}}(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') \cdot k_{\text{compact}}(\boldsymbol{x}, \boldsymbol{x}').$$

The tapered kernel matrix is an elementwise product of two positive-definite matrices; hence, it is positive-definite, too (Horn and Johnson, 1994, Theorem 5.2.1). The primary motivation of this kernel is to introduce sparsity to the matrix. The supporting theory is drawn on the confidence interval (cf. (4)) rather than on the prediction (3). It is cast in the setting of fixed-domain asymptotics, which is similar to a usual practice in machine learning—a prescaling of each attribute to within a finite interval. The theory hints that if the spectral density of k_{compact} has a lighter tail (i.e., the spectrum of the corresponding kernel matrix decays faster) than that of k, then the ratio between the prediction variance by using the tapered kernel k_{taper} and that by using the original kernel k tends to a finite limit, as the number of training data increases to infinity in the domain. The theory holds a guarantee on the prediction confidence if we choose k_{compact} judiciously. Tapering is generally applicable to heavy-tailed kernels (e.g., Matérn kernels with low smoothness) rather than light-tailed kernels such as the Gaussian. Nevertheless, a drawback of this approach is similar to the one we stated earlier for using a compactly supported kernel alone: the range of the support must be sufficiently small for sparse linear algebra to be efficient.

1.3 Proposed Kernel

In this paper, we propose a novel approach for constructing approximate kernels motivated by low-rank kernels and cross-domain independent kernels. The construction aims at deriving a kernel that (a) maintains the (strict) positive-definiteness, (b) leverages the advantages of low-rank and independent approaches, (c) facilitates the evaluation of the kernel matrix $K(X, X)$ and the out-of-sample extension $k(X, \boldsymbol{x})$, and (d) admits fast algorithms for a variety of matrix operations. The premise of the idea is a hierarchical partitioning of the data domain and a recursive approximation across the hierarchy. Space partitioning is a frequently encountered idea for kernel matrix approximations (Si et al., 2014; March et al., 2014; Yu et al., 2017; Si et al., 2017), but maintaining positive definiteness is quite challenging. Moreover, when the approximation is performed in a hierarchical fashion and is focused on only the matrix, it is not always easy to generalize to out of samples. A particularly intriguing property of the approach proposed in this article is that both positive definiteness and out-of-sample extensions are guaranteed, because the construction acts on the kernel function itself.

2. Hierarchically Compositional Kernel

The low-rank kernel (in particular, the Nyström approximation $k_{\text{Nyström}}$) and the cross-domain independent kernel $k_{\text{independent}}$ are complementary to each other in the following sense: the former acts on the global space, where the covariance at every pair of points $\boldsymbol{x}$ and $\boldsymbol{x}'$ are deducted by a conditional covariance based on the conditioned set $\underline{X}$ chosen globally; whereas the latter preserves all the local information but completely ignores the interrelationship outside the local domain. We argue that an organic composition of the two will carry both advantages and alleviate the shortcomings. Further, a hierarchical composition may reduce the information loss in nearby local domains.

2.1 Composition of Low-Rank Kernel with Cross-Domain Independent Kernel

Let the domain S be partitioned into disjoint sub-domains $\bigcup S_j = S$. Let $\underline{X}$ be a set of landmark points in S. For generality, $\underline{X}$ needs not be a subset of the training data X. Consider the function

$$k_{\text{compositional}}(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} k(\boldsymbol{x}, \boldsymbol{x}'), & \text{if } \boldsymbol{x}, \boldsymbol{x}' \in S_j \text{ for some } j, \\ k(\boldsymbol{x}, \underline{X}) K(\underline{X}, \underline{X})^{-1} k(\underline{X}, \boldsymbol{x}'), & \text{otherwise.} \end{cases}$$

Clearly, $k_{\text{compositional}}$ leverages both (6) and (8). When two points $\boldsymbol{x}$ and $\boldsymbol{x}'$ are located in the same domain, they maintain the full covariance (8); whereas when they are located in separate domains, their covariance comes from the low-rank kernel $k_{\text{Nyström}}$ (6). Such a composition complements missing information across domains in $k_{\text{independent}}$ and also complements the information loss in local domains caused by the Nyström approximation. The following result is straightforward in light of the fact that if $\boldsymbol{x} \in \underline{X}$, then $K(\underline{X}, \underline{X})^{-1} k(\underline{X}, \boldsymbol{x})$ is a column of the identity matrix where the only nonzero element (i.e., 1) is located with respect to the location of $\boldsymbol{x}$ inside $\underline{X}$.

Proposition 1 *We have*

$$k_{compositional}(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}'),$$

if $\boldsymbol{x}, \boldsymbol{x}' \in S_j$ for some j, or if either of $\boldsymbol{x}, \boldsymbol{x}'$ belongs to $\underline{X}$.

An alternative view of the kernel $k_{\text{compositional}}$ is that it is an additive combination of a globally low-rank approximation and local Schur complements within each sub-domain. Hence, the kernel is (strictly) positive-definite. See Lemma 2 and Theorem 3 in the following.

Lemma 2 *The Schur-complement function*

$$k_{Schur}(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') - k(\boldsymbol{x}, \underline{X}) K(\underline{X}, \underline{X})^{-1} k(\underline{X}, \boldsymbol{x}')$$

is positive-definite, if k is strictly positive-definite, or if k is positive-definite and $K(\underline{X}, \underline{X})$ is invertible.

Proof For any set X, let $Y = X \cup \underline{X}$. It amounts to showing that the corresponding kernel matrix $K_{\text{Schur}}(Y, Y)$ is positive semi-definite; then, $K_{\text{Schur}}(X, X)$ as a principal submatrix is also positive semi-definite.

Denote by $\underline{X}^c = Y \backslash \underline{X}$, which could possibly be empty, and let the points in $\underline{X}$ be ordered before those in $\underline{X}^c$. Then,

$$K_{\text{Schur}}(Y,Y) = \begin{bmatrix} 0 & 0 \\ 0 & K(\underline{X}^c,\underline{X}^c) - K(\underline{X}^c,\underline{X})K(\underline{X},\underline{X})^{-1}K(\underline{X},\underline{X}^c) \end{bmatrix}.$$

By the law of inertia, the matrices

$$\begin{bmatrix} K(\underline{X},\underline{X}) & K(\underline{X},\underline{X}^c) \\ K(\underline{X}^c,\underline{X}) & K(\underline{X}^c,\underline{X}^c) \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} K(\underline{X},\underline{X}) & 0 \\ 0 & K(\underline{X}^c,\underline{X}^c) - K(\underline{X}^c,\underline{X})K(\underline{X},\underline{X})^{-1}K(\underline{X},\underline{X}^c) \end{bmatrix}$$

have the same number of positive, zero, and negative eigenvalues, respectively. If k is strictly positive-definite, then the eigenvalues of both matrices are all positive. If k is positive-definite, then the eigenvalues of both matrices are all nonnegative. In both cases, the Schur-complement matrix $K(\underline{X}^c,\underline{X}^c) - K(\underline{X}^c,\underline{X})K(\underline{X},\underline{X})^{-1}K(\underline{X},\underline{X}^c)$ is positive semi-definite and thus so is $K_{\text{Schur}}(Y,Y)$. ■

Theorem 3 *The function $k_{compositional}$ is positive-definite if k is positive-definite and $K(\underline{X},\underline{X})$ is invertible. Moreover, $k_{compositional}$ is strictly positive-definite if k is so.*

Proof Write $k_{\text{compositional}} = k_1 + k_2$, where $k_1(\boldsymbol{x},\boldsymbol{x}') = k(\boldsymbol{x},\underline{X})K(\underline{X},\underline{X})^{-1}k(\underline{X},\boldsymbol{x}')$ and

$$k_2(\boldsymbol{x},\boldsymbol{x}') = \begin{cases} k(\boldsymbol{x},\boldsymbol{x}') - k(\boldsymbol{x},\underline{X})K(\underline{X},\underline{X})^{-1}k(\underline{X},\boldsymbol{x}'), & \boldsymbol{x},\boldsymbol{x}' \in S_j \text{ for some } j, \\ 0, & \text{otherwise.} \end{cases}$$

Clearly, k_1 is positive-definite; by Lemma 2, k_2 is so, too. Thus, $k_{\text{compositional}}$ is positive-definite.

We next show the strict definiteness when k is strictly positive-definite. That is, for any set of points $\{\boldsymbol{x}_i\}$ and any set of coefficients $\{\alpha_i\}$ that are not all zero, the bilinear form $\sum_{il} \alpha_i\alpha_l k_{\text{compositional}}(\boldsymbol{x}_i,\boldsymbol{x}_l)$ cannot be zero. Note that $k_2(\boldsymbol{x},\boldsymbol{x}') = 0$ whenever $\boldsymbol{x}$ or $\boldsymbol{x}' \in \underline{X}$. Moreover, we have seen in the proof of Lemma 2 that the Schur-complement matrix $K(\underline{X}^c,\underline{X}^c) - K(\underline{X}^c,\underline{X})K(\underline{X},\underline{X})^{-1}K(\underline{X},\underline{X}^c)$ is positive-definite when k is strictly positive-definite. Therefore, we have $\sum_{il} \alpha_i\alpha_l k_2(\boldsymbol{x}_i,\boldsymbol{x}_l) = 0$ only when $\alpha_i = 0$ for all i satisfying $\boldsymbol{x}_i \notin \underline{X}$. In such a case,

$$\sum_{il} \alpha_i\alpha_l k_1(\boldsymbol{x}_i,\boldsymbol{x}_k) = \sum_{\boldsymbol{x}_i,\boldsymbol{x}_l \in \underline{X}} \alpha_i\alpha_l k(\boldsymbol{x}_i,\underline{X})K(\underline{X},\underline{X})^{-1}k(\underline{X},\boldsymbol{x}_l).$$

Because of the strict positive-definiteness of k, the above summation cannot be zero if any of the involved α_i (that is, those satisfying $\boldsymbol{x}_i \in \underline{X}$) is nonzero. Then, $\sum_{il} \alpha_i\alpha_l[k_1(\boldsymbol{x}_i,\boldsymbol{x}_l) + k_2(\boldsymbol{x}_i,\boldsymbol{x}_l)] = 0$ only when all α_i are zero. ■

Since the composition replaces the Nyström approximation in local domains by the full covariance, it bares no surprise that $k_{\text{compositional}}$ improves over $k_{\text{Nyström}}$ in terms of matrix approximation.

Theorem 4 *Given a set $\underline{X}$ of landmark points and for any set $X \neq \underline{X}$,*

$$\|K(X,X) - K_{compositional}(X,X)\| < \|K(X,X) - K_{Nyström}(X,X)\|,$$

where $\|\cdot\|$ is the 2-norm or the Frobenius norm.

Proof Based on the split $k_{\text{compositional}} = k_1 + k_2$ in the proof of Theorem 3, one easily sees that

$$K - K_{\text{compositional}} = (K - K_{\text{Nyström}}) - \text{block-diag}(K - K_{\text{Nyström}}),$$

where block-diag means keeping only the diagonal blocks of a matrix. Denote by $A = K - K_{\text{Nyström}}$ and $D = \text{block-diag}(A)$. In what follows we show that

$$\|A - D\| < \|A\|. \tag{9}$$

Because $A = K(X,X) - K(X,\underline{X})K(\underline{X},\underline{X})^{-1}K(\underline{X},X)$ is positive semi-definite and nonzero, its diagonal cannot be zero. Then, eliminating the block-diagonal part D reduces the Frobenius norm. Thus, (9) holds for the Frobenius norm. To see that (9) also holds for the 2-norm, let $Y = X \backslash \underline{X}$ and let the points in Y be ordered before those in $X \backslash Y$. Then,

$$A = \begin{bmatrix} K(Y,Y) - K(Y,\underline{X})K(\underline{X},\underline{X})^{-1}K(\underline{X},Y) & 0 \\ 0 & 0 \end{bmatrix}.$$

Because the zero rows and columns do not contribute to the 2-norm, and because the top-left block of A is positive-definite, it suffices to prove (9) for any positive-definite matrix A.

Note the following two straightforward inequalities

$$\lambda_{\min}(A - D) \geq \lambda_{\min}(A) - \lambda_{\max}(D), \tag{10}$$

$$\lambda_{\max}(A - D) \leq \lambda_{\max}(A) - \lambda_{\min}(D). \tag{11}$$

Because D consists of the diagonal blocks of A, the interlacing theorem of eigenvalues states that for each diagonal block D_i, we have

$$\lambda_{\min}(A) \leq \lambda_{\min}(D_i) \leq \lambda_{\max}(D_i) \leq \lambda_{\max}(A).$$

Then, taking the max/min eigenvalues of all blocks, we obtain

$$\lambda_{\min}(A) \leq \lambda_{\min}(D) \leq \lambda_{\max}(D) \leq \lambda_{\max}(A). \tag{12}$$

Substituting (12) into (10) and (11), together with $\lambda_{\min}(A) > 0$, we obtain

$$\lambda_{\min}(A - D) > -\lambda_{\max}(A) \quad \text{and} \quad \lambda_{\max}(A - D) < \lambda_{\max}(A),$$

which immediately implies that $\|A - D\|_2 < \|A\|_2$. ■

2.2 Hierarchical Composition

While $k_{\text{compositional}}$ maintains the full information inside each domain S_j, the information loss across domains caused by the low-rank approximation may still be dramatic. Consider the scenario of a large number of disjoint domains S_j; such a scenario is necessarily typical for the purpose of reducing the computational cost. If each domain is adjacent to only a few neighboring domains, it is possible to reduce the information loss in nearby domains.

The idea is to form a hierarchy. Let us first take a two-level hierarchy for example. A few of the neighboring domains S_j form a super-domain $S_J = \bigcup_{j \in J} S_j$. These super-domains are formed such that they are disjoint and they collectively partition the whole domain; i.e., $\bigcup S_J = S$. Under such a hierarchical formation, instead of using landmark points $\underline{X}$ in S to define the covariance across the bottom-level domains S_j, we may use landmark points $\underline{X}_J$ chosen from the super-domain S_J to define the covariance. The intuition is that the conditional covariance $k(\boldsymbol{x}, \boldsymbol{x}'|\underline{X}_J)$ for $\boldsymbol{x}, \boldsymbol{x}' \in S_J$ tends to be smaller than $k(\boldsymbol{x}, \boldsymbol{x}'|\underline{X})$, because $\boldsymbol{x}$ and $\boldsymbol{x}'$ are geographically closer to $\underline{X}_J$ than to $\underline{X}$. Then, the information loss is reduced for points inside the same super-domain S_J.

To formalize this idea, we consider an arbitrary hierarchy, which is represented by a rooted tree T. See Figure 1 for an example. The root node 1 is associated with the whole domain $S =: S_1$. Each nonleaf node i possesses a set of children $Ch(i)$; correspondingly, the associated domain S_i is partitioned into disjoint sub-domains S_j satisfying $\bigcup_{j \in Ch(i)} = S_i$. The partitioning tree T is almost the most general rooted tree, except that no nodes in T have exactly one child.

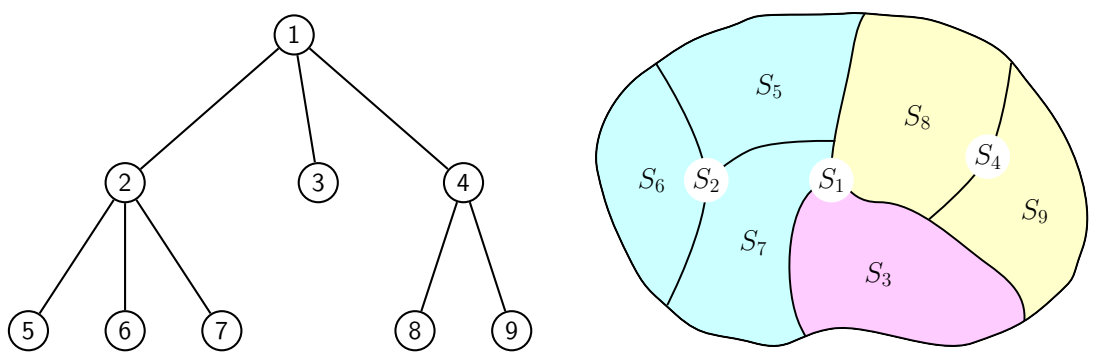


Figure 1: Partitioning tree T and the domain S.

Each nonleaf node i is associated with a set $\underline{X}_i$ of landmark points, located within the domain S_i. We now recursively define the kernel on domains across levels. Continuing the example of Figure 1, node 4 has two children 8 and 9. Since these two children are leaf nodes, the covariance within S_8 (or S_9) comes from the original kernel k, whereas the covariance across S_8 and S_9 comes from the Nyström approximation by using landmark points $\underline{X}_4$. That is, the kernel is equal to $k(\boldsymbol{x}, \boldsymbol{x}')$ if $\boldsymbol{x}$ and $\boldsymbol{x}'$ are both in S_8 (or in S_9), and equal to $k(\boldsymbol{x}, \underline{X}_4) K(\underline{X}_4, \underline{X}_4)^{-1} k(\underline{X}_4, \boldsymbol{x}')$ if they are in S_8 and S_9 separately. Such a covariance bares less information loss caused by the conditioned set, compared with the use of landmark points located within the whole domain S.

Next, consider the covariance between child domains of S_2 and those of S_4 (say, S_6 and S_9, respectively). At a first glance, we could have used $k(\boldsymbol{x}, \underline{X}_1)K(\underline{X}_1, \underline{X}_1)^{-1}k(\underline{X}_1, \boldsymbol{x}')$ to define the kernel, because $\underline{X}_1$ consists of landmark points located in the domain that covers both S_6 and S_9. However, such a definition cannot guarantee the positive-definiteness of the overall kernel. Instead, we approximate $k(\boldsymbol{x}, \underline{X}_1)$ by using the Nyström approximation $k(\boldsymbol{x}, \underline{X}_2)K(\underline{X}_2, \underline{X}_2)^{-1}K(\underline{X}_2, \underline{X}_1)$ based on the landmark points $\underline{X}_2$. Then, the covariance for $\boldsymbol{x} \in S_6$ and $\boldsymbol{x}' \in S_9$ is defined as

$$\Big[k(\boldsymbol{x}, \underline{X}_2)K(\underline{X}_2, \underline{X}_2)^{-1}K(\underline{X}_2, \underline{X}_1)\Big]K(\underline{X}_1, \underline{X}_1)^{-1}\Big[K(\underline{X}_2, \underline{X}_1)K(\underline{X}_2, \underline{X}_2)^{-1}k(\underline{X}_2, \boldsymbol{x}')\Big].$$

Formally, for a leaf node j and $\boldsymbol{x}, \boldsymbol{x}' \in S_j$, define $k^{(j)}(\boldsymbol{x}, \boldsymbol{x}') \equiv k(\boldsymbol{x}, \boldsymbol{x}')$. For a nonleaf node i and $\boldsymbol{x}, \boldsymbol{x}' \in S_i$, define

$$k^{(i)}(\boldsymbol{x}, \boldsymbol{x}') := \begin{cases} k^{(j)}(\boldsymbol{x}, \boldsymbol{x}'), & \text{if } \boldsymbol{x}, \boldsymbol{x}' \in S_j \text{ for some } j \in Ch(i), \\ \psi^{(i)}(\boldsymbol{x}, \underline{X}_i)K(\underline{X}_i, \underline{X}_i)^{-1}\psi^{(i)}(\underline{X}_i, \boldsymbol{x}'), & \text{otherwise,} \end{cases} \tag{13}$$

where if j is a child of i and if $\boldsymbol{x} \in S_j$, then

$$\psi^{(i)}(\boldsymbol{x}, \underline{X}_i) := \begin{cases} k(\boldsymbol{x}, \underline{X}_i), & \text{if } j \text{ is a leaf node,} \\ \psi^{(j)}(\boldsymbol{x}, \underline{X}_j)K(\underline{X}_j, \underline{X}_j)^{-1}K(\underline{X}_j, \underline{X}_i), & \text{otherwise.} \end{cases} \tag{14}$$

The $k^{(i)}$ at the root level gives the *hierarchically compositional kernel* of this paper:

$$k_{\text{hierarchical}} := k^{(\text{root})}. \tag{15}$$

Clearly, the kernel $k_{\text{compositional}}$ in Section 2.1 is a special case of $k_{\text{hierarchical}}$ when the partitioning tree consists of only the root and the leaf nodes (which are children of the root).

Expanding the recursive formulas (13) and (14), for two distinct leaf nodes j and l, we see that the covariance between $\boldsymbol{x} \in S_j$ and $\boldsymbol{x}' \in S_l$ is

$$\begin{aligned} k_{\text{hierarchical}}(\boldsymbol{x}, \boldsymbol{x}') &= k^{(r)}(\boldsymbol{x}, \boldsymbol{x}') \\ &= \underbrace{k(\boldsymbol{x}, \underline{X}_{j_1})K(\underline{X}_{j_1}, \underline{X}_{j_1})^{-1}K(\underline{X}_{j_1}, \underline{X}_{j_2})\cdots K(\underline{X}_{j_s}, \underline{X}_{j_s})^{-1}K(\underline{X}_{j_s}, \underline{X}_r)}_{\psi^{(r)}(\boldsymbol{x}, \underline{X}_r)} K(\underline{X}_r, \underline{X}_r)^{-1} \\ &\qquad \cdot \underbrace{K(\underline{X}_r, \underline{X}_{l_t})K(\underline{X}_{l_t}, \underline{X}_{l_t})^{-1}\cdots K(\underline{X}_{l_2}, \underline{X}_{l_1})K(\underline{X}_{l_1}, \underline{X}_{l_1})^{-1}k(\underline{X}_{l_1}, \boldsymbol{x}')}_{\psi^{(r)}(\underline{X}_r, \boldsymbol{x}')}, \end{aligned} \tag{16}$$

where r is the least common ancestor of j and l, and $(j, j_1, j_2, \ldots, j_s, r)$ and $(l, l_1, l_2, \ldots, l_t, r)$ are the paths connecting r and the two leaf nodes, respectively. Therefore, we have the following result.

Proposition 5 *Based on the notation in the preceding paragraph, for $\boldsymbol{x} \in S_j$ and $\boldsymbol{x}' \in S_l$, we have*

$$k_{hierarchical}(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}'),$$

whenever $\boldsymbol{x} \in \underline{X}_j \cap \underline{X}_{j_1} \cdots \cap \underline{X}_{j_s}$, $\boldsymbol{x}' \in \underline{X}_l \cap \underline{X}_{l_1} \cdots \cap \underline{X}_{l_t}$, and either of $\boldsymbol{x}, \boldsymbol{x}'$ belongs to $\underline{X}_r$.

Proof On (16), recursively apply the fact that if $\boldsymbol{x} \in \underline{X}$, then $K(\underline{X}, \underline{X})^{-1} k(\underline{X}, \boldsymbol{x})$ is a column of the identity matrix where the only nonzero element (i.e., 1) is located with respect to the location of $\boldsymbol{x}$ inside $\underline{X}$. ■

The following theorem guarantees the validity of the kernel. Its proof strategy is similar to that of Theorem 3, but it is complex because of recursion. We defer the proof to Appendix A.

Theorem 6 *The function $k_{hierarchical}$ is positive-definite if k is positive-definite and $K(\underline{X}_i, \underline{X}_i)$ is invertible for all sets of landmark points $\underline{X}_i$ associated with the nonleaf nodes i. Moreover, $k_{hierarchical}$ is strictly positive-definite if k is so.*

3. Matrix View

The kernel matrix $K_{\text{hierarchical}}(X, X)$ for a set of training points X exhibits a hierarchical block structure. Figure 2 pictorially shows such a structure for the example in Figure 1. To avoid degenerate empty blocks, we assume that $X \cap S_j \neq \emptyset$ for all leaf nodes j in the partitioning tree T.

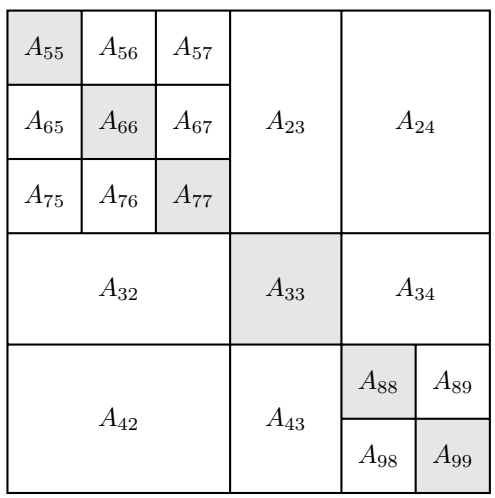


Figure 2: The matrix $K_{\text{hierarchical}}$ corresponding to the partitioning tree T in Figure 1.

Formally, for any node i, let $X_i = X \cap S_i$; that is, X_i consists of the training points that fall within the domain S_i. Then, define a matrix $A \in \mathbb{R}^{n \times n}$ with the following structure:

1. For every node i, A_{ii} is a diagonal block whose rows and columns correspond to X_i; for every pair of sibling nodes i and j, A_{ij} is an off-diagonal block whose rows correspond to X_i and columns to X_j.
2. For every leaf node i, $A_{ii} = K(X_i, X_i)$.
3. For every pair of sibling nodes i and j, $A_{ij} = U_i \Sigma_p U_j^T$, where p is the parent of i and j, with Σ_p and U_i defined next.

4. For every nonleaf node p, $\Sigma_p = K(\underline{X}_p, \underline{X}_p)$.

5. For every leaf node i, $U_i = K(X_i, \underline{X}_p)K(\underline{X}_p, \underline{X}_p)^{-1}$, where p is the parent of i.

6. For every pair of child node i and parent node p not being the root, the block of U_p corresponding to node i is $U_i W_p$, where $W_p = K(\underline{X}_p, \underline{X}_r)K(\underline{X}_r, \underline{X}_r)^{-1}$ and r is the parent of p. That is, in the matrix form,

$$U_p = \begin{bmatrix} \vdots \\ U_i \\ \vdots \end{bmatrix}_{i \in Ch(p)} \cdot W_p.$$

One sees that A is exactly equal to $K_{\text{hierarchical}}(X, X)$ by verifying against the definition of $k_{\text{hierarchical}}$ in (13)–(15).

Such a hierarchical block structure is a special case of the *recursively low-rank compressed matrix* studied in Chen (2014b). In this matrix, off-diagonal blocks are recursively compressed into low rank through change of basis, while the main-diagonal blocks at the leaf level remain intact. Its connections and distinctions with related matrices (e.g., FMM matrices (Sun and Pitsianis, 2001) and hierarchical matrices (Hackbusch, 1999)) were discussed in detail in Chen (2014b). The signature of a recursively low-rank compressed matrix is that many matrix operations can be performed with a cost, loosely speaking, linear in n. The matrix structure in this paper, which results from the kernel $k_{\text{hierarchical}}$, specializes a general recursively low-rank compressed matrix in the following aspects: (a) the matrix is symmetric; (b) the middle factor Σ_p in each A_{ij} is the same for all child pairs i, j of p; and (c) the change-of-basis factor W_p is the same for all children i of p. Hence, storage and time costs of matrix operations are reduced by a constant factor compared with those of the algorithms in Chen (2014b). In what follows, we discuss the algorithmic aspects of the matrix operations needed to carry out kernel method computations, but defer the complexity analysis in Section 4. Discussions of additional matrix operations useful in a general machine learning context are made toward the end of the paper.

3.1 Matrix-Vector Multiplication

First, we consider computing the matrix-vector product $y = Ab$. Let the blocks of a vector be labeled in the same manner as those of the matrix A. That is, for any node i, b_i denotes the part of b that corresponds to the point set X_i. Then, the vector y is clearly an accumulation of the smaller matrix-vector products $A_{ij}b_j$ in the appropriate blocks, for all sibling pairs i, j and for all leaf nodes $i = j$ (cf. Figure 2). When $i = j$ is a leaf node, the computation of $A_{ij}b_j$ is straightforward. On the other hand, when i and j constitute a pair of sibling nodes, we consider a descendant leaf node l of i. The block of $A_{ij}b_j$ corresponding to node l admits an expanded expression

$$U_l W_{l_1} W_{l_2} \cdots W_{l_s} W_i \Sigma_p W_j^T \left(\sum_{q_t \in Ch(j)} W_{q_t}^T \cdots \left(\sum_{q_2 \in Ch(q_3)} W_{q_2}^T \left(\sum_{q_1 \in Ch(q_2)} W_{q_1}^T \left(\sum_{q \in Ch(q_1)} U_q^T b_q \right) \right) \right) \right), \tag{17}$$

where $(l, l_1, l_2, \ldots, l_s, i, p)$ is the path connecting l and the parent p of i. This expression assumes that all the leaf descendants of j are on the same level so that the expression is not overly complex; but the subsequent reasoning applies to the general case. The terms inside the nested parentheses of (17) motivate the following recursive definition:

$$c_i = \begin{cases} U_i^T b_i, & i \text{ is a leaf node,} \\ W_i^T \sum_{j \in Ch(i)} c_j, & \text{otherwise.} \end{cases}$$

Clearly, all c_j's can be computed within one pass of a post-order tree traversal. Upon the completion of the traversal, we rewrite (17) as $U_l W_{l_1} W_{l_2} \cdots W_{l_s} W_i \Sigma_p c_j$. Then we note that for any leaf node l, y_l is a sum of these expressions with all p along the path connecting l and the root, and additionally, of $A_{ll} b_l$. In other words, we have

$$y_l = A_{ll} b_l + \left(\sum_{l' \in Ch(l_1) \setminus \{l\}} U_l \Sigma_{l_1} c_{l'} \right) + \left(\sum_{l' \in Ch(l_2) \setminus \{l_1\}} U_l W_{l_1} \Sigma_{l_2} c_{l'} \right) + \cdots + \left(\sum_{l' \in Ch(\text{root}) \setminus \{l_t\}} U_l W_{l_1} W_{l_2} \cdots W_{l_t} \Sigma_{\text{root}} c_{l'} \right),$$

where $(l, l_1, l_2, \ldots, l_t, \text{root})$ is the path connecting l and the root. Therefore, we recursively define another quantity

$$d_j = W_i d_i + \sum_{j' \in Ch(i) \setminus \{j\}} \Sigma_i c_{j'}$$

for all nonroot nodes j with parent i. Clearly, all d_j's can also be computed within one pass of a pre-order tree traversal. Upon the completion of this second traversal, we have $y_l = A_{ll} b_l + U_l d_l$, which concludes the computation of the whole vector y.

As such, computing the matrix-vector product $y = Ab$ consists of one post-order tree traversal, followed by a pre-order one. We refer the reader to Chen (2014b) for a full account of the computational steps. For completeness, we summarize the pesudocode in Algorithm 1 as a reference for computer implementation.

3.2 Matrix Inversion

Next, we consider computing A^{-1}, for which we use the tilded notation $\tilde{A} := A^{-1}$. One can show (Chen, 2014b) that $\tilde{A}$ has exactly the same hierarchical block structure as does A. In other words, the structure of $\tilde{A}$ can be described by reusing the verbatim at the beginning of Section 3, with the factors A_{ii}, A_{ij}, U_i, Σ_p, W_p replaced by the tilded version $\tilde{A}_{ii}$, $\tilde{A}_{ij}$, $\tilde{U}_i$, $\tilde{\Sigma}_p$, $\tilde{W}_p$, respectively. The proof is both inductive and constructive, providing explicit formulas to compute $\tilde{A}_{ii}$, $\tilde{A}_{ij}$, $\tilde{U}_i$, $\tilde{\Sigma}_p$, $\tilde{W}_p$ level by level. The essential idea is that if $(A_{ii} - U_i \Sigma_p U_i^T)^{-1} = \tilde{A}_{ii} - \tilde{U}_i \tilde{\Sigma}_p \tilde{U}_i^T$ have been computed for all children i of a node p, and if r is the parent of p, then

$$A_{pp} - U_p \Sigma_r U_p^T = \begin{bmatrix} \ddots & & \\ & A_{ii} - U_i \Sigma_p U_i^T & \\ & & \ddots \end{bmatrix} + \begin{bmatrix} \vdots \\ U_i \\ \vdots \end{bmatrix} \Sigma_p \begin{bmatrix} \cdots & U_i^T & \cdots \end{bmatrix} - U_p \Sigma_r U_p^T.$$

Algorithm 1 Computing $y = Ab$

```
Initialize c_i ← 0, d_i ← 0 for each nonroot node i of the tree
UPWARD(root)
DOWNWARD(root)

function UPWARD(i)
    if i is leaf then
        c_i ← U_i^T b_i;   y_i ← A_ii b_i
    else
        for all children j of i do
            UPWARD(j)
            c_i ← c_i + W_i^T c_j, if i is not root
        end for
    end if
    if i is not root then
        for all siblings l of i do d_l ← d_l + Σ_p c_i end for        ▷ p is parent of i
    end if
end function

function DOWNWARD(i)
    if i is leaf then y_i ← y_i + U_i d_i and return end if
    for all children j of i do
        d_j ← d_j + W_i d_i, if i is not root
        DOWNWARD(j)
    end for
end function
```

Hence, the inversion of $A_{pp} - U_p \Sigma_r U_p^T$ can be done by applying the Sherman–Morrison–Woodbury formula, which results in a form $\tilde{A}_{pp} - \tilde{U}_p \tilde{\Sigma}_r \tilde{U}_p^T$, where $\tilde{W}_r$ is determined based on the change of basis from $\tilde{U}_i$ to $\tilde{U}_p$ and the middle factor $\tilde{\Sigma}_r$ is also determined. In addition, the factors $\tilde{\Sigma}_j$ (computed previously) for all descendants j of r must be corrected because the diagonal block of $\tilde{A}_{pp}$ corresponding to node j is a low-rank correction of the corresponding diagonal block of $\tilde{A}_{ii}$ with the same basis. Such a down-cascading correction occurs whenever the induction proceeds from one child level to the parent level; but computationally, the correction on any node can be accumulated during the whole induction process. The net result of the accumulation is that each $\tilde{\Sigma}_i$ needs be corrected only once, which ensures an efficient computation.

We refer the reader to Chen (2014b) for a full account of the computational steps. Similar to the matrix-vector multiplication, the overall computation here consists of one post-order tree traversal (corresponding to the induction), followed by a pre-order one (corresponding to the down-cascading correction). For completeness, we summarize the pseudocode in Algorithm 2 as a reference for computer implementation.

Algorithm 2 Computing $\tilde{A} = A^{-1}$

```
UPWARD(root)
DOWNWARD(root)

function UPWARD(i)
    if i is leaf then
        Ã_ii ← (A_ii − U_i Σ_p U_i^T)^{−1};   Ũ_i ← Ã_ii U_i;   Θ̃_i ← U_i^T Ũ_i        ▷ p is parent of i
        return
    end if
    for all children j of i do
        UPWARD(j)
        W̃_j ← (I + Σ̃_j Ξ̃_j) W_j if j is not leaf
        Θ̃_j ← W_j^T Ξ̃_j W̃_j if j is not leaf
    end for
    Ξ̃_i ← Σ_{j∈Ch(i)} Θ̃_j
    if i is not root then Λ̃_i ← Σ_i − W_i Σ_p W_i^T else Λ̃_i ← Σ_i end if    ▷ p is parent of i
    Σ̃_i ← −(I + Λ̃_i Ξ̃_i)^{−1} Λ̃_i
    for all children j of i do
        Ẽ_j ← W̃_j Σ̃_i W̃_j^T if j is not leaf
    end for
    Ẽ_i ← 0 if i is root
end function

function DOWNWARD(i)
    if i is leaf then
        Ã_ii ← Ã_ii + Ũ_i Σ̃_p Ũ_i^T if i is not root                    ▷ p is parent of i
    else
        Ẽ_i ← Ẽ_i + W̃_i Ẽ_p W̃_i^T if i is not root                      ▷ p is parent of i
        Σ̃_i ← Σ̃_i + Ẽ_i
        for all children j of i do DOWNWARD(j) end for
    end if
end function
```

3.3 (Implicit) Out-of-Sample Construction

Now, we consider the vector $k_{\text{hierarchical}}(X, \boldsymbol{x})$ for an existing training set X and a new point $\boldsymbol{x}$ in the testing set. Generally, this vector is not used alone but it appears in the inner product with some other vector w (see (2)). Whereas the construction of $k_{\text{hierarchical}}(X, \boldsymbol{x})$ can be done at $O(n)$ cost (details omitted here), we shall consider instead the computation of the inner product $w^T k_{\text{hierarchical}}(X, \boldsymbol{x})$, because the cost of computing this inner product is proportional to only the height of the tree per $\boldsymbol{x}$ after an $O(n)$ preprocessing. The

preprocessing cost is amortized on each $\boldsymbol{x}$ and thus is generally negligible for a large number of $\boldsymbol{x}$'s.

The computation of $w^T k_{\text{hierarchical}}(X, \boldsymbol{x})$ was not described in Chen (2014b), but the idea is similar to that of the matrix-vector multiplication in Section 3.1. To simplify notation, let $v \equiv k_{\text{hierarchical}}(X, \boldsymbol{x})$. Assume that $\boldsymbol{x}$ falls in the domain S_j for some leaf node j. Then, $v_j = k(X_j, \boldsymbol{x})$ and for any leaf node $l \neq j$,

$$v_l = U_l W_{l_1} W_{l_2} \cdots W_{l_s} \Sigma_p W_{j_t}^T \cdots W_{j_2}^T W_{j_1}^T K(\underline{X}_{j_1}, \underline{X}_{j_1})^{-1} k(\underline{X}_{j_1}, \boldsymbol{x}),$$

where p is the least common ancestor of l and j, and $(l, l_1, l_2, \ldots, l_s, p)$ and $(j, j_1, j_2, \ldots, j_t, p)$ are the paths connecting l and p, and j and p, respectively. Therefore, if we define

$$d_{j_u} = W_{j_u}^T d_{j_{u-1}}, \quad \cdots \quad d_{j_2} = W_{j_2}^T d_{j_1}, \quad d_{j_1} = W_{j_1}^T d_j, \quad d_j = K(\underline{X}_{j_1}, \underline{X}_{j_1})^{-1} k(\underline{X}_{j_1}, \boldsymbol{x}), \tag{18}$$

where $(j, j_1, j_2, \ldots, j_u, \text{root})$ is the path connecting j and the root, then we have

$$w^T v = w_j^T k(X_j, \boldsymbol{x}) + \sum_{l \neq j,\ l \text{ is leaf}} w_l^T U_l W_{l_1} W_{l_2} \cdots W_{l_s} \Sigma_p d_{j_t}. \tag{19}$$

If we further define

$$e_l = \begin{cases} U_l^T w_l, & l \text{ is leaf}, \\ \displaystyle\sum_{i \in Ch(l)} W_l^T e_i, & \text{otherwise}, \end{cases} \qquad \text{and} \qquad c_q = \Sigma_p^T e_l, \tag{20}$$

where p is the parent of the sibling pair q and l, then (19) is simplified to

$$w^T v = w_j^T k(X_j, \boldsymbol{x}) + \sum_{j_t \in \text{path}(j, \text{root})} c_{j_t}^T d_{j_t}. \tag{21}$$

Based on (18)–(21), we see that $w^T v \equiv w^T k_{\text{hierarchical}}(X, \boldsymbol{x})$ can be computed in the following two phases. In the first phase, we compute e_l and c_l for all nonleaf nodes l according to (20). Such a computation can be done by using a post-order tree traversal and is independent of $\boldsymbol{x}$. Thus, this computation is the preprocessing step. In the second phase, we compute d_{j_t} for all j_t along the path $(j, j_1, j_2, \ldots, j_u, \text{root})$ according to (18). Once they are computed, the summation (21) is straightforward and we thus conclude the computation. Note that the second phase is always conducted along a certain path connecting the root and the leaf j. As long as the determination of which leaf j the point $\boldsymbol{x}$ falls in is restricted on this path, the cost of the second phase is always asymptotically smaller than that of a tree traversal. We summarize the pseudocode in Algorithm 3.

4. Practical Considerations

Sections 2 and 3 provide a general framework for the interpretation of and the computation with the proposed kernel $k_{\text{hierarchical}}$; they, however, have not covered a range of practical issues. Some issues are dimension dependent (this paper concerns particularly dimensions higher than three, which dominates the use of related matrices, e.g., FMM and hierarchical matrices, in scientific computing); whereas others are numerically related. In this section, we discuss the handling of several of these issues and ensure that the proposed kernel is computationally efficient.

Algorithm 3 Computing $z = w^T k_{\text{hierarchical}}(X, \boldsymbol{x})$ for $\boldsymbol{x} \notin X$

Prefactorize $K(\underline{X}_p, \underline{X}_p)$ for all parents p of leaf nodes
Initialize $c_i \leftarrow 0$, $d_i \leftarrow 0$ for each nonroot node i of the tree
Common-Upward(`root`)
▷ The above three steps are independent of $\boldsymbol{x}$ and are treated as precomputation. In computer implementation, the intermediate results c_i are carried over to the next step Second-Upward, whereas the contents of d_i are discarded and the allocated memory is reused.
Second-Upward(`root`)

function Common-Upward(i)
  if i is leaf **then**
    $d_i \leftarrow U_i^T w_i$
  else
    for all children j of i **do**
      Common-Upward(j)
      $d_i \leftarrow d_i + W_i^T d_j$, **if** i is not root
    end for
  end if
  if i is not root **then**
    for all siblings l of i **do** $c_l \leftarrow \Sigma_p^T d_i$ **end for** ▷ p is parent of i
  end if
end function

function Second-Upward(i)
  if i is leaf **then**
    $d_i \leftarrow K(\underline{X}_p, \underline{X}_p)^{-1} k(\underline{X}_p, \boldsymbol{x})$ ▷ p is parent of i
    $z \leftarrow w_i^T k(X_i, \boldsymbol{x})$
  else
    Find the child j (among all children of i) where $\boldsymbol{x}$ lies on
    Second-Upward(j)
    $d_i \leftarrow W_i^T d_j$ if i is not root
  end if
  $z \leftarrow z + c_i^T d_i$ if i is not root
end function

4.1 Partitioning of Domain

Whereas the partitioning of a low-dimensional domain can be made regular, the partitioning of a high-dimensional domain is inherently difficult because of the curse of dimensionality. Several data-driven approaches appear to be natural choices but they have pros and cons.

The k-d tree (Bentley, 1975) approach is known to be efficient for nearest neighbor search. Generally, the approach iteratively selects an axis of the bounding box that contains the training points and partitions the axis such that the numbers of points on both sides are balanced. A drawback of this approach, particularly in the context of classification, lies in

the case that some attributes of the data are binary and the counts are highly imbalanced. These attributes sometimes result from a conversion of categorical attributes to numeric ones. Partitioning on these axes is unlikely to be balanced. An alternative is to partition the axis into two segments of equal length. This approach often results in highly imbalanced partitioning because data are generally not evenly distributed.

The k-means (MacQueen, 1967) approach partitions a point set through a k-means clustering and hence the partitioning is a Voronoi diagram of the cluster centers. The resulting tree is not necessarily binary if there exhibits more than two natural clusters in the data on some level. An advantage of this approach is that the clustering often results in a tight grouping of the points, such that the subsequent Nyström approximations bare a good quality. A disadvantage is that the approach suffers from loss of clusters during iterations and it is less robust if only one set of initial guesses is used. Our experience indicates that a robust implementation of k-means sometimes costs much more than does the rest of the training computation.

The PCA (Pearson, 1901) approach recursively partitions the data according to the principal axis (the direction along which the data varies the most). This approach is based on a Gaussian assumption and often results in compact partitions if the data indeed conforms to this assumption. However, it ignores the skewness and other higher order moments, which are also crucial to the shape of data in high dimensions. In the standard case, the hyperplane that partitions the data passes through the mean, which may result in highly imbalanced partitions. An alternative is to move the hyperplane along the principal direction such that the two partitions are always balanced. Computationwise, this approach requires computing the dominant singular vectors of the shifted data matrix (Chen et al., 2009), which can be achieved by using a power iteration (Golub and Van Loan, 1996) or the Lanczos algorithm (Saad, 2003). However, even if we compute the singular vectors only approximately, we find that the cost is still too high (see e.g., Section 5.2).

For computational efficiency, we recommend the random projection (Johnson and Lindenstrauss, 1984) approach. This approach uses a random vector as the normal direction of the partitioning hyperplane and positions the hyperplane such that the numbers of points on the two sides are balanced. Computationwise, it amounts to projecting the points along the random direction and splitting them in two halves around the median. This approach is motivated by the study of dimension reduction, where it was observed that the projection quality on a random subspace is as good as that on the space spanned by the dominant singular vectors, in the sense that Euclidean distances are approximately preserved (Dasgupta and Gupta, 2002). Moreover, the heuristic use of a one-dimensional subspace for projection is robust and is computationally highly efficient. In such a case, the purpose is not to preserve the Euclidean distance, but rather, to serve as an efficient procedure for partitioning. In our experience, the eventual regression/classification performance of using this approach is almost identical to that of the PCA approach (see Section 5.2).

Under random projection, one can quickly determine which partition a new point $\boldsymbol{x}$ lies in, by comparing its projected coordinate with the median of those of the training points.

4.2 Choice of Landmark Points

The set of landmark points $\underline{X}_i$ associated to each domain S_i simply consists of uniformly random samples of the point set X_i.

Some alternatives exist. Using the k-means centers of X_i as the landmark points may improve the Nyström approximations, but computing them for every nonleaf node i is often much more expensive than the rest of the training computation. On the other hand, using the uniformly random samples of the domain S_i as the landmark points may sound theoretically preferable, but they are difficult to compute because the domains are polyhedra rather than regular shapes (e.g., boxes).

We mentioned in Section 2.2 that the compositional kernel $k_{\text{compositional}}$ is a special case of $k_{\text{hierarchical}}$, if in the partitioning tree every child of the root is a leaf node. Interestingly, another interpretation exists. If we relax the requirement that the landmark points $\underline{X}_i$ must reside within the domain S_i, then for any partitioning tree, using the same set of landmark points $\underline{X}_i$ for every i also results in the compositional kernel $k_{\text{compositional}}$. By following the proof of Theorem 6, one sees that the relation $\underline{X}_i \subset S_i$ is in fact nonessential. Hence, the requirement $\underline{X}_i \subset S_i$ reflects only the modeling desire of a better kernel approximation in local domains; it is not a necessary condition for kernel validity.

4.3 Numerical Stability

A notorious numerical challenge for kernel matrices K is that they are exceedingly ill-conditioned as the size increases. Hence, the regularization λI serves as an important rescue of matrix inversion. This challenge is less severe for the matrix of the proposed kernel, $K_{\text{hierarchical}}(X, X)$, but since the components of this matrix are defined with the term $K(\underline{X}_i, \underline{X}_i)^{-1}$ for nonleaf nodes i, one must ensure that each $K(\underline{X}_i, \underline{X}_i)$ is not too ill conditioned. Generally, when the set $\underline{X}_i$ is not large, the conditioning of $K(\underline{X}_i, \underline{X}_i)$ is acceptable; however, for a robust safeguard, we use a small $\lambda' < \lambda$ for help. Instead of treating $k(\boldsymbol{x}, \boldsymbol{x}')$ as the base kernel and λ as the amount of regularization, we treat $k'(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') + \lambda' \delta_{\boldsymbol{x},\boldsymbol{x}'}$ as the base kernel and $\lambda - \lambda'$ the regularization, where δ is the Kronecker delta. Then, we use k' to build the kernel $k'_{\text{hierarchical}}$ and add to the matrix $K'_{\text{hierarchical}}$ a regularization $(\lambda - \lambda')I$.

4.4 Sizes

Two parameters exist for the proposed kernel. First, the recursive partitioning needs a stopping criterion; we say that a set of points is not further partitioned when the cardinality is less than or equal to n_0. Second, we need to specify the sizes of the conditioned sets $\underline{X}_i$. Whereas one may argue that the conditioned set needs to grow with the size of the point set for maintaining the approximation quality, a substantial increase in the size of $\underline{X}_i$ across levels will result in too high a computational cost, forfeiting the purpose of the kernel. Therefore, we mandate that each conditioned set have the same size r.

It will be convenient if we can consolidate the two parameters so that different approximate kernels are comparable. It will also be beneficial to have a guided choice of them. These requirements are indeed achievable. Under a balanced binary partitioning, n_0 can effectively take only these values: $\lceil n/2^j \rceil$, for $j = 0, 1, \ldots \lfloor \log_2 n \rfloor$. On the other hand, it

is not sensible to have the rank r greater than n_0 because the smallest off-diagonal blocks have a size at most $n_0 \times n_0$. Then, we take

$$n_0 = \lceil n/2^j \rceil \quad \text{and} \quad r = \lfloor n/2^j \rfloor \quad \text{for some } j. \tag{22}$$

4.5 Cost Analysis

We are now ready to perform a full analysis of the computational costs based on the practical choices made in the preceding subsections.

Memory Cost Let us first consider the memory cost of storing the matrix A. For simiplicity, assume that n is a power of 2 (and thus $n_0 = r$). Then, the partitioning tree has n/n_0 leaf nodes and $n/n_0 - 1$ nonleaf nodes. Hence, the memory costs for the factors A_{ii}, U_i, Σ_p, and W_p are $n_0^2 n/n_0$, nr, $r^2(n/n_0 - 1)$, and $r^2(n/n_0 - 1)$, respectively. Therefore, the total memory cost is approximately $4nr$.

Arithmetic Cost, Algorithm 1 The arithmetic cost of matrix-vector multiplication (Algorithm 1) is $O(nr)$, because the subroutines Upward and Downward for each node performs $O(r^2)$ calculations, discounting the recursion calls. These recursions essentially constitute a tree traversal, which visits all the $2n/n_0 - 1$ nodes. Therefore, the total cost is $O(nr)$. In fact, a more detailed analysis may reveal the constant inside the big-O notation. Note that the matrix-vector multiplications $W_i^T c_j$ and $W_i d_i$ occur for each node j, $\Sigma_p c_i$ occurs for each node i, and $U_i^T b_i$, $A_{ii} b_i$, and $U_i d_i$ occur for each leaf node i. The matrix size of all these multiplications is $r \times r$. Therefore, if each multiplication requires $2r^2$ arithmetic operations, then the total cost is approximately $2r^2 \times (2n/n_0 + 2n/n_0 + 2n/n_0 + n/n_0 + n/n_0 + n/n_0) = 18nr$, ignoring lower-order terms.

Arithmetic Cost, Algorithm 2 Based on a similar argument, the arithmetic cost of matrix inversion (Algorithm 2) is $O(nr^2)$. A more precise estimate is $37nr^2$, if factorizing a general $r \times r$ matrix, factorizing a symmetric positive-definite matrix, performing a triangular solve, and multiplying two $r \times r$ matrices require $2r^3/3$, $r^3/3$, r^2, and $2r^3$ operations, respectively. As before, we ignore the lower-order terms.

Arithmetic Cost, Algorithm 3, Precomputation Phase Now consider the precomputation phase of the inner product $w^T k_{\text{hierarchical}}(X, \boldsymbol{x})$ (Algorithm 3). In computer implementation, we move the first line (prefactorizing $K(\underline{X}_p, \underline{X}_p)$) to the matrix construction part (to be discussed in Section 4.5). Hence, the precomputation of Algorithm 3 is dominated by the third line Common-Upward(`root`). The arithmetic cost is $O(nr)$, or $10nr$ in a more precise account.

Arithmetic Cost, Algorithm 3, $\boldsymbol{x}$-Dependent Phase Assume that evaluating the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$ requires $O(\text{nz}(\boldsymbol{x}) + \text{nz}(\boldsymbol{x}'))$ arithmetic operations, where $\text{nz}(\cdot)$ denotes the number of nonzeros of a data point. Further assume that for two point sets Y and Z of sizes n_Y and n_Z respectively, evaluating the vector $k(Y, \boldsymbol{x})$ requires $O(n_Y \,\text{nz}(\boldsymbol{x}) + \text{nz}(Y))$ operations, and evaluating the matrix $K(Y, Z)$ requires $O(n_Y \,\text{nz}(Z) + n_Z \,\text{nz}(Y))$ operations, where the notation $\text{nz}(\cdot)$ is extended for point sets in a straightforward manner.

Now we consider the cost of the $\boldsymbol{x}$-dependent phase of Algorithm 3. Line 20 costs $O(r \,\text{nz}(\boldsymbol{x}) + \text{nz}(\underline{X}_p)) + O(r^2)$, where the first term comes from evaluating $k(\underline{X}_p, \boldsymbol{x})$ and the

second term a linear-solve. Note that the matrix $K(\underline{X}_p, \underline{X}_p)$ has already been evaluated and prefactorized prior to this phase. In fact, such a computation is done when constructing $K_{\text{hierarchical}}$. Similarly, line 21 costs $O(n_0 \operatorname{nz}(\boldsymbol{x}) + \operatorname{nz}(X_i)) + O(n_0)$. Line 23 costs $O(\operatorname{nz}(\boldsymbol{x}))$, because it amounts to projecting $\boldsymbol{x}$ on a direction. Then, the overall cost of the $\boldsymbol{x}$-dependent phase of Algorithm 3 is

$$O\Big(r \operatorname{nz}(\boldsymbol{x}) + \operatorname{nz}(X_i) + \operatorname{nz}(\underline{X}_p) + (r^2 + \operatorname{nz}(\boldsymbol{x})) \log_2(n/r)\Big), \tag{23}$$

where $\boldsymbol{x}$ lies in the leaf node i and p is its parent.

Arithmetic Cost, Matrix Construction We shall also consider the cost of hierarchical partitioning and the instantiation of the matrix $A = K_{\text{hierarchical}}$. In the partitioning of a set of n points X, generating a random projection direction costs $O(d)$, computing the projected coordinates costs $O(\operatorname{nz}(X))$, finding the median costs $O(n)$, and permuting the points costs $O(\operatorname{nz}(X))$. Then, counting recursion, the total cost is $O((d + n + \operatorname{nz}(X)) \log_2(n/r))$. On the other hand, in the instantiation of the matrix A, computing A_{ii} for all leaf nodes i costs $O(r \operatorname{nz}(X))$, computing U_i for all leaf nodes i costs $O(r[n + \operatorname{nz}(X) + \sum_{i \text{ is leaf}} \operatorname{nz}(\underline{X}_p)])$, computing Σ_p for all nonleaf nodes p costs $O(r[\sum_{p \text{ is nonleaf}} \operatorname{nz}(\underline{X}_p)])$, factorizing these Σ_p's costs $O(nr^2)$, computing W_i for all nonleaf and nonroot nodes i costs $O(r[n + \sum_{i \text{ is not leaf and not root}} \operatorname{nz}(\underline{X}_i) + \operatorname{nz}(\underline{X}_p)])$, and finding all landmark point sets costs $O(n)$. Therefore, the total cost of instantiating A is $O(r[nr + \operatorname{nz}(X) + \sum_{i \text{ is not leaf}} \operatorname{nz}(\underline{X}_i)])$.

Arithmetic Cost Summary, Training and Testing Summarizing the above analysis, we see that the overall cost of training (including partitioning, instantiating A, matrix inversion, matrix-vector multiplication, and preprocessing of Algorithm 3) is

$$O\left((d + n + \operatorname{nz}(X)) \log_2(n/r) + nr^2 + r\left[\operatorname{nz}(X) + \sum_{i \text{ is not leaf}} \operatorname{nz}(\underline{X}_i)\right]\right). \tag{24}$$

The overall cost of testing per $\boldsymbol{x}$ has been given in (23).

We remark that the term $\log_2(n/r)$ in both (23) and (24) is dominated by r when $n \leq r2^r$. Moreover, when the data X is dense, the training cost (24) simplifies to $O(nr^2 + ndr)$ and the testing cost per point (23) simplifies to $O(r^3 + dr)$. Suppressing the data dimension d, these costs are further simplified to $O(nr^2)$ and $O(r^3)$, respectively.

5. Experimental Results

In this section, we present comprehensive experiments to demonstrate the effectiveness of the proposed kernel. The experiments were performed on one node of a computing cluster with 512 GB memory and an IBM POWER8 12-core processor, unless otherwise stated. The programs were implemented in C++. The linear algebra routines (BLAS and LAPACK) were linked against the IBM ESSL library and some obviously parallelizable for-loops were decorated by OpenMP pragmas, so that the experiments with large data sets can be completed in a reasonable time frame. However, the programs were neither fully optimized nor fully parallelized, leaving room for improvement on both memory consumption and execution time.

Table 1: Data sets.

Name	Type	d	n Train	n' Test
cadata	regression	8	16,512	4,128
YearPredictionMSD	regression	90	463,518	51,630
ijcnn1	binary classification	22	35,000	91,701
covtype.binary	binary classification	54	464,809	116,203
SUSY	binary classification	18	4,000,000	1,000,000
mnist	10 classes	780	60,000	10,000
acoustic	3 classes	50	78,823	19,705
covtype	7 classes	54	464,809	116,203

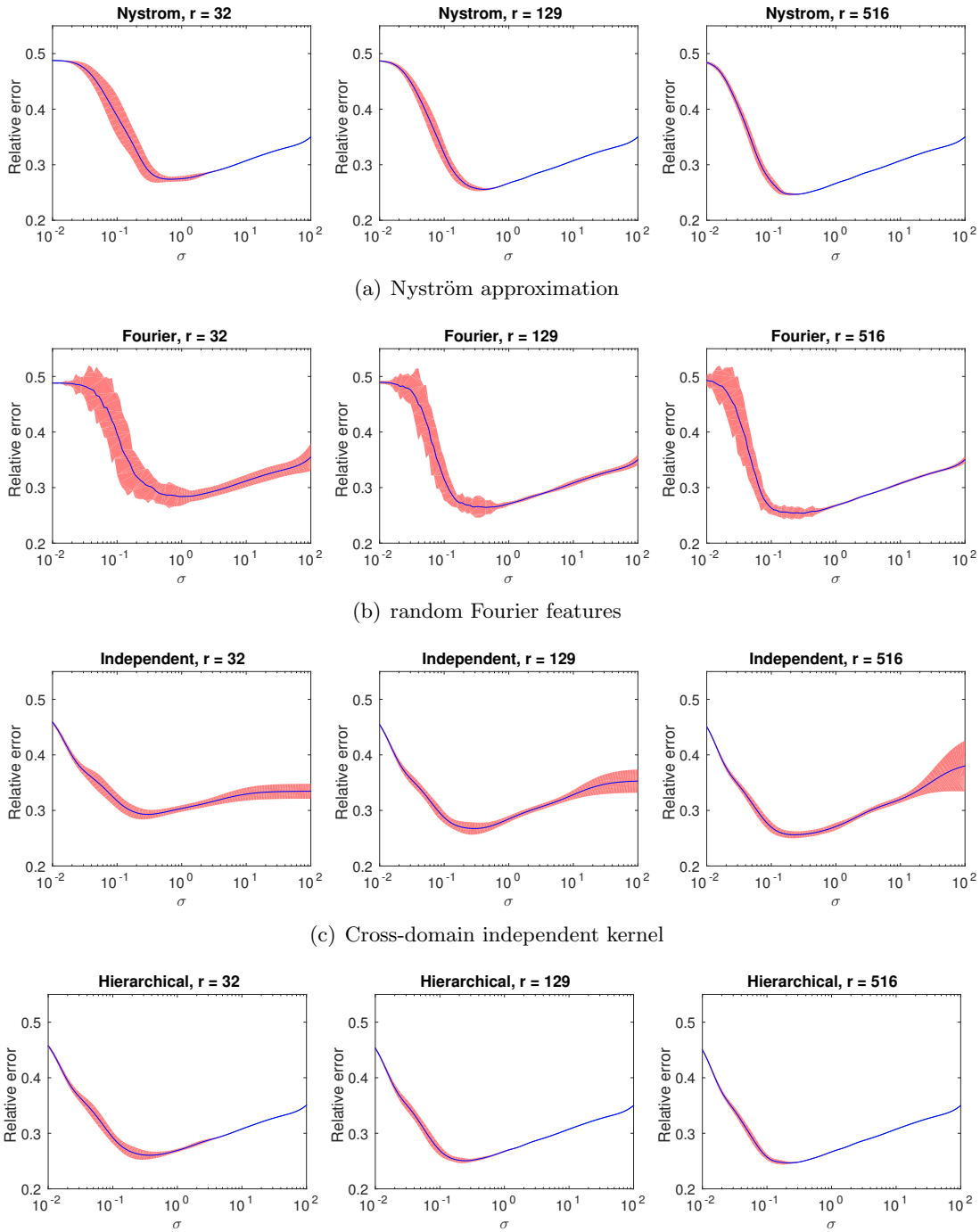
Table 1 summarizes the data sets used for experiments. They were all downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/` and are widely used as benchmarks of kernel methods. We selected these data sets primarily because of their varying sizes. Some of the data sets come with a training and a testing part; for those not, we performed a 4:1 split to create the two parts, respectively. In some of the data sets, the attributes had already been normalized to within $[0, 1]$ or $[-1, 1]$; for those not, we performed such a normalization. We also preprocessed the data sets by removing duplicate and conflicting records (whose labels are inconsistent) in the training sets. Such records are infrequent.

5.1 Effect of Randomness

Throughout Section 5, we will compare various approximate kernels: Nyström approximation $k_{\text{Nyström}}$, random Fourier features k_{Fourier}, cross-domain independent kernel $k_{\text{independent}}$, and the proposed kernel $k_{\text{hierarchical}}$. The partitioning in the cross-domain independent kernel is the same as that in the proposed kernel, except that the hierarchy is flattened. Because of the random nature of all these kernels (e.g., landmark points, sampling, and partitioning), we first study how the performance is affected by randomization. Note that the quantity r is comparable across kernels, even though its specific meaning is different.

The data set for demonstration is cadata. We use the Gaussian kernel (5) as an example. As hinted earlier, the choice of the range parameter σ affects the quality of various kernels. Therefore, the experiment setup is to use a reasonable regularization $\lambda = 0.01$ and to vary the choice of σ in a large interval (between 0.01 and 100) such that the optimal σ falls within the interval. We set the rank r (and the leaf size n_0) according to (22) with three particular choices: $r = 32$, 129, and 516.

For each r, we repeat 30 times with different random seeds; but the seed always stays the same every time when the range of σ is swept. The results (relative testing error versus σ) are summarized in Figure 3, with mean (blue curve) and standard deviation (red band) plotted. One sees that the red bands of random Fourier features are not smooth; this is because each single error curve from a fixed seed is nonsmooth. Moreover, the error curves for Nyström approximation have a nonnegligible variation when σ is small, whereas those for the independent kernel vary significantly when σ is large. The error curves of the proposed kernel are the most stable, with the narrowest standard deviation band. Generally



(a) Nyström approximation

(b) random Fourier features

(c) Cross-domain independent kernel

(d) Hierarchically compositional kernel

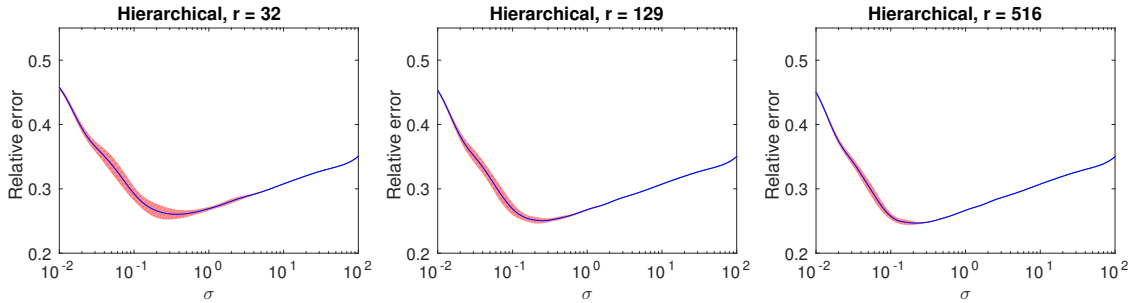
Figure 3: Regression error curves versus σ. Results are summarized from 30 repeated runs and the red bands show three times standard deviation. Data set: cadata.

speaking, as r increases, all approximate kernels yield a more and more stable error curve, except the peculiar case of the independent kernel at large σ.

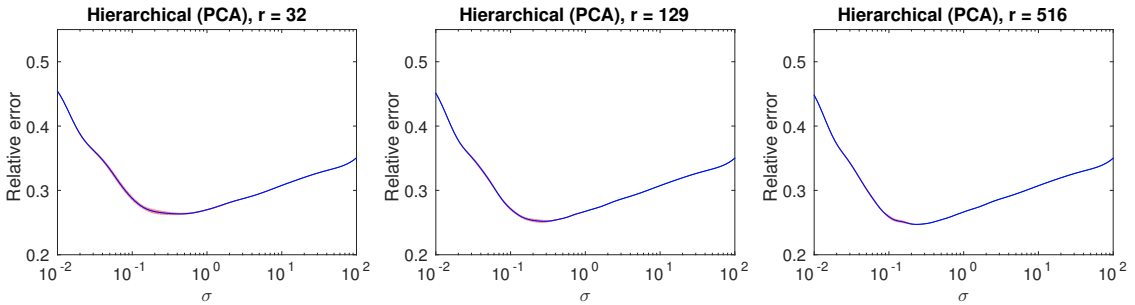
The unstable performance caused by randomness is unfavorable for parameter estimation, because the valley, where the optimal parameter stands, may move substantially. The nonsmoothness exhibited in the Fourier approach also renders difficulty. Although the unfavorable behaviors are substantially alleviated when r increases to 516 in this particular data set, to our experience, the relieving size for r correlates with the data size; that is, the larger n, the higher r needs to increase to. In this regard, the proposed kernel is the most favorable because its performance is relatively stable even for small r.

5.2 Partitioning Approaches

We further compare the methods for partitioning needed by the proposed kernel. The comparison focuses on three aspects: effect of randomness, testing error, and computational efficiency. Among the several possible choices discussed in Section 4.1, only the PCA approach and the random projection approach (which is recommended) yield a balanced partitioning; hence, we compare only these two.



(a) Random partitioning



(b) PCA partitioning

Figure 4: Regression error curves of the proposed method. Data are recursively partitioned by using random projections (top row) and by PCA (bottom row), respectively.

Figure 4 summarizes the error curves (mean and standard deviation) resulting from the two approaches, based on the same setting as in the preceding subsection. The top row corresponds to the recommended approach whereas the bottom row PCA. One sees that

the mean error curves are almost identical in both approaches. PCA is less influenced by randomization. Such a result should not be surprising, because PCA bares no randomness at all on partitioning; the only randomness comes from the landmark points. Although the variation of the error curves of the random projection approach is higher, such a variation is acceptable, particularly when it is compared with that of other approximate kernels in the preceding subsection.

The essential reason why random projection is favored over PCA comes from the consideration of computational efficiency. To generate the normal direction of a partitioning hyperplane, random projection amounts to only generating d random numbers, whereas PCA requires computing the dominant singular vector of the shifted data matrix. Table 2 presents the time costs of the additional singular vector computation, against the partitioning cost and the overall training cost by using random projection. We call the singular vector computation "overhead." The overhead is much higher with respect to the partitioning step, because such a step has a negligible cost compared to the overall training. The overhead is also generally higher when r is smaller because there requires more partitioning. The overhead with respect to partitioning easily exceeds 100% for quite a few of the data sets, and sometimes it is even a few thousand percents. For the data set mnist, which has the largest dimension d, the overhead with respect to overall training ranges from approximately 50% to 800%.

5.3 Performance Results for Various Data Sets

We now compare various approximate kernels on all the data sets listed in Table 1. For each kernel and each r, we obtain the performance result through a grid search of the optimal parameters σ and λ. Because of the high cost of parameter tuning, no repetitions are run and hence the performances may be susceptible to randomization. Hence, conclusions are drawn across data sets and we refrain from over-interpreting the results of an individual data set. We run experiments with a few r's and we are particularly interested in the performance trend as r increases.

The results are summarized in Figures 5 and 6. In the panel of the plots, each row corresponds to one data set. The three columns are performance versus r, training time, and memory consumption, respectively. The performance is measured as the relative error in the regression case and the accuracy in the classification case. The memory cost is estimated and normalized. According to the analysis in Section 4.5, the memory cost of the proposed kernel is approximately $4r$ per training point, whereas for the other approximate kernels we use an estimation of r per point.

A few observations follow. First, the proposed kernel almost always yields the best performance versus r, except for the data set YearPredictionMSD. Such an observation confirms the effectiveness of the proposed method, which combines the advantages of the Nyström method globally and the cross-domain independent kernel locally.

Second, the Fourier method generally runs the fastest, followed by Nyström and independent, whose speeds are comparably similar; and the proposed method falls behind. Although all methods have the same asymptotic cost $O(nr^2)$, in a finer level of analysis, Fourier wins over Nyström because the generation of random features in the former is often less expensive than the kernel evaluation in the latter. Of course, such a fact could also be

Table 2: PCA overhead with respect to partitioning and to overall training, under different r.

cadata		
r	partition.	train.
32	91.16%	9.48%
64	139.23%	7.59%
129	51.33%	2.29%
258	37.93%	0.94%
516	52.86%	0.78%

YearPredictionMSD		
r	partition.	train.
56	687.69%	71.74%
113	616.68%	50.97%
226	630.07%	21.07%
452	226.84%	5.64%
905	216.37%	2.66%

ijcnn1		
r	partition.	train.
34	199.52%	15.95%
68	8.80%	0.51%
136	153.16%	4.23%
273	58.42%	1.37%
546	64.74%	0.58%

covtype.binary		
r	partition.	train.
56	86.40%	7.03%
113	74.75%	4.18%
226	56.26%	1.88%
453	28.75%	0.59%
907	93.58%	0.66%

SUSY		
r	partition.	train.
61	85.40%	4.08%
122	99.44%	3.21%
244	52.10%	1.61%
488	27.41%	0.40%
976	79.71%	0.49%

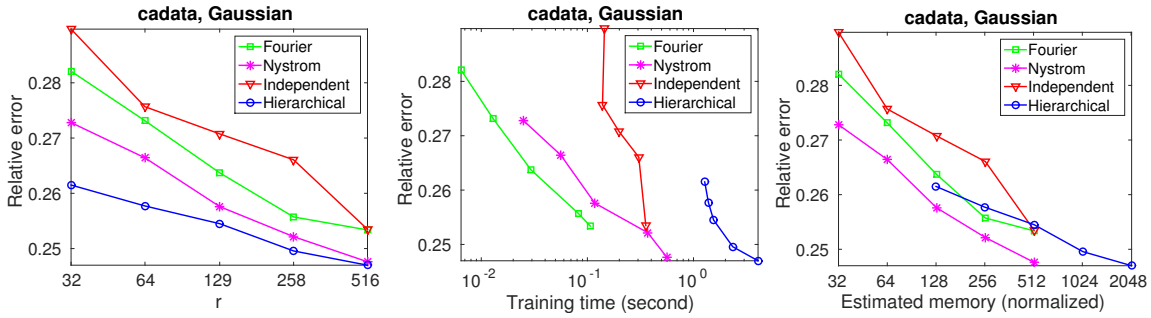
mnist		
r	partition.	train.
58	3973.02%	805.67%
117	3775.47%	508.89%
234	4341.62%	383.34%
468	3126.89%	151.64%
937	2175.98%	51.17%

acoustic		
r	partition.	train.
38	664.45%	53.07%
76	411.56%	29.15%
153	435.01%	21.66%
307	264.07%	7.62%
615	164.62%	2.28%

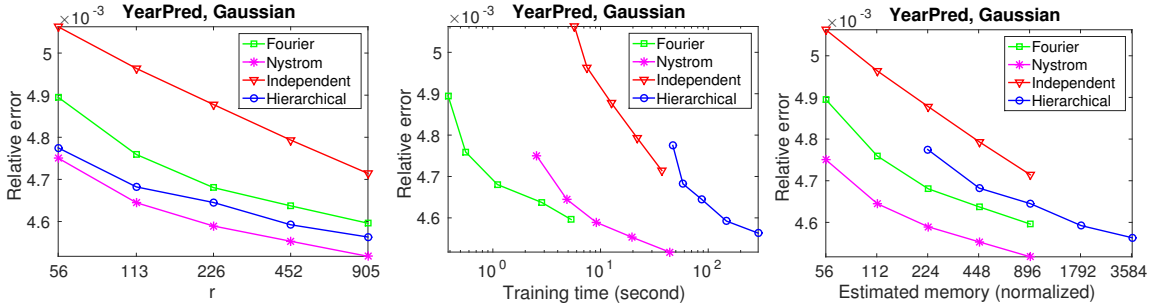
covtype		
r	partition.	train.
56	96.86%	7.02%
113	155.34%	6.29%
226	75.37%	2.77%
453	105.68%	1.60%
907	32.17%	0.30%

interpreted on the other hand as a weakness of the Fourier method: It is applicable to only a numeric data representation, whereas Nyström assumes no numeric form of data but a kernel. Nyström has a similar time cost as does the independent kernel; the former requires more kernel evaluations but the latter needs computing the sequence of subdomains each point belongs to. The independent kernel and the proposed kernel need similar computations (small matrix multiplications and factorizations), but the number of such operations is a constant times more for the proposed kernel; hence, not surprisingly it requires more computational effort.

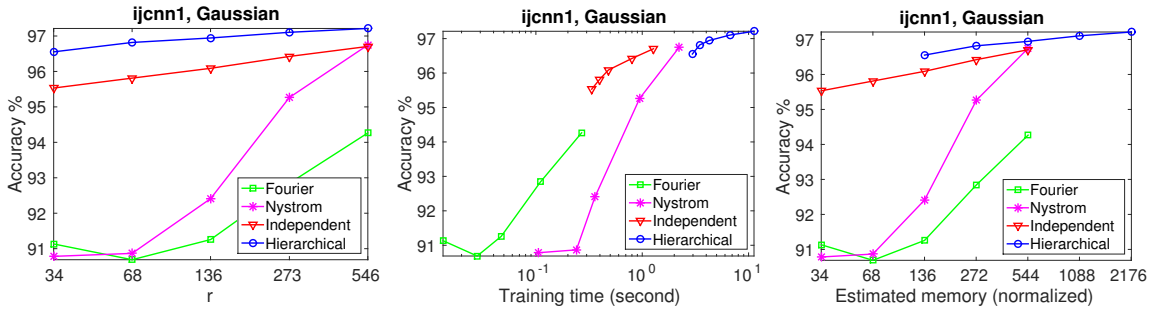
Note that in the Fourier and the Nyström methods, large matrices are operated in a small number of times. The implementation straightforwardly makes use of sophisticated linear algebra libraries, where parallelism and cache reuse have been aggressively optimized (Anderson et al., 1999; Goto and Geijn, 2008). On the other hand, in the independent and the proposed kernels, small matrices are operated in a large number of times. Moreover, there exist fragmented computations such as tree traversals and data shuffles. Optimizing these computations in terms of parallelism and cache reuse is possible and the executation times have the potential to be substantially improved.



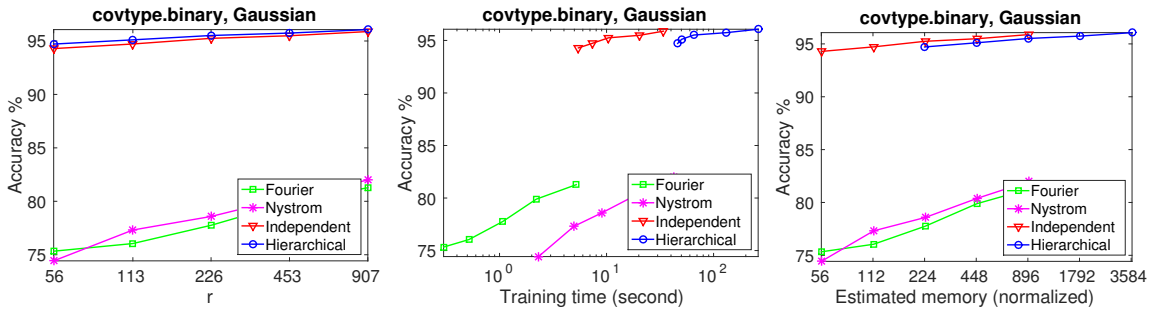
(a) cadata, regression



(b) YearPredictionMSD, regression

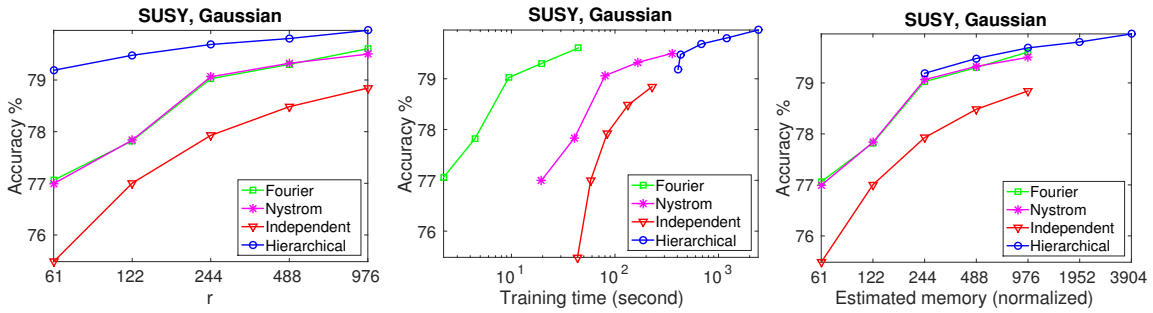


(c) ijcnn1, binary classification

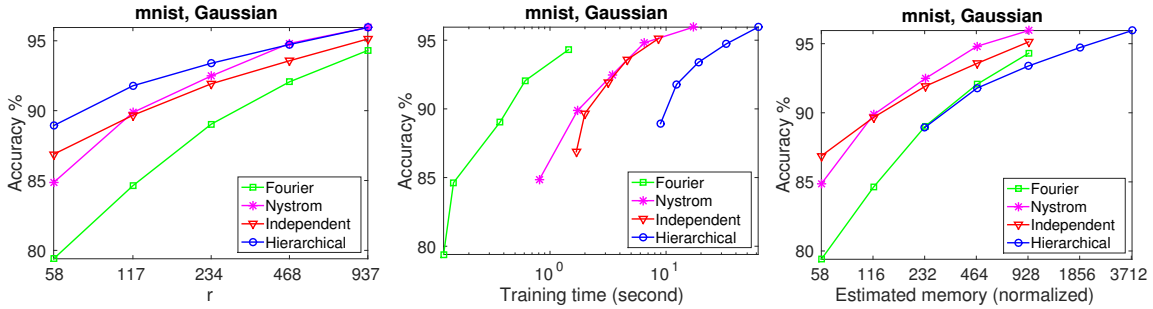


(d) covtype.binary, binary classification

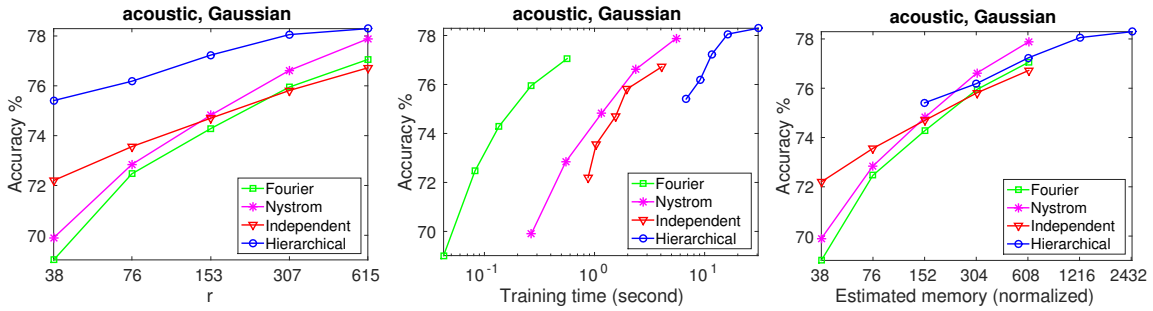
Figure 5: Performance versus r, time, and memory. Gaussian kernel.



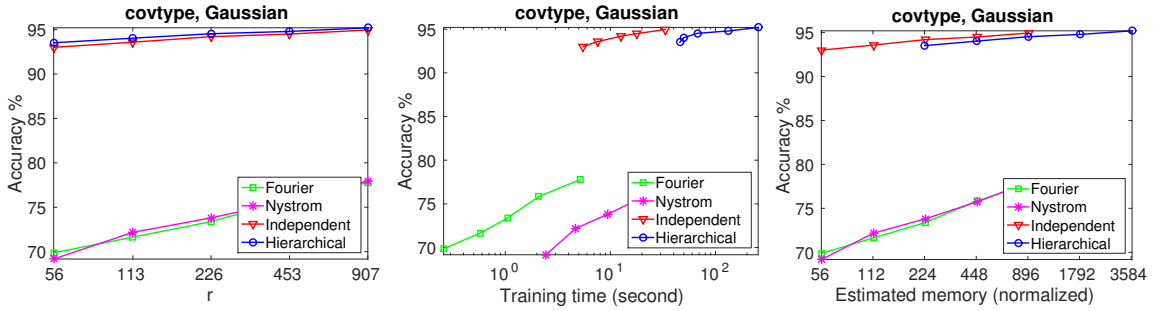
(a) SUSY, binary classification



(b) mnist, multiclass classification



(c) acoustic, multiclass classification



(d) covtype, multiclass classification

Figure 6: (Continued) Performance versus r, time, and memory. Gaussian kernel.

The third observation of Figures 5 and 6 focuses on the performance versus memory consumption. Since all the other methods are estimated to use the same amount of memory, the plots are essentially moving only the curve of the proposed kernel to the right compared with the performance-versus-r plots. Then, with the same amount of memory, some methods behave better than others in some data sets. There does not exist a consistent winner. One sees that the proposed method is the best performing for the data sets ijcnn1 and SUSY.

The last observation is that for the data set covtype (both the binary and the multiclass case), a significant performance gap occurs between the low-rank (Fourier and Nyström) and the full-rank (independent and hierarchical) kernels. This is the exemplified case where the eigenvalues in the kernel matrix decay too slowly, such that low-rank kernels with an insufficiently large rank r are clearly underperforming.

5.4 A Different Base Kernel

We consider using a different base kernel and inspect whether the observations made in the preceding subsection still hold. Here, we use the Laplace kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_1}{\sigma}\right)$$

popularized by Rahimi and Recht (2007). Clearly, this kernel is the tensor product of one-dimensional exponential kernels $\exp(-|x_i - x_i'|/\sigma)$ for all attributes i. In the GP context, the exponential kernel gives rise to the well-known Ornstein-Uhlenbeck process. Both the exponential kernel and the Gaussian kernel are special cases of the Matérn family of kernels, but their characteristics are opposite: the latter yields an extremely smooth process whereas the former highly rough (Stein, 1999). Hence, one might expect that their results substantially differ.

However, the answer is to the contrary. To avoid cluttering, we leave the plots to the appendix (see Figures 9 and 10); they are similar to those shown in Figures 5 and 6 of the Gaussian kernel. Specifically, the general trends and the specific performance values are quite close across the two base kernels. A possible reason is that the (optimal) regularization λ generally lies on the order of 0.01 to 0.1, which is relatively large compared with the kernel values, whose peak occurs at $k(0) = 1$. In the GP context, the noise level λ is so high that it eclipses the effect of smoothness—the rough variation of data may as well be interpreted as noise instead. Then, the smoothness of the kernel matters little and thus the results of different base kernels look similar.

Based on this observation, we additionally perform experiments with the inverse multiquadric kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \frac{\sigma^2}{\sqrt{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2 + \sigma^2}}.$$

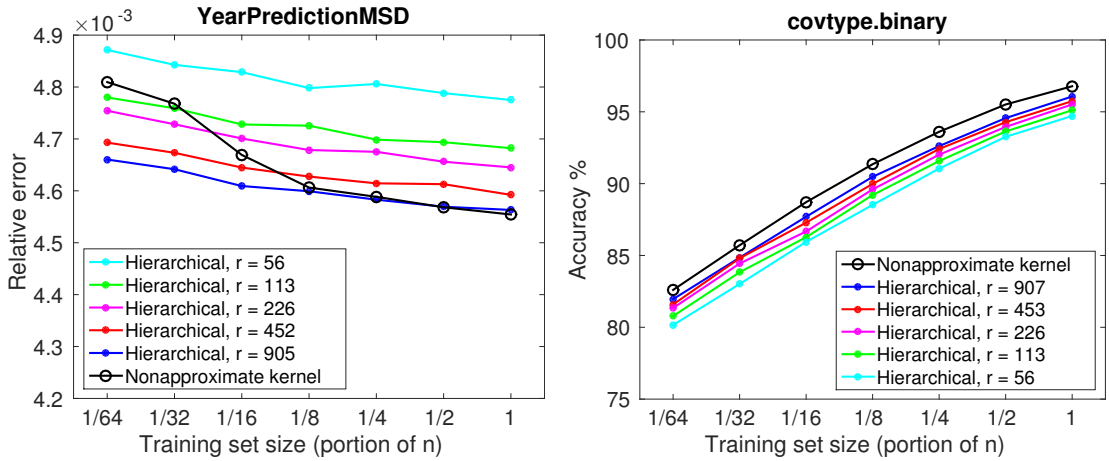
The strict positive definiteness of this kernel is proved in Micchelli (1986); but its Fourier transform is little known and hence we do not compare with the random Fourier method. The results are shown in the appendix (see Figures 11 and 12). One sees again that they are quite similar to those of the Gaussian and Laplace kernels.

5.5 Trade-Off between n and r

A folk wisdom in machine learning is that more data beats a more complex model (Domingos, 2012). The applicability of this knowledge lies in the regime where the hypothesis space has not been saturated by data. For kernel methods, a natural question to ask is whether subsampling is viable, considering that the computational effort is nontrivial with respect to the data size n. A relevant question in the context of approximate kernels is whether a trade-off exists between n and r, given a fixed budget nr. This budget comes from the memory constraint, because the memory cost is $O(nr)$ in various approximate kernels. Note that if nr is fixed, the arithmetic cost $O(nr^2)$ will increase when n becomes smaller, but the kernels tend to admit a better approximation quality.

Another scenario that resolves the interplay between n and r is the following: Suppose the current computational resources have been fully leveraged for computation and one is offered an upgrade such that the memory capacity is increased t times. To expect the best performance improvement, should one seek t times more training samples (provided feasible), or a smaller increase in training samples but meanwhile also an increase in r?

To answer this question, we use two data sets, YearPredictionMSD and covtype.binary, and progressively downsize them by a factor of two. Recall that the former is a regression problem whereas the latter classification. We plot in Figure 7 the performance curves versus the training size, for a few r's in the progression of approximately a factor of two. For comparison, we also perform the full-fledged computation with the nonapproximate kernel through solving (2) directly by using a preconditioned Krylov method on a cluster of AWS EC2 machines (see Avron et al. (2016) for details). The performance curve is black with circle markers.



(a) YearPredictionMSD, regression

(b) covtype.binary, binary classification

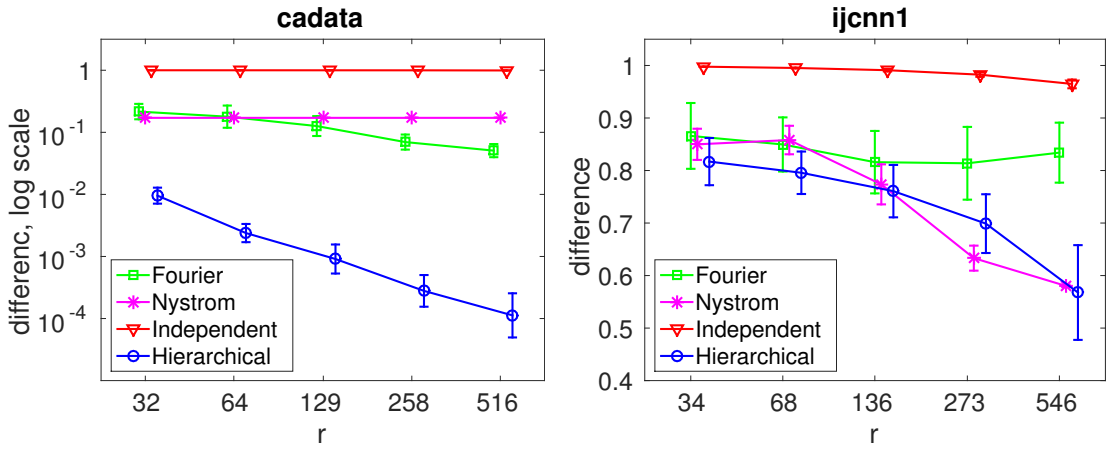
Figure 7: Performance under different training sizes and r. Kernel: Gaussian.

The question does not seem to bare a clear and simple answer after one investigates the two plots in Figure 7. Commonly, the performance of the proposed kernel improves in a consistent pace as r increases. For covtype.binary, the curves approach that of the nonapproximate kernel; however, such is not the case for YearPredictionMSD. In this data

set, when the training size is small, increasing r may surpass the performance of the non-approximate kernel. Furthermore, for covtype.binary, increasing the training size clearly yields a much higher performance than does increasing r by the same factor; however, for YearPredictionMSD, increasing r is to the contrary more beneficial. Hence, what the trade-off between n and r favors appears to be data set dependent.

5.6 Kernel PCA

Kernel principal component analysis (kernel PCA; see Schölkopf et al. (1998)) is another popular application of kernel methods. The standard PCA defines the embedding of a point $\boldsymbol{x}$ as the projected coordinates along the principal components of the training set X; whereas kernel PCA defines the embedding as the projected coordinates of $\phi(\boldsymbol{x})$ along the principal components of $\phi(X)$, where ϕ denotes the mapping from the input space to the feature space. For low-rank kernels (e.g., Nyström and Fourier) that give the explicit feature map ϕ, kernel PCA may be straightforwardly computed through singular value decomposition of the feature points $\phi(X)$ after centering. For other kernels (e.g., cross-domain independent kernel and the proposed kernel), one may leverage the relation $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$ and compute the embedding through eigenvalue decomposition of the kernel matrix $K(X, X)$ after centering.



(a) cadata, $n = 16,512$ (b) ijcnn1, $n = 35,000$

Figure 8: Kernel PCA: Embedding alignment difference (embedding dimension = 3). The horizontal placement of each curve is slightly jittered to avoid cluttering.

We follow Zhang et al. (2008) and evaluate the embedding quality of different kernels against that of the base kernel. Specifically, let U be the embedding matrix, where each row corresponds to the embedding of one point $\boldsymbol{x}$; similarly, let $\widetilde{U}$ be the embedding matrix resulting from an approximate kernel. We use a matrix M to align $\widetilde{U}$ with U; that is, M minimizes $\|U - \widetilde{U}M\|_F$. Then, we show in Figure 8 the alignment difference $\|U - \widetilde{U}M\|_F / \|U\|_F$ for different approximate kernels with various (rank) r. The embedding dimension is fixed to 3 and the base kernel is Gaussian, with the choice of bandwidth σ

approximately equal to the optimum found in the preceding experiments. One sees that generally the proposed kernel results in the smallest alignment difference.

6. Concluding Remarks

Kernel methods enjoy a strong theoretical support for data interpretation and predictive modeling; however, they come with a prohibitive computational price. Substantial efforts exist in the search of approximate kernels. Low-rank kernels are one of the most popular approaches because their costs have a linear dependency on the data size n and because of their relatively simple linear algebra implementation. From the matrix angle, the effectiveness of a low-rank approximation, however, depends heavily on the decay of eigenvalues, which is often found to be insufficiently fast for large data. We may interpret that low-rank approximations tackle the problem globally, because its purpose is to approximate the whole matrix. On the other hand, block diagonal approximations (e.g., the cross-domain independent kernel) focus on the preservation of local information only. They may approximate very poorly the kernel matrix, but they often produce surprisingly good results for regression and classification (see the covtype example in Section 5). Hence, the hierarchically compositional kernel proposed in this paper is motivated by the separate characteristics of these approximations: it preserves the full information in local domains and gradually applies low-rank approximations when the domains expand in a hierarchical fashion. The net effect is that the kernel admits an $O(nr)$ storage and an $O(nr^2)$ arithmetic cost similar to that of others, but the factor r may be reduced for a matching performance when n is large. This is favorable when computational resources are constrained; that is, as the data size n increases, the additional factor r must remain small in order for computation to be affordable.

The computational efficiency of the proposed kernel is supported by a delicate matrix structure. The resulting kernel matrix $K_{\text{hierarchical}}$ is a special case of the *recursively low-rank compressed matrices* studied in Chen (2014b). Such a hierarchical structure allows many otherwise $O(n^2)$–$O(n^3)$ dense matrix operations done with an $O(nr)$–$O(nr^2)$ cost. Examples of these matrix operations are matrix-vector multiplication, matrix inversion, log-determinant calculation (Chen, 2014b), and square-root decomposition (Chen, 2014a). In addition to the focus of this paper—regression and classification, such matrix operations are crucial in many machine learning and statistical settings, including Gaussian process modeling, simulation of random processes, likelihood calculation, Markov Chain Monte Carlo, and Bayesian inference. The proposed kernel hints a direction for performing scalable computations facing an ever increasing n.

The recursively low-rank compressed structure is not a unique discovery in the prosperous literature of structured matrix computations. Many practically useful structures in scientific computing (e.g., FMM matrices (Sun and Pitsianis, 2001) and hierarchical matrices (Hackbusch, 1999)) exploit the same low-rank property exhibited in the off-diagonal blocks of the matrix, but they differ from ours in fine details, including the admissibility of low-rank approximation, the method of approximation, the nesting of basis, the shape of the hierarchy tree, and the supported matrix operations. Apart from these differences, the most distinguishing feature of our work is that the matrix structure is amenable for high dimensional data, an arena where other structured matrix methods are rarely applicable.

This peculiarity roots in the fact that the decay of singular values of the off-diagonal blocks becomes much slower when the data dimension increases. These blocks generally have a full, or nearly full, rank when the data dimension is larger than three or four. Hence, it is challenging to develop computational methods from the angle of matrix approximations, if evaluation of quality is based on the approximation error. Instead, we develop theory from the perspective of kernels and ensure the preservation of positive-definiteness. Although the new kernel arises from some approximation of the old one, we may bypass the approximation interpretation and consider them separate kernels. The choice of kernels will then be justified via model selection. An additional benefit is that out-of-sample extension is straightforward.

Although the setting of this paper casts the data domain $\mathcal{X}$ as an Euclidean space for simplicity, it is possible to generalize the proposed method to a more abstract space (e.g., a metric space). In fact, the construction of the proposed kernel and the instantiation of the kernel matrix requires as minimal as the fact that the base kernel k is strictly positive-definite. The only exception is the partitioning of the data domain, where the partition membership must be inferred for every data point (including out of samples). In a metric space, one solution is the k-means clustering, because the clustering straightforwardly defines the partitions for the training data; moreover, the partition membership for a testing point is determined by the minimal distance (metric) from the point to the cluster centers.

Through experimentation, we have demonstrated the complex performance curves with respect to kernel parameters. The curves may vary significantly due to randomization, may be multimodal, and may even be nonsmooth. These phenomena pose a challenge for selecting the optimal parameters through cross validation and grid search. To make things worse, grid search is applicable only when the number of parameters is very small. An example when one may want more parameters is to introduce anisotropy to the kernel through specifying one range parameter for each or a few attributes. Hence, a more principled approach, which in theory can incorporate an arbitrary number of parameters, is to take the GP view and to maximize the Gaussian log-likelihood

$$L(\boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^T K(\boldsymbol{\theta})^{-1}\boldsymbol{y} - \frac{1}{2}\log\det K(\boldsymbol{\theta}) - \frac{n}{2}\log 2\pi \tag{25}$$

through numerical optimization, where $\boldsymbol{\theta}$ is the vector of unknown parameters and $K(\boldsymbol{\theta})$ is the kernel matrix depending on $\boldsymbol{\theta}$. This approach, coined *maximum likelihood estimation* (MLE), is also a central subject in estimation theory. Optimizing (25) nevertheless is nontrivial because of the $O(n^3)$ cost for evaluating the objective function and its derivatives when K is dense. Existing research (Anitescu et al., 2012; Stein et al., 2013) solves an approximate problem and bounds the variance of the result against that of the original problem (25). Fortunately, most of the approximate kernels discussed in this paper allow an $O(nr^2)$ cost for evaluating $L(\boldsymbol{\theta})$. For example, the algorithm for computing the log-determinant term for a recursively low-rank compressed matrix is described in Chen (2014b). An avenue of future work is to adapt the log-determinant calculation discussed in Chen (2014b) to the proposed kernel and to develop robust optimization for parameter estimation.

Acknowledgments

We would like to acknowledge support for this work by the XDATA program of the Defense Advanced Research Projects Agency (DARPA), administered through Air Force Research Laboratory contract FA8750-12-C-0323. We are indebted to the editor and the referees whose comments have substantially improved the paper.

Appendix A. Proof of Theorem 6

The proof technique is similar to that of Theorem 3: we decompose the kernel $k_{\text{hierarchical}}$ as a sum of positive-definite terms and show that if k is strictly positive-definite, then the bilinear form cannot be zero if the coefficients are not all zero. To prevent cluttering, we use the short-hand notation $K_{\underline{i},\underline{j}}$ to denote $K(\underline{X}_i, \underline{X}_j)$.

For any node i with parent p (if any), define a function $\xi^{(i)}$ for $\boldsymbol{x}, \boldsymbol{x}' \in S_i$:

$$\xi^{(i)}(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} k(\boldsymbol{x}, \boldsymbol{x}') - k(\boldsymbol{x}, \underline{X}_p) K^{-1}_{\underline{p},\underline{p}} k(\underline{X}_p, \boldsymbol{x}'), & i \text{ is leaf}, \\ \psi^{(i)}(\boldsymbol{x}, \underline{X}_i) K^{-1}_{\underline{i},\underline{i}} [K_{\underline{i},\underline{i}} - K_{\underline{i},\underline{p}} K^{-1}_{\underline{p},\underline{p}} K_{\underline{p},\underline{i}}] K^{-1}_{\underline{i},\underline{i}} \psi^{(i)}(\underline{X}_i, \boldsymbol{x}'), & i \text{ is neither leaf nor root}, \\ \psi^{(i)}(\boldsymbol{x}, \underline{X}_i) K^{-1}_{\underline{i},\underline{i}} \psi^{(i)}(\underline{X}_i, \boldsymbol{x}'), & i \text{ is root}. \end{cases} \tag{26}$$

Through telescoping, one sees that $k_{\text{hierarchical}}(\boldsymbol{x}, \boldsymbol{x}') = k^{(\text{root})}(\boldsymbol{x}, \boldsymbol{x}')$ is a sum of $\xi^{(i)}(\boldsymbol{x}, \boldsymbol{x}')$ for all nodes i where $\boldsymbol{x}, \boldsymbol{x}' \in S_i$. Because each $\xi^{(i)}$ is clearly positive-definite, so is $k_{\text{hierarchical}}$.

We now show the strict definiteness when k is strictly positive-definite. For any set of points $\{\boldsymbol{x}_i\}$ and any set of coefficients $\{\alpha_i\}$, the bilinear form $\sum_{jl} \alpha_j \alpha_l k_{\text{hierarchical}}(\boldsymbol{x}_j, \boldsymbol{x}_l)$ is zero if and only if

$$\sum_{\boldsymbol{x}_j, \boldsymbol{x}_l \in S_i} \alpha_j \alpha_l \xi^{(i)}(\boldsymbol{x}_j, \boldsymbol{x}_l) = 0, \quad \forall \text{ nodes } i. \tag{27}$$

If (27) holds, we shall prove the following statement:

(Q) If i is not the root, it holds that $\alpha_j = 0$ for all j satisfying

$$\boldsymbol{x}_j \in S_i \Bigg\backslash \left[\bigcap_{q \in Des(i)} \underline{X}_q \cap \underline{X}_p \right],$$

where p is the parent of i and $Des(i)$ means all the descendant nonleaf nodes of i, including i itself.

Once we prove this statement, we see that the only α_j that can possibly be nonzero are those satisfying $\boldsymbol{x}_j \in S \cap_{\text{all nodes } q} \underline{X}_q$. However, if any of these α_j is nonzero, then applying the third case of (26) on the root yields

$$\begin{aligned} \sum_{\boldsymbol{x}_j, \boldsymbol{x}_l \in S_{\text{root}}} \alpha_j \alpha_l \xi^{(\text{root})}(\boldsymbol{x}_j, \boldsymbol{x}_l) &= \sum_{\boldsymbol{x}_j, \boldsymbol{x}_l \in S_{\text{root}}} \alpha_j \alpha_l \psi^{(\text{root})}(\boldsymbol{x}_j, \underline{X}_{\text{root}}) K^{-1}_{\underline{\text{root}},\underline{\text{root}}} \psi^{(\text{root})}(\underline{X}_{\text{root}}, \boldsymbol{x}_l) \\ &= \sum_{\boldsymbol{x}_j, \boldsymbol{x}_l \in S \cap_{\text{all nodes } q} \underline{X}_q} \alpha_j \alpha_l e_j^T K_{\underline{\text{root}},\underline{\text{root}}} e_l \neq 0, \end{aligned}$$

contradicting (27). Here, e_j means the jth column of the identity matrix. Therefore, the bilinear form $\sum_{jl} \alpha_j \alpha_l k_{\text{hierarchical}}(\boldsymbol{x}_j, \boldsymbol{x}_l)$ cannot be zero if the coefficients are not all zero.

We now prove statement (Q) by induction. The base case is when i is a leaf node. Applying the same argument as that in the proof of Theorem 3 on the first case of (26), we have that $\alpha_j = 0$ for all j satisfying $\boldsymbol{x}_j \in S_i \backslash \underline{X}_p$. In the induction step, if (Q) holds for all children nodes i of p and if r is the parent of p, we apply the second case of (26) and obtain

$$0 = \sum_{\boldsymbol{x}_j, \boldsymbol{x}_l \in S_p} \alpha_j \alpha_l \xi^{(p)}(\boldsymbol{x}_j, \boldsymbol{x}_l) = \sum_{\boldsymbol{x}_j, \boldsymbol{x}_l \in S_p} \alpha_j \alpha_l \psi^{(p)}(\boldsymbol{x}_j, \underline{X}_p) K^{-1}_{\underline{p},\underline{p}} [K_{\underline{p},\underline{p}} - K_{\underline{p},\underline{r}} K^{-1}_{\underline{r},\underline{r}} K_{\underline{r},\underline{p}}] K^{-1}_{\underline{p},\underline{p}} \psi^{(p)}(\underline{X}_p, \boldsymbol{x}_l)$$
$$= \sum_{\boldsymbol{x}_j, \boldsymbol{x}_l \in S_p \backslash \cap_{q \in Des(p)} \underline{X}_q} \alpha_j \alpha_l e_j^T [K_{\underline{p},\underline{p}} - K_{\underline{p},\underline{r}} K^{-1}_{\underline{r},\underline{r}} K_{\underline{r},\underline{p}}] e_l. \tag{28}$$

For the matrix $K_{\underline{p},\underline{p}} - K_{\underline{p},\underline{r}} K^{-1}_{\underline{r},\underline{r}} K_{\underline{r},\underline{p}}$ inside the square bracket, the block corresponding to $\underline{X}_p \backslash \underline{X}_r$ has full rank, whereas the rest is zero. Then, in order for (28) to be zero, one must have that $\alpha_j = 0$ for all j satisfying $\boldsymbol{x}_j \in \underline{X}_p \backslash \underline{X}_r$. Making a disjunction of this relation with that in the induction hypothesis: $\boldsymbol{x}_j \in S_p \backslash \cap_{q \in Des(p)} \underline{X}_q$, we obtain

$$\boldsymbol{x}_j \in S_p \Bigg\backslash \left[\bigcap_{q \in Des(p)} \underline{X}_q \cap \underline{X}_r \right],$$

which concludes the induction and subsequently concludes the proof of Theorem 6.

Appendix B. Performance Results with the Laplace Kernel

See Figures 9 and 10. These plots are qualitatively similar to those of the Gaussian kernel (Figures 5 and 6).

Appendix C. Performance Results with the Inverse Multiquadric Kernel

See Figures 11 and 12. These plots are qualitatively similar to those of the Gaussian kernel (Figures 5 and 6).

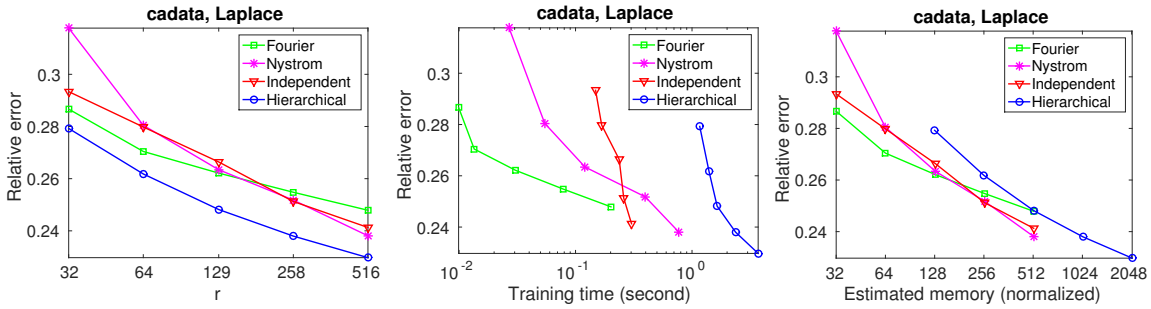
References

E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1999.

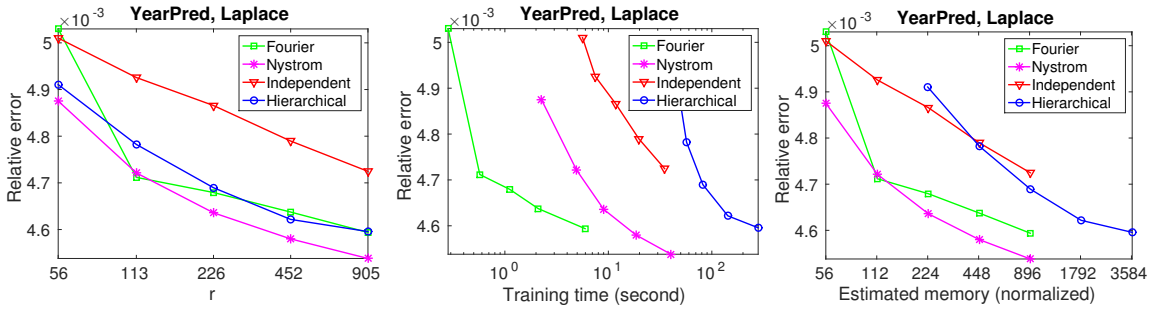
Mihai Anitescu, Jie Chen, and Lei Wang. A matrix-free approach for solving the parametric Gaussian process maximum likelihood problem. *SIAM Journal on Scientific Computing*, 34(1):A240–A262, 2012.

Haim Avron and Vikas Sindhwani. High-performance kernel machines with implicit distributed optimization and randomization. *Technometrics*, 58(3):341–349, 2016.

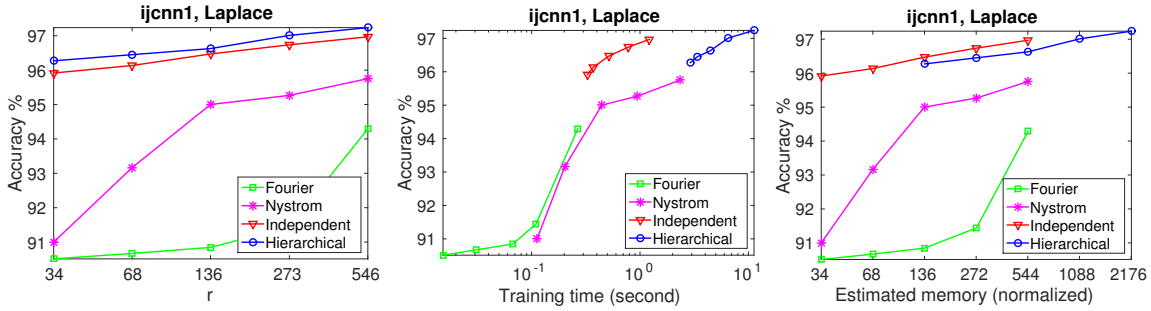
Haim Avron, Kenneth Clarkson, and David Woodruff. Faster kernel ridge regression using sketching and preconditioning. Submitted, 2016.



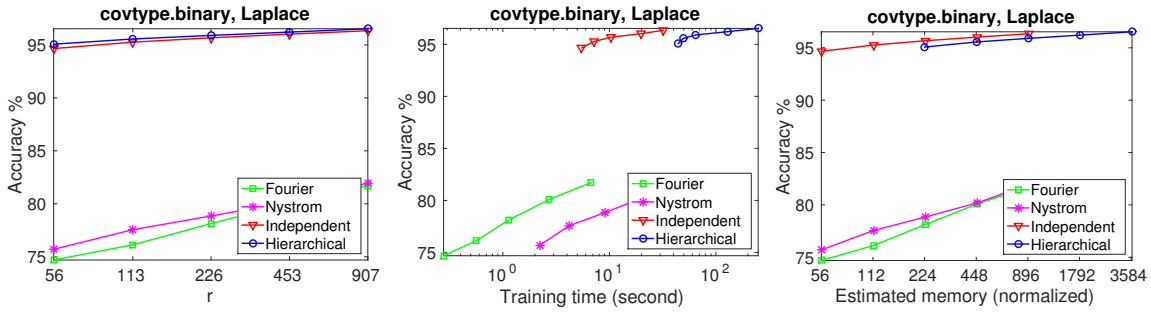
(a) cadata, regression



(b) YearPredictionMSD, regression

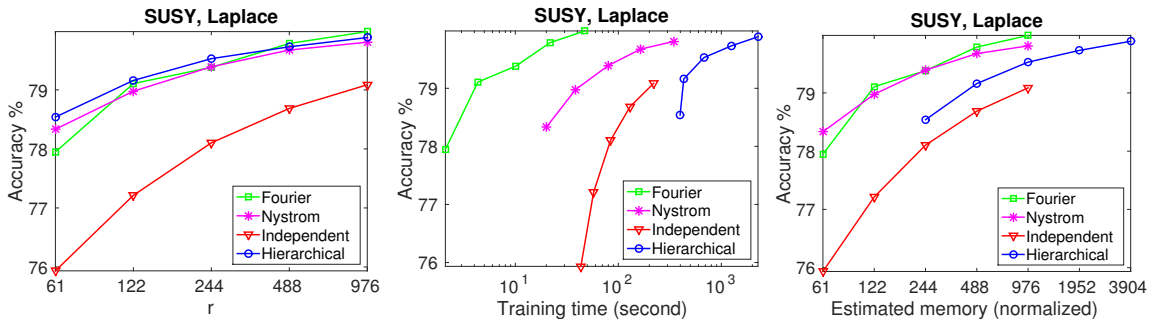


(c) ijcnn1, binary classification

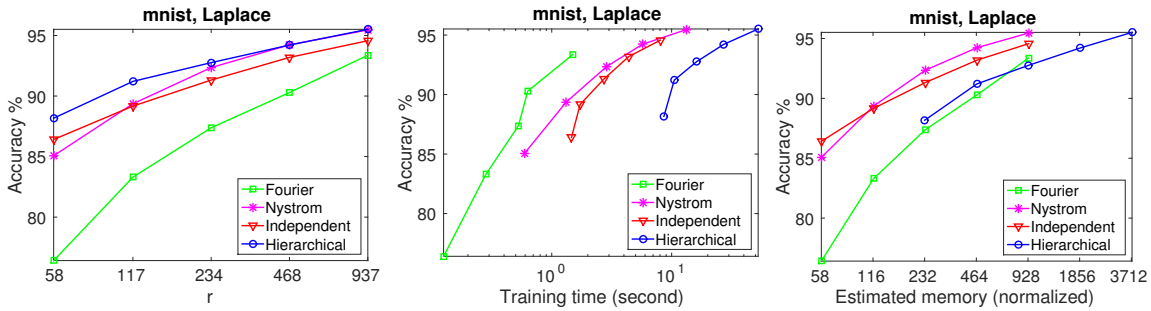


(d) covtype.binary, binary classification

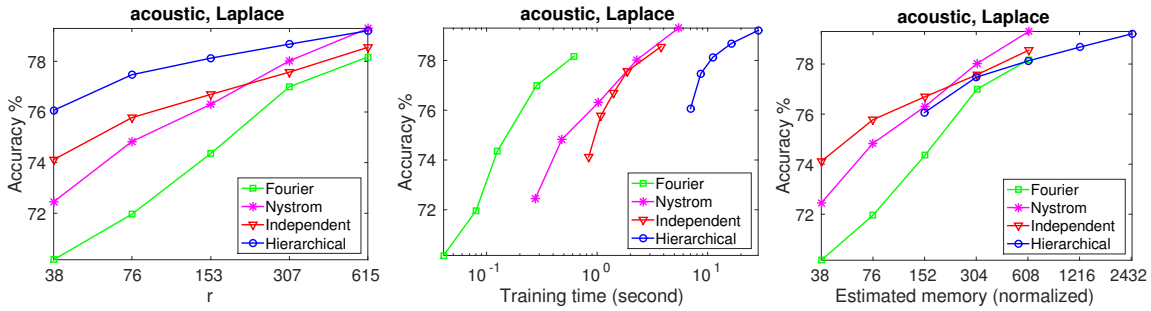
Figure 9: Performance versus r, time, and memory. Laplace kernel.



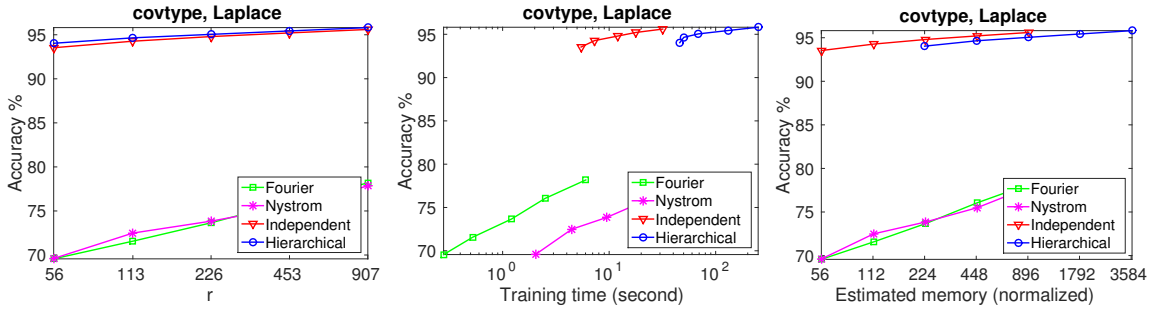
(a) SUSY, binary classification



(b) mnist, multiclass classification

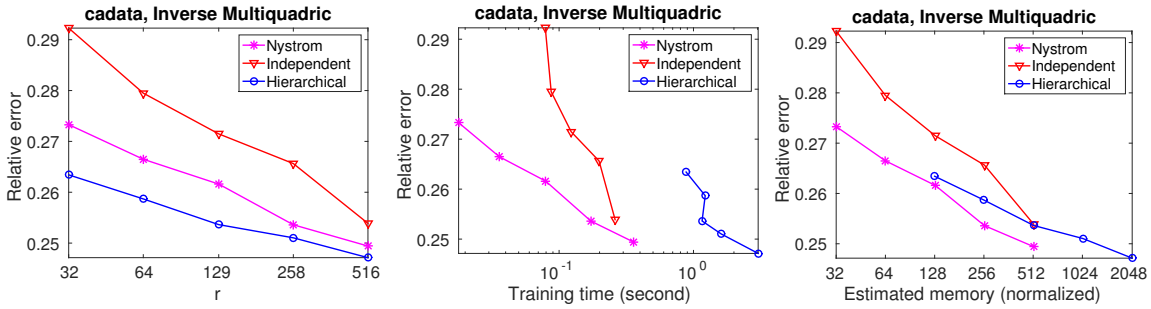


(c) acoustic, multiclass classification

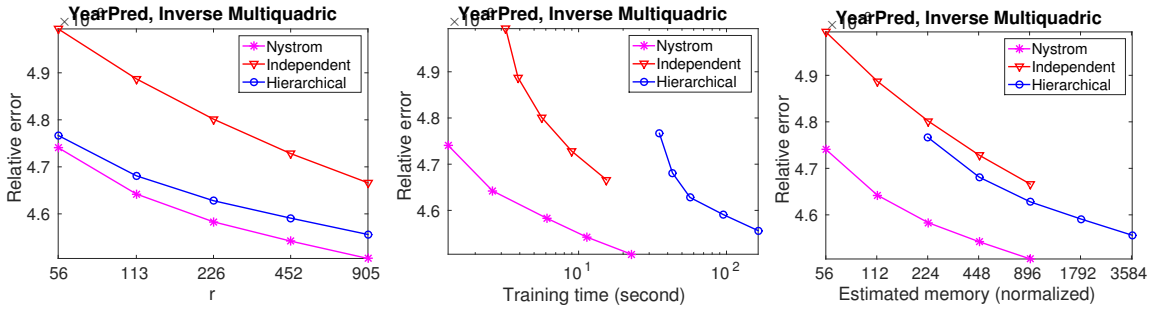


(d) covtype, multiclass classification

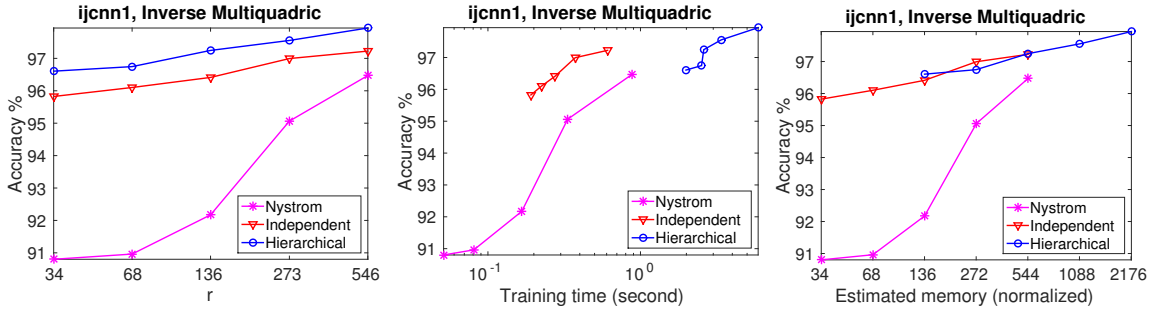
Figure 10: (Continued) Performance versus r, time, and memory. Laplace kernel.



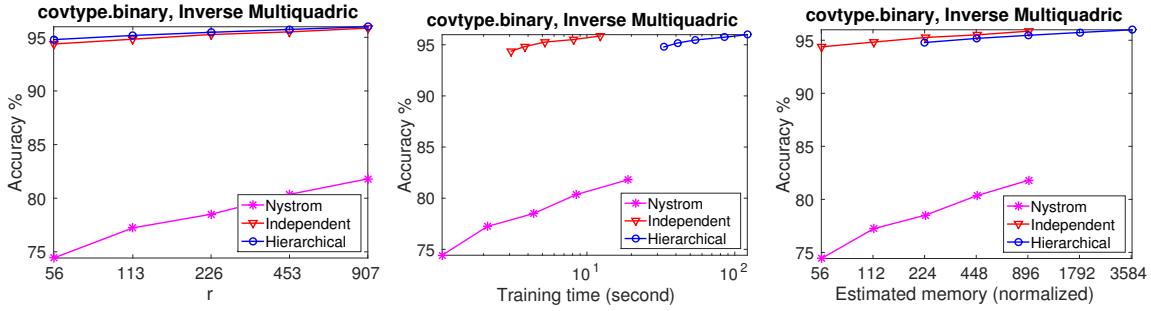
(a) cadata, regression



(b) YearPredictionMSD, regression

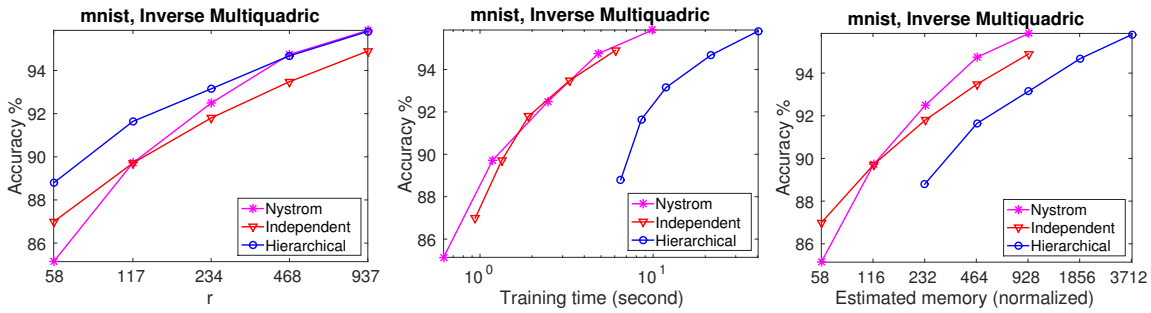


(c) ijcnn1, binary classification

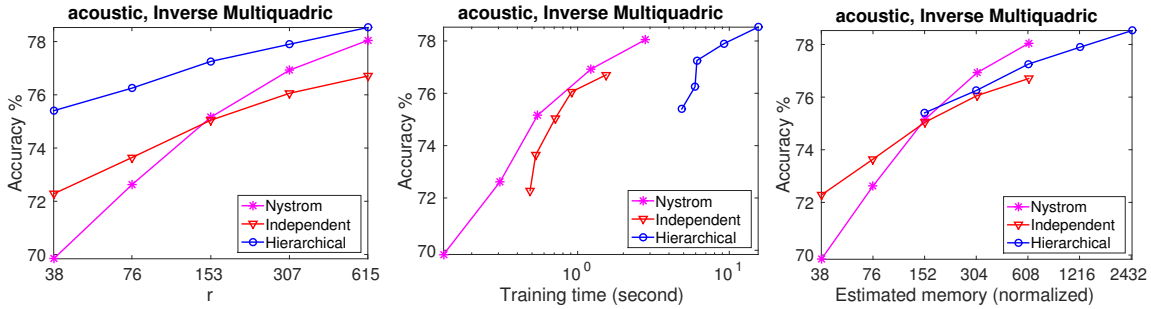


(d) covtype.binary, binary classification

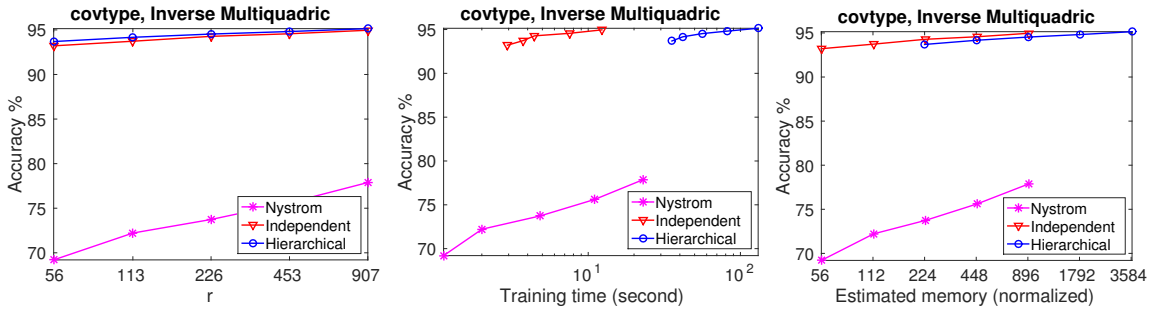
Figure 11: Performance versus r, time, and memory. Inverse multiquadric kernel.



(a) mnist, multiclass classification



(b) acoustic, multiclass classification



(c) covtype, multiclass classification

Figure 12: (Continued) Performance versus r, time, and memory. Inverse multiquadric kernel.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 8(9):509–517, 1975.

B. Boots, A. Gretton, and G. J. Gordon. Hilbert space embeddings of predictive state representations. In *Proc. 29th Intl. Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2013.

Jie Chen. On the use of discrete Laplace operator for preconditioning kernel matrices. *SIAM Journal on Scientific Computing*, 35(2):A577–A602, 2013.

Jie Chen. Computing square root factorization for recursively low-rank compressed matrices. Technical Report RC25499, IBM Thomas J. Watson Research Center, 2014a.

Jie Chen. Data structure and algorithms for recursively low-rank compressed matrices. Technical Report ANL/MCS-P5112-0314, Argonne National Laboratory, 2014b.

Jie Chen, Haw-Ren Fang, and Yousef Saad. Fast approximate kNN graph construction for high dimensional data via recursive Lanczos bisection. *Journal of Machine Learning Research*, 10(Sep):1989–2012, 2009.

Jean-Paul Chilès and Pierre Delfiner. *Geostatistics: Modeling Spatial Uncertainty*. Wiley, 2012.

Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2002.

Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

Petros Drineas and Michael W. Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.

Reinhard Furrer, Marc G Genton, and Douglas Nychka. Covariance tapering for interpolation of large spatial datasets. *Journal of Computational and Graphical Statistics*, 15(3): 502–523, 2006.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

Kazushige Goto and Robert Van De Geijn. High-performance implementation of the level-3 BLAS. *ACM Transactions on Mathematical Software*, 35(1), 2008.

W. Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part I: Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

Z. Harchaoui, F. Bach, O. Cappe, and E. Moulines. Kernel-based methods for hypothesis testing: A unified view. *Signal Processing Magazine, IEEE*, 30(4):87–97, July 2013.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, second edition, 2009.

Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1994.

P. Huang, H. Avron, T. N. Sainath, V. Sindhwani, and B. Ramabhadran. Kernel methods match deep neural networks on TIMIT. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014.

William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference on Modern Analysis and Probability*, volume 26, pages 189–206. American Mathematical Society, 1984.

Cari Kaufman, Mark Schervish, and Douglas Nychka. Covariance tapering for likelihood-based estimation in large spatial data sets. *Journal of the American Statistical Association*, 103:1545–1555, 2008.

J. R. Koehler and A. B. Owen. *Handbook of statistics*, volume 13, chapter 9 Computer Experiments, pages 261–308. Elsevier B.V., 1996.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.*, volume 1, pages 281–297, 1967.

William B. March, Bo Xiao, and George Biros. ASKIT: Approximate skeletonization kernel-independent treecode in high dimensions. Preprint arXiv:1410.0260, 2014.

Charles A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2(1):11–22, 1986.

J. J. Monaghan and J. C. Lattanzio. A refined particle method for astrophysical problems. *Astronomy and Astrophysics*, 149(1):135–143, 1985.

A. O'Hagan and J. F. C. Kingman. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society. Series B (Methodological)*, 40(1):1–42, 1978.

K Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(11):559–572, 1901.

Ali Rahimi and Ben Recht. Random features for large-scale kernel machines. In *Neural Infomration Processing Systems*, 2007.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition, 2003.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2001.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. *Lecture Notes in Computer Science*, 2111:416426, 2001.

S. Si, C.-J. Hsieh, and I.S. Dhillon. Memory efficient kernel approximation. In *Proceedings of the 25th international conference on Machine learning*, 2014.

Si Si, Cho-Jui Hsieh, and Inderjit S. Dhillon. Memory efficient kernel approximation. *Journal of Machine Learning Research*, 18(20):1–32, 2017.

Ralph C. Smith. *Uncertainty Quantification: Theory, Implementation, and Applications.* Society for Industrial and Applied Mathematics, 2013.

L Song, B. Boots, S. Siddiqi, G. Gordon, and A. Smola. Hilbert space embeddings of hidden Markov models. In *Proc. of the 30th International Conference on Machine Learning (ICML)*, 2013.

Michael L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging.* Springer, 1999.

Michael L. Stein. Limitations on low rank approximations for covariance matrices of spatial data. *Spatial Statistics*, 8:1–19, 2014.

Michael L. Stein, Jie Chen, and Mihai Anitescu. Stochastic approximation of score functions for Gaussian processes. *Annals of Applied Statistics*, 7(2):1162–1191, 2013.

Xiaobai Sun and Nikos P. Pitsianis. A matrix version of the fast multipole method. *SIAM Review*, 43(2):289–300, 2001.

Holger Wendland. *Scattered Data Approximation.* Cambridge University Press, 2004.

Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, 2000.

J. Yang, V. Sindhwani, Q. Fan, H. Avron, and M. Mahoney. Random Laplace feature maps for semigroup kernels on histograms. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

Tianbao Yang, Yu feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström method vs random Fourier features: A theoretical and empirical comparison. In *Advances in Neural Information Processing Systems 25*, 2012.

I. Yen, T. Lin, S. Lin, P. Ravikumar, and I. Dhillon. Sparse random feature algorithm as coordinate descent in Hilbert space. In *Neural Information Processing Systems*, pages 2456–2464, 2014.

Chenhan D. Yu, William B. March, and George Biros. An $n \log n$ parallel fast direct solver for kernel matrices. Preprint arXiv:1701.02324, 2017.

K. Zhang, J. Peters, D. Janzing, and B. Scholkopf. Kernel based conditional independence test and application in causal discovery. In *Proc. 27th Intl. Conf. on Uncertainty in Artificial Intelligence*, 2011.

Kai Zhang and James T. Kwok. Clustered Nyström method for large scale manifold learning and dimension reduction. *IEEE Transactions on Neural Networks*, 21(10):1576–1587, 2010.

Kai Zhang, Ivor W. Tsang, and James T. Kwok. Improved Nyström low-rank approximation and error analysis. In *Proceedings of the 25th international conference on Machine learning*, 2008.