SIAM J. MATRIX ANAL. APPL.
Vol. 40, No. 4, pp. 1497–1526

BLOCK BASIS FACTORIZATION FOR SCALABLE KERNEL EVALUATION*

RUOXI WANG†, YINGZHOU LI‡, MICHAEL W. MAHONEY§, AND ERIC DARVE¶

Abstract. Kernel methods are widespread in machine learning; however, they are limited by the quadratic complexity of the construction, application, and storage of kernel matrices. Low-rank matrix approximation algorithms are widely used to address this problem and reduce the arithmetic and storage cost. However, we observed that for some datasets with wide intraclass variability, the optimal kernel parameter for smaller classes yields a matrix that is less well-approximated by low-rank methods. In this paper, we propose an efficient structured low-rank approximation method—the block basis factorization (BBF)—and its fast construction algorithm to approximate radial basis function kernel matrices. Our approach has linear memory cost and floating point operations for many machine learning kernels. BBF works for a wide range of kernel bandwidth parameters and extends the domain of applicability of low-rank approximation methods significantly. Our empirical results demonstrate the stability and superiority over the state-of-the-art kernel approximation algorithms.

Key words. kernel matrix, low-rank approximation, data-sparse representation, machine learning, high-dimensional data, RBF

AMS subject classifications. 15A23, 65D05, 65F10, 62G08, 68W20, 68W25

DOI. 10.1137/18M1212586

1. Introduction.

Kernel methods are mathematically well-founded nonparametric methods for learning. The essential part of kernel methods is a kernel function $\mathcal{K} : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$. It is associated with a feature map Ψ from the original input space $\mathcal{X} \in \mathbb{R}^d$ to a higher-dimensional Hilbert space $\mathcal{H}$ such that

$$\mathcal{K}(\mathbf{x}, \mathbf{y}) = \langle \Psi(\mathbf{x}), \Psi(\mathbf{y}) \rangle_{\mathcal{H}}.$$

Presumably, the underlying function for data in the feature space is linear. Therefore, the kernel function enables us to build expressive nonlinear models based on the machinery of linear models. In this paper, we consider the radial basis function (RBF) kernel that is widely used in machine learning.

The kernel matrix is an essential part of kernel methods in the training phase and is defined in what follows. Given n data points $\{\mathbf{x}_i\}_{i=1}^n$, the (i, j)th entry in a kernel matrix is $K_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$. For example, the solution to a kernel ridge regression is the same as the solution to the linear system

$$(K + \delta I)\boldsymbol{\alpha} = \mathbf{y}.$$

Regrettably, any operations involving kernel matrices can be computationally expensive. Their construction, application, and storage complexities are quadratic in

*Received by the editors September 12, 2018; accepted for publication (in revised form) by M. W. Berry September 18, 2019; published electronically December 3, 2019.
https://doi.org/10.1137/18M1212586
Funding: The work of the third author was partially supported by ARO, DARPA, NSF, and ONR. The work of the fourth author was supported by NSF, the Department of Energy, and the Army Research Laboratory.

†Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA 94305 (ruoxi.rw@gmail.com).
‡Department of Mathematics, Duke University, Durham, NC 27708-0320 (yingzhou.li@duke.edu).
§International Computer Science Institute and Department of Statistics, University of California, Berkeley, CA 94720 (mmahoney@stat.berkeley.edu).
¶Department of Mechanical Engineering and Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA, 94305 (darve@stanford.edu).

the number of data points n. Moreover, for solving linear systems involving these matrices, the complexity is even higher. It is $O(n^3)$ for direct solvers [19] and $O(n^2T)$ for iterative solvers [28, 33], where T is the iteration number. This is prohibitive in large-scale applications. One popular solution to address this problem and reduce the arithmetic and storage cost is using matrix approximation. If we are able to approximate the matrix such that the number of entries that need to be stored is reduced, then the timing for iterative solvers will be accelerated (assuming memory is a close approximation of the running time for a matrix-vector multiplication).

In machine learning, low-rank matrix approximations are widely used [19, 32, 27, 31, 34, 15, 18, 14, 44]. When the kernel matrix has a large spectrum gap, a high approximation accuracy can be guaranteed by theoretical results. Even when the matrix does not have a large spectrum gap or fast spectrum decay, these low-rank algorithms are still popular practical choices to reduce the computational cost; however, the approximation would be less accurate.

The motivation of our algorithm is that in many machine learning applications, the RBF kernel matrices cannot be well-approximated by low-rank matrices [37]; nonetheless, they are not arbitrary high-rank matrices and are often of certain structure. In the rest of the introduction, we first discuss the importance of higher-rank matrices and then introduce the main idea of our algorithm that takes advantage of those structures. The RBF kernel $f(\|\mathbf{x}-\mathbf{y}\|/h)$ has a *bandwidth parameter* h that controls the size of the neighborhood, i.e., how many nearby points to be considered for interactions. The numerical rank of a kernel matrix depends strongly on this parameter. As h decreases from large to small, the corresponding kernel matrix can be approximated by a low-rank matrix whose rank increases from $O(1)$ to $O(n)$. In the large-h regime, traditional low-rank methods are efficient; however, in the small-h regime, these methods fall back to quadratic complexity. The bandwidth parameter is often chosen to maximize the overall performance of regression/classification tasks, and its value is closely related to the smoothness of the underlying function. For kernel regressions and kernelized classifiers, the hypothesis function classes are $\sum_i \alpha_i \mathcal{K}_h(\mathbf{x}, \mathbf{x}_i)$ and $\sum_i \alpha_i y_i \mathcal{K}_h(\mathbf{x}, \mathbf{x}_i)$, respectively. Both can be viewed as interpolations on the training data points. Clearly, the optimal value of h should align with the smoothness of the underlying function. Although many real-world applications have found large h to lead to good overall performances, in a lot of cases a large h will hurt the performance. For example, in kernel regression, when the underlying function is nonsmooth such as those with sharp local changes, using a large bandwidth will smooth out the local structures; in kernelized classifiers, when the true decision surfaces that separate two classes are highly nonlinear, choosing a large bandwidth imposes smooth decision surfaces on the model and ignores local information near the decision surfaces. In practice, the previous situations where relatively small bandwidths are needed are very common. One example is that for classification datasets with imbalanced classes, often the optimal h for smaller classes is relatively small. Hence, if we are particularly interested in the properties of smaller classes, a small h is appropriate. As a consequence, matrices of higher ranks occur frequently in practice.

Therefore, for certain machine learning problems, low-rank approximations of dense kernel matrices are inefficient. This motivates the development of approximation algorithms that extend the applicability of low-rank algorithms to matrices of higher ranks, i.e., that work efficiently for a wider range of kernel bandwidth parameters.

In the field of scientific computing (which also considers kernel matrices, but typically for very different ends), hierarchical algorithms [20, 21, 12, 13, 17, 42] ef-

ficiently approximate the forward application of full-rank PDE kernel matrices in low dimensions. These algorithms partition the data space recursively using a tree structure and separate the interactions into near- and far-field interactions, where the near-field interactions are calculated hierarchically and the far-field interactions are approximated using low-rank factorizations. Later, hierarchical matrices ($\mathcal{H}$-matrix, $\mathcal{H}^2$-matrix, HSS matrix, HODLR matrix) [24, 26, 25, 8, 40, 4] were proposed as algebraic variants of these hierarchical algorithms. Based on the algebraic representation, the application of the kernel matrix as well as its inverse, or its factorization can be processed in quasi-linear ($O(n \log^k n)$) operations. Due to the tree partitioning, the extension to high-dimensional kernel matrices is problematic. Both the computational and storage costs grow exponentially with the data dimension, spoiling the $O(n)$ or $O(n \log n)$ complexity of those algorithms.

In this paper, we adopt some ideas from hierarchical matrices and butterfly factorizations [29, 30] and propose a block basis factorization (BBF) structure that generalizes the traditional low-rank matrix, along with its efficient construction algorithm. We apply this scientific computing–based method to a range of problems, with an emphasis on machine learning problems. We will show that the BBF structure achieves significantly higher accuracy than plain low-rank matrices, given the same memory budget, and the construction algorithm has a linear in n complexity for many machine learning kernel learning tasks.

The key of our structure is realizing that in most machine learning applications, the submatrices representing the interactions from one cluster to the entire dataset are numerically low-rank. For example, Wang et al. [39] mathematically proved that if the diameter of a cluster $\mathcal{C}$ is smaller than that of the entire dataset $\mathcal{X}$, then the rank of the submatrix $\mathcal{K}(\mathcal{C}, \mathcal{X})$ is lower than the rank of the entire matrix $\mathcal{K}(\mathcal{X}, \mathcal{X})$. If we partition the data such that each cluster has a small diameter, and the clusters are as far apart as possible from each other, then we can take advantage of the low-rank property of the submatrix $\mathcal{K}(\mathcal{C}, \mathcal{X})$ to obtain a presentation that is more memory-efficient than low-rank representations.

The application of our BBF structure is not limited to RBF kernels or machine learning applications. There are many other types of structured matrices for which the conventional low-rank approximations may not be satisfactory. Examples include but are not limited to covariance matrices from spatial data [38], frontal matrices in the multifrontal method for sparse matrix factorizations [3], and kernel method in dynamic systems [7].

1.1. Main contributions. Our main contribution is threefold. First, we showed that for classification datasets whose decision surfaces have small radius of curvature, a small kernel bandwidth parameter is needed for high accuracy. Second, we proposed a novel matrix approximation structure that extends the applicability of low-rank methods to matrices whose ranks are higher. Third, we developed a corresponding construction algorithm that produces errors with small variance (the algorithm uses randomized steps) and that has linear, i.e., $O(n)$ complexity, for most machine learning kernel learning tasks. Specifically, our contributions are as follows:

- For several datasets with imbalanced classes, we observed an improvement in accuracy for smaller classes when we set the kernel bandwidth parameter to be smaller than that selected from a cross-validation procedure. We attribute this to the nonlinear decision surfaces, which we quantify as the smallest radius of curvature of the decision boundary.

- We proposed a novel matrix structure called the BBF for machine learning applications. BBF approximates the kernel matrix with linear memory and is efficient for a wide range of bandwidth parameters.
- We proposed a construction algorithm for the BBF structure that is accurate, stable, and linear for many machine learning kernels. This is in contrast to most algorithms to calculate the singular value decomposition (SVD), which are more accurate but lead to a cubic complexity, or naïve random sampling algorithms (e.g., uniform sampling), which are linear but often inaccurate or unstable for incoherent matrices. We also provided a fast precomputation algorithm to search for suggested input parameters for BBF.

Our algorithm involves three major steps. First, it divides the data into k distinct clusters and permutes the matrix according to these clusters. The permuted matrix has k^2 blocks, each representing the interactions between two clusters. Second, it computes the column basis for every row-submatrix (the interactions between one cluster and the entire dataset) by first selecting representative columns using a randomized sampling procedure and then compressing the columns using a randomized SVD. Finally, it uses the corresponding column- and row- basis to compress each of the k^2 blocks, also using a randomized subsampling algorithm. Consequently, our method computes an approximation for the k^2 blocks using a set of only k bases. The resulting framework yields a rank-R approximation and achieves a similar accuracy as the best rank-R approximation, where R refers to the approximation rank. The memory complexity for BBF is $\mathcal{O}(nR/k + R^2)$, where k is upper bounded by R. This should be contrasted with a low-rank scheme that gives a rank-R approximation with $\mathcal{O}(nR)$ memory complexity. BBF achieves a similar approximation accuracy to the best rank-R approximation with a factor of k saving on memory.

1.2. Related research. There is a large body of research that aims to accelerate kernel methods by low-rank approximations [19]. Given a matrix $K \in \mathbb{R}^{n\times n}$, a rank-$r$ approximation of K is given by $K \approx UV^\top$, where $U, V \in \mathbb{R}^{n\times r}$ and r is related to accuracy. The SVD provides the most accurate rank-r approximation of a matrix in terms of both 2-norm and Frobenius-norm; however, it has a cubic cost. Recent work [34, 31, 27, 32] has reduced the cost to $\mathcal{O}(n^2 r)$ using randomized projections. These methods require the construction of the entire matrix to proceed. Another line of the low-rank approximation research is the Nyström method [15, 18, 5], which avoids constructing the entire matrix. A naïve Nyström algorithm uniformly samples columns and reconstructs the matrix with the sampled columns, which is computationally inexpensive but which works well only when the matrix has uniform leverage scores, i.e., low coherence. Improved versions [16, 44, 14, 18, 1] of Nyström have been proposed to provide more sophisticated ways of column sampling.

There are several methods proposed to address the same problem as in this paper. The clustered low-rank approximation (CLRA) [35] performs a blockwise low-rank approximation of the kernel matrix from social network data with quadratic construction complexity. The memory-efficient kernel approximation (MEKA) [36] successfully avoids the quadratic complexity in CLRA. Importantly, these previous methods did not consider the class size and parameter size issues as we did in detail. Also, in our benchmark, we found that under multiple trials, MEKA is not robust; i.e., it often failed to be accurate and produced large errors. This is due to its inaccurate structure and its simple construction algorithm. We briefly discuss some significant differences between MEKA and our algorithm. In terms of the structure, the basis

in MEKA is computed from a smaller column space and is inherently a less accurate representation, making it more straightforward to achieve a linear complexity; in terms of the algorithm, the uniform sampling method used in MEKA is less accurate and less stable than the sophisticated sampling method used in BBF that is strongly supported by theory.

There is also a strong connection between our algorithm and the improved fast Gauss transform (IFGT) [41], which is an improved version of the fast Gauss transform [22]. Both BBF and IFGT use a clustering approach for space partitioning. Differently, the IFGT approximates the kernel function by performing an analytic expansion, while BBF uses an algebraic approach based on sampling the kernel matrix. This difference has made BBF more adaptive and achieves higher approximation accuracy. Along the same line of adopting ideas from hierarchical matrices, Chen et al. [10] combined the hierarchical matrices and the Nyström method to approximate kernel matrices rising from machine learning applications.

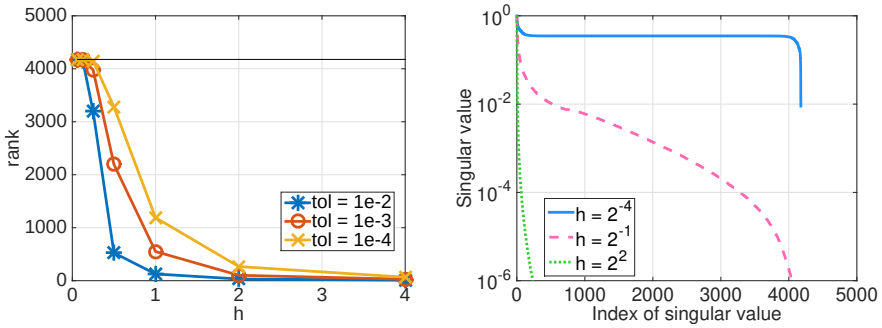
The paper is organized as follows. Section 2 discusses the motivations behind extending low-rank structures and designing efficient algorithms for higher-rank kernel matrices. Section 3 proposes a new structure that better approximates higher-rank matrices and remains efficient for lower-rank matrices, along with its efficient construction algorithm. Finally, section 4 presents our experimental results, which show the advantages of our proposed BBF over the state of the art in terms of the structure, algorithm, and applications to kernel regression problems.

2. Motivation: Kernel bandwidth and class size. In this section, we discuss the motivations behind designing an algorithm that remains computationally efficient when the matrix rank increases. Three main motivations are as follows. First, the matrix rank depends strongly on the kernel bandwidth parameters (chosen based on the particular problem); the smaller the parameter, the higher the matrix rank. Second, a small bandwidth parameter (higher-rank matrix) imposes high nonlinearity on the model; hence, it is useful for regression problems with nonsmooth function surfaces and classification problems with complex decision boundaries. Third, when the properties of smaller classes are of particular interest, a smaller bandwidth parameter would be appropriate, and the resulting matrix would be of higher rank. In the following, we focus on the first two motivations.

2.1. Dependence of matrix rank on kernel parameters. We consider first the bandwidth parameters, and we will show that the matrix rank depends strongly on the parameter. Take the Gaussian kernel $\exp(-\|\mathbf{x}-\mathbf{y}\|^2/h^2)$ as an example. The bandwidth h controls the function smoothness. As h increases, the function becomes more smooth, and, consequently, the matrix numerical rank decreases. Figure 1 constructs a matrix from a real dataset and shows the numerical rank versus h with varying tolerances *tol*. As h increases from 2^{-4} to 2^2, the numerical rank decreases from full (4177) to low (66 with tol $= 10^{-4}$, 28 with tol $= 10^{-3}$, 11 with tol $= 10^{-2}$).

Low-rank matrix approximations are efficient in the large-h regime, and in such a regime, the matrix rank is low. Unfortunately, in the small-h regime, they fall back to models with quadratic complexity. One natural question is whether the situation where a relatively small h is useful occurs in machine learning or whether low-rank methods are sufficient. We answer this question in the following section, where we study kernel classifiers on real datasets and investigate the influence of h on accuracy.

2.2. Optimal kernel bandwidth. We study the optimal bandwidth parameters used in practical situations, and, in particular, we consider kernel classifiers. In



(a) Numerical ranks (b) Singular value decay

FIG. 1. *Numerical ranks of the kernel matrix versus h. The data used are Abalone and are normalized in each dimension. The numerical rank is computed as $\text{argmin}_k(\sigma_k < tol \cdot \sigma_1)$, where $\sigma_1 \geq \sigma_2 \geq, \ldots, \geq \sigma_n$ are the singular values and tol is the tolerance. The plot on the right shows the decay patterns of singular values for varying h.*

TABLE 1

Statistics for classification datasets and their selected *classes. r_i is the median distance to the center for class i; n_i is the number of points in class i.*

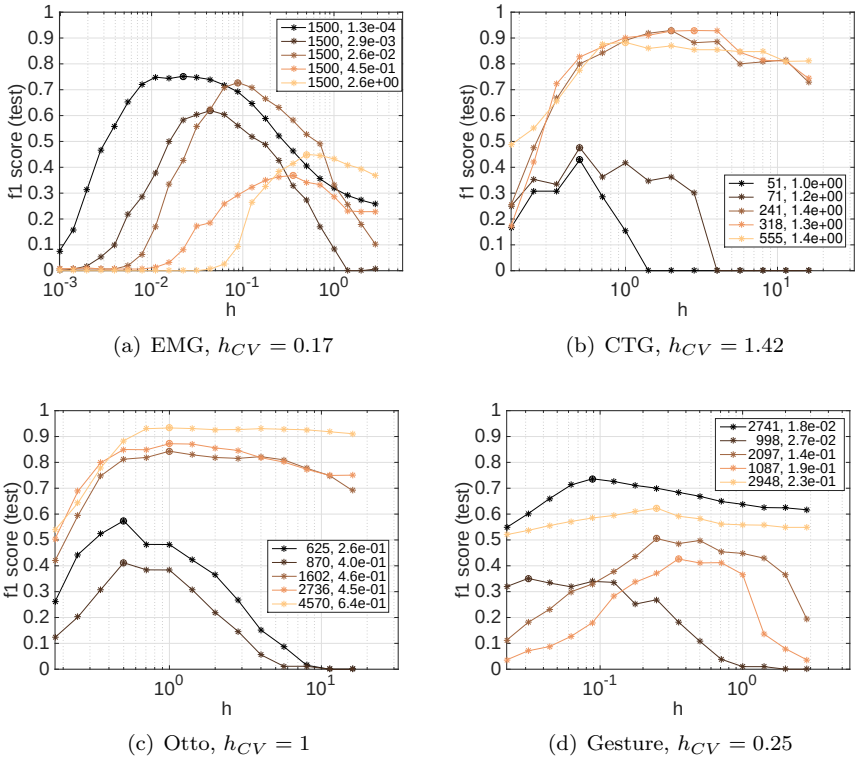
Data	n	d		Selected classes			(Other classes not shown)	
				1	2	3	4	5
EMG	28500	8	n_i	1500	1500	1500	1500	1500
			r_i^2	$\mathbf{1.3 \times 10^{-4}}$	2.9×10^{-3}	2.6×10^{-2}	4.5×10^{-1}	$\mathbf{2.6 \times 10^{0}}$
CTG	2126	23	n_i	**51**	71	241	318	**555**
			r_i^2	1.0	1.2	1.4	1.3	1.4
Gesture	9873	32	n_i	2741	998	2097	1087	2948
			r_i^2	$\mathbf{1.8 \times 10^{-2}}$	2.7×10^{-2}	1.4×10^{-1}	1.9×10^{-1}	$\mathbf{2.3 \times 10^{-1}}$
Otto	20000	93	n_i	**625**	870	1602	2736	**4570**
			r_i^2	0.26	0.40	0.46	0.45	0.64

practice, the parameter h is selected by a cross-validation procedure combined with a grid search, and we denote such parameter as h_{CV}. For datasets with wide intravariability, we observed that the optimal parameters of some small classes turned out to be smaller than h_{CV}. By small classes, we refer to those with fewer points or smaller diameters.

Table 1 lists some classification datasets with wide intravariability. This class imbalance has motivated us to study the individual performance of each class. We found that there can be a significant discrepancy between h_{CV} which is optimal overall for the entire dataset and the optimal h for a specific class. In Figure 2, we use kernel SVM classifier under a wide range of h and measure the performance by F_1 score on the test data. The F_1 score is the harmonic mean of the precision and recall, i.e.,

$$\frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

The data were randomly divided into an 80% training set and a 20% testing set. Figure 2 shows the test F_1 score versus h for selected classes. We see that for some smaller classes represented by darker colors, the F_1 score peaks at a value for h that is smaller than h_{CV}. Specifically, for the smallest class (black curve) of each dataset, as



(a) EMG, $h_{CV} = 0.17$

(b) CTG, $h_{CV} = 1.42$

(c) Otto, $h_{CV} = 1$

(d) Gesture, $h_{CV} = 0.25$

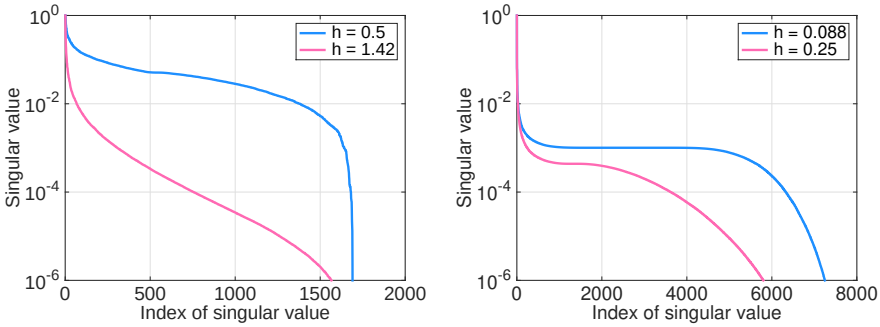
FIG. 2. *Test F_1 score of selected class for different datasets. Each curve represents one class, and the solid circle represents the maximum point along the curve. The legend represents a pair (n_i, r_i^2), where n_i is the number of point in each class, r_i is the median distance to class center, and h_{CV} is the parameter obtained from cross validation. We see that for smaller classes (represented by darker colors), the F_1 score peaks at an h that is smaller than h_{CV}.*

h increases from their own optimal h to h_{CV}, the test F_1 scores drop by 21%, 100%, 16%, and 5% for EMG, CTG, Otto, and Gesture datasets, respectively. To interpret the value of h in terms of matrix rank, we plotted the singular values for different values of h for the CTG and Gesture dataset in Figure 3. We see that when using h_{CV}, the numerical rank is much lower than using a smaller h, which leads to a better performance on smaller classes.

The above observation suggests that the value of h_{CV} is mostly influenced by large classes and using h_{CV} may degrade the performance of smaller classes. Therefore, to improve the prediction accuracy for smaller classes, one way is to reduce the bandwidth h. Unfortunately, a decrease in h increases the rank of the corresponding kernel matrix, making low-rank algorithms inefficient. Moreover, even if we create the model using h_{CV}, as discussed previously, the rank of the kernel matrix will not be very low in most cases. These altogether stress the importance of developing an algorithm that extends the domain of applicability of low-rank algorithms.

2.3. Factors affecting the optimal kernel bandwidth. This section complements the previous section by investigating some data properties that influence the optimal kernel bandwidth parameter h.

We studied synthetic two-dimensional data, and our experiments suggested that the optimal h depends strongly on the *smallest radius of curvature* of the decision



(a) CTG (b) Gesture

FIG. 3. *Singular value decay for CTG dataset and Gesture dataset on selected bandwidths. In subplot* (a), $h_{CV} = 1.42$ *and* $h = 0.5$*, respectively, is the optimal paraemter for the full dataset and the smallest dataset. In subplot* (b), $h_{CV} = 0.25$ *is the optimal parameter for the full dataset, and* $h = 0.088$ *achieves a good* F_1 *score for clusters of small radii.*

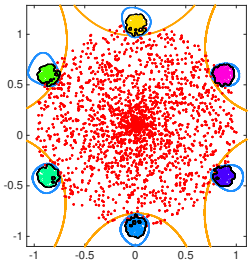
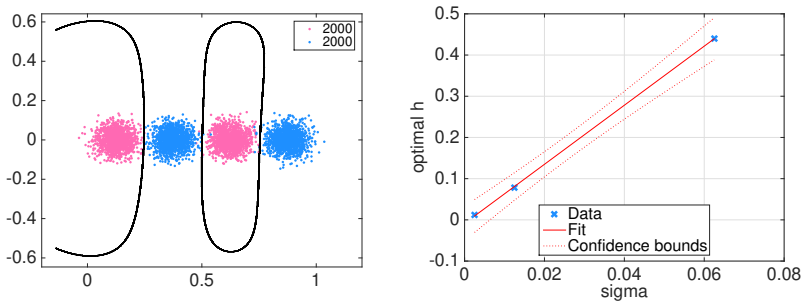


FIG. 4. *Decision boundary with varying smallest radius of curvatures. Dots represent data points, and different classes are color coded. The curves represent the decision boundaries, of which the radii of curvature are large, median, and small for the orange, blue, and black (those surrounding the small clusters) curves, respectively.*

surface (depicted in Figure 4). By optimal, we mean the parameter that yields the highest accuracy, and if multiple such parameters exist, we refer to the largest one to be optimal and denote it as h^*.

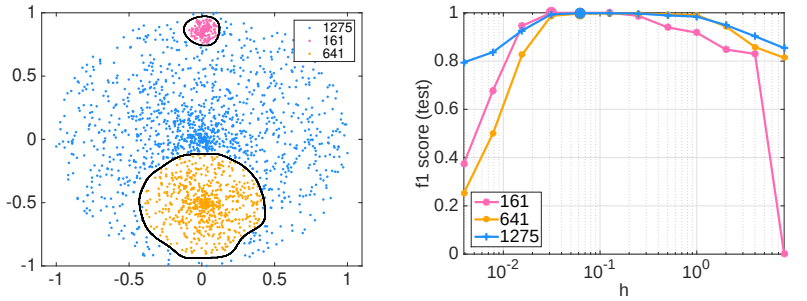
We first experimentally study the relation between h^* and the smallest radius of curvature of the decision boundary. Figure 5 shows Gaussian clusters with alternating labels that are color coded. We decrease the radius of curvature of the decision boundary by decreasing the radius of each cluster while keeping the box size fixed. We quantify the smallest radius of curvature of the decision boundary approximately by the standard deviation σ of each cluster. Figure 5b shows a linear correlation between σ and h^*.

We study a couple more examples. Figure 6 shows two smaller circles with different radii surrounded by a large circle. For this example, the smallest radius of curvature of the decision boundary depends strongly on the cluster radius. Hence, the optimal h for the smaller class (pink colored) should be smaller than that for the larger class (orange colored), which was verified by the F_1 score. Compared to the large cluster, the F_1 score for the small cluster peaks at a smaller h and drops faster as h increases. A similar observation was made in higher-dimensional data as well. We generated two clusters of different radii which are surrounded by a larger cluster



(a) Data with 4 clusters (b) Relation of σ and optimal h

FIG. 5. *Left: data (4 clusters) with two alternating labels (20 and 100 clusters cases are not shown). Each cluster is generated from a Gaussian distribution with standard deviation σ. The decision boundary (black curve) is associated with $h^* = 0.44$. Right: linear relation of the standard deviation σ ($\approx$ half of cluster radius) of each cluster and h^*.*



(a) Data and decision boundary (b) F_1 score for test data

FIG. 6. *Left: classes surrounded by a larger one; the legend shows the number of points in each class. The decision boundary (black curve) is for $h = 0.125$. Right: the test F_1 score for the test data versus h.*

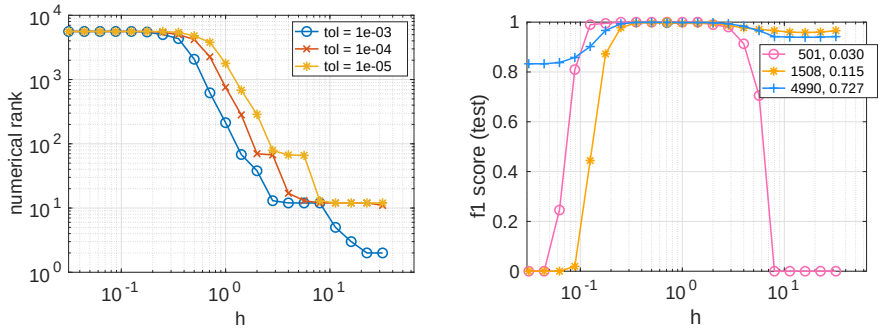
in dimension 10. Figure 7 shows that the intuition in dimension 2 nicely extends to dimension 10. Another cluster example is in Figure 4, which shows multiple small clusters overlapping with a larger cluster at the boundary. The 3 reference decision boundaries correspond to h being 1.5 (orange), 0.2 (blue), and 0.02 (black), respectively. The highest accuracy was achieved at $h = 0.5$, which is close to the small cluster radius 0.2 and is large enough to tolerate the noises in the overlapping region.

The above examples, along with many that are not shown in this paper, have experimentally suggested that the optimal parameter h and the smallest radius of curvature of the decision surface are positively correlated. Hence, for datasets whose decision surfaces are highly nonlinear, i.e., of small radius of curvature, a relatively small h is very likely needed to achieve a high accuracy.

In the following section, we will introduce our novel scheme to accelerate kernel evaluations, which remains efficient in cases where traditional low-rank methods are inefficient.

3. BBF.

In this section, we propose the BBF that extends the availability of traditional low-rank structures. Subsection 3.1 describes the BBF structure. Subsection 3.2 proposes its fast construction algorithm.



(a) Numerical ranks (b) F_1 score for test data

FIG. 7. *Left: numerical ranks of the kernel matrix evaluated on the training data versus h. Right: the F_1 score for the test data versus h. The synthetic data are in dimension 10 and have three clusters of different radii. The legend represents a pair (n_i, r_i), where n_i is the number of points in cluster i and r_i is the cluster radius.*

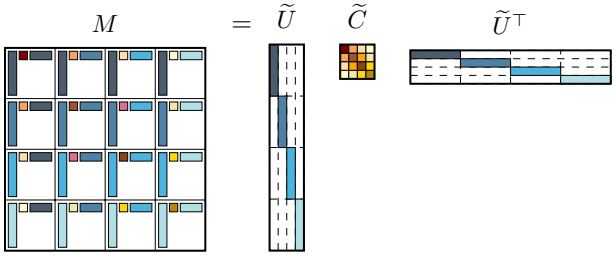


FIG. 8. $M = \widetilde{U}\widetilde{C}\widetilde{U}^\top$.

3.1. BBF structure. This section defines and analyzes the BBF. Given a symmetric matrix $M \in \mathbb{R}^{n\times n}$ partitioned into k by k blocks, let $M_{i,j}$ denote the (i,j)th block for $i,j = 1,\ldots,k$. Then the BBF of M is defined as

$$M = \widetilde{U}\widetilde{C}\widetilde{U}^\top, \tag{1}$$

where $\widetilde{U}$ is a block diagonal matrix with the ith diagonal block U_i being the column basis of $M_{i,j}$ for all j and $\widetilde{C}$ is a k by k block matrix with the (i,j)th block denoted by $C_{i,j} = U_i^\top M_{i,j} U_j$. The BBF structure is depicted in Figure 8.

We discuss the memory cost for the BBF structure. If the numerical ranks of all the base U_i are bounded by r, then the memory cost for the BBF is $O(nr + (rk)^2)$. Further, if $k \le \sqrt{n}$ and r is a constant independent of n, then the BBF gives a data-sparse representation of matrix M. In this case, the complexity for both storing the BBF structure and applying it to a vector will be linear in n.

It is important to distinguish between our BBF and a block low-rank (BLR) structure [2]. There are two main differences: (1) The memory usage of the BBF is much less than BLR. BBF has one basis for all the blocks in the same row, while BLR has a separate basis for each block. The memory for BBF is $nr + (rk)^2$, whereas for BLR it is $2nkr$. (2) It is more challenging to construct BBF in linear complexity while remaining accurate. A direct approach using SVD to construct the low-rank base has a cubic cost, while a simple randomized approach would be inaccurate and unstable.

In the next section, we will propose an efficient method to construct the BBF structure, which uses randomized methods to reduce the cost while still providing a robust approach. Our method is linear in n for many kernels used in machine learning applications.

3.2. Fast construction algorithm for BBF. In this section, we first introduce a theorem in subsubsection 3.2.1 that reveals the motivation behind our BBF structure and addresses the applicable kernel functions. We then propose a fast construction algorithm for BBF in subsubsection 3.2.2.

3.2.1. Motivations. Consider a RBF kernel function $\mathcal{K} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$. The following theorem in [39] provides an upper bound on the error for the low-rank representation of kernel $\mathcal{K}$. The error is expressed in terms of the function smoothness and the diameters of the source domain and the target domain.

THEOREM 3.1. *Consider a function f and kernel $\mathcal{K}(\mathbf{x},\mathbf{y}) = f(\|\mathbf{x}-\mathbf{y}\|_2^2)$ with $\mathbf{x} = (x_1,\ldots,x_d)$ and $\mathbf{y} = (y_1,\ldots,y_d)$. We assume that $x_i \in [0, D/\sqrt{d}]$, $y_i \in [0, D/\sqrt{d}]$, where D is a constant independent of d. This implies that $\|\mathbf{x}-\mathbf{y}\|_2^2 \le D^2$. We assume further that there are $D_{\mathbf{x}} < D$ and $D_{\mathbf{y}} < D$ such that $\|\mathbf{x}_i - \mathbf{x}_j\|_2 \le D_{\mathbf{x}}$ and $\|\mathbf{y}_i - \mathbf{y}_j\|_2 \le D_{\mathbf{y}}$.*

Let $f_p(x) = \sum_n \mathcal{T} \circ f(x + 4nD^2)$ be a $4D^2$-periodic extension of $f(x)$, where $\mathcal{T}(\cdot)$ is 1 on $[-D^2, D^2]$ and smoothly decays to 0 outside of this interval. We assume that f_p and its derivatives through $f_p^{(q-1)}$ are continuous and that the qth derivative is piecewise continuous with its total variation over one period bounded by V_q.

Then $\forall\ M_f, M_t > 0$ with $9M_f \le M_t$, the kernel $\mathcal{K}$ can be approximated in a separable form whose rank is at most $R = R(M_f, M_t, d) = 4M_f\binom{M_t+d}{d}$:

$$\mathcal{K}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{R} g_i(\mathbf{x}) h_i(\mathbf{y}) + \epsilon_{M_f,M_t}.$$

The L_∞ error is bounded by

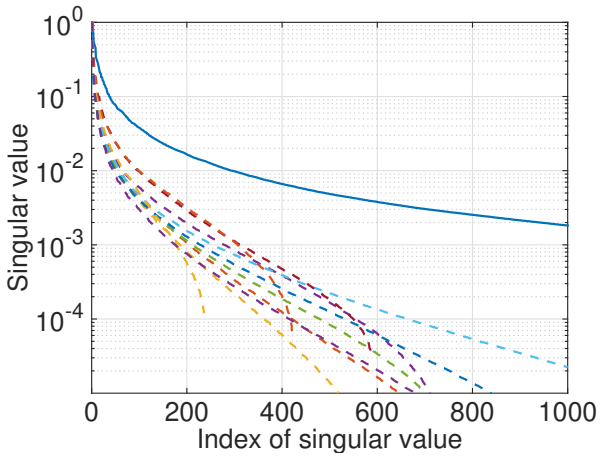
$$|\epsilon_{M_f,M_t}| \le \|f\|_\infty \left(\frac{D_{\mathbf{x}} D_{\mathbf{y}}}{D^2}\right)^{M_t+1} + \frac{V_q}{\pi q}\left(\frac{2D^2}{\pi M_f}\right)^q.$$

In Theorem 3.1, the error is up bounded by the summation of two terms. We first study the second term, which is independent of $D_{\mathbf{x}}$ or $D_{\mathbf{y}}$. The second term depends on the smoothness of the function and decays exponentially as the smoothness of the function increases. Many kernel functions used in machine learning are sufficiently smooth; hence, the second term is usually smaller than the first term. Regarding the first term, the domain diameter information influences the error through the factor $(\frac{D_{\mathbf{x}} D_{\mathbf{y}}}{D^2})^{M_t+1}$, which suggests that for a fixed rank (positively related to M_t), reducing either $D_{\mathbf{x}}$ or $D_{\mathbf{y}}$ reduces the error bound. It also suggests that for a fixed error, reducing either $D_{\mathbf{x}}$ or $D_{\mathbf{y}}$ reduces the rank. This has motivated us to cluster points into distinct clusters of small diameters, and by the theorem, the rank of the submatrix that represents the local interactions from one cluster to the entire dataset would be lower than the rank of the entire matrix.

Hence, we seek linear-complexity clustering algorithms that are able to separate points into clusters of small diameters; k-means and k-centers algorithms are natural choices. Both algorithms partition n data points in dimension d into k clusters at

Cluster	Radius	Size
1	42.4	752
2	41.3	603
3	37.6	729
4	24.3	733
5	24.2	1571
6	23.0	588
7	21.8	1006
8	21.6	421
9	21.4	236
10	21.3	855
Full	62.8	7494

(a)



(b)

FIG. 9. *Left* (a): *clustering result of the pendigits dataset. Right* (b): *normalized singular value decay. In subplot* (b), *the solid curve represents the entire matrix* M, *and the dash curves represent the row-submatrices* $M(C_i,:)$. *The kernel used was the Gaussian kernel with bandwidth parameter* $h = 2$.

a cost of $O(nkd)$ per iteration. Moreover, they are based on the Euclidean distance between points, which is consistent with the RBF kernels which are functions of the Euclidean distance. In practice, the algorithms converge to slightly different clusters due to different objective functions, but neither is absolutely superior. Importantly, the clustering results from these algorithms yield a more memory-efficient BBF structure than random clusters. A more task-specific clustering algorithm will possibly yield better result; however, the main focus of this paper is on factorizing the matrix efficiently rather than proposing new approaches to identify good clusters.

We experimentally verify our motivation on a real-world dataset. We clustered the pendigits dataset into 10 clusters $(C_1, C_2, \ldots, C_{10})$ using the k-means algorithm and reported the statistics of each cluster in Figure 9(a). We see that the radius of each cluster is smaller than that of the full dataset. We further plotted the normalized singular values of the entire matrix M and its submatrices $M(C_i,:)$ in Figure 9(b). Notably, the normalized singular value of the submatrices shows a significantly faster decay than that of the entire matrix. This suggests that the ranks of submatrices are much lower than that of the entire matrix. Hence, by clustering the data into clusters of smaller radius, we are able to capture the local interactions that are missed by the conventional low-rank algorithms which only consider global interactions. As a result, we achieve a similar level of accuracy with a much less memory cost.

3.2.2. BBF construction algorithm. This section proposes a fast construction algorithm for the BBF structure. For simplicity, we assume that the data points are evenly partitioned into k clusters, $\mathcal{C}_1, \ldots, \mathcal{C}_k$ and that the numerical rank for each submatrix is r. We first permute the matrix according to the clusters:

$$M = PKP^\top = \begin{array}{c} \\ \mathcal{C}_1 \\ \mathcal{C}_2 \\ \vdots \\ \mathcal{C}_k \end{array} \begin{array}{c} \begin{array}{cccc} \mathcal{C}_1 & \mathcal{C}_2 & \cdots & \mathcal{C}_k \end{array} \\ \begin{pmatrix} M_{1,1} & M_{1,2} & \cdots & M_{1,k} \\ M_{2,1} & M_{2,2} & \cdots & M_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ M_{k,1} & M_{k,2} & \cdots & M_{k,k} \end{pmatrix} \end{array}, \tag{2}$$

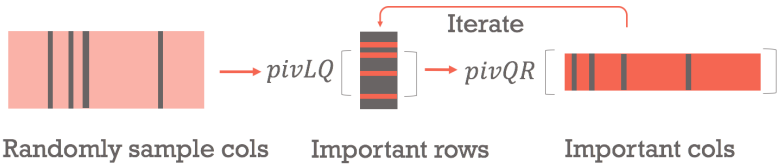


FIG. 10. *A pictorial description of the sampling algorithm. We start with sampling random columns and iterate between important rows (by pivoted LQ) and important columns (by pivoted QR) to obtain our refined important columns. This procedure is usually repeated a few times to ensure the stability of the important indices.*

where P is a permutation matrix and $M_{i,j} = \mathcal{K}(\mathcal{C}_i, \mathcal{C}_j)$ is the interaction matrix between cluster $\mathcal{C}_i$ and cluster $\mathcal{C}_j$.

Our fast construction algorithm consists of two components: basis construction and inner matrix construction. In the following, we adopt MATLAB's notation for submatrices. We use the colon to represent `1:end`, e.g., $M_{i,:} = \begin{pmatrix} M_{i,1} & \cdots & M_{i,k} \end{pmatrix}$, and use the index vectors $\mathcal{I}$ and $\mathcal{J}$ to represent subrows and subcolumns; e.g., $M(\mathcal{I}, \mathcal{J})$ represents the intersection of rows and columns, whose indices are $\mathcal{I}$ and $\mathcal{J}$, respectively.

1. Basis construction

We consider first the basis construction algorithm. The most accurate approach is to explicitly construct the submatrix $M_{i,:}$ and apply an SVD to obtain the column basis; regrettably, it has a cubic cost to compute all the bases. Randomized SVD [27] reduces the cost to quadratic while being accurate; however, a quadratic complexity is still expensive in practice. In the following, we describe a linear algorithm that is accurate and stable. Since the proposed algorithm adopts randomness, by "stable" we mean that the variance of the output is small under multiple runs. The key idea is to restrict us in a subspace by sampling columns of large volume.

The algorithm is composed of two parts. In the first part, we select some columns of $M_{i,:}$ that are representative of the column space. By representative, we mean that the r sampled columns have volume approximating the maximum r-dimensional volume among all column sets of size r. In the second part, we apply the randomized SVD algorithm to the representative columns to extract the column basis.

Part 1: *Randomized sampling algorithm*

We seek a sampling method that samples columns with approximate maximum volume. Strong rank revealing QR (RRQR) [23] returns columns whose volume is proportional to the maximum volume obtained by SVD. QR with column pivoting (pivoted QR) is a practical replacement for the strong RRQR due to its inexpensive computational cost. To ensure a linear complexity, we use the pivoted QR factorization with a randomized approach.

We describe the randomized sampling method [16] used in our BBF algorithm; the algorithm detail is in Algorithm 1 with the procedure depicted in Figure 10. The complexity of sampling r columns from an $m \times n$ matrix is $O(r^2(m+n))$. The size of the output index sets Π_r and Π_c could grow as large as qr, but it can be controlled by some practical algorithmic modifications. One is that given a tolerance, we truncate the top columns based on the magnitudes of the diagonal entries of matrix R from the pivoted QR. Another is to apply an early stopping once the important column index set does not change for two consecutive iterations. For the numerical results reported in this paper, we used $q = 2$. Note that any linear sampling algorithm can

substitute Algorithm 1, and in practice, Algorithm 1 returns columns whose volume is very close to the largest.

Algorithm 1: Randomized sampling algorithm for each submatrix.

Function *Randomized_Sampling*($M_{i,:}$, r_i, q)
- **input :** (1) Row-submatrix $M_{i,:}$ to sample from in its implicit form (given data and kernel function); (2) Sample size r_i; (3) Iteration parameter q
- **output:** Important column index set Π_c for $M_{i,:}$
- $\Pi_r = \emptyset$
- **for** *iter*=1, ..., q **do**
  - *Important columns*. Uniformly sample r_i rows, and denote the index set as Γ_r, update $\Pi_r = \Pi_r \cup \Gamma_r$. Apply a pivoted QR factorization on $M_{i,:}(\Pi_r, :)$ to get the important columns index set, denoted as Π_c.
  - *Important rows*. Uniformly sample r_i columns, and denote the index set as Γ_c. Update $\Pi_c = \Gamma_c \cup \Pi_c$. Apply a pivoted LQ factorization on $M_{i,:}(:, \Pi_c)$ to get the important row index set, denoted as Π_r.
- **end**
- **return** Π_c

Note: The pivoted QR is the QR factorization with column pivoting based on the largest column norm.

Algorithm 2: BBF sampling algorithm.

Function *BBF_Sampling*($\{M_{i,:}\}_{i=1}^k$, $\{r_i\}_{i=1}^k$, q)
- **input :** (1) Submatrices $\{M_{i,:}\}_{i=1}^k$ to sample from in their implicit forms (given data and kernel function); (2) Sample sizes $\{r_i\}_{i=1}^k$ for each submatrix $M_{i,:}$; (3) Iteration parameter q
- **output:** Important column index set Π_i for each row-submatrix
- **for** $i = 1, \ldots, k$ **do**
  - Π_i = Randomized_Sampling($M_{i,:}(:, \Gamma), r_i, q$) (using Algorithm 1)
- **end**
- **return** Π_i for $i = 1, \ldots, k$

Applying Algorithm 1 to k submatrices $\{M_{i,:}\}_{i=1}^k$ will return the desired k sets of important columns for BBF, which is described in Algorithm 2. The complexity of Algorithm 2 depends on k (see subsubsection 3.2.3 for details), and we can remove this dependence by applying Algorithm 1 on a preselected and refined set of columns instead of all the columns. This leads to a more efficient procedure to sample columns for the k submatrices as described in Algorithm 3. Our final BBF construction algorithm will use Algorithm 2 for column sampling.

Part 2: *Orthogonalization algorithm*

Having sampled the representative columns $M_{i,:}(:, \Pi_i)$, the next step is to obtain the column basis that approximates the span of the selected columns. This can be achieved through any orthogonalization methods, e.g., pivoted QR, SVD, randomized SVD [27], etc. According to Algorithm 1, the size of the sampled index set Π_c can be as large as qr. In practice, we found that the randomized SVD works efficiently. The randomized SVD algorithm was proposed to reduce the cost of computing a rank-r approximation of an $m \times n$ matrix to $O(mnr)$. The algorithm is described in

Algorithm 3: More efficient BBF sampling algorithm.

Function *BBF_Sampling_I*($\{M_{i,:}\}_{i=1}^k$, $\{r_i\}_{i=1}^k$, q)
input : (1) Submatrices $\{M_{i,:}\}_{i=1}^k$ to sample from in their implicit forms (given data and kernel function); (2) Sample sizes $\{r_i\}_{i=1}^k$ for each submatrix $M_{i,:}$; (3) Iteration parameter q
output: Important column index set Π_i for each row-submatrix
for $i = 1, \ldots, k$ **do**
  Randomly sample r_i columns from M_i and denote the index set as Π_i
  Apply a pivot LQ on $M_{i,:}(:, \Pi_i)$ to obtain r important rows, and we denote the index set as Γ_i
end
Stack all the sampled rows $\Gamma = [\Gamma_1, \ldots, \Gamma_k]$
for $i = 1, \ldots, k$ **do**
  Π_i = Randomized_Sampling($M_{i,:}(:, \Gamma), r_i, q$) (using Algorithm 1)
end
return Π_i for $i = 1, \ldots, k$

Algorithm 4: Randomized SVD.

Function *Randomized_SVD*(M, r, q)
input : (1) Matrix $M \in \mathbb{R}^{m \times n}$; (2) desired rank r; (3) iteration parameter q
output: U, Σ, and V such that $M \approx U\Sigma V^\top$
Randomly generate a Gaussian matrix $\Omega \in \mathbb{R}^{n \times r}$
$M\Omega = QR$
for $i = 1, \ldots, q$ **do**
  $M^\top Q = \hat{Q}\hat{R}$
  $M\hat{Q} = QR$
end
$\hat{U}\Sigma V^\top = Q^\top M$
$U = Q\hat{U}$
return U, Σ, V

Algorithm 4. The practical implementation of Algorithm 4 involves an oversampling parameter ℓ to reduce the iteration parameter q. For simplicity, we eliminate ℓ from the pseudocode.

2. Inner matrix construction

We then consider the inner matrix construction. Given column base U_i and U_j, we seek a matrix $C_{i,j}$ such that it minimizes

$$\|M_{i,j} - U_i C_{i,j} U_j^\top\|.$$

The minimizer is given by $C_{i,j} = U_i^\dagger M_{i,j} (U_j^\top)^\dagger$. Computing $C_{i,j}$ exactly has a quadratic cost. Again, we restrict ourselves in a subspace and propose a sampling-based approach that is efficient yet accurate. The following proposition provides a key theoretical insight behind our algorithm.

PROPOSITION 3.2. *If a matrix $M \in \mathbb{R}^{m \times n}$ can be written as $M = UCV^\top$, where $U \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{n \times r}$. Further, if for some index set $\mathcal{I}$ and $\mathcal{J}$, $U(\mathcal{I}, :)$ and $V(\mathcal{J}, :)$*

are full rank, then the inner matrix C is given by

$$C = (U(\mathcal{I},:))^{\dagger}\ M(\mathcal{I},\mathcal{J})\ (V(\mathcal{J},:)^{\top})^{\dagger}, \tag{3}$$

where $\dagger$ denotes the pseudoinverse of the matrix.

Proof. To simplify the notations, we denote $\widehat{U} = U(\mathcal{I},:)$, $\widehat{V} = V(\mathcal{J},:)$, and $\widehat{M} = M(\mathcal{I},\mathcal{J})$, where $\mathcal{I}$ is the sampled row index set for U and $\mathcal{J}$ is the sampled row index set for V. We apply the sampling matrices $P_{\mathcal{I}}$ and $P_{\mathcal{J}}$ (matrices of 0s and 1s) to both sides of equation $M = UCV^{\top}$ and obtain

$$P_{\mathcal{I}} M P_{\mathcal{J}}^{\top} = P_{\mathcal{I}} U C V^{\top} P_{\mathcal{J}}^{\top},$$

i.e.,

$$\widehat{M} = \widehat{U} C \widehat{V}^{\top}.$$

The assumption that $\widehat{U}$ and $\widehat{V}$ are tall and skinny matrices with full column ranks implies that $\widehat{U}^{\dagger}\widehat{U} = I$ and $\widehat{V}^{\top}(\widehat{V}^{\top})^{\dagger} = I$. We then multiply $\widehat{U}^{\dagger}$ and $(\widehat{V}^{\top})^{\dagger}$ on both sides and obtain the desired result:

$$\widehat{U}^{\dagger}\widehat{M}(\widehat{V}^{\top})^{\dagger} = \widehat{U}^{\dagger}\widehat{U}C\widehat{V}^{\top}(\widehat{V}^{\top})^{\dagger} = C. \qquad \square$$

Proposition 3.2 provides insights into an efficient, stable, and accurate construction of the inner matrix. In practice, the equality $M = UCV^{\top}$ in Proposition 3.2 often holds with an error term, and we seek index sets $\mathcal{I}$ and $\mathcal{J}$ such that the computation for C is accurate and numerically stable. Equation 3 suggests that a good choice leads to an $M(\mathcal{I},\mathcal{J})$ with a large volume. However, finding such a set can be computationally expensive, and a heuristic is required for efficiency. We used a simplified approach where we sample $\mathcal{I}$ (resp., $\mathcal{J}$) such that $U(\mathcal{I},:)$ (resp., $V(\mathcal{J},:)$) has a large volume. This leads to good numerical stability because having a large volume is equivalent to being nearly orthogonal, which implies a good condition number. In principle, a pivoted QR strategy could be used, but fortunately we are able to skip it by using the results from the basis construction. Recall that in the basis construction, the important rows were sampled using a pivoted LQ factorization; hence, they already have large volumes.

Therefore, the inner matrix construction is described in what follows. We first uniformly sample r column indices Γ_j and r row indices Γ_i, respectively, from $\mathcal{C}_j$ and $\mathcal{C}_i$. Then the index sets are constructed as $\mathcal{I} = \Pi_i \cup \Gamma_i$ and $\mathcal{J} = \Pi_j \cup \Gamma_j$, where Π_i and Π_j are the important row index sets from the basis construction. Finally, $C_{i,j}$ is given by

$$(U_i(\mathcal{I},:))^{\dagger}\ M_{i,j}(\mathcal{I},\mathcal{J})\ (U_j(\mathcal{J},:)^{\top})^{\dagger}.$$

We also observed small entries in some off-diagonal blocks of the inner matrix. Those blocks normally represent far-range interactions. We can set the blocks for which the norm is below a preset threshold to 0. In this way, the dense inner matrix becomes a blockwise sparse matrix, further reducing the memory.

Having discussed the details for the construction algorithm, we summarize the procedure in Algorithm 5, which is the algorithm used for all the numerical results.

In this section, for simplicity, we only present BBF for symmetric kernel matrices. However, the extension to general nonsymmetric cases is straightforward by applying similar ideas, and the computational cost will be roughly doubled. Asymmetric BBF can be useful in compressing the kernel matrix in the testing phase.

Algorithm 5: Main Algorithm—Fast construction algorithm for BBF.

Function *BBF_Construction* (k, $\{\mathcal{C}_i\}_{i=1}^k$, $\{r_i\}_{i=1}^k$, M, q_{Samp}, q_{SVD})

Input : (1) Number of clusters k; (2) Clustering assignments $\{\mathcal{C}_i\}_{i=1}^k$; (3) Rank $\{r_i\}_{i=1}^k$ for each column basis; (4) Matrix M in its implicit form; (5) Iteration parameter q_{Samp} for randomized sampling; (6) Iteration parameter q_{SVD} for randomized SVD.

Output: Block diagonal matrix $\widetilde{U}$ and blockwise sparse matrix $\widetilde{C}$ s.t. $M \approx \widetilde{U}\widetilde{C}\widetilde{U}^\top$

$[\Pi_1, \ldots, \Pi_k] = \text{BBF\_Sampling}(\{M_{i,:}\}_{i=1}^k, \{r_i\}_{i=1}^k, q_{\text{Samp}})$ (Algorithm 2)

for $i = 1, \ldots, k$ **do**

  $U_i = \text{Randomized\_SVD}(M_{i:}(:, \Pi_i), r_i, q_{\text{SVD}})$ (Algorithm 4)

end

for $i = 1, \ldots, k$ **do**

  for $j = 1, \ldots, i$ **do**

    if *cutoff criterion is not satisfied* **then**

      Uniformly sample Γ_i and Γ_j from $\mathcal{C}_i$ and $\mathcal{C}_j$, respectively

      $\mathcal{I} = \Pi_i \cup \Gamma_i$ and $\mathcal{J} = \Pi_j \cup \Gamma_j$

      $C_{i,j} = (U_i(\mathcal{I}, :))^\dagger M_{i,j}(\mathcal{I}, \mathcal{J})(U_j(\mathcal{J}, :)^\top)^\dagger$

      $C_{j,i} = C_{i,j}^\top$

    else

      $C_{i,j} = 0$

      $C_{j,i} = 0$

    end

  end

end

return $\tilde{U}$, $\tilde{C}$

3. Precomputation: Parameter selection

We present a heuristic algorithm to identify input parameters for BBF. The algorithm takes n input points $\{\mathbf{x}_i\}_{i=1}^n$ and a requested error (tolerance) ϵ and outputs the suggested parameters for the BBF construction algorithm, specifically the number of clusters k, the index set for each cluster $\mathcal{I}$, and the estimated rank r_i for the submatrix corresponding to the cluster $\mathcal{I}$. We seek a set of parameters that minimizes the memory cost while keeping the approximation error below ϵ.

Choice of column ranks. Given the tolerance ϵ and the number of clusters k, we describe our method of identifying the column ranks. To maintain a low cost, the key idea is to consider only the diagonal blocks instead of the entire row-submatrices. For each row-submatrix in the RBF kernel matrices (after permutation), the diagonal block, which represents the interactions within a cluster, usually has a slower spectral decay than that of off-diagonal blocks, which represent the interactions between clusters. Hence, we minimize the input rank for the diagonal block and use this as the rank for those off-diagonal blocks in the same row.

Specifically, we denote $\sigma_{1,i} \geq \sigma_{2,i} \geq \cdots \geq \sigma_{n_i,i}$ as the singular values for $M_{i,i}$. Then for block $M_{i,i} \in \mathbb{R}^{n_i \times n_i}$, the rank r_i is chosen as

$$r_i = \min\left\{ m \;\middle|\; \sum_{p=m+1}^{n_i} \sigma_{p,i}^2 < \frac{n_i^2}{n^2} \|M_{i,i}\|_F^2 \, \epsilon^2 \right\}.$$

Choice of number of clusters k. Given the tolerance ϵ, we consider the number of clusters k. For k clusters, the upper bound on the memory usage of BBF is $\sum_{i=1}^{k} n_i r_i + (\sum_{i=1}^{k} r_i)^2$, where r_i is computed as described above. Hence, the optimal k is the solution to the following optimization problem:

$$\begin{aligned} &\text{minimize}_k \;\; g(k) = \sum_{i=1}^{k} n_i r_i + \left(\sum_{i=1}^{k} r_i \right)^2 \\ &\text{subj to} \;\; r_i = \min \left\{ m \;\middle|\; \sum_{p=m+1}^{n_i} \sigma_p^2 < \frac{n_i^2}{n^2} \|M_{i,i}\|_F^2 \; \epsilon^2 \right\} \;\; \forall i. \end{aligned}$$

We observed empirically that in general, $g(k)$ is close to convex in the interval $[1, O(\sqrt{n})]$, which enables us to perform a dichotomy search algorithm with complexity $O(\log n)$ for the minimal point.

3.2.3. Complexity analysis. In this section, we analyze the algorithm complexity. We will provide detailed analysis on the factorization step, including the basis construction and the inner matrix construction, and skip the analysis for the precomputation step. We first introduce some notations.

Notations. Let k denote the number of clusters, $\{n_i\}_{i=1}^{k}$ denote the number of point in each cluster, $\{r_i\}_{i=1}^{k}$ denote the requested rank for the blocks in the ith submatrix, and l denote the oversampling parameter.

Basis construction. The cost comes from two parts: the column sampling and the randomized SVD. We first calculate the cost for the ith row-submatrix. For the column sampling, the cost is $n_i(r_i + l)^2 + n(r_i + l)^2$, where the first term comes from the pivoted LQ factorization and the second term comes from the pivoted QR factorization. For the randomized SVD, the cost is $n_i(r_i + l)^2$. Summing up the costs from all the submatrices, we obtain the overall complexity

$$O\left(\sum_{i=1}^{k} n_i(r_i + l)^2 + n(r_i + l)^2 + n_i(r_i + l)^2 \right).$$

We simplify the result by denoting the maximum numerical rank of all blocks as r. Then the above complexity is simplified to $O(nkr^2)$.

Inner matrix construction. The cost for computing inner matrix $C_{i,j}$ with sampled $M_{i,j}$, U_i, and U_j is $r_i^2 r_j + r_i r_j^2$. Summing over all the k^2 blocks, the overall complexity is given by

$$\sum_{i=1}^{k} \sum_{j=1}^{k} r_i^2 r_j + r_i r_j^2.$$

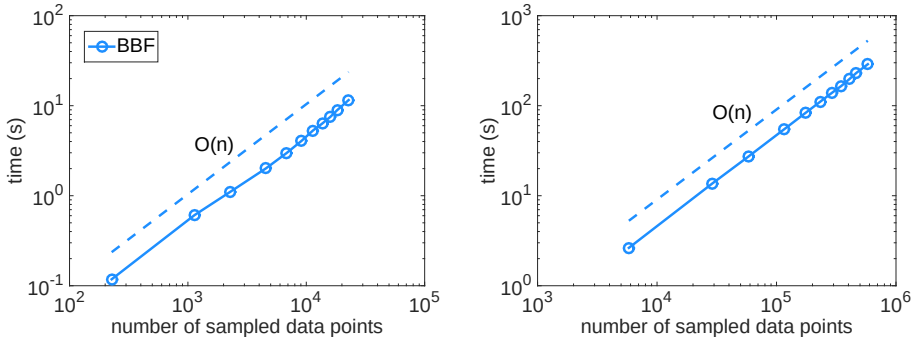
With the same assumptions as above, the simplified complexity is $O(k^2 r^3)$. Note that k can reach up to $O(\sqrt{n})$ while still maintaining a linear complexity for this step.

Finally, we summarize the complexity of our algorithm in Table 2.

From Table 2, we note that the factorization and application cost (storage) depend quadratically on the number of clusters k. This suggests that a large k will spoil the linearity of the algorithm. However, this may not be the case for most machine learning kernels, and we will discuss the influence of k on three types of kernel matrices: (1) well-approximated by a low-rank matrix, (2) full-rank but approximately sparse, and (3) full-rank and dense:

TABLE 2
Complexity table; n is the total number of points, k is the number of clusters, and r is the numerical rank for column basis.

Precomputation		Factorization		Application
Each block	$O(n_i r^2)$	Basis	$O(nkr^2)$	$O(nr + (rk)^2)$
Compute $g(k)$	$O(nr^2)$	Inner matrix	$O(k^2 r^3)$	
Total	$O(r^2 n \log n)$	Total	$O(nkr^2 + k^2 r^3)$	



(a) Census housing, $h = 2$ (b) Forest covertype, $h = 2$

FIG. 11. *Factorization time (log-log scale) for kernel matrices from real datasets. To illustrate the linear growth of the complexity, we generated datasets with a varying number of points with the following strategy. We first clustered the data into 15 groups and sampled a portion p% from each group, then increased p. To avoid the influence from other factors on the timing, we fixed the input rank for each block. As we can see, the timing grows linearly with the data size (matrix size).*

1. *Well-approximated by a low-rank matrix*. When the kernel matrix is well-approximated by a low-rank matrix, kr is up bounded by a constant (up to the approximation accuracy). In this case, both the factorization and application costs are linear.
2. *Full-rank but approximately sparse*. When the kernel matrix is full-rank ($kr = O(n)$) but approximately sparse, the application cost (storage) remains linear due to the sparsity. By sparsity, we mean that as h decreases, the entries in the off-diagonal blocks of the inner matrices become small enough that setting them to 0 does not cause much accuracy loss. The factorization cost, however, becomes quadratic when using Algorithm 2. One solution is to use Algorithm 3 for column sampling, which removes the dependence on k, assuming $k < O(\sqrt{n})$.
3. *Full-rank and dense*. In this case, BBF would be suboptimal. However, we experimentally observed that many kernel matrices generated by RBF functions with high-dimensional data are in case 1 or 2.

In the end, we empirically verify the linear complexity of our method. Figure 11 shows the factorization time (in seconds) versus the number of data points on some real datasets. The trend is linear, confirming the linear complexity of our algorithm.

4. Experimental results. In this section, we experimentally verify the advantages of the BBF structure in subsection 4.1 and the BBF algorithm in subsection 4.2. By BBF algorithm, we refer to the BBF structure and the proposed fast construction algorithm.

TABLE 3
Real datasets used in the experiments.

Dataset	Abalone	Mushroom	Cpusmall
# Instance	4177	8124	8192
# Attributes	8	112	16
Dataset	Pendigits	Census house	Forest covertype
# Instance	10992	22748	581012
# Attributes	11	16	54

The datasets are listed in Tables 3 and 1, and they can be downloaded from the UCI repository [6], the libsvm website [9], and Kaggle. All the data were normalized such that each dimension has mean 0 and standard deviation 1. All the experiments were performed on a computer with a 2.4 GHz CPU and 8 GB memory.

4.1. BBF structure. In this section, we will experimentally analyze the key factors in our BBF structure that contribute to its advantages over competing methods. Many factors contribute, and we will focus our discussions on the following two: (1) The BBF structure has its column base constructed from the entire row-submatrix, which is an inherently more accurate representation than from diagonal blocks only (see MEKA), and (2) the BBF structure considers local interactions instead of only global interactions used by a low-rank scheme.

4.1.1. Basis from the row-submatrix versus diagonal blocks. We verify that computing the column basis from the entire row-submatrix $M_{i,:}$ is generally more accurate than from the diagonal blocks $M_{i,i}$ only. Column basis computed from the diagonal blocks only preserves the column space information in the diagonal blocks and will be less accurate in approximating the off-diagonal blocks Figure 12 shows that computing the basis from the entire row-submatrix is more accurate.

4.1.2. BBF structure versus low-rank structure. We compare the BBF structure and the low-rank structure. The BBF structure refers to Figure 8, and the low-rank structure means $K \approx UU^\top$, where U is a tall and skinny matrix. For a fair comparison, we fixed all the factors to be the same except for the structure. For example, for both the BBF and the low-rank schemes, we used the same sampling method for the column selection and computed the inner matrices exactly to avoid randomness introduced in that step. The columns for BBF and low-rank scheme, respectively, were sampled from each row-submatrix $M_{i,:} \in R^{n_i \times n}$ and the entire matrix $M \in R^{n \times n}$. For BBF with leverage-score sampling, we sampled columns of $M_{i,:}$ based on its column leverage scores computed from the algorithm in [14].

Figure 13 shows the relative error versus the memory cost for different sampling methods. The relative error is computed by $\frac{\|\hat{K}-K\|_F}{\|K\|_F}$, where $\hat{K}$ is the approximated kernel matrix, K is the exact kernel matrix, and $\|\cdot\|_F$ denotes the Frobenius norm. As can be seen, the BBF structure is strictly a generalization of the low-rank scheme and achieves lower approximation error regardless of the sampling method used. Moreover, for most sampling methods, the BBF structure outperforms the best low-rank approximation computed by an SVD, which strongly implies that the BBF structure is favorable.

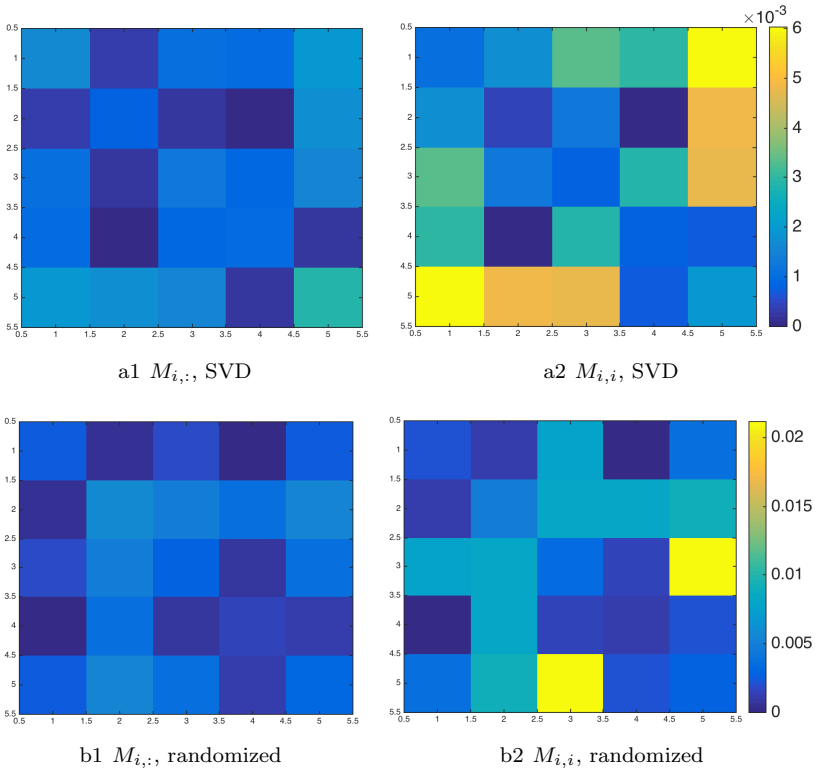
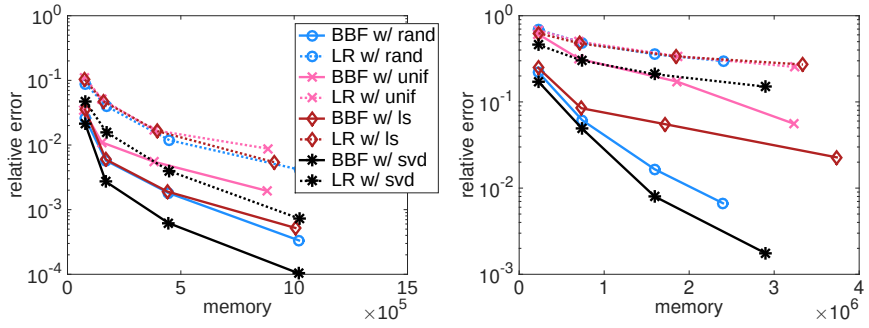


FIG. 12. *Errors for each block in the approximated matrix from the Abalone dataset. Warmer color represents larger error. Subplots* (a1) *and* (a2) *share the same colorbar, and* (b1) *and* (b2) *share the same colorbar. The error for block* (i, j) *is computed as* $\|M_{i,j} - \widehat{M}_{i,j}\|_F / \|M\|_F$, *where* $\widehat{M}_{i,j}$ *is the approximation of* $M_{i,j}$. *The basis in subplots* (a1) *and* (a2) *is computed by an SVD and in* (b1) *and* (b2) *by the randomized sampling algorithm. As we can see, computing the column basis from the diagonal blocks leads to lower error in the diagonal blocks; however, the errors in the off-diagonal blocks are much larger. The relative error in subplots* (a1), (a2), (b1), *and* (b2) *are* 1.4×10^{-3}, 6.3×10^{-3}, 1.5×10^{-2}, *and* 4.0×10^{-2}, *respectively.*



(a) Abalone, $h = 2$ (b) Pendigits, $h = 2$

FIG. 13. *Kernel approximation error versus memory cost for BBF and low-rank structure with different sampling methods. Gaussian kernel is used. The results are averaged over* 5 *runs. BBF (solid lines) uses the structure described in Figure* 8, *and LR (dash lines) uses a low-rank structure. "rand": randomized sampling; "unif": uniform sampling; "ls": leverage score sampling; "svd": an SVD is used for computing the basis.*

4.2. BBF algorithm. In this section, we experimentally evaluate the performance of our BBF algorithm with other state-of-the-art kernel approximation methods. Subsections 4.2.1 and 4.2.2 examine the matrix reconstruction error under varying memory budget and kernel bandwidth parameters. Subsubsection 4.2.3 applies the approximations to the kernel ridge regression problem. Finally, subsubsection 4.2.4 compares the linear complexity of BBF with the IFGT [41]. Throughout the experiments, we use BBF to denote our algorithm, whose input parameters are computed from our precomputation algorithm.

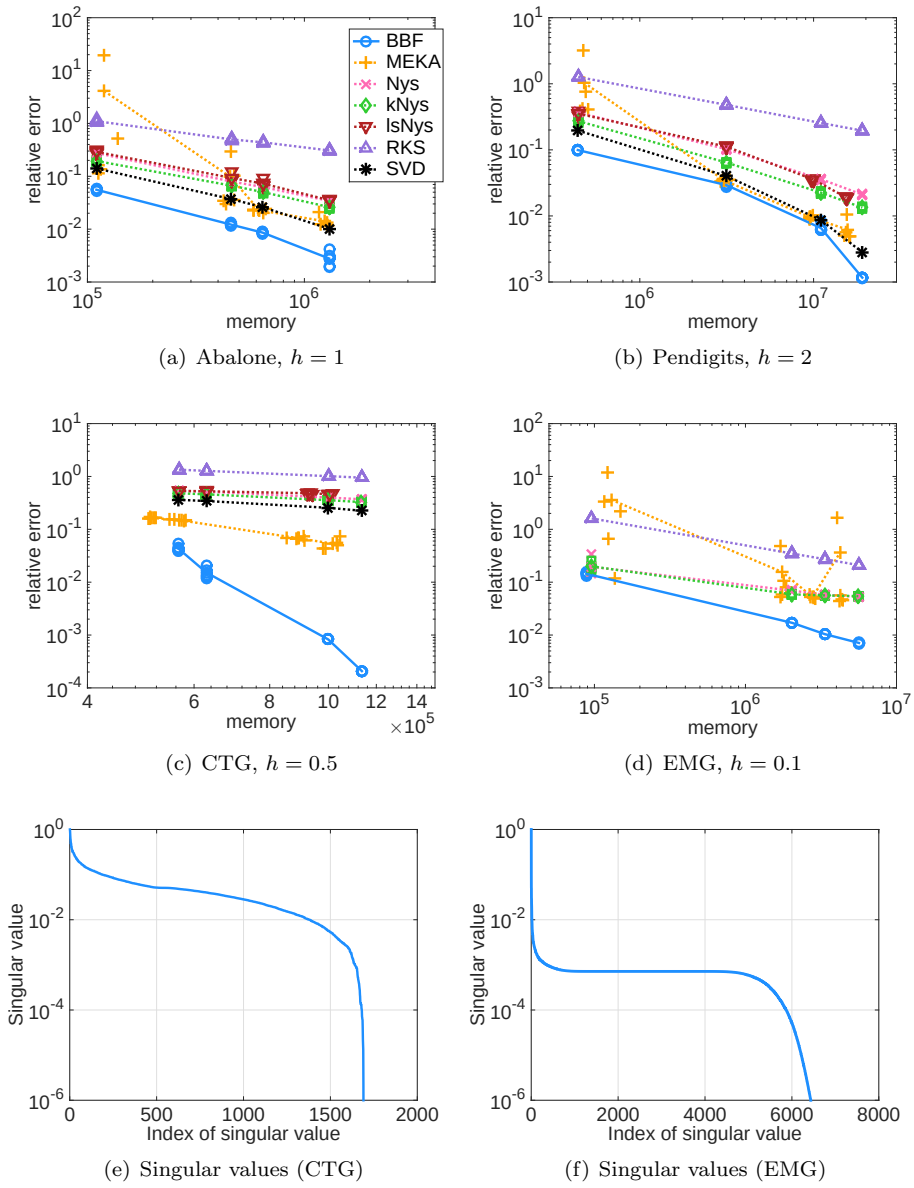
In what follows, we briefly introduce some implementation and input parameter details for the methods we are comparing to:

- *The naïve Nyström (Nys).* We uniformly sampled $2k$ columns without replacement for a rank k approximation.
- *k-means Nyström (kNys).* It uses k-means clustering and sets the centroids to be the landmark points. We used the code provided by the author.
- *Leverage score Nyström (lsNys).* It samples columns with probabilities proportional to the statistical leverage scores. We calculated the approximated leverage scores [14] and sampled $2k$ columns with replacement for a rank-k approximation.
- *MEKA.* We used the code provided by the author.
- *Random kitchen sinks (RKS).* We used our own MATLAB implementation based on their algorithm.
- *IFGT.* We used the `C++` code provided by the author.

4.2.1. Approximation with varying memory budget. We consider the reconstruction errors from different methods when the memory cost varies. The memory cost (storage) is also a close approximation of the running time for a matrix-vector multiplication. In addition, computing memory is more accurate than running time, which is sensitive to the implementation and algorithmic details. In our experiments, we indirectly increased the memory cost by requesting a lower tolerance in BBF. The memories for all the methods were fixed to be roughly the same in the following way. For low-rank methods, the input rank was set to be the memory of BBF divided by the matrix size. For MEKA, the input number of clusters was set to be the same as BBF; the "eta" parameter (the percentage of blocks to be set to zeros) was also set to be similar as BBF.

Figures 14 and 15 show the reconstruction error versus memory cost on real datasets and two-dimensional synthetic datasets, respectively. We see that BBF achieves comparable and often significantly lower error than the competing methods regardless of the memory cost. There are two observations worth noting. First, the BBF outperforms the exact SVD, which is the best rank-r approximation, and it outperforms with a factorization complexity that is only linear rather than cubic. This has demonstrated the superiority of the BBF structure over the low-rank structure. Second, even when compared to a similar structure as MEKA, BBF achieves a lower error whose variance is also smaller, and it achieves so with a similar factorization complexity. These have verified that the representation of BBF is more accurate and that the constructing algorithm is more stable.

4.2.2. Approximation with varying kernel bandwidth parameters. We consider the reconstruction errors with varying decay patterns of singular values, which we achieve by choosing a wide range of kernel bandwidth parameters. The memory for all methods are fixed to be roughly the same.



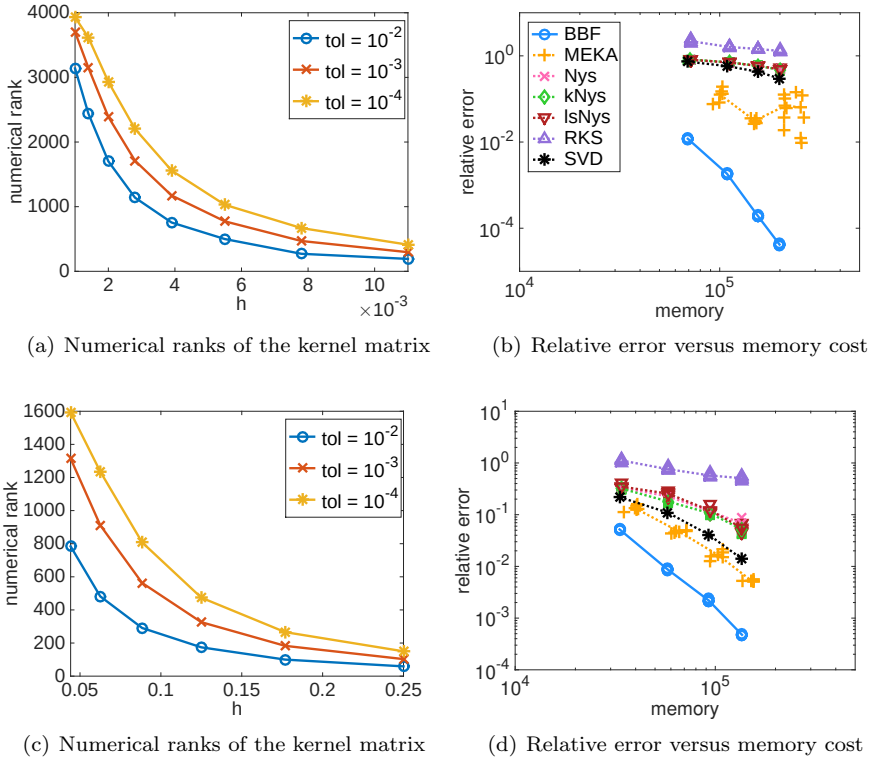
(a) Abalone, $h = 1$ (b) Pendigits, $h = 2$

(c) CTG, $h = 0.5$ (d) EMG, $h = 0.1$

(e) Singular values (CTG) (f) Singular values (EMG)

FIG. 14. *Comparisons of BBF (our algorithm) with competing methods. Top four plots (log-log scale) share the same legend. For each method, we show the improvement in error as more memory is available. For each memory footprint, we report the error of* 5 *runs of each algorithm. Each run is shown with a marker, while the lines represent the average error. For CTG and EMG datasets, the parameter* h *was chosen to achieve higher* F_1 *score on smaller classes, which leads to matrices with higher ranks, as shown by the plateau or slow decay of singular values in the bottom plots subplots* (e) *and* (f).

The plots on the left of Figure 16 show the average matrix reconstruction error versus $1/h^2$. We see that for all the low-rank methods, the error increases when h decreases. When h becomes smaller, the kernel function becomes less smooth, and consequently the matrix rank increases. This relation between h and the matrix rank



(a) Numerical ranks of the kernel matrix (b) Relative error versus memory cost

(c) Numerical ranks of the kernel matrix (d) Relative error versus memory cost

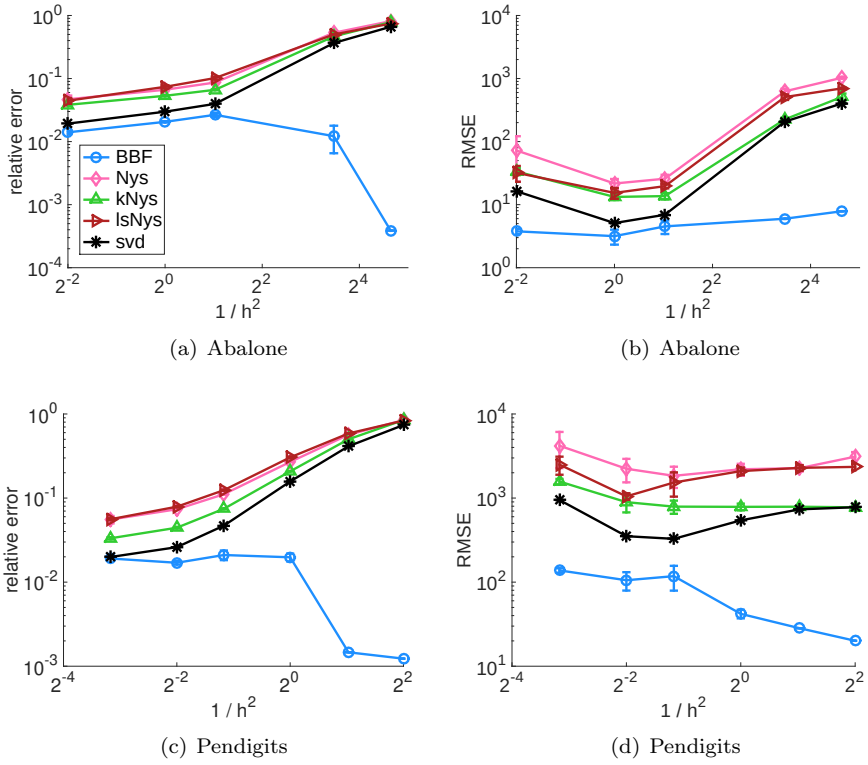
FIG. 15. *The left plots report the numerical ranks of matrices versus h and the right plots report the relative error versus memory. The data for top plots have alternating labels (see Figure* 5 *with* 100 *clusters), while the data for the bottom plots have smaller clusters surrounded by larger ones (see Figure* 6*). The values of h reported in subplot* (a) *and* (c) *yield test accuracy greater than* 0.99*. The h in* (b) *and* (d) *are the largest optimal h with values* 0.0127 *and* 0.25*, respectively. For each memory cost, we report the relative error of* 5 *runs of each algorithm. The number of clusters for BBF was fixed at* 20 *for subplot* (b) *and selected automatically for subplot* (d).

is revealed in some statistics listed in Table 4. The results in the table are consistent with the results shown in [18] for varying kernel bandwidth parameters.

In the large-h regime, the gap in error between BBF and other methods is small. In such a regime, the matrix is low-rank, and the low-rank algorithms work effectively. Hence, the difference in error is not significant. In the small-h regime, the gap starts to increase. In this regime, the matrix becomes close to diagonal dominant, and the low-rank structure, as a global structure, cannot efficiently capture the information along the diagonal, while for BBF, the precomputation procedure will increase the number of clusters to better approximate the diagonal part, and the off-diagonal blocks can be set to 0 due to their small entries. By efficiently using the memory, BBF is favorable in all cases, from low-rank to nearly diagonal.

4.2.3. Kernel ridge regression. We consider the kernel ridge regression. The standard optimization problem for the kernel ridge regression is

$$\min_{\boldsymbol{\alpha}} \|K\boldsymbol{\alpha} - \mathbf{y}\|^2 + \lambda\|\boldsymbol{\alpha}\|^2, \tag{4}$$



(a) Abalone (b) Abalone

(c) Pendigits (d) Pendigits

FIG. 16. *Plots for relative error vers $1/h^2$ for different kernel approximation methods. The memory costs for all methods and all kernel parameters are fixed to be roughly the same. The kernel function used is the Gaussian kernel. As the rank of the matrix increases (h decreases), the gap in error between low-rank approximation methods and BBF increases.*

TABLE 4

Summary statistics for abalone and pendigits datasets with the Gaussian kernel, where r is the rank, M is the exact matrix, and M_r is the best rank-r approximation for M. $\lceil \frac{\|M\|_F^2}{\|M\|_2^2} \rceil$ is referred to as the stable rank and is an underestimate of the rank; l_r represents the rth largest leverage score scaled by $\frac{n}{r}$.

Abalone ($r = 100$)				Pendigits ($r = 252$)			
$\frac{1}{h^2}$	$\lceil \frac{\|M\|_F^2}{\|M\|_2^2} \rceil$	$100\frac{\|M_r\|_F}{\|M\|_F}$	l_r	$\frac{1}{h^2}$	$\lceil \frac{\|M\|_F^2}{\|M\|_2^2} \rceil$	$100\frac{\|M_r\|_F}{\|M\|_F}$	l_r
0.25	2	99.99	4.34	0.1	3	99.99	2.39
1	4	99.86	2.03	0.25	6	99.79	1.83
4	5	97.33	1.94	0.44	8	98.98	1.72
25	15	72.00	5.20	1	12	93.64	2.02
100	175	33.40	12.60	2	33	77.63	2.90
400	931	19.47	20.66	4	207	49.60	4.86
1000	1155	16.52	20.88	25	2794	19.85	14.78

where K is a kernel matrix, $\mathbf{y}$ is the target, and $\lambda > 0$ is the regularization parameter. The minimizer is given by the solution of the following linear system:

$$(K + \lambda I)\boldsymbol{\alpha} = \mathbf{y}. \tag{5}$$

The linear system can be solved by an iterative solver, e.g., MINRES [33], and the complexity is $O(n^2T)$, where n^2 is from matrix-vector multiplications and T denotes the iteration number. If we can approximate K by $\widehat{K}$, which can be represented in lower memory, then the solving time can be accelerated. This is because the memory is a close approximation for the running time of a matrix-vector multiplication. We could also solve the approximated system directly when the matrix can be well-approximated by a low-rank matrix; that is, we compute the inversion of $\widehat{K}$ first by the Woodbury formula[1] and then apply the inversion to $\mathbf{y}$.

In the experiments, we approximated K by $\widehat{K}$ and solved the following approximated system with MINRES:

$$(\widehat{K} + \lambda I)\boldsymbol{\alpha} = \mathbf{y}. \tag{6}$$

The dataset was randomly divided into training set (80%) and testing set (20%). The kernel used is the Laplacian kernel $K(\mathbf{x}, \mathbf{y}) = \exp(\|\mathbf{x} - \mathbf{y}\|/h)$ for this subsection. We report the test root-mean-square error (RMSE), which is defined as

$$\sqrt{\frac{1}{n_{\text{test}}}\|K_{\text{test}}\hat{\boldsymbol{\alpha}} - \mathbf{y}_{\text{test}}\|_F^2}, \tag{7}$$

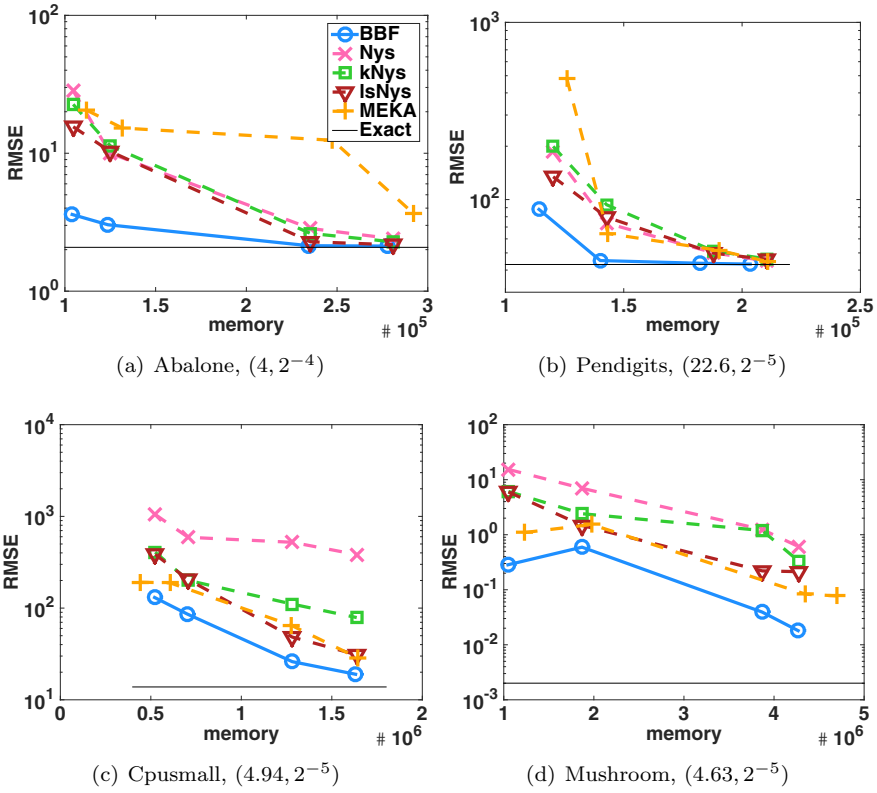
where K_{test} is the interaction matrix between the test data and training data, $\hat{\boldsymbol{\alpha}}$ is the solution from solving (6), and $\mathbf{y}_{\text{test}}$ is the true test target. Figure 17 shows the test RMSE with varying memory cost of the approximation. We see that with the same memory footprint, the BBF achieves lower test error.

Discussion. For downstream prediction tasks, better generalization error could be achieved by using the surrogate kernel, which is the kernel matrix between the testing points and landmark points, instead of the exact kernel matrix for naïve Nyström, k-means Nyström, leverage score Nyström, and random kitchen sink. Based on our experience, using surrogate kernels with Nyström methods and random Fourier methods achieves testing accuracy competitive with that of BBF. Hence, although BBF significantly outperforms Nyström methods and random Fourier methods in the approximation of kernel matrices, the advantage of BBF in prediction compared with surrogate kernels is less pronounced.

Meanwhile, an easy modification of BBF can be used to construct a surrogate kernel for downstream predictions as well. Specifically, for U_i, we can set U_i as the carefully sampled important columns with points denoted as X_i instead of the column basis of those sampled columns. This further reduces our factorization cost due to the removal of the orthonormalization step. Once these important columns are available, the middle matrix C can be constructed identical to that in Algorithm 5. These steps construct the modified BBF, which can be used to accelerate the linear system solve of (6) and obtain $\boldsymbol{\alpha}$ efficiently. Then the coefficient for the surrogate kernel is computed as $\widetilde{\boldsymbol{\alpha}} = CU^{\dagger}\boldsymbol{\alpha}$. We denote $\widetilde{\boldsymbol{\alpha}}_i$ as the coefficient for cluster i. The downstream prediction task, then, is divided into two steps. First, for a testing point $\mathbf{x}_{test}$, we find the cluster i that $\mathbf{x}_{test}$ belongs to. Second, we compute the predictions y_{test} as $K(\mathbf{x}_{test}, X_i)\widetilde{\boldsymbol{\alpha}_i}$.

With this modified BBF and the corresponding prediction procedure, assuming a surrogate kernel of the same size is used, it will be more efficient to compute the coefficients of the surrogate kernel as well as the predictions through BBF than through Nyström methods or random Fourier methods.

[1] https://en.wikipedia.org/wiki/Woodbury_matrix_identity.



(a) Abalone, $(4, 2^{-4})$ (b) Pendigits, $(22.6, 2^{-5})$

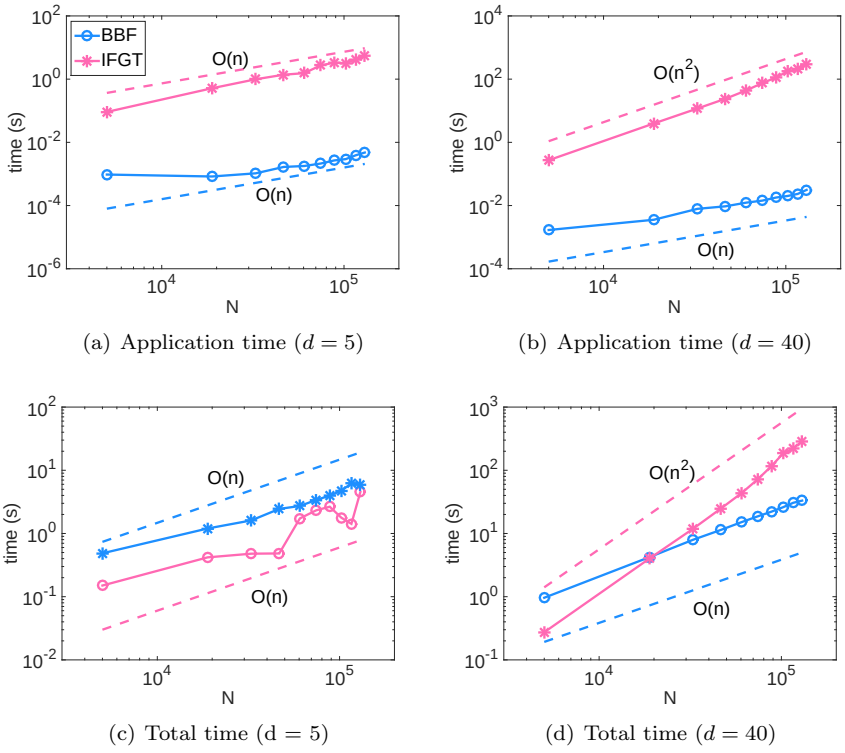
(c) Cpusmall, $(4.94, 2^{-5})$ (d) Mushroom, $(4.63, 2^{-5})$

FIG. 17. *Test RMSE versus memory for kernel ridge regression. For each memory cost, we report the results averaged over* 5 *runs. The black line on the bottom of each plot represents the test RMSE when using the exact matrix. The kernel parameter* h *and the regularization parameter* λ *were selected by a* 5*-fold cross validation with a grid search, and the selected* (h, λ) *pairs are listed in the subcaption.*

4.2.4. Comparison with IFGT. We benchmarked the linear complexity of the IFGT [41] and BBF. IFGT was proposed to alleviate the dimensionality issue for the Gaussian kernel. For a fixed dimension d, the IFGT has a linear complexity in terms of application time and memory cost; regrettably, when d increases (e.g., $d \geq 10$), the algorithm requires a large number of data points n to make this linear behavior visible. BBF, on the other hand, does not require a large n to observe a linear growth.

We verify the influence of dimension d on the complexity of BBF and the IFGT on synthetic datasets. We fixed the tolerance to be 10^{-3} throughout the experiments. Figure 18 shows the time versus the number of points. We focus only on the trend of time instead of the absolute value because the IFGT was implemented in `C++`, while BBF was in MATLAB. We see that the growth rate of IFGT is linear when $d = 5$ but falls back to quadratic when $d = 40$; the growth rate of BBF, however, remains linear.

5. Conclusions and future work. In this paper, we observed that for classification datasets whose decision boundaries are complex, i.e., of small radius of curvature, a small bandwidth parameter is needed for a high prediction accuracy. In practical datasets, this complex decision boundary occurs frequently when there exist a large variability in class sizes or radii. These small bandwidths result in ker-



(a) Application time ($d = 5$) (b) Application time ($d = 40$)

(c) Total time (d = 5) (d) Total time ($d = 40$)

FIG. 18. *Timing (loglog scale) for IFGT and BBF on a synthetic dataset with dimensions* $d = 5$ *and* 40. *We generated* 10 *centers uniformly at random in a unit cube, and around each center we randomly generated data with standard deviation* 0.1 *along each dimension. The tolerance and the kernel parameter* h *were set to* 10^{-3}, *and* 0.5, *respectively. All the plots share the same legends. The top plots show the application time (matrix-vector product), and bottom plots show the total time (factorization and application). The timing for BBF is linear for all dimensions, while the timing for the IFGT falls back to quadratic when* d *increases.*

nel matrices whose ranks are not low, and hence traditional low-rank methods are no longer efficient. Moreover, for many machine-learning applications, low-rank approximations of dense kernel matrices are inefficient. Hence, we are interested in extending the domain of availability of low-rank methods and retain computational efficiency. Specifically, we proposed a structured low-rank–based algorithm that is of linear memory cost and floating point operations and that remains accurate even when the kernel bandwidth parameter is small, i.e., when the matrix rank is not low. We experimentally demonstrated that the algorithm works in fact for a wide range of kernel parameters. Our algorithm achieves comparable and often orders-of-magnitude-higher accuracy than other state-of-the-art kernel approximation methods, with the same memory cost. It also produces errors with smaller variance, thanks to the sophisticated randomized algorithm. This is in contrast with other randomized methods whose error fluctuates much more significantly. Applying our algorithm to the kernel ridge regression also demonstrates that our method competes favorably with the state-of-the-art approximation methods.

There are a couple of future directions. One direction is on the efficiency and performance of the downstream inference tasks. The focus of this paper is on the approximation of the kernel matrix itself. Although the experimental results have

demonstrated good performance in real-world regression tasks, we could further improve the downstream tasks by relaxing the orthogonality constrains of the U_i matrices. That is, we can generate U_i from interpolative decomposition [11], which allows us to share the same advantages as algorithms using Nys and RKS. Another direction is on the evaluation metric for the kernel matrix approximation. This paper used the conventional Frobenius norm to measure the approximation performance. Zhang et al. [43] proposed a new metric that better measures the downstream performance. The new metric suggests that to achieve a good generalization performance, it is important to have a high-rank approximation. This suggestion aligns well with the design of BBF and further emphasizes its advantage. Evaluating BBF under the new metric will be explored in the future. Last but not least, the BBF construction did not consider the regularization parameter used in many learning algorithms. We believe that the regularization parameter could facilitate the low-rank compression of the kernel matrix in our BBF, while the strategy requires further exploration.

REFERENCES

[1] A. E. Alaoui and M. W. Mahoney, *Fast randomized kernel ridge regression with statistical guarantees*, in Proceedings of the 28th International Conference on Neural Information Processing Systems—Volume 1, NIPS'15, MIT Press, Cambridge, MA, 2015, pp. 775–783, http://dl.acm.org/citation.cfm?id=2969239.2969326.

[2] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J. L'Excellent, and C. Weisbecker, *Improving multifrontal methods by means of block low-rank representations*, SIAM J. Sci. Comput., 37 (2015), pp. A1451–A1474, https://doi.org/10.1137/120903476.

[3] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker, *Improving multifrontal methods by means of block low-rank representations*, SIAM J. Sci. Comput., 37 (2015), pp. A1451–A1474.

[4] A. Aminfar, S. Ambikasaran, and E. Darve, *A fast block low-rank dense solver with applications to finite-element matrices*, J. Comput. Phys., 304 (2016), pp. 170–188.

[5] F. Bach, *Sharp analysis of low-rank kernel matrix approximations*, in Proceedings of the 26th Annual Conference on Learning Theory, 2013, pp. 185–209.

[6] K. Bache and M. Lichman, *UCI Machine Learning Repository*, http://archive.ics.uci.edu/ml (2013).

[7] J. Bouvrie and B. Hamzi, *Kernel methods for the approximation of nonlinear systems*, SIAM J. Control Optim., 55 (2017), pp. 2460–2492, https://doi.org/10.1137/14096815X.

[8] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[9] C.-C. Chang and C.-J. Lin, *LIBSVM: A library for support vector machines*, ACM Trans. Intell. Syst. Technol., 2 (2011), 27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html.

[10] J. Chen, H. Avron, and V. Sindhwani, *Hierarchically compositional kernels for scalable nonparametric learning*, J. Mach. Learn. Res., 18 (2017), pp. 2214–2255, https://doi.org/10.5555/3122009.3176810.

[11] H. Cheng, Z. Gimbutas, P.-G. Martinsson, and V. Rokhlin, *On the compression of low rank matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1389–1404, https://doi.org/10.1137/030602678.

[12] E. Darve, *The fast multipole method* I: *Error analysis and asymptotic complexity*, SIAM J. Numer. Anal., 38 (2000), pp. 98–128.

[13] E. Darve, *The fast multipole method: Numerical implementation*, J. Comput. Phys., 160 (2000), pp. 195–240.

[14] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff, *Fast approximation of matrix coherence and statistical leverage*, J. Mach. Learn. Res., 13 (2012), pp. 3475–3506.

[15] P. Drineas and M. W. Mahoney, *On the Nyström method for approximating a gram matrix for improved kernel-based learning*, J. Mach. Learn. Res., 6 (2005), pp. 2153–2175.

[16] B. Engquist, L. Ying, et al., *A fast directional algorithm for high frequency acoustic scattering in two dimensions*, Commun. Math. Sci., 7 (2009), pp. 327–345.

[17] W. Fong and E. Darve, *The black-box fast multipole method*, J. Comput. Phys., 228 (2009), pp. 8712–8725.

[18] A. Gittens and M. W. Mahoney, *Revisiting the Nyström method for improved large-scale machine learning*, J. Mach. Learn. Res., 17 (2016), pp. 3977–4041.
[19] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Vol. 3, John Hopkins University Press, Baltimore, MD, 2012.
[20] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.
[21] L. Greengard and V. Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, Acta Numer., 6 (1997), pp. 229–269.
[22] L. Greengard and J. Strain, *The fast Gauss transform*, SIAM J. Sci. Stat. Comput., 12 (1991), pp. 79–94, https://doi.org/10.1137/0912004.
[23] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.
[24] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part* I*: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.
[25] W. Hackbusch and S. Börm, *Data-sparse approximation by adaptive $\mathcal{H}^2$ matrices*, Computing, 69 (2002), pp. 1–35.
[26] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic*, Computing, 64 (2000), pp. 21–47.
[27] N. Halko, P.-G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.
[28] M. R. Hestenes and E. Stiefel, *Methods of Conjugate Gradients for Solving Linear Systems*, Vol. 49, NBS, Washington, DC, 1952.
[29] Y. Li, H. Yang, E. R. Martin, K. L. Ho, and L. Ying, *Butterfly factorization*, Multiscale Model. Simul., 13 (2015), pp. 714–732.
[30] Y. Li, H. Yang, and L. Ying, *Multidimensional butterfly factorization*, Appl. Comput. Harmon. Anal., 44 (2018), pp. 737–758, https://doi.org/10.1016/j.acha.2017.04.002.
[31] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert, *Randomized algorithms for the low-rank approximation of matrices*, Proc. Natl. Acad. Sci., 104 (2007), pp. 20167–20172.
[32] M. W. Mahoney, *Randomized algorithms for matrices and data*, Found. Trends Mach. Learn., 3 (2011), pp. 123–224.
[33] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629.
[34] T. Sarlos, *Improved approximation algorithms for large matrices via random projections*, in 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), IEEE, New York, 2006, pp. 143–152.
[35] B. Savas, I. S. Dhillon, et al., *Clustered low rank approximation of graphs in information science applications*, in SIAM International Conference on Data Mining, (SDM), SIAM, Philadelphia, 2011, pp. 164–175.
[36] S. Si, C.-J. Hsieh, and I. S. Dhillon, *Memory efficient kernel approximation*, J. Mach. Learn., 18 (2017), pp. 682–713.
[37] M. L. Stein, *Limitations on low rank approximations for covariance matrices of spatial data*, Spat. Stat., 8 (2014), pp. 1–19, https://doi.org/10.1016/j.spasta.2013.06.003.
[38] M. L. Stein, *Limitations on low rank approximations for covariance matrices of spatial data*, Spat. Stat., 8 (2014), pp. 1–19.
[39] R. Wang, Y. Li, and E. Darve, *On the numerical rank of radial basis function kernels in high dimensions*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 1810–1835, https://doi.org/10.1137/17M1135803.
[40] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.
[41] C. Yang, R. Duraiswami, N. A. Gumerov, and L. Davis, *Improved fast Gauss transform and efficient kernel density estimation*, in Proceedings of the Ninth IEEE International Conference on Computer Vision, 2003, IEEE, New York, 2003, pp. 664–671.
[42] L. Ying, G. Biros, and D. Zorin, *A kernel-independent adaptive fast multipole algorithm in two and three dimensions*, J. Comput. Phys., 196 (2004), pp. 591–626.
[43] J. Zhang, A. May, T. Dao, and C. Ré, *Low-Precision Random Fourier Features for Memory-Constrained Kernel Approximation*, http://arxiv.org/abs/1811.00155, 2019.
[44] K. Zhang and J. T. Kwok, *Clustered Nyström method for large scale manifold learning and dimension reduction*, IEEE Trans. Neural Netw., 21 (2010), pp. 1576–1587.