Block Basis Factorization for Scalable Kernel Evaluation

Ruoxi Wang ∗ Yingzhou Li † Michael W. Mahoney ‡ 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 intra-class 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 (RBF) kernel matrices. Our approach has linear memory cost and floating point operations. 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-art kernel approximation algorithms.

1 Introduction

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

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

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.

Kernel matrix is an essential part in most kernel methods and is defined in what follows. Given $n$ data points $\{x_i\}_{i=1}^n$, the $(i,j)$-th entry in a kernel matrix is $K_{ij} = K(x_i, x_j)$. For example, the solution to a kernel ridge regression is the same as the solution to the linear system

$$(K + \delta I)\alpha = y.$$

Regrettably, any operations involving kernel matrices can be computationally expensive. Their construction, application and storage complexities are quadratic in 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 [15] and $O(n^2T)$ for iterative solvers [23, 27], 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 [15, 26, 22, 25, 28, 11, 14, 10, 36]; however, the success of any low-rank algorithm depends on a large spectrum gap or fast spectrum decay of the matrix.

∗Institute for Computational and Mathematical Engineering, Stanford University, Email: ruoxi@stanford.edu
†Department of Mathematics, Duke University, Email: yingzhou.li@duke.edu
‡International Computer Science Institute and Department of Statistics, University of California, Berkeley, Email: mmahoney@stat.berkeley.edu
§Department of Mechanical Engineering, Stanford University, Email: darve@stanford.edu
The motivation for our algorithm is that in many machine learning applications, the RBF kernel matrices cannot be well-approximated by low-rank approximations. The RBF kernel $f(||x - y||/h)$ has a bandwidth parameter $h$ that controls the size of the neighborhood, i.e., how many nearby points to consider 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. In many machine learning applications, the optimal bandwidth parameter is relatively small due to the high nonlinearity of the underlying function. The hypothesis function to quadratic complexity. In many machine learning applications, the optimal bandwidth parameter is relatively small due to the high nonlinearity of the underlying function. The hypothesis function classes for kernel regressions and kernelized classifiers are $\sum_i \alpha_i K(x, x_i)$ and $\sum_i \alpha_i y_i K(x, x_i)$, respectively. Both can be viewed as interpolations on the training data points. In kernel regression, when the underlying function is non-smooth such as those with sharp local changes, using a large bandwidth will smooth out the local structures. Similar arguments apply to 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. The situations where relatively small bandwidths are needed are very common, and 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 motivated 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, hierarchical algorithms [16, 17, 8, 9, 13, 35] efficiently 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) [19, 21, 20, 6, 33, 3, 24] 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 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 propose a block basis factorization (BBF) structure that generalizes the traditional low-rank matrix. Given the same memory budget, the BBF structure achieves significantly higher accuracy than low-rank matrices. Along with the structure, we also propose a construction algorithm with linear cost in both floating-point operations and storage.

The key of our structure is realizing that in most machine learning applications, the sub-matrices representing the interactions from one cluster to the entire data set are numerically low-rank. For example, Wang [32] mathematically proved that if the diameter of a cluster $C$ is smaller than that of the entire dataset $X$, then the rank of the sub-matrix $K(C, X)$ is lower than the rank of the entire matrix $K(X, X)$. 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 sub-matrix $K(C, X)$ 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. There are many other types of structured matrices for which the naïve low-rank approximation may not be satisfactory. Examples include, but are not limited to, covariance matrices from spatial data [31], and frontal matrices in the multi-frontal method for sparse matrix factorizations [2].

1.1 Main Contributions

Our main contribution is two-fold. 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 developed a novel matrix approximation algorithm that extends the applicability of low-rank methods to matrices whose ranks are higher, that produces errors with small variance (the algorithm uses randomized steps), and that has linear, i.e., $O(n)$ complexity. 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 Block Basis Factorization (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 linear, accurate and stable. 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 random sampling algorithms which are linear but often inaccurate or unstable. We also provided a fast pre-computation algorithm to search for near-optimal input parameters for BBF.

Our algorithm involves three major steps. First, it divides the data into \( k \) distinct clusters, 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. Last, it uses the corresponding column- and row- basis to compress each of the \( k^2 \) blocks using a set of only \( k \) bases. The resulting framework yields a rank-\( r \) approximation with \( O(nr/k + r^2) \) storage. This should be contrasted with a low-rank scheme that gives a rank-\( r \) approximation using \( O(nr) \) memory. BBF achieves the same rank 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 [15]. Given a matrix \( K \in \mathbb{R}^{n \times n} \), a rank-\( r \) approximation of \( K \) is given by \( K \approx UV^T \) 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 [28, 25, 22, 26] have reduced the cost to \( O(n^2r) \) 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 [11, 14, 4], 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 [12, 36, 10, 14, 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) [29] performs a block-wise low-rank approximation of the kernel matrix from social network data with quadratic construction complexity. The memory efficient kernel approximation (MEKA) [30] 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.

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 linear-complexity construction algorithm. Finally, Section 4 presents our experimental results, which show the advantages of our proposed BBF over the state-of-arts in terms of the structure, algorithm and applications to kernel regression problems.
2 Motivation

In this section, we discuss the motivations behind designing an algorithm that remains computationally efficient when the matrix rank increases. Two 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 non-smooth function surfaces and classification problems with complex decision boundaries.

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 \( \exp\left(-\frac{\|x - y\|^2}{h^2}\right) \) 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 \( \text{tol} \). As \( h \) increases from \( 2^{-4} \) to \( 2^2 \), the numerical rank decreases from full (4177) to low (66 with \( \text{tol} = 10^{-4} \), 28 with \( \text{tol} = 10^{-3} \), 11 with \( \text{tol} = 10^{-2} \)).

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

Low-rank matrix approximations are efficient in the large-\( h \) regime, and in such 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 occur 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 practice, the parameter \( h \) is selected by a cross-validation procedure combined with a grid search, and we denote such parameter as \( h_{\text{CV}} \). For datasets with wide intra-variability, we observed that the optimal parameters of some small classes turned out to be smaller than \( h_{\text{CV}} \). By small classes, we refer to those with fewer points or smaller diameters.

Table 1 lists some classification datasets with wide intra-variability. This class imbalance has motivated us to study the individual performance of each class. We found that there can be a significant discrepancy between \( h_{\text{CV}} \) which is optimal overall for the entire dataset and the optimal \( h \) for a specific class. In Figure 2, we used 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 was randomly divided into 80% training set and 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 \(h\) 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.

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.

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 \(i\).

<table>
<thead>
<tr>
<th>Data</th>
<th>(n)</th>
<th>(d)</th>
<th>Selected Classes</th>
<th>(\text{(other classes not shown)})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(n_i)</td>
<td>(r_i^2)</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>EMG</td>
<td>28,500</td>
<td>8</td>
<td>1,500</td>
<td>1,500</td>
</tr>
<tr>
<td></td>
<td>(r_i^2)</td>
<td>1.3 (\times) (10^{-4})</td>
<td>2.9 (\times) (10^{-3})</td>
<td>2.6 (\times) (10^{-2})</td>
</tr>
<tr>
<td>CTG</td>
<td>2,126</td>
<td>23</td>
<td>51</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>(r_i^2)</td>
<td>1.0</td>
<td>1.2</td>
<td>1.4</td>
</tr>
<tr>
<td>Gesture</td>
<td>9,873</td>
<td>32</td>
<td>2.741</td>
<td>998</td>
</tr>
<tr>
<td></td>
<td>(r_i^2)</td>
<td>1.8 (\times) (10^{-2})</td>
<td>2.7 (\times) (10^{-2})</td>
<td>1.4 (\times) (10^{-1})</td>
</tr>
<tr>
<td>Otto</td>
<td>20,000</td>
<td>93</td>
<td>625</td>
<td>870</td>
</tr>
<tr>
<td></td>
<td>(r_i^2)</td>
<td>0.26</td>
<td>0.40</td>
<td>0.46</td>
</tr>
</tbody>
</table>

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 2D data and our experiments suggested that the optimal \(h\) depends strongly on the smallest radius of curvature of the decision surface (depicted in Figure 3). 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 establish the relation between \(h^*\) and the smallest radius of curvature of the decision boundary. Figure 4 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 \(\sigma\) of each cluster. Figure 4b shows a linear correlation between \(\sigma\) and \(h^*\).

We study another two examples. Figure 5 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. Figure 3 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.
Figure 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. $h_{CV}$ 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 $h_{CV}$.

Figure 3: 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.
Figure 4: 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 $\sigma$. The decision boundary (black curve) is associated with $h^* = 0.44$. Right: Linear relation of the standard deviation $\sigma$ ($\approx$ half of cluster radius) of each cluster and $h^*$.

Figure 5: 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 $h$. 
The above examples, along with many that are not shown in this paper, have experimentally established the linear relation between the optimal parameter $h$ and the smallest radius of curvature of the decision surface. This suggests that for datasets whose decision surfaces are highly nonlinear, i.e., of small radius of curvature, a relatively small $h$ is 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 Block Basis Factorization

In this section, we propose the block basis factorization (BBF) that extends the availability of traditional low-rank structures. Section 3.1 describes the BBF structure. Section 3.2 proposes its fast construction algorithm.

3.1 BBF Structure

This section defines and analyzes the Block Basis Factorization (BBF). Given a 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:

$$M = \tilde{U} \tilde{C} \tilde{V}^T,$$

where $\tilde{U}$ is a block diagonal matrix with the $i$-th diagonal block $U_i$ being the column basis of $M_{i,j}$ for all $j$, $\tilde{V}$ is a block diagonal matrix with the $j$-th diagonal block $V_j$ being the row basis of $M_{i,j}$ for all $i$, and $\tilde{C}$ is a $k$ by $k$ block matrix with the $(i,j)$-th block denoted by $C_{i,j} = U_i^T M_{i,j} V_j$. The BBF structure is depicted in Figure 6.

![Figure 6: $M = \tilde{U} \tilde{C} \tilde{V}^T$](image)

We discuss the memory cost for the BBF structure. If the numerical ranks of all the base are bounded by $r$, then the memory cost for the BBF is $O(nr + (rk)^2)$. Further, if $k \leq \sqrt{n}$ 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.

It is important to distinguish between our BBF and a block low-rank (BLR) structure. 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 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 $O(n)$ method, which uses randomized methods to reduce the cost while still providing a robust approach.

3.2 Fast construction algorithm for BBF

In this section, we first introduce a theorem in Section 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 Section 3.2.2.
3.2.1 Motivations

Consider a RBF kernel function $K : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$. The following theorem in [32] provides an upper bound on the error for the low-rank representation of kernel $K$. 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 $K(x, y) = f(||x - y||^2)$ with $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$. 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 $||x - y||^2 \leq D^2$. We assume further that there are $D_x < D$ and $D_y < D$, such that $||x_i - x_j||_2 \leq D_x$ and $||y_j - y_j||_2 \leq D_y$.

Let $f_p(x) = \sum_{s} T \circ f(x + 4\pi D^2)$ be a $4D^2$-periodic extension of $f(x)$, where $T(\cdot)$ 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 the $q$-th derivative is piecewise continuous with its total variation over one period bounded by $V_q$.

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

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

The $L_\infty$ error is bounded by

$$|\epsilon_{M_f, M_t}| \leq ||f||_\infty \left(\frac{D_x D_y}{D^2}\right)^{M_t+1} + \frac{V_q}{\pi q} \left(\frac{2D^2}{M_f}\right)^q.$$

In Theorem 3.1, the domain diameter information influences the error through the factor $\left(\frac{D_x D_y}{D^2}\right)^{M_t+1}$, which suggests for a fixed rank (positively related to $M_t$), reducing either $D_x$ or $D_y$ reduces the error bound. It also suggests that for a fixed error, reducing either $D_x$ or $D_y$ 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 a cost of $O(nkd)$. 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.

3.2.2 BBF Construction Algorithm

This section proposes a fast construction algorithm for the BBF structure. For simplicity, we assume the data points are evenly partitioned into $k$ clusters, $C_1, \ldots, C_k$, and the numerical rank for each submatrix is $r$. We first permute the matrix according to the clusters:

$${\bf M} = {\bf P}{\bf K}{\bf P}^T$$

$$M = PKPT = \begin{pmatrix} C_1 & C_2 & \cdots & C_k \\ C_1 & M_{1,1} & M_{1,2} & \cdots & M_{1,k} \\ C_2 & M_{2,1} & M_{2,2} & \cdots & M_{2,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ C_k & M_{k,1} & M_{k,2} & \cdots & M_{k,k} \end{pmatrix}$$

where $P$ is a permutation matrix, and $M_{i,j} = K(C_i, C_j)$ is the interaction matrix between cluster $C_i$ and cluster $C_j$.

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., $M_{i,:} = (M_{i,1} \cdots M_{i,k})$, and use the index vectors $I$ and $J$ to represent sub-rows and sub-columns, e.g., $M(I,J)$ represents the intersection of rows and columns whose indices are $I$ and $J$, 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 [22] 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. 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 the $r$ sampled columns have 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) [18] 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 [12] used in our BBF algorithm; the algorithm detail is in Algorithm 1 with the procedure depicted in Figure 7. The complexity of sampling $r$ columns from an $m \times n$ matrix is $O(r^2(m+n))$. 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

```python
1 Function Randomized_Sampling(M, r, l)
2 Input: (1) Matrix M to sample from in its implicit form (given data and kernel function); (2) Sample size r; (3) Oversampling parameter l
3 Output: Important column index set $\Pi_c$
4 Initialization. Uniformly sample $r$ columns from $M$ and denote the index set as $\Pi_c$.
5 Important rows. Uniformly sample $l$ columns, and denote the index set as $\Gamma_c$. Update $\Pi_c \leftarrow \Gamma_c \cup \Pi_c$. Apply a pivoted LQ factorization on $M(:, \Pi_c)$ to get the top $r$ important row index set, denoted as $\Pi_r$.
6 Important columns. Uniformly sample $l$ rows, and denote the index set as $\Gamma_r$, update $\Pi_r \leftarrow \Pi_r \cup \Gamma_r$. Apply a pivoted QR factorization on $M(\Pi_r, :)\) to get the top $r$ important columns index set, denoted as $\Pi_c$.
7 Refinement. Alternating between important columns and important rows for a couple of times will provide reliable important columns of $M$.
8 return $\Pi_c$
9 Note: The pivoted QR is the QR factorization with column pivoting based on the largest column norm.
```

The overall complexity of Algorithm 1, as will be analyzed in Section 3.2.3, depends on the number of clusters $k$. A slight modification of Algorithm 1, as stated in Algorithm 2, will remove the dependence on $k$. The key idea is to apply Algorithm 1 on a pre-selected and refined set of columns instead of all the columns.

Figure 7: 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 for a few times to ensure the stability of the important indices.
The minimizer is given by $V$ if a matrix $C$ is given by Proposition 3.2. Yet accurate. The following proposition provides a key theoretical insight behind our algorithm. Again, we restrict ourselves in a subspace and propose a sampling-based approach that is efficient to extract the corresponding column basis.

**Proof.** To simplify the notations, we denote $\hat{\Sigma}=\text{Pseudoinverse}(C)$. Where $\hat{Q}^\top\hat{Q}=Q^\top Q=I$. Let $\hat{I} = \text{rank}(C)$, it is easy to prove that $\hat{V}^\top \hat{C} \hat{V}$ is a $\hat{I} \times \hat{I}$ inner matrix, where $\hat{C}$ is an oversampling parameter. The algorithm is described in Algorithm 3.

**Algorithm 3:** 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: $U$, $\Sigma$, and $V$ such that $M \approx U\Sigma V^T$

1. Randomly generate a Gaussian matrix $\Omega \in \mathbb{R}^{n \times (r+q)}$
2. $M\Omega = QR$
3. for $i = 1, \ldots, q$
4.   $B \leftarrow Q^* M$
5.   $B = \hat{U}\hat{\Sigma}\hat{V}^*$
6. end
7. $U \leftarrow \hat{U}$, $\Sigma \leftarrow \hat{\Sigma}$, $V \leftarrow \hat{V}$
8. return $U$, $\Sigma$, $V$
```

2. Inner matrix construction

We then consider the inner matrix construction. Given column base $U_i$ and $V_j$, we seek a matrix $C_{i,j}$ such that it minimizes $\|M_{i,j} - U_i C_{i,j} V_j^T\|$. The minimizer is given by $C_{i,j} = U_i^\top M_{i,j} (V_j^T)^\dagger$. 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 $M = UCV^T$, where $U \in \mathbb{R}^{m \times r}$ and $V \in \mathbb{R}^{n \times r}$. Further, if for some index set $I$ and $J$, $U(I,:)$ and $V(J,:)$ are full rank, then, the inner matrix $C$ is given by

$$C = (U(I,:))^\dagger M(I,J) (V(J,:)^T)^\dagger,$$

where $\dagger$ denotes the pseudo-inverse of the matrix.

**Proof.** To simplify the notations, we denote $\hat{U} = U(I,:)$, $\hat{V} = V(J,:)$, and $\hat{M} = M(I,J)$, where $I$ is the sampled row index set for $U$ and $J$ is the sampled row index set for $V$. We apply the sampling matrices $P_I$ and $P_J$ (matrices of 0’s and 1’s) to both sides of equation $M = UCV^T$, and obtain

$$P_I M P_J^T = P_I U C V^T P_J^T,$$
The assumption that \( \tilde{U} \) and \( \tilde{V} \) are tall and skinny matrices with full column ranks implies that \( \tilde{U}^\dagger \tilde{U} = I \) and \( \tilde{V}^T (\tilde{V}^T)^\dagger = I \). We then multiply \( \tilde{U}^\dagger \) and \( (\tilde{V}^T)^\dagger \) on both sides and obtain the desired result:
\[
\tilde{U}^\dagger \tilde{M} (\tilde{V}^T)^\dagger = \tilde{U}^\dagger \tilde{U} \tilde{C} \tilde{V}^T (\tilde{V}^T)^\dagger = C.
\]

Prop. 3.2 provides insights into an efficient, stable and accurate construction of the inner matrix. In practice, the equality in Prop. 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 \( \Gamma_r \) and \( r \) row indices \( \Gamma_r \), respectively, from \( C_i \) and \( C_j \). Then, the index sets are constructed as \( \mathcal{I} = \Pi_i \cup \Gamma_r \) and \( \mathcal{J} = \Pi_j \cup \Gamma_r \), where \( \Pi_i \) and \( \Pi_j \) are the important row index sets from the basis construction. Finally, \( C_{i,j} \) is given by
\[
(U_i(\mathcal{I}, :))(M_{i,j}(\mathcal{I}, \mathcal{J}))(U_j(\mathcal{J}, :)^T)^\dagger.
\]

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 block-wise sparse matrix, further reducing the memory.

Having discussed the details for the construction algorithm, we summarize the procedure in Algorithm 4.

---

### Algorithm 4: Fast construction algorithm for BBF

**Input:**
- \( k \), \( \{C_i\}_{i=1}^k \), \( \{r_i\}_{i=1}^k \), \( M \), \( q \), \( l \)

**Output:** Block diagonal matrix \( \tilde{U} \) and block-wise sparse matrix \( \hat{C} \) s.t. \( M \approx \tilde{U} \hat{C} \tilde{U}^T \)

1. **Function BBF Construction**
   1. for \( i = 1, \ldots, k \) do
   2.     \( \Pi_i = \text{Randomized Sampling}(M_i, r_i, l) \) (using Algorithm 1)
   3.     \( U_i = \text{Randomized SVD}(M_i(:, \Pi_i), r_i, q, l) \) (using Algorithm 3)
   4.   end
   5. for \( i = 1, \ldots, k \) do
   6.     for \( j = 1, \ldots, i \) do
   7.       if cutoff criterion is not satisfied then
   8.         Uniformly sample \( \Gamma_i \) and \( \Gamma_j \) from \( C_i \) and \( C_j \), respectively
   9.         \( \mathcal{I} \leftarrow \Pi_i \cup \Gamma_i \) and \( \mathcal{J} \leftarrow \Pi_j \cup \Gamma_j \)
   10.        \( C_{i,j} \leftarrow (U_i(\mathcal{I}, :))(M_{i,j}(\mathcal{I}, \mathcal{J}))(U_j(\mathcal{J}, :)^T)^\dagger \)
   11.        \( C_{j,i} \leftarrow C_{i,j} \)
   12.      end
   13.      \( C_{i,j} \leftarrow 0 \), \( C_{j,i} \leftarrow 0 \)
   14.   end
   15. end
   16. return \( \tilde{U}, \hat{C} \)

---

### 3. Pre-computation: Near-optimal Parameter Selection
We present a heuristic algorithm to identify near-optimal input parameters for BBF. The algorithm takes \(n\) input points \(\{x_i\}_{i=1}^n\) and a requested error (tolerance) \(\epsilon\), and outputs the near-optimal parameters for the BBF construction algorithm, specifically, the number of clusters \(k\), the index set for each cluster \(I_k\), and the estimated rank \(r_i\) for the submatrix corresponding to the cluster \(I_k\). We seek a set of parameters that minimizes the memory cost while keeping the approximation error below \(\epsilon\).

**Choice of column ranks.** Given the tolerance \(\epsilon\) and the number of clusters \(k\), we describe our method of identifying the near-optimal 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 \(\hat{M}_{i,j}\) as the approximation of \(M_{i,j}\), \(\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n\), 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

\[
    r_i = \min \left\{ m \mid \sum_{p=m+1}^{n_i} \sigma_p^2 < \frac{n_i^2}{n_i} \| M_{i,i} \|_F^2 \epsilon^2 \right\}.
\]

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

\[
\begin{align*}
    \text{minimize}_k & \quad g(k) = \sum_{i=1}^{k} n_i r_i + \left( \sum_{i=1}^{k} r_i \right)^2 \\
    \text{subj to} & \quad r_i = \min \left\{ m \mid \sum_{p=m+1}^{n_i} \sigma_p^2 < \frac{n_i^2}{n_i} \| M_{i,i} \|_F^2 \epsilon^2 \right\}, \forall i
\end{align*}
\]

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 pre-computation step. We first introduce some notations.

**Notations.** Let \(k\) denote the number of clusters, \(\{n_i\}_{i=1}^{k}\) denote the number of points in each cluster, \(\{r_i\}_{i=1}^{k}\) denote the requested rank for the blocks in the \(i\)-th 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 \(i\)-th 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 assuming that the clusters are of equal size and the numerical rank for each block is \(r\). 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 \(r_i^2r_j + r_i^2\). Summing over all the \(k^2\) blocks, the overall complexity is given by

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

With the same assumptions as above, the simplified complexity is \(O(k^2r^3)\). 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.

Table 2: Complexity table. $n$ is the total number of points, $k$ is the number of clusters, $r$ is the numerical rank for column basis.

<table>
<thead>
<tr>
<th>Pre-computation</th>
<th>Factorization</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each block</td>
<td>$O(nr^2)$</td>
<td>Basis $O(nr^2)$</td>
</tr>
<tr>
<td>Compute $g(k)$</td>
<td>$O(nr^2)$</td>
<td>Inner matrix $O(k^2r^3)$</td>
</tr>
<tr>
<td>Total</td>
<td>$O(r^2n \log n)$</td>
<td>Total $O(nr + (rk)^2)$</td>
</tr>
</tbody>
</table>

From Table 2, we note that the application cost, i.e., the storage, depends 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 because the complexity did not consider the block sparsity in the inner matrices, and we will discuss the influence of $k$ in what follows.

1. **Storage.** For a fixed accuracy and for RBF kernel matrices whose rank is medium, we observed that the storage of BBF as a function of $k$ is close to being convex. In practice, we were able to use optimization algorithms for convex functions and get satisfactory results. The general shape of the storage as a function of $k$ is as follows. In the small-$k$ regime, the BBF structure is close to the low-rank structure, which is inefficient, and hence the storage is relatively large. In the large-$k$ regime, e.g., the extreme case being $k = n$, the scheme reverts to a quadratic storage. In addition, for a fixed accuracy and for rapidly decaying kernels that lead to high ranks, increasing $k$ typically decreases the storage. In that case, as the rank increases (e.g., $h$ decreases), the entries in off-diagonal blocks of the inner matrices become small enough that setting them to 0 does not cause much accuracy loss. This leads to further reductions in the storage.

2. **Factorization complexity.** The factorization complexity can be further reduced to be independent of $k$ by avoiding the construction of $M_i, (\Pi_r, :)$ as described in Algorithm 2, assuming $k < O(\sqrt{n})$.

In the end, we empirically verify the linear complexity of our method. Figure 8 shows the factorization time (in second) versus the number of data points on some real datasets. The trend is linear, confirming the linear complexity of our algorithm.

![Figure 8: Factorization time (loglog 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).](image-url)
4 Experimental Results

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

The datasets are listed in Table 3 and Table 1, and they can be downloaded from the UCI repository [5], the libsvm website [7] 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 2.4 GHz CPU and 8 GB memory.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Abalone</th>
<th>Mushroom</th>
<th>Cpusmall</th>
</tr>
</thead>
<tbody>
<tr>
<td># Instance</td>
<td>4,177</td>
<td>8,124</td>
<td>8,192</td>
</tr>
<tr>
<td># Attributes</td>
<td>8</td>
<td>112</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 3: Real datasets used in the experiments.

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); 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 $M_{i,\cdot}$ 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 9 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 6, and the low-rank structure means $K \approx UU^T$, 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.

Figure 10 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.

4.2 BBF algorithm

In this section, we experimentally evaluate the performance of our BBF algorithm with other state-of-art kernel approximation methods. Section 4.2.1 and Section 4.2.2 examine the matrix reconstruction error under varying memory budget and kernel bandwidth parameters. Section 4.2.3
Figure 9: Errors for each block in the approximated matrix from the Abalone dataset. Warmer color represents larger error. Subplot (a1) and (a2) shares the same colorbar; and (b1) and (b2) shares the same colorbar. The error for block \((i,j)\) is computed as \(\|M_{i,j} - \hat{M}_{i,j}\|_F / \|M\|_F\). The basis in subplot (a1) and (a2) are computed by an SVD; and in (b1) and (b2) are computed 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 subplot (a1), (a2), (b1) and (b2) are \(1.4 \times 10^{-3}\), \(6.3 \times 10^{-3}\), \(1.5 \times 10^{-2}\), \(4.0 \times 10^{-2}\), respectively.

Figure 10: 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 6, and LR (dash lines) uses a low-rank structure. “rand”: randomized sampling; “unif”: uniform sampling; “ls”: exact leverage score sampling; “svd”: an SVD is used for computing the basis.
applies the approximations to the kernel ridge regression problem. Finally, Section 4.2.4 compares
the linear complexity of BBF with the improved fast gauss transform (IFGT). Throughout the
experiments, we use BBF to denote our algorithm, whose input parameters are computed from
our pre-computation 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 land-
  mark 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 exact leverage scores ($O(n^2)$) and sampled $2k$
  columns with replacement for a rank-$k$ approximation.

- **Memory Efficient Kernel Approximation (MEKA).** We used the code provided by the author.

- **Random Kitchen Sinks (RKS).** We used our own MATLAB implementation based on their
  algorithm.

- **Improved Fast Gauss Transform (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 multi-
lication. 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.

Figure 11 and Figure 12 show the reconstruction error versus memory cost on real datasets and
2D 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 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.

The plots on the left of Figure 13 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 are revealed in some statistics listed in Table 4. The results
in the table are consistent with the results shown in [14] for varying kernel bandwidth parameters.

In the large-$h$ regime, the gap in error between BBF and other methods is small. In such
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 pre-computation
procedure will increase the number of clusters to better approximate the diagonal part, and the
Figure 11: Comparisons of BBF (our algorithm) with competing methods. Top four plots (loglog 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).
Figure 12: 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 has alternating labels (see Figure 4 with 100 clusters) while the data for the bottom plots has smaller clusters surrounded by larger ones (see Figure 5). 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 value 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).

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 $M$. $\frac{\|M\|_F^2}{\|M_r\|_F^2}$ is referred to as the stable rank and is an underestimate of the rank. $l_r$ represents the $r$-th largest leverage score scaled by $\frac{1}{n}$. 

<table>
<thead>
<tr>
<th></th>
<th>Abalone ($r = 100$)</th>
<th>Pendigits ($r = 252$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\frac{1}{n^2}$</td>
<td>$\frac{|M|_F^2}{|M|_F^2}$</td>
</tr>
<tr>
<td>0.25</td>
<td>2</td>
<td>99.99</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>99.86</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>97.33</td>
</tr>
<tr>
<td>25</td>
<td>15</td>
<td>72.00</td>
</tr>
<tr>
<td>100</td>
<td>175</td>
<td>33.40</td>
</tr>
<tr>
<td>400</td>
<td>931</td>
<td>19.47</td>
</tr>
<tr>
<td>1000</td>
<td>1155</td>
<td>16.52</td>
</tr>
</tbody>
</table>
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.

Figure 13: 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.

### 4.2.3 Kernel ridge regression

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

$$
\min_\alpha \| K\alpha - y \|^2 + \lambda \|\alpha\|^2,
$$

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

$$
(K + \lambda I)\alpha = y.
$$

The linear system can be solved by an iterative solver, e.g. MINRES [27], and the complexity is $O(n^2 T)$ where $n^2$ is from matrix-vector multiplications and $T$ denotes the iteration number. If we can approximate $K$ by $\hat{K}$ 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.

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

$$
(\hat{K} + \lambda I)\alpha = y.
$$

The dataset was randomly divided into training set (80%) and testing set (20%). We report the test root-mean-square error (RMSE) which is defined as

$$
\sqrt{\frac{1}{n_{\text{test}}} \| K_{\text{test}} \hat{\alpha} - y_{\text{test}} \|^2_F},
$$

where $F$ denotes the Frobenius norm.
where $K_{\text{test}}$ is the interaction matrix between the test data and training data, $\hat{\alpha}$ is the solution from solving Equation 6, and $y_{\text{test}}$ is the true test target. Figure 14 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.

![Figure 14: 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 $\lambda$ were selected by a 5-fold cross-validation with a grid search, and the selected $(h, \lambda)$ pairs are listed in the subcaption.](image)

4.2.4 Comparison with the improved fast gauss transform (IFGT)

We benchmarked the linear complexity of the Improved Fast Gauss Transform (IFGT) [34] 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 15 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.
Figure 15: Timing (loglog scale) for IFGT and BBF on a synthetic dataset with dimension $d = 5$ 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.
5 Conclusions

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 kernel 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, 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-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-art approximation methods.
References


