Journal of Machine Learning Research 18 (2017) 1-32 Submitted 1/15; Revised 2/17; Published 3/17

Memory Efficient Kernel Approximation

Si Si* SISIDAISY@GOOGLE.COM
Google Research
Mountain View, CA 94043, USA
Cho-Jui Hsieh CHOHSIEH@UCDAVIS.EDU
Departments of Computer Science and Statistics
University of California, Davis
Davis, CA 95616, USA
Inderjit S. Dhillon INDERJIT@CS.UTEXAS.EDU
Department of Computer Science
University of Texas at Austin
Austin, TX 78701, USA

Editor: Le Song

Abstract

Scaling kernel machines to massive data sets is a major challenge due to storage and computation issues in handling large kernel matrices, that are usually dense. Recently, many papers have suggested tackling this problem by using a low-rank approximation of the kernel matrix. In this paper, we first make the observation that the structure of shift-invariant kernels changes from low-rank to block-diagonal (without any low-rank structure) when varying the scale parameter. Based on this observation, we propose a new kernel approximation framework – Memory Efficient Kernel Approximation (MEKA), which considers both low-rank and clustering structure of the kernel matrix. We show that the resulting algorithm outperforms state-of-the-art low-rank kernel approximation methods in terms of speed, approximation error, and memory usage. As an example, on the covtype dataset with half a million samples, MEKA takes around 70 seconds and uses less than 80 MB memory on a single machine to achieve 10% relative approximation error, while standard Nyström approximation is about 6 times slower and uses more than 400MB memory to achieve similar approximation. We also present extensive experiments on applying MEKA to speed up kernel ridge regression.

Keywords: kernel approximation, Nyström method, kernel methods

1. Introduction

Kernel methods (Schölkopf and Smola, 2002) are a class of machine learning algorithms that first map samples from input space to a high-dimensional feature space. In the high-dimensional feature space, various methods can be applied depending on the machine learning task, for example, kernel support vector machine (SVM) (Cortes and Vapnik, 1995) (Hsieh, Si, and Dhillon, 2014a) and kernel ridge regression (Saunders, Gammerman, and Vovk, 1998). A key issue in scaling up kernel machines is the storage and computation

∗. This work was done before joining Google.

of the kernel matrix, which is usually dense. Storing the dense matrix takes $O(n^2)$ space, while computing it requires $O(n^2 d)$ operations, where n is the number of data points and d is the dimension. A common approach to achieve scalability is to approximate the kernel matrix using limited memory storage. This approach not only resolves the memory issue, but also speeds up kernel machine solvers, because the time complexity for using the kernel is usually proportional to the amount of memory used to represent the kernel. Most kernel approximation methods aim to form a low-rank approximation $G \approx CC^T$ for the kernel matrix G, with $C \in \mathbb{R}^{n \times k}$ and rank $k \ll n$. Although it is well known that Singular Value Decomposition (SVD) yields the best rank-k approximation, it often cannot be applied as it requires the entire kernel matrix to be computed and stored. To overcome this issue, many methods have been proposed to approximate the best rank-k approximation of a kernel matrix, including Greedy basis selection techniques (Smola and Schölkopf, 2000), incomplete Cholesky decomposition (Fine and Scheinberg, 2001), and Nyström methods (Williams and Seeger, 2001).

However, it is unclear whether low-rank approximation is the most memory efficient way to approximate a kernel matrix. In this paper, we first make the observation that for practically used shift-invariant kernels, the kernel structure varies from low-rank to block-diagonal as the scaling parameter γ varies from 0 to ∞. This observation suggests that even the best rank-k approximation can have extremely large approximation error when γ is large, so it is worth exploiting the block structure of the kernel matrix. Based on this idea, we propose a Memory Efficient Kernel Approximation (MEKA) framework to approximate the kernel matrix. Our proposed framework considers and analyzes the use of clustering in the input space to efficiently exploit the block structure of shift-invariant kernels. We show that the individual blocks generated by kmeans clustering have low-rank structure, which motivates us to apply Nyström low-rank approximation to each block separately. Between-cluster blocks are then approximated in a memory-efficient manner. Our approach only needs $O(nk + (ck)^2)$ memory to store a rank-ck approximation(where $c \ll n$ is the number of clusters), while traditional low-rank methods need $O(nk)$ space to store a rank-k approximation. Therefore, using the same amount of storage, our method can achieve lower approximation error than the commonly used low-rank methods. Moreover, our proposed method takes less computation time than other low-rank methods to achieve a given approximation error.

Theoretically, we show that under the same amount of storage, the error bound of our approach can be better than standard Nyström if the gap between the $k+1$-st and $ck+1$-st singular values of G is larger than $\|\Delta\|_2$ where Δ consists of all between-cluster blocks. On real datasets, our proposed algorithm consumes less memory and computation time to achieve comparable reconstruction error. For example, on the covtype dataset with half million samples, MEKA takes around 70 seconds and uses less than 80 MB memory on a single machine to achieve 10% relative approximation error, while standard Nyström approximation takes more than 400 seconds and uses more than 400MB memory to achieve similar approximation. Also, MEKA is faster for kernel ridge regression compared with other state-of-the-art kernel approximation methods. As an example, on the mnist2m dataset with 2 million samples, our method takes 550 seconds on a single machine using less than 500 MBytes memory to achieve accuracy comparable with standard Nyström approximation,

which takes more than 2700 seconds and uses more than 2 GBytes memory on the same problem.

Parts of this paper have appeared previously in (Si, Hsieh, and Dhillon, 2014a). In this paper, we provide: (1) a more detailed survey of state-of-the-art methods; (2) much more comprehensive experimental comparisons; (3) thorough investigation of the influence of the parameters in our method; (4) the application of applying the block structure of kernel matrix to speed up kernel SVM; and (5) more discussion including how to achieve stable results and solve non-psd issues in MEKA.

The rest of the paper is outlined as follows. We first present related work in Section 2. We then explain the popular Nyström approximation method and present motivation for our framework in Section 3. We then show the block structure of kernel matrix and its application to speed up kernel SVM in Section 4. Our main kernel approximation algorithm MEKA is proposed and analyzed in Section 5. Experimental results are given in Section 6, and conclusion and discussion are provided in Section 7.

2. Related Research

To approximate the kernel matrix using limited memory, one common way is to use a low-rank approximation. The best rank-k approximation can be obtained by the SVD, but it is computationally prohibitive when n grows to tens of thousands. To address the scalability issue of SVD, approximate SVD solvers such as randomized SVD (Halko, Martinsson, and Tropp, 2011) have been widely used for large-scale data. To exploit the sparse structure of large-scale network data, an alternative is to apply CUR matrix decomposition (Mahoney and Drineas, 2009) that explicitly expresses the low-rank decomposition in terms of a small number of rows and columns of the original data matrix. Another way is building a hierarchical tree to initialize a block Lanczos algorithm to efficiently compute the spectral decomposition of large-scale graphs (Si, Shin, Dhillon, and Parlett, 2014b). Unfortunately, to approximate kernel matrices, all the above approaches need to compute the entire kernel matrix, so the time complexity is at least $O(dn^2)$.

Many algorithms have been proposed to overcome the prohibitive time and space complexity of SVD for approximating kernel matrices. They can be categorized into two classes: methods that explicitly approximate kernel matrices, and methods that approximate the kernel function.

Approximating the kernel matrix. The first class of approaches approximate the kernel matrix based on a subset of sampled elements; as a result, all of them are data dependent. The Nyström method (Williams and Seeger, 2001) is the most widely used technique to approximate the kernel matrix given a sampled subset of columns. To approximate a rank-k approximation, Nyström approximation requires $O(nk^2 + k^3)$ time to form the rank-k approximation. Many strategies have been proposed to improve over the basic Nyström approximation, including ensemble Nyström (Kumar, Mohri, and Talwalkar, 2009), Nyström with k-means to obtain benchmark points (Zhang, Tsang, and Kwok, 2008; Zhang and Kwok, 2010), randomized Nyström (Li, Kwok, and Lu, 2010), Nyström approximation with shift (Wang et al., 2014), adding "pseudo landmark points" to the Nyström approximation (Hsieh, Si, and Dhillon, 2014b), and fast-Nys that uses fast transform to

generate structured landmark points to speed up Nyström approximation (Si, Hsieh, and Dhillon, 2016).

Different Nyström sampling strategies are analyzed and compared in (Kumar, Mohri, and Talwalkar, 2012; Gittens and Mahoney, 2013). Besides Nyström approximation, Fine and Scheinberg (2001) use the incomplete Cholesky decomposition with pivoting for approximating kernel matrices, which requires $O(nk^2 + nkd)$ time for computing a rank-k approximation. Bach and Jordan (2005) incorporate side information (labels) into the incomplete Cholesky decomposition, and show that the resulting problem can be solved with the same $O(nk^2 + nkd)$ time complexity. Finally, Achlioptas, McSherry, and Schölkopf (2001) propose a sampling and reweighted approach to obtain an unbiased estimator of the kernel-vector product, and use subspace iteration to approximate the top k eigenvectors.

Approximating the kernel function. The second class of methods is to directly approximate the kernel function without computing elements of the kernel matrix, so the approximation does not depend on the data. To approximate the kernel function, a typical approach is to find a feature mapping $Z : \mathbb{R}^d \to \mathbb{R}^k$ where the kernel function $K(\boldsymbol{x}, \boldsymbol{y})$ can be approximated by $Z(\boldsymbol{x})^T Z(\boldsymbol{y})$. Rahimi and Recht (2007, 2008) define the random feature map for shift invariant kernel functions based on the Fourier transform. In the resulting Random Kitchen Sinks (RKS) algorithm, the main computation turns out to be the matrix vector multiplication $W\boldsymbol{x}_i$ for each instance $\boldsymbol{x}_i$, where W is a Gaussian random matrix. To improve efficiency, Le, Sarlos, and Smola (2013) show that the computation of $W\boldsymbol{x}_i$ can be sped up by the fast Hadamard transform. On the other hand, Yang et al. (2014) propose to use a quasi Monte Carlo approach to improve the approximation performance of RKS. In addition to shift invariant kernels, Kar and Karnick (2012) construct the random feature map for polynomial kernels, and Hamid et al. (2014) propose the condensed random feature map to improve performance.

Besides the above approaches based on random feature maps, there are other methods that directly approximate the kernel function. Cotter, Keshet, and Srebro (2011) approximate the Gaussian kernel by the t-th order Taylor expansion, but it requires $O(d^t)$ features, which is computationally burdensome for large d or t. Chang et al. (2010) propose to use the kernel expansion for low-degree polynomial kernels. Recently, Yang et al. (2012) showed that the Nyström method has a better generalization error bound than the RKS approach if the gap in the eigen-spectrum of the kernel matrix is large.

Most of the above methods can be viewed as faster ways to find a rank-k approximation of the kernel matrix, so the approximation error is always worse than the top-k SVD when using $O(nk)$ memory. As we show in Section 3, the kernel matrix typically changes from low-rank to block structure as the scaling parameter γ increases. However, the block structure of the kernel matrix has *never* been considered in dense kernel approximation, although it has been studied for approximation of other types of matrices. For example, Savas and Dhillon (2011) applied Clustered Low Rank Approximation (CLRA) to approximate large and sparse social networks. CLRA applies spectral or graph clustering to the adjacency matrix of a social network, runs SVD on each diagonal block, and uses matrix projection to capture off-diagonal information. All these steps require storage of the entire matrix, thus they are infeasible for large-scale dense kernel matrices. For example, computing the entire kernel matrix of the mnist2m dataset would require about 8 TBytes of memory; moreover the time for computing the SVD and projection steps is prohibitive. On the same dataset

our proposed approach can obtain accurate results in 10 minutes with only 500 MBytes memory (as shown in Section 6). To achieve this, we need totally different algorithms than the ones in CLRA for clustering and approximating blocks to yield our memory-efficient scheme.

More specifically, CLRA was designed to approximate sparse adjacency matrices, but cannot be directly applied to large kernel matrices as that would require computation and storage of the entire kernel matrix at a cost of $O(n^2d)$ time and $O(n^2)$ space. To overcome this problem, we propose the following innovations: (1) We perform clustering, and then apply Nyström approximation to within-cluster blocks to avoid computing all within-block entries; (2) We theoretically justify the use of kmeans clustering to explore the block structure of the kernel; (3) We propose a sampling approach to capture between-block information; (4) We theoretically show the error bound of our method and compare it with the traditional Nyström approach.

3. Preliminaries and Motivation

Let $K(\cdot,\cdot)$ denote the kernel function, and $G \in \mathbb{R}^{n\times n}$ be the corresponding kernel matrix where $G_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $\boldsymbol{x}_i$, $\boldsymbol{x}_j \in \mathbb{R}^d$ are data points. Computing and storing the kernel matrix G usually takes $O(n^2d)$ time and $O(n^2)$ space, which is prohibitive when there are millions of samples. One way to deal with these challenges is to approximate the dense kernel matrix G by a low-rank approximation $\tilde{G}$. By doing this, kernel machines are transformed to linear problems which can be solved efficiently. The best rank-k approximation of G is given by its singular value decomposition(SVD), i.e., $G \approx U_k\Sigma_k{U_k}^T$, where Σ_k is the diagonal matrix of largest k singular values and U_k contains the corresponding singular vectors.

However, computing the SVD of G is computationally prohibitive and memory intensive. Many fast kernel approximation algorithms have thus been proposed and studied. Nyström kernel approximation is a widely used approximation approach, which uses a sample of m data points and does not need to form the entire G explicitly to generate its low-rank approximation. In standard Nyström approximation (proposed in Williams and Seeger (2001)), we first uniformly at random sample m data points and assemble the corresponding m columns of G as the $n \times m$ matrix C. Let M be the $m \times m$ kernel matrix between the m sampled points, then the standard Nyström method generates a rank-k approximation to G as

$$G \approx \tilde{G} = CM_k^+C^T, \tag{1}$$

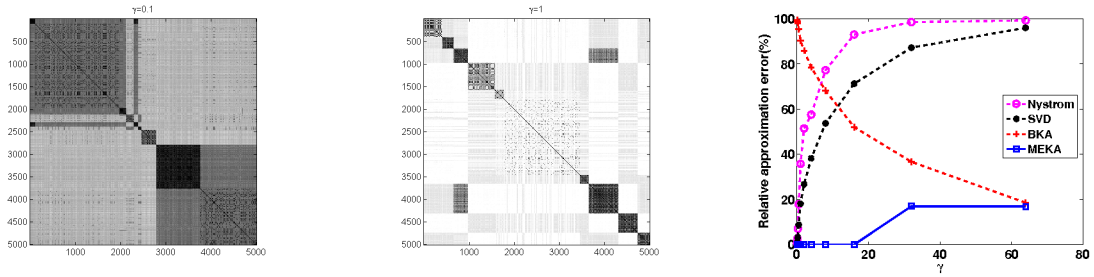
where M_k is the best rank-k approximation of M (by SVD) and M_k^+ is its pseudo-inverse. Various extensions to this Nyström based kernel approximation have been proposed. For example, k-means Nyström(Zhang, Tsang, and Kwok, 2008; Zhang and Kwok, 2010) uses clusters centroids as the landmark points to form C; ensemble Nyström(Kumar, Mohri, and Talwalkar, 2009) combines a collection of standard Nyström approximations. We will compare state-of-the-art Nyström based methods in Section 6.

We use the Gaussian kernel as an example to discuss the structure of the kernel matrix under different scale parameters. Given two samples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, the Gaussian kernel is given by $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma\|\boldsymbol{x}_i-\boldsymbol{x}_j\|_2^2}$, where γ is a scale or width parameter; the corresponding kernel matrix entries are $G_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Low-rank approximation has been widely used

to obtain an approximation for kernel matrices. However, under different scale parameters, the kernel matrix has quite different structures, suggesting that different approximation strategies should be used for different γ.

Let us examine two extreme cases of the Gaussian kernel: when $\gamma \to 0$, $G \to \boldsymbol{e}\boldsymbol{e}^T$ where $\boldsymbol{e} = [1, \ldots, 1]^T$. As a consequence, G is close to low-rank when γ is small. However, at the other extreme as $\gamma \to \infty$, G changes to the identity matrix, which has full rank with all eigenvalues equal to 1. In this case, G does not have a low-rank structure, but has a block/clustering structure. This observation motivates us to consider both low rank and clustering structure of the kernel matrix. Figures 1a and 1b give an example of the structure of a Gaussian kernel with different γ on a real dataset by randomly sampling 5000 samples from the covtype dataset.

Before discussing further details, we first contrast the use of block and low-rank approximations on the same dataset. We compare approximation errors for different methods when they use the same amount of memory in Figure 1c. Clearly, low-rank approximation methods work well only for very small γ values. Block Kernel Approximation (BKA), as proposed in Section 4.1, is a simple way to use clustering structure of G that is effective for large γ. Our proposed algorithm, MEKA, considers both block and low-rank structure of the kernel, and thus performs better than others under different γ values as seen in Figure 1c.



(a) The Gaussian kernel matrix with $\gamma = 0.1$ on covtype dataset

(b) The Gaussian kernel matrix with $\gamma = 1$ on covtype dataset

(c) Comparison of different kernel approximation methods for various γ.

Figure 1: (a) and (b) show that the structure of the Gaussian kernel matrix $K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma \|\boldsymbol{x}-\boldsymbol{y}\|^2}$ for the covtype data tends to become more block diagonal as γ increases(dark regions correspond to large values, while lighter regions correspond to smaller values). Plot (c) shows that low-rank approximations work only for small γ, and Block Kernel Approximation (BKA) works for large γ, while our proposed method MEKA works for small as well as large γ.

4. Block Kernel Approximation

In this section, we first introduce Block Kernel Approximation (BKA), a simple way to exploit the clustering structure of kernel matrices, and then show its application for speeding up kernel SVM.

4.1 Clustering Structure of Shift-invariant Kernel Matrices

There has been substantial research on approximating shift-invariant kernels (Rahimi and Recht, 2007). A kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is shift-invariant if the kernel value depends only on $\boldsymbol{x}_i - \boldsymbol{x}_j$, that is, $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = f(\eta(\boldsymbol{x}_i - \boldsymbol{x}_j))$ where $f(\cdot)$ is a function that maps $\mathbb{R}^d$ to $\mathbb{R}$, and $\eta > 0$ is a constant to determine the "scale" of the data. η is very crucial to the performance of kernel machines and is usually chosen by cross-validation. We further define $g_{\boldsymbol{u}}(t) = f(\eta t \boldsymbol{u})$ to be a one variable function along $\boldsymbol{u}$'s direction where $\boldsymbol{u}$ is an unit vector. We assume the kernel function satisfies the following property:

Assumption 1 *$g_{\boldsymbol{u}}(t)$ is differentiable for all $t \neq 0$.*

Most of the practically used shift-invariant kernels satisfy the above assumption, for example, the Gaussian kernel ($K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma\|\boldsymbol{x}-\boldsymbol{y}\|_2^2}$), and the Laplacian kernel ($K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\gamma\|\boldsymbol{x}-\boldsymbol{y}\|_1}$). It is clear that η^2 is equivalent to γ for the Gaussian kernel if written in the form of $K(\boldsymbol{x}, \boldsymbol{y}) = f(\eta(\boldsymbol{x}-\boldsymbol{y}))$. When η is large, off-diagonal blocks of shift-invariant kernel matrices will become small, and most of the information is concentrated in the diagonal blocks. To approximate the kernel matrix by exploiting this clustering structure, we first present a simple Block Kernel Approximation (BKA) as follows. Given a good partition $\mathcal{V}_1, \ldots, \mathcal{V}_c$ of the data points, where each $\mathcal{V}_s$ is a subset of $\{1, \ldots, n\}$, BKA approximates the kernel matrix as:

$$G \approx \tilde{G} \equiv \begin{bmatrix} G^{(1,1)} & 0 & \ldots & 0 \\ 0 & G^{(2,2)} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & G^{(c,c)} \end{bmatrix}. \tag{2}$$

Here, $G^{(s,s)}$ denotes the kernel matrix for block $\mathcal{V}_s$ – note that this implies that diagonal blocks $\tilde{G}^{(s,s)} = G^{(s,s)}$ and all the off-diagonal blocks, $\tilde{G}^{(s,t)} = 0$ with $s \neq t$.

BKA is useful when η is large. By analyzing its approximation error, we now show that k-means in the input space can be used to capture the clustering structure for shift-invariant kernels. The approximation error equals $\|\tilde{G}-G\|_F^2 = \sum_{i,j} K(\boldsymbol{x}_i, \boldsymbol{x}_j)^2 - \sum_{s=1}^c \sum_{i,j\in\mathcal{V}_s} K(\boldsymbol{x}_i, \boldsymbol{x}_j)^2$. Since the first term is fixed, minimizing the error $\|\tilde{G} - G\|_F^2$ is the same with maximizing the second term, the sum of squared within-cluster entries $D = \sum_{s=1}^c \sum_{i,j\in\mathcal{V}_s} K(\boldsymbol{x}_i, \boldsymbol{x}_j)^2$.

However, directly maximizing D will not give a useful partition – the maximizer will assign all the data into one cluster. The same problem occurs in graph clustering (Shi and Malik, 2000; von Luxburg, 2007). A common approach is to normalize D by each cluster's size $|\mathcal{V}_s|$. The resulting spectral clustering objective (also called ratio association) is:

$$D^{\text{kernel}}(\{\mathcal{V}_s\}_{s=1}^c) = \sum_{s=1}^{c} \frac{1}{|\mathcal{V}_s|} \sum_{i,j\in\mathcal{V}_s} K(\boldsymbol{x}_i, \boldsymbol{x}_j)^2. \tag{3}$$

Maximizing (3) usually yields a balanced partition, but the computation is expensive because we have to compute all the entries in G. In the following theorem, we derive a lower bound for $D^{\text{kernel}}(\{\mathcal{V}_s\}_{s=1}^c)$:

Theorem 1 *For any shift-invariant kernel that satisfies Assumption 1,*

$$D^{kernel}(\{\mathcal{V}_s\}_{s=1}^c) \geq \bar{C} - \eta^2 R^2 D^{kmeans}(\{\mathcal{V}_s\}_{s=1}^c) \tag{4}$$

where $\bar{C} = \frac{nf(0)^2}{2}$, *$R$ is a constant depending on the kernel function, and* $D^{kmeans} \equiv \sum_{s=1}^c \sum_{i\in\mathcal{V}_s} \|\boldsymbol{x}_i - \boldsymbol{m}_s\|_2^2$ *is the k-means objective function, where* $\boldsymbol{m}_s = (\sum_{i\in\mathcal{V}_s}\boldsymbol{x}_i)/|\mathcal{V}_s|$, $s=1,\cdots,c$, *are the cluster centers.*

Proof We use $\boldsymbol{u}$ to denote the unit vector in the direction of $\boldsymbol{x}_i - \boldsymbol{x}_j$ ($\boldsymbol{x}_i \neq \boldsymbol{x}_j$). By the mean value theorem, we have

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = g_{\boldsymbol{u}}(\eta\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2) = g_{\boldsymbol{u}}(0) + \eta g'_{\boldsymbol{u}}(s)\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2$$

for some $s \in (0, \eta\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2)$. By definition, $f(\mathbf{0}) = g_{\boldsymbol{u}}(0)$, so

$$f(\mathbf{0}) \leq K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \eta R\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2, \tag{5}$$

$$\text{where } R := \sup_{\theta\in\mathbb{R}, \|\boldsymbol{v}\|=1} |g'_{\boldsymbol{v}}(\theta)|. \tag{6}$$

Squaring both sides of (5) we have

$$f(\mathbf{0})^2 \leq K(\boldsymbol{x}_i, \boldsymbol{x}_j)^2 + \eta^2 R^2\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2 + 2K(\boldsymbol{x}_i, \boldsymbol{x}_j)(\eta R\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2).$$

From the classical arithmetic and geometric mean inequality, we can upper bound the last term by

$$2K(\boldsymbol{x}_i, \boldsymbol{x}_j)(\eta R\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2) \leq K(\boldsymbol{x}_i, \boldsymbol{x}_j)^2 + \eta^2 R^2\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2,$$

therefore

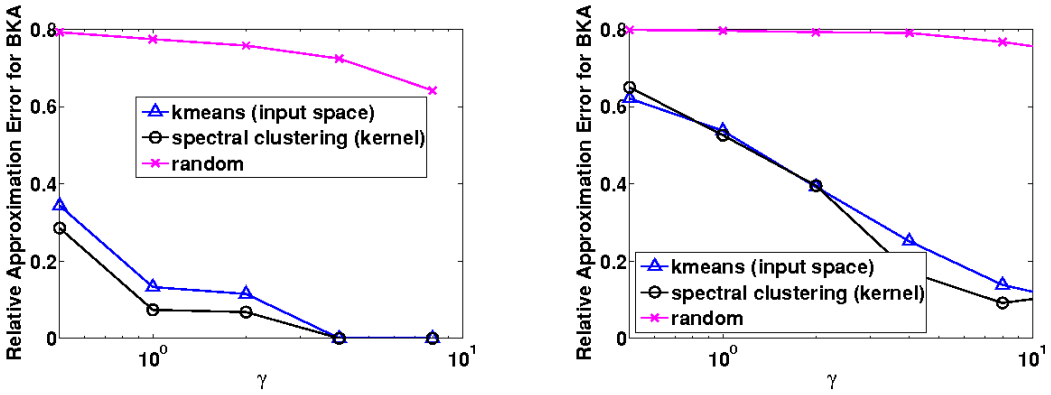
$$\frac{f(\mathbf{0})^2}{2} \leq K(\boldsymbol{x}_i, \boldsymbol{x}_j)^2 + \eta^2 R^2\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2. \tag{7}$$

Plugging (7) into (3), we have

$$\begin{aligned}\mathcal{D}^{\text{kernel}}(\{\mathcal{V}_s\}_{s=1}^c) &\geq \sum_{s=1}^c \frac{1}{|\mathcal{V}_s|} \sum_{i,j\in\mathcal{V}_s} \left(\frac{f(\mathbf{0})^2}{2} - \eta^2 R^2\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2\right) \\ &\geq \frac{nf(\mathbf{0})^2}{2} - \eta^2 R^2 \sum_{s=1}^c \frac{1}{|\mathcal{V}_s|} \sum_{i,j\in\mathcal{V}_s} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2,\end{aligned}$$

which can be manipulated to prove the desired bound (4). ■

Interestingly, the right hand side of (4) can be maximized when the k-means objective function D^{kmeans} is minimized. Therefore, although optimal solutions for k-means and ratio association might be different (consider the two circles data, when each circle forms a cluster), Theorem 1 shows that conducting k-means in the input space will provide a reasonably good way to exploit the clustering structure of shift-invariant kernels, especially when it is infeasible to perform spectral clustering on G which might need precomputation of the entire kernel matrix. Figure 2 shows that the partition from k-means can often work as well as spectral clustering on G, which directly optimizes D^{kernel}, and both of them are much better than random partitions. One advantage of conducting k-means is that the time complexity of each iteration is $O(ndc)$, which is much less than computing the kernel when the dimensionality $d \ll n$.



(a) covtype (b) cadata

Figure 2: The Gaussian kernel approximation error of BKA using different ways to generate five partitions on 500 samples from covtype and cadata; on these data sets k-means in the input space performs similarly to spectral clustering on the kernel matrix, but is more efficient.

4.2 Speeding up Kernel SVM with BKA

In this section we will show how to use block kernel approximation(BKA) to divide kernel SVM problem into subproblems and significantly speed up the computation. Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i), i = 1, \ldots, n, \boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, -1\}$, the main task in training the kernel SVM is to solve the following quadratic optimization problem:

$$\min_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \boldsymbol{e}^T \boldsymbol{\alpha}, \quad \text{s.t. } 0 \leq \boldsymbol{\alpha} \leq C, \tag{8}$$

where $\boldsymbol{e}$ is the vector of all ones; C is the balancing parameter between loss and regularization in the SVM primal problem; $\boldsymbol{\alpha} \in \mathbb{R}^n$ is the vector of dual variables; and Q is an $n \times n$ matrix with $Q_{ij} = y_i y_j G_{ij}$, where $G_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the kernel value between i-th and j-th sample. Letting $\boldsymbol{\alpha}^*$ denote the optimal solution of (8), the decision value for a test data $\boldsymbol{x}$ can be computed by

$$\sum_{i=1}^{n} \alpha_i^* y_i K(\boldsymbol{x}, \boldsymbol{x}_i). \tag{9}$$

Due to high computation cost of directly solving kernel SVM, we can approximate the kernel matrix G by BKA to divide whole kernel SVM problem into subproblems, where each subproblem can be handled efficiently and independently.

To do this, we first partition the dual variables into k subsets $\{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$, where $\{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$ are partitions generated by performing kmeans on the data points, and then solve the respective subproblems independently

$$\min_{\boldsymbol{\alpha}_{(c)}} \frac{1}{2}(\boldsymbol{\alpha}_{(c)})^T Q_{(c,c)} \boldsymbol{\alpha}_{(c)} - \boldsymbol{e}^T \boldsymbol{\alpha}_{(c)}, \text{ s.t. } 0 \leq \boldsymbol{\alpha}_{(c)} \leq C, \tag{10}$$

where $c = 1, \ldots, k$, $\boldsymbol{\alpha}_{(c)}$ denotes the subvector $\{\alpha_i \mid i \in \mathcal{V}_c\}$ and $Q_{(c,c)}$ is the submatrix of Q with row and column indexes $\mathcal{V}_c$.

dataset	Number of training samples	Number of testing samples	d
ijcnn1	49,990	91,701	22
census	159,619	39,904	409
covtype	464,810	116,202	54

Table 1: Dataset statistics

The quadratic programming problem (8) has n variables, and takes at least $O(n^2)$ time to solve in practice. By dividing it into k subproblems (10) with equal sizes, the time complexity for solving the subproblems can be reduced to $O(k \cdot (\frac{n}{k})^2) = O(n^2/k)$. Moreover, the space requirement is also reduced from $O(n^2)$ to $O(n^2/k^2)$.

After computing all the subproblem solutions, we concatenate them to form an approximate solution for the whole problem $\bar{\boldsymbol{\alpha}} = [\bar{\boldsymbol{\alpha}}_{(1)}, \ldots, \bar{\boldsymbol{\alpha}}_{(k)}]$, where $\bar{\boldsymbol{\alpha}}_{(c)}$ is the optimal solution for the c-th subproblem.

4.3 Comparing BKA-SVM with Low-rank Kernel SVM Solvers(BKA-SVM)

We now compare block structure based kernel SVM solver–BKA-SVM with low-rank structure based kernel SVM solvers. All the experiments are conducted on an Intel 2.66GHz CPU with 8G RAM. We use 3 benchmark datasets as shown in Table 1. The thee datasets can be downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets` or the UCI data repository. We use a random 80%-20% split for covtype, and the original training/testing split for other datasets.

4.3.1 Competing Methods

We include the following exact kernel SVM solvers (LIBSVM), approximate low-rank SVM solvers (LLSVM, FastFood) in our comparison:

1. LIBSVM: the implementation in the LIBSVM library (Chang and Lin, 2011) with a small modification to handle SVM without the bias term – we observe that LIBSVM has similar test accuracy with/without bias. We also include the results for using LIBSVM with random 1/5 subsamples on each dataset in Table 2.

2. LLSVM: improved Nyström method for nonlinear SVM by (Wang et al., 2011). We solve the resulting linear SVM problem by the dual coordinate descent solver in LIBLINEAR (Hsieh et al., 2008).

3. FastFood: use random Fourier features to approximate the kernel function (Le et al., 2013). We solve the resulting linear SVM problem by the dual coordinate descent solver in LIBLINEAR.

Both LLSVM and FastFood use low-rank representation to approximate kernel matrix so that to speed up solving kernel SVM.

	ijcnn1		census		covtype	
	$C = 32, \gamma = 2$		$C = 512, \gamma = 2^{-9}$		$c = 32, \gamma = 32$	
	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)
BKA-SVM	**5**	98.49	838	**94.72**	**436**	96.05
LIBSVM	115	**98.69**	2920	94.2	83631	**96.15**
LIBSVM (subsample)	6	98.24	641	93.2	5330	92.46
LLSVM	38	98.23	1212	92.8	4451	84.21
FastFood	87	95.95	851	91.6	8550	80.1

Table 2: Comparison on real datasets using the RBF kernel.

4.3.2 Parameter Setting

We consider the RBF kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2)$. We use same kernel function for both training and test phases. We chose the balancing parameter C and kernel parameter γ by 5-fold cross validation on a grid of points: $C = [2^{-10}, 2^{-9}, \ldots, 2^{10}]$ and $\gamma = [2^{-10}, \ldots, 2^{10}]$ for ijcnn1, census, and covtype. For BKA-SVM, we set the number of clusters to be 64 for these three datasets. There is a tradeoff between the number of clusters and prediction accuracy. If we increase the number of clusters, BKA-SVM will become faster, but the prediction accuracy will mostly decrease. On the other hand, if the number of clusters is set smaller, BKA-SVM can achieve higher accuracy (in most cases), while takes more time to train. The following are parameter settings for other methods in Table 2: the rank is set to be 3000 in LLSVM; number of Fourier features is 3000 in Fastfood[1]; the tolerance in the stopping condition for LIBSVM is set to 10^{-3} (the default setting of LIBSVM).

Tables 2 present time taken and test accuracies. Experimental results show that the BKA-SVM achieves near-optimal test performance. Also we observe that BKA-SVM performs better than low-rank approximation based methods for kernel SVM problem showing the benefit of using block structure of kernel matrix.

We can see that BKA exploits the block structure of kernel matrix, and can speed up the training of kernel SVM. As shown in Tandon et al. (2016), BKA can also be used for speeding up kernel ridge regression problem. About the memory requirement, which is the main theme of this paper, BKA takes $O(\frac{n^2}{k})$ memory to approximate the kernel matrix, while popular low-rank based kernel approximation methods are more memory efficient, and only need linear memory to represent the kernel matrix.

5. Memory Efficient Kernel Approximation

There are two main drawbacks of the BKA approach: (i) it ignores all off-diagonal blocks, which results in large error when η is small (as seen in Figure 1(c)); (ii) for large-scale kernel approximation, it is too expensive to compute and store all the diagonal block entries. To

1. In Fastfood we control the number of blocks so that number of Fourier features is close to 3000 for each dataset.

overcome these two drawbacks, we propose to use low-rank representation for each block in the kernel matrix.

5.1 Low Rank Structure of Each Block

To motivate the use of low-rank representation in our proposed method, we first present the following bound:

Theorem 2 *Given data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$, and a partition $\mathcal{V}_1, \ldots, \mathcal{V}_c$, and assume f is Lipschitz continuous, then for any s, t ($s = t$ or $s \neq t$)*

$$\|G^{(s,t)} - G_k^{(s,t)}\|_F \leq 4Ck^{-1/d}\sqrt{|\mathcal{V}_s||\mathcal{V}_t|}\min(r_s, r_t),$$

where $G_k^{(s,t)}$ is the best rank-k approximation to $G^{(s,t)}$; C is the Lipschitz constant of the shift-invariant function f; r_s is the radius of the s-th cluster.

Proof To prove this theorem, we use the ϵ-net theorem in (Cucker and Smale, 2001), which states that when all the data $\boldsymbol{x}_i \in \mathbb{R}^d$, $i = 1, \cdots, n$ are in a ball with radius r, there exist $T = (\frac{4r}{\bar{r}})^d$ balls of radius $\bar{r}$ that cover all the data points. If we set T to be k, then $\bar{r} = k^{-1/d}4r$.

Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_s}$ be the data points in the s-th cluster, and let $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{n_t}$ be the data points in the t-th cluster, and $n_s = |\mathcal{V}_s|, n_t = |\mathcal{V}_t|$. Our goal is to show that $G^{(s,t)}$ is low-rank, where $G_{i,j}^{(s,t)} = K(\boldsymbol{x}_i, \boldsymbol{y}_j)$. Assume r_t is the radius of the t-th cluster, therefore we can find k balls with radius $\bar{r} = k^{-1/d}4r_t$ to cover $\{\boldsymbol{y}_j\}_{j=1}^{n_t}$.

Assume centers of the balls for t-th cluster are $\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_k$, then we can form a low-rank matrix $\bar{G}^{(s,t)} = \bar{U}\bar{V}^T$, where for all $i = 1, \ldots, n_s, j = 1, \ldots, n_t$, and $q = 1, \ldots, k$,

$$\bar{U}_{i,q} = K(\boldsymbol{x}_i, \boldsymbol{m}_q) \text{ and } \bar{V}_{j,q} = \begin{cases} 1 & \text{if } \boldsymbol{y}_j \in Ball(\boldsymbol{m}_q), \\ 0 & \text{otherwise.} \end{cases}$$

Assume $\boldsymbol{y}_j$ is in ball q, then

$$\begin{aligned}(G_{ij}^{(s,t)} - \bar{G}_{ij}^{(s,t)})^2 &= (f(\boldsymbol{x}_i - \boldsymbol{y}_j) - f(\boldsymbol{x}_i - \boldsymbol{m}_q))^2 \\ &\leq C^2\|(\boldsymbol{x}_i - \boldsymbol{y}_j) - (\boldsymbol{x}_i - \boldsymbol{m}_q)\|^2 \\ &= C^2\|\boldsymbol{y}_j - \boldsymbol{m}_q\|_2^2 \\ &\leq C^2\bar{r}^2.\end{aligned}$$

Therefore, if $(G^{(s,t)})^*$ is the best rank-k approximation for $G^{(s,t)}$, then

$$\|G^{(s,t)} - (G^{(s,t)})^*\|_F \leq \|G^{(s,t)} - \bar{G}^{(s,t)}\|_F \leq Ck^{-1/d}4r_t\sqrt{n_s n_t}. \tag{11}$$

Similarly, by covering s-th cluster $\{\boldsymbol{x}_i\}_{i=1}^{n_s}$ with k balls we can get the following inequality:

$$\|G^{(s,t)} - (G^{(s,t)})^*\|_F \leq Ck^{-1/d}4r_s\sqrt{n_s n_t}. \tag{12}$$

Combining (11) and (12) gives the results. ■

16	14	13	7	7
14	29	13	9	9
13	13	20	10	10
7	9	10	29	11
7	9	10	11	28

(a) k-means clustering.

139	99	101	44	45
99	116	86	43	44
101	86	131	46	47
44	43	46	47	45
45	44	47	45	49

(b) a random partition.

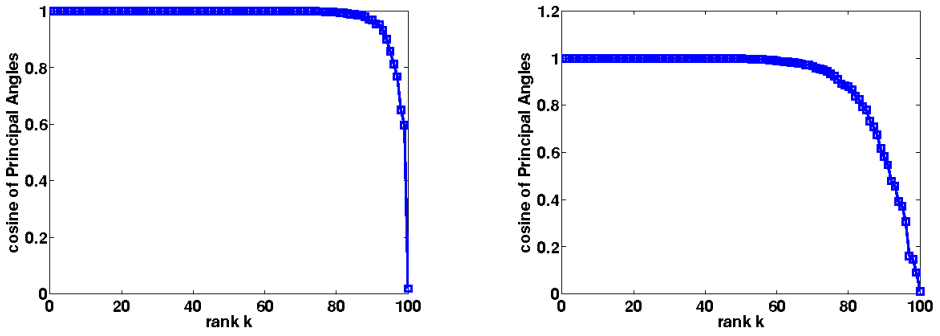
Table 3: Rank of each of the 5 blocks (from a subsampled ijcnn1 data set) using different partition strategies: (a) by k-means clustering; (b) by a random partition.

Theorem 2 suggests that each block(diagonal or off-diagonal block) of the kernel matrix will be low-rank if we find the partition by k-means in the input space and the radius of the cluster is small. In the following we present empirical confirmation of this result. In Table 3, we present the numerical rank of each block, where numerical rank for a m by n matrix A is defined as the number of singular values with magnitude larger than $\max(n, m)\|A\|_2\delta$ where δ is a small tolerance 10^{-6}. We sample 4000 data points from the ijcnn1 dataset and generate 5 clusters by k-means and random partition. Table 3a shows the numerical rank for each block using k-means, while Table 3b shows the numerical rank for each block when the partitions are random. We observe that by using k-means, the rank of each block is fairly small.

5.2 Memory Efficient Kernel Approximation (MEKA)

Based on the above observation, we propose a fast and memory efficient scheme to approximate shift-invariant kernel matrices. As suggested by Theorem 2, each block tends to be low-rank after k-means clustering; thus we can form a rank-k approximation for each of the c^2 blocks separately to achieve low error; however, this approach would require $O(cnk)$ memory, which can be prohibitive. Therefore, our proposed method first performs k-means clustering, and after rearranging the matrix according to clusters, it computes the low-rank basis only for diagonal blocks (which are more dominant than off-diagonal blocks) and uses them to approximate off-diagonal blocks. Empirically, we observe that the principal angles between the dominant singular subspaces of diagonal block and off-diagonal block are small (as shown in Figure 3). In Figure 3, we randomly sampled 1000 data points from the covtype and ijcnn1 datasets and generated 5 clusters by k-means for each dataset. The blue line shows the cosines of the principal angles between the dominant singular subspace of a diagonal block $G^{(s,s)}$ and that of an off-diagonal block $G^{(s,t)}$ for different ranks k, where s and t are randomly chosen. We can observe that most of the cosines are close to 1, showing that there is substantial overlap between the dominant singular subspaces of the diagonal and off-diagonal block.

By using our proposed approach, we focus on diagonal blocks, and spend less effort on the off-diagonal blocks. Assume the rank-k_s approximation of the s^{th} diagonal block is



(a) covtype (b) ijcnn1

Figure 3: The cosines of the principal angles between the dominant singular subspaces of diagonal block and off-diagonal block of a Gaussian kernel ((a): $\gamma = 0.1$ and 1000 random samples from covtype; (b): $\gamma = 1$ and 1000 random samples from ijcnn1) with respect to different ranks. The cosines of the principal angles are close to 1 showing that two subspaces are similar.

$W^{(s)}L^{(s,s)}(W^{(s)})^T$, we form the following memory-efficient kernel approximation:

$$\tilde{G} = WLW^T, \tag{13}$$

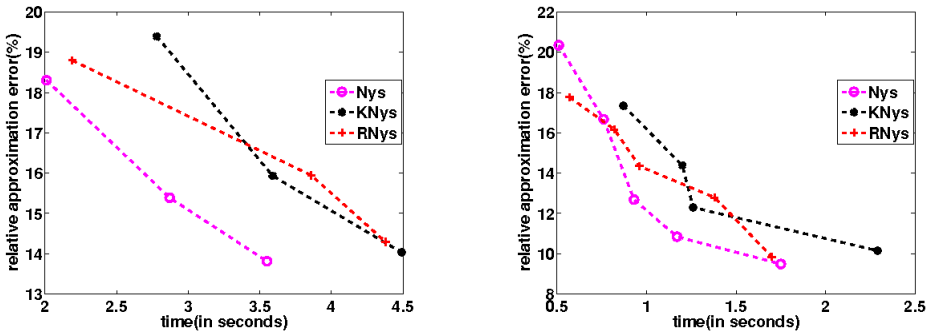
where W is a diagonal matrix as

$$W \equiv \begin{bmatrix} W^{(1)} & 0 & \dots & 0 \\ 0 & W^{(2)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & W^{(c)} \end{bmatrix}; \tag{14}$$

and L is a "link" matrix consisting of c^2 blocks, where each $k_s \times k_t$ block $L^{(s,t)}$ captures the interaction between the s^{th} and t^{th} clusters. For now, let us first assume $k_s = k \; \forall s$ (we will discuss different strategies to choose k_s later). Note that if we were to restrict L to be a block diagonal matrix, $\tilde{G}$ would still be a block diagonal approximation of G. However, we consider the more general case that L is dense. In this case, each off-diagonal block $G^{(s,t)}$ is approximated as $W^{(s)}L^{(s,t)}(W^{(t)})^T$, and this approximation is memory efficient as only $O(k^2)$ additional memory is required to represent the (s,t) off-diagonal block. If a rank-k approximation is used within each cluster, then the generated approximation has rank ck, but only needs a total of $O(nk + (ck)^2)$ storage.

Computing $W^{(s)}$. Since we aim to deal with dense kernel matrices of huge size, we use the standard Nyström approximation to compute low-rank "basis" for each diagonal block. When applying the standard Nyström method to a $n_s \times n_s$ block $G^{(s,s)}$, we sample m columns from $G^{(s,s)}$, evaluate their kernel values, compute the rank-k pseudo-inverse of an $m \times m$ matrix, and form $G^{(s,s)} \approx W^{(s)}L^{(s,s)}(W^{(s)})^T$. The time required per block is $O(n_s m(k+d) + m^3)$, and thus our method requires a total of $O(nm(k+d) + cm^3)$ time to form W. We can replace Nyström by any other low-rank approximation method discussed

in Section 2. In Figure 4 we compare using standard Nyström(Nys)(Williams and Seeger, 2001), k-means Nyström(KNys)(Zhang and Kwok, 2010), and Nyström with randomized SVD(RNys)(Li, Kwok, and Lu, 2010) to generate $W^{(s)}$ on both ijcnn1 and cadata datasets. As shown in Figure 4, we observe that the standard Nyström method combined with MEKA gives excellent performance.



(a) ijcnn1 (b) cadata

Figure 4: Comparison of using Nys, KNys, and RNys to obtain the basis $W^{(s)}$ for diagonal blocks in MEKA on ijcnn1 and cadata datasets. The x-axis shows the computation time and y-axis shows the relative kernel approximation error(%).

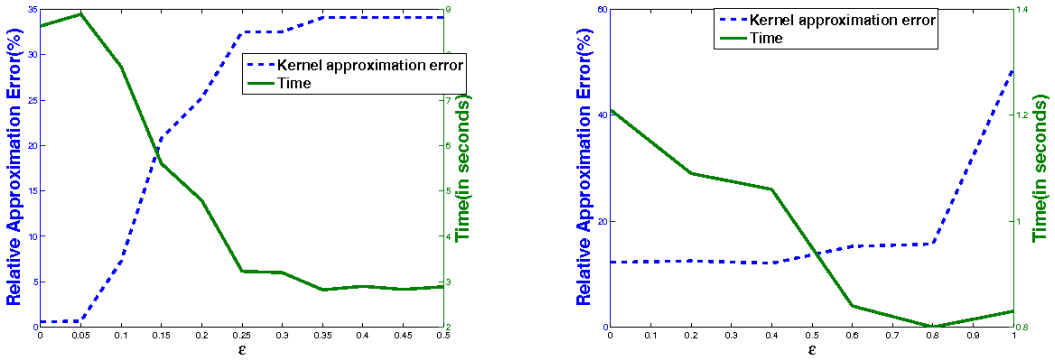
Computing $L^{(s,t)}$. The optimal least squares solution for $L^{(s,t)}(s \neq t)$ is the minimizer of the local approximation error $\|G^{(s,t)} - W^{(s)}L^{(s,t)}(W^{(t)})^T\|_F$. However, forming the entire $G^{(s,t)}$ block can be time consuming. For example, computing the whole kernel matrix for mnist2m with 2 million data points takes more than a week. Therefore, to compute $L^{(s,t)}$, we propose to randomly sample a $(1+\rho)k \times (1+\rho)k$ submatrix $\hat{G}^{(s,t)}$ from $G^{(s,t)}$, and then find $L^{(s,t)}$ that minimizes the error on this submatrix. If the row/column index set for the subsampled submatrix $\hat{G}^{(s,t)}$ in $G^{(s,t)}$ is v_s/v_t, then $L^{(s,t)}$ can be computed in closed form:

$$L^{(s,t)} = ((W^{(s)}_{v_s})^T W^{(s)}_{v_s})^{-1}(W^{(s)}_{v_s})^T \hat{G}^{(s,t)} W^{(t)}_{v_t}((W^{(t)}_{v_t})^T W^{(t)}_{v_t})^{-1},$$

where $W^{(s)}_{v_s}$ and $W^{(t)}_{v_t}$ are formed by the rows in $W^{(s)}$ and $W^{(t)}$ with row index sets v_s and v_t respectively.

Since there are only k^2 variables in $L^{(s,t)}$, we do not need too many samples for each block, and the time to compute $L^{(s,t)}$ is $O((1+\rho)^3k^3)$. In practice, we observe that setting ρ to be 2 or 3 is enough for a good approximation, so the time complexity is $O(k^3)$. Empirically, many values in the off-diagonal blocks are close to zero, and only a few of them have large values as shown in Figure 1. Based on this observation, we further propose a thresholding technique to reduce the time for storing and computing $L^{(s,t)}$. Since the distance between cluster centers is a good indicator for the values in an off-diagonal block, we can set the whole block $L^{(s,t)}$ to 0 if $K(\boldsymbol{m}_s, \boldsymbol{m}_t) \leq \epsilon$ for some thresholding parameter $\epsilon > 0$, where $\boldsymbol{m}_s$ and $\boldsymbol{m}_t$ are the k-means centroid for the s-th and t-th cluster respectively. Obviously, to choose ϵ, we need to achieve a balance between speed and accuracy. When ϵ is small, we will approximate more off-diagonal blocks; while when ϵ is large, we will set more off-diagonal blocks to be 0, but increase the approximation error. We test the influence

of thresholding parameter ϵ on the ijcnn1 and cadata data in Figure 5. When ϵ is large, although MEKA yields higher approximation error(because it omits more off-diagonal information), it is faster. On the other hand, for small ϵ, when more off-diagonal information is considered, we notice an increase in time and smaller in approximation error. In practice, we need to use cross-validation to select ϵ.

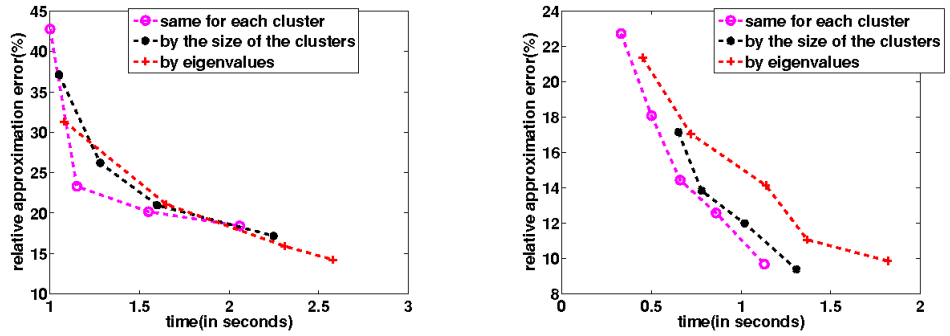


(a) ijcnn1 (b) cadata

Figure 5: Time (in seconds) and kernel approximation quality of MEKA when varying the thresholding parameter ϵ for setting off-diagonal blocks in L to be zero.

Choosing the rank k_s for each cluster. We need to decide the rank for the $s^{th}(s = 1, \cdots, c)$ cluster, k_s, which can be done in different ways: (i) the same k for all the clusters; (ii) k_s is proportional to the size of s^{th} cluster; (iii) eigenvalues based approach. For (iii), suppose $M^{(s)}$ is the $m_s \times m_s$ matrix consisting of the intersection of m_s sampled columns in $G^{(s,s)}$, and M is the $cm \times cm (\sum_{s=1}^{c} m_s = cm)$ block-diagonal matrix with $M^{(s)}$ as diagonal block. We can choose k_s such that the set of top-ck eigenvalues of M is the union of the eigenvalues of $M^{(s)}$ in each cluster, that is, $[\sigma_1(M), \ldots, \sigma_{ck}(M)] = \cup_{s=1}^{c}[\sigma_1(M^{(s)}), \ldots, \sigma_{k_s}(M^{(s)})]$. To use (iii), we can oversample points in each cluster, e.g., sample $2k$ points from each cluster, perform eigendecompostion of a $2k \times 2k$ kernel matrix, sort the eigenvalues from c clusters, and finally select the top-ck eigenvalues and their corresponding eigenvectors. Comparing these three strategies, (i) achieves lower memory usage and is fastest, and (ii) is more accurate than (i) with more memory usage, while (iii) is slowest but achieves lower error for diagonal blocks. In Figure 6, we compare these three sampling strategies to choose k_s on ijcnn1 and cadata datasets. It is shown that these three methods perform similarly well and choosing k_s to be same for each cluster performs sightly better than by the size of each cluster and singular values based approach. In the experiment, we set all the clusters to have the same rank k. We show that this simple choice of k_s already outperforms state-of-the-art kernel approximation methods.

We are now ready to present our main algorithm in Algorithm 1. In Table 4, we compare the time and space complexity for our method with SVD, standard Nyström, and RKS. We can see that MEKA is more memory efficient. For the time complexity, both T_L (time for computing off-diagonal L) and T_C (time for clustering) are small as (1) we use thresholding to force some $L^{(s,t)}$ blocks to be zero, and perform least squares on small blocks, which



(a) ijcnn1 (b) cadata

Figure 6: Comparison of different strategies to choose the rank k_s of each cluster in MEKA on ijcnn1 and cadata datasets.

Method	Storage	Rank	Time Complexity
RKS (Rahimi and Recht, 2008)	$O(cnk)$	ck	$O(cnkd)$
Nyström(Williams and Seeger, 2001)	$O(cnk)$	ck	$O(cnm(ck+d)+(cm)^3)$
SVD	$O(cnk)$	ck	$O(n^3+n^2d)$
MEKA	$O(nk+(ck)^2)$	ck	$O(nm(k+d)+cm^3+T_L+T_C)$

Table 4: Memory and time analysis of various kernel approximation methods, where T_L is the time to compute the matrix L and T_C is the time for clustering in MEKA.

means T_L can at most be $O(\frac{1}{2}c^2k^3)$; (2)T_C is proportional to the number of samples. For a large dataset, we sample 20000 points for k-means, and thus the clustering is more efficient than working on the entire data set.

Algorithm 1 Memory Efficient Kernel Approximation (MEKA)

Input : Data points $\{(\boldsymbol{x}_i)\}_{i=1}^n$, scaling parameter γ, rank k, and no. of clusters c.

Output: The rank-ck approximation $\tilde{G} = WLW^T$ using $O(nk+(ck)^2)$ space

Generate the partition $\mathcal{V}_1, \ldots, \mathcal{V}_c$ by k-means;

for $s = 1, \ldots, c$ **do**

  Perform the rank-k approximation $G^{(s,s)} \approx W^{(s)}L^{(s,s)}(W^{(s)})^T$ by standard Nyström;

end

forall $(s,t)(s \neq t)$ **do**

  Sample a submatrix $\bar{G}^{(s,t)}$ from $G^{(s,t)}$ with row index set v_s and column index set v_t;

  Form $W_{v_s}^{(s)}$ by selecting the rows in $W^{(s)}$ according to index set v_s;

  Form $W_{v_t}^{(t)}$ by selecting the rows in $W^{(t)}$ according to index set v_t;

  Solve the least squares problem: $\bar{G}^{(s,t)} \approx W_{v_s}^{(s)}L^{(s,t)}(W_{v_t}^{(t)})^T$ to obtain $L^{(s,t)}$;

end

Dataset	T_C	T_W	T_L
pendigit	0.05	0.69	0.35
ijcnn1	0.15	1.27	0.84
covtype	1.83	9.82	12.23

Table 5: Time (in seconds) for each step of MEKA, where T_W is the time to compute low-rank approximation for the diagonal block matrices; T_L is the time to form the "link" matrix L; T_C is the time for performing k-means clustering.

In Table 5, we show the time cost for each step of MEKA on pendigit, ijcnn1, and covtype datasets. The execution time of our proposed algorithm mainly consists of three parts:(1) time for performing k-means clustering (T_C), (2) time for forming the "basis" W from the diagonal blocks (T_W), (3) time to compute the link matrix L from off-diagonal blocks (T_L). From Table 5, we observe that compared with T_W and T_L, T_C is fairly small and T_W dominates the whole process in most cases. For covtype data set, since we choose c to be large, T_L is sightly larger than T_W. We will analyze T_W, T_L, and T_C for different c in the experiment part.

5.3 Analysis

We now bound the approximation error for our proposed method. We show that when $\sigma_{k+1} - \sigma_{ck+1}$ is large, where σ_{k+1} and σ_{ck+1} are the $k+1^{st}$ and $ck+1^{st}$ singular values of G respectively, and entries in off-diagonal blocks are small, MEKA has a better approximation error bound compared to standard Nyström that uses similar storage.

Theorem 3 *Let Δ denote a matrix consisting of all off-diagonal blocks of G, so $\Delta^{(s,t)} = G^{(s,t)}$ for $s \neq t$ and all zeros when $s = t$. We sample cm points from the dataset uniformly at random without replacement and split them according to the partition from k-means, such that each cluster has m_s benchmark points and $\sum_{s=1}^{c} m_s = cm$. Let G_{ck} be the best rank-ck approximation of G, and $\tilde{G}$ be the rank-ck approximation from MEKA. Suppose we choose the rank k_s for each diagonal block using the eigenvalue based approach as mentioned in Section 5.1, then with probability at least $1-\delta$, the following inequalities hold for any sample of size cm:*

$$\|G - \tilde{G}\|_2 \leq \|G - G_{ck}\|_2 + \frac{1}{\sqrt{c}} \frac{2n}{\sqrt{m}} G_{\max}(1+\theta) + 2\|\Delta\|_2,$$

$$\|G - \tilde{G}\|_F \leq \|G - G_{ck}\|_F + \left(\frac{64k}{m}\right)^{\frac{1}{4}} n G_{\max}(1+\theta)^{\frac{1}{2}} + 2\|\Delta\|_F$$

where $\theta = \sqrt{\frac{n-m}{n-0.5} \frac{1}{\beta(m,n)} \log \frac{1}{\delta}} d^G_{\max} / G^{\frac{1}{2}}_{\max}$; $\beta(m,n) = 1 - \frac{1}{2\max\{m,n-m\}}$; $G_{\max} = \max_i G_{ii}$; and $d^G_{\max}$ represents the distance $\max_{ij} \sqrt{G_{ii} + G_{jj} - 2G_{ij}}$.

Proof Let B denote the matrix formed by the diagonal block of G, that is,

$$B \equiv \begin{bmatrix} G^{(1)} & 0 & \dots & 0 \\ 0 & G^{(2)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & G^{(c)} \end{bmatrix}. \tag{15}$$

According to the definition of Δ, $G = B + \Delta$. In MEKA, the error $\|\tilde{G} - G\|_2$ consists of two components,

$$\|\tilde{G} - G\|_2 = \|\tilde{B} - B + (\tilde{\Delta} - \Delta)\|_2 \leq \|\tilde{B} - B\|_2 + \|\tilde{\Delta} - \Delta\|_2 \tag{16}$$

where $\tilde{B}$ and $\tilde{\Delta}$ are the approximations for B and Δ in MEKA respectively.

Let us first consider the error in approximating the diagonal blocks $\|\tilde{B} - B\|_2$. Since we sample cm benchmark points from n data points uniformly at random without replacement and distribute them according to the partition coming from k-means, the s-th cluster now has m_s benchmark points with $\sum_{s=1}^{s=c} m_s = cm$. For the s-th diagonal block $G^{(s)}$, we will perform the rank-k_s approximation using standard Nyström, so we have $G^{(s)} \approx E^{(s)}(M_{k_s}^{(s)})^+ E^{(s)}$, where $E^{(s)}$ denotes the matrix formed by m_s sampled columns from $G^{(s)}$ and $M_{k_s}^{(s)}$ is a $m_s \times m_s$ matrix consisting of the intersection of sampled m_s columns.

Suppose we use the singular value based approach to choose k_s for s-th cluster as described in Section 5.1, and

$$M_{ck}^{+} equiv \begin{bmatrix} (M_{k_1}^{(1)})^+ & 0 & \dots & 0 \\ 0 & (M_{k_2}^{(2)})^+ & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & (M_{k_c}^{(c)})^+, \end{bmatrix} \tag{17}$$

where M is the $cm \times cm$ block diagonal matrix that consists of the intersection of the sampled cm columns. Then we can see that approximating the diagonal blocks B is equivalent to directly performing standard Nyström on B by sampling cm benchmark points uniformly at random without replacement to achieve the rank-ck approximation. The standard Nyström's norm-2 and Frobenius error bound are given in (Kumar et al., 2009), so $\|B - \tilde{B}\|_2$ can be bounded with probability at least $1 - \delta$ as

$$\|B - \tilde{B}\|_2 \leq \|B - B_{ck}\|_2 + \frac{2n}{\sqrt{cm}} B_{\max}[1 + \sqrt{\frac{n - cm}{n - 0.5} \frac{1}{\beta(cm, n)} \log \frac{1}{\delta}} d_{\max}^B / B_{\max}^{\frac{1}{2}}],$$

where B_{ck} denotes the best rank-ck approximation to B; $B_{\max} = \max_i B_{ii}$; $d_{\max}^B$ represents the distance $\max_{ij} \sqrt{B_{ii} + B_{jj} - 2B_{ij}}$.

To bound $\|\tilde{\Delta} - \Delta\|_2$, recall that some off-diagonal blocks in MEKA are set to 0 by thresholding and $\mathbf{0}$ is one special solution of least squares problem to compute $L^{(s,t)}$, we have $\|\tilde{\Delta} - \Delta\|_2 \leq \|\Delta\|_2$.

Furthermore, according to perturbation theory (Stewart and Ji-Guang, 1990), we have

$$\|B - B_{ck}\|_2 \leq \|G - G_{ck}\|_2 + \|\Delta\|_2. \tag{18}$$

The inequality in (16) combined with (18) gives a bound on $\|\tilde{G} - G\|_2$ as,

$$\begin{aligned}
\|\tilde{G} - G\|_2 &\leq \|B - B_{ck}\|_2 + \|\Delta\|_2 + \frac{2n}{\sqrt{cm}} B_{\max}[1 + \sqrt{\frac{n-cm}{n-0.5}\frac{1}{\beta(cm,n)}\log\frac{1}{\delta}} d^B_{\max}/B^{\frac{1}{2}}_{\max}] \\
&\leq \|G - G_{ck}\|_2 + 2\|\Delta\|_2 + \frac{2n}{\sqrt{cm}} B_{\max}[1 + \sqrt{\frac{n-cm}{n-0.5}\frac{1}{\beta(cm,n)}\log\frac{1}{\delta}} d^B_{\max}/B^{\frac{1}{2}}_{\max}] \\
&\leq \|G - G_{ck}\|_2 + 2\|\Delta\|_2 + \frac{2n}{\sqrt{cm}} G_{\max}[1 + \sqrt{\frac{n-cm}{n-0.5}\frac{1}{\beta(cm,n)}\log\frac{1}{\delta}} d^G_{\max}/G^{\frac{1}{2}}_{\max}] \\
&\leq \|G - G_{ck}\|_2 + 2\|\Delta\|_2 + \frac{1}{\sqrt{c}}\frac{2n}{\sqrt{m}} G_{\max}[1 + \sqrt{\frac{n-m}{n-0.5}\frac{1}{\beta(m,n)}\log\frac{1}{\delta}} d^G_{\max}/G^{\frac{1}{2}}_{\max}],
\end{aligned}$$

where G_{ck} denotes the best rank-ck approximation to G; $G_{\max} = \max_i G_{ii}$; $d^G_{\max}$ represents the distance $\max_{ij}\sqrt{G_{ii} + G_{jj} - 2G_{ij}}$. The third inequality is because $G = B + \Delta$, $B_{\max} \leq G_{\max}$ and $d^B_{\max} \leq d^G_{\max}$. The last inequality is because $n \gg m$ and $n \gg cm$.

Similarly by using perturbation theory and upper bounds for the Frobenius error of standard Nyström, the result follows. ■

When $k_s (s = 1, \cdots, c)$ is balanced (meaning k_s is approximately the same for each cluster) and n is large, MEKA provides a rank-ck approximation using roughly the same amount of storage as rank-k approximation by standard Nyström. Interestingly, from Theorem 3, if $\|G - G_k\|_2 - \|G - G_{ck}\|_2 \geq 2\|\Delta\|_2$, then

$$\|G - \tilde{G}\|_2 \leq \|G - G_k\|_2 + \frac{1}{\sqrt{c}}\frac{2n}{\sqrt{m}} G_{\max}(1 + \theta).$$

The second term in the right hand side of above inequality is only $\frac{1}{\sqrt{c}}$ of that in the spectral norm error bound for standard Nyström that uniformly samples m columns without replacement in G to obtain the rank-k approximation as shown in (Kumar, Mohri, and Talwalkar, 2009). Thus, if there is a large enough gap between σ_{k+1} and σ_{ck+1}, the error bound for our proposed method is better than standard Nyström that uses similar storage. Furthermore, when γ is large, G tends to have better clustering structure, suggesting in Theorem 3 that $\|\Delta\|$ is usually quite small. Note that when using the same rank k for all the clusters, the above bound can be worse because of some extreme cases, e.g., all the top-ck eigenvalues are in the same cluster. In practice we do not observe those extreme situations. We also want to mention that both $\|\Delta\|_F$ and $\|\Delta\|_2$ will be affected by the number of clusters c.

6. Experimental Results

In this section, we empirically demonstrate the benefits of our proposed method, MEKA on various data sets[2] that are listed in Table 6. Experiment results here are mainly based on

2. All the datasets are downloaded from `www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets` and UCI repository (Bache and Lichman, 2013). Note that the census datasets in Table 1 and Table 6 come from different sources.

Dataset	n	d	Dataset	n	d
wine	6,497	11	census	22,784	137
cpusmall	8,192	12	ijcnn1	49,990	22
pendigit	10,992	16	covtype	581,012	54
cadata	20,640	8	mnist2m	2,000,000	784

Table 6: Data set statistics (n: number of samples; d: dimension of samples).

the Gaussian kernel, but we observe similar behavior on other shift-invariant kernels (see Section 6.2). Note that the same kernel function is used in both training and test phases. We compare our method with six state-of-the-art kernel approximation methods:

1. The standard Nyström method (denoted by Nys)(Williams and Seeger, 2001). In the experiment, we uniformly sample $2k$ columns of G without replacement, and run Nyström for rank-k approximation.

2. Kmeans Nyström (denoted by KNys)(Zhang and Kwok, 2010), where the landmark points are the cluster centroids. As suggested in (Zhang et al., 2012), we sample 20000 points for clustering when the total number of data samples is larger than 20000.

3. Random Kitchen Sinks (denoted by RKS)(Rahimi and Recht, 2008), which approximates the shift-invariant kernel based on its Fourier transform.

4. Fastfood with "Hadamard features" (denoted by fastfood)(Le, Sarlos, and Smola, 2013).

5. Ensemble Nyström (denoted by ENys) (Kumar, Mohri, and Talwalkar, 2009). Due to concern for the computation cost, we set the number of "experts" in ENys 3.

6. Nyström using randomized SVD (denoted by RNys)(Li, Kwok, and Lu, 2010). We set the number of power iterations $q = 1$ and oversampling parameter $p = 10$.

We compare all the methods on two different tasks: kernel low-rank approximation and kernel ridge regression. We do not compare with BKA in this section, because (1) the approximation error of BKA is large when γ is small (as shown in Figure 1); (2) BKA is time consuming because it needs to compute all the diagonal blocks' kernel values, which needs $O(\frac{n^2 d}{c})$ time; (3) BKA is not memory efficient to approximate the kernel matrix, because it needs $O(\frac{n^2}{c})$ space to store the approximation. All the experiments are conducted on a machine with an Intel Xeon X5440 2.83GHz CPU and 32G RAM.

6.1 Kernel Approximation Quality

We now compare the kernel approximation quality for the above methods.

Main results. The kernel approximation results are shown in Figure 7 and Table 7. We use relative kernel approximation error $\|G - \tilde{G}\|_F / \|G\|_F$ to measure the quality. We randomly sampled 20000 rows of G to evaluate the relative approximation error for ijcnn1 and covtype. In Figure 7, we show the kernel approximation performance of different methods by varying k and γ. The rank (k) varies from 100 to 600 for ijcnn1 and covtype

Dataset	k	γ	c	Nys	RNys	KNys	ENys	RKS	fastfood	MEKA
pendigit	128	2	5	0.1325	0.1361	0.0828	0.2881	0.4404	0.4726	**0.0811**
ijcnn1	128	1	10	0.0423	0.0385	0.0234	0.1113	0.2972	0.2975	**0.0082**
covtype	256	10	15	0.3700	0.3738	0.2752	0.5646	0.8825	0.8920	**0.1192**

Table 7: Comparison of approximation error of our proposed method with six other state-of-the-art kernel approximation methods on real datasets, where γ is the Gaussian scaling parameter; c is the number of clusters in MEKA; k is the rank of each diagonal-block in MEKA and the rank of the approximation for six other methods. Note that for a given k, every method has roughly the same amount of memory. All results show relative kernel approximation errors for each k.

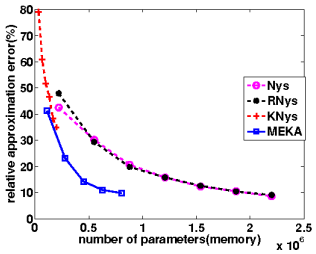
and from 20 to 200 for the pendigit data. Figure 7 shows that our proposed approximation scheme always achieves lower error with less time and memory. The main reason is that using similar amount of time and memory, our method aims to approximate the kernel matrix by a rank-ck approximation, while all other methods are only able to form a rank-k approximation.

In Table 7, we fix the rank k and γ, so that each method has the same memory usage of low-rank representation, and compare MEKA with them in terms of relative approximation error. As it can be seen, under the same amount of memory, our proposed method consistently yields lower approximation error than other methods.

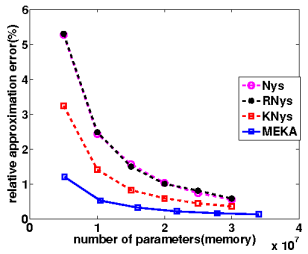
Also as we can see from Table 7 and Figure 8, Nyström based methods perform much better than random features based methods (including RKS and Fastfood here) in terms of kernel approximation quality. Therefore we do not show their performance in Figure 7, so that we could see the difference between MEKA and other Nyström based methods.

Robustness to the Gaussian scaling parameter γ. To show the robustness of our proposed algorithm with different γ as explained in Section 3, we test its performance on the ijcnn1 (Figure 9a), cadata (Figure 9b) and sampled covtype datasets (Figure 1c). The relative approximation errors for different γ values are shown in the figures using a fixed amount of memory. For large γ, the kernel matrix tends to have a block structure, so our proposed method yields lower error than other methods. The gap becomes larger as γ increases. Interestingly, Figure 1c shows that the approximation error of MEKA is superior to even the exact SVD, as it is much more memory efficient. Even for small γ where the kernel exhibits low-rank structure, our proposed method performs better than Nyström based methods, suggesting that it can get the low-rank structure of the kernel matrix.

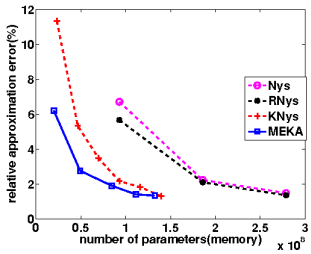
Robustness to the number of clusters c. Compared with Nyström, one main extra parameter for our method is the number of clusters c. In Figure 10, we test our method with different values of c on ijcnn1 dataset. In Figure 10a, we show how the approximation error changes when c is varied from 5 to 25 (with rank $k = 100$ in each cluster). The memory usage is $nk + (ck)^2$, so the storage increases as c increases. For a fair comparison, we increase the rank of other methods as c increases, so that all the methods use the same amount of memory. Figure 10a shows that the performance of our proposed method for varying choices of c. We can observe that under different c, MEKA always performs better than others. Furthermore, in Figure 10b we show the time cost for each step of MEKA on



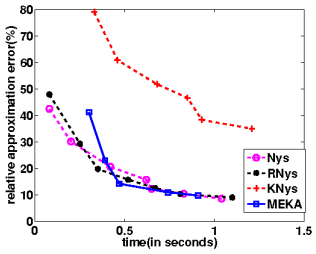
(a) pendigit ($\gamma = 2$), memory vs approx. error.



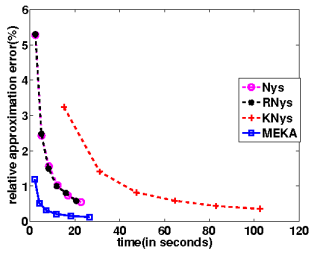
(b) ijcnn1 ($\gamma = 1$), memory vs approx. error.



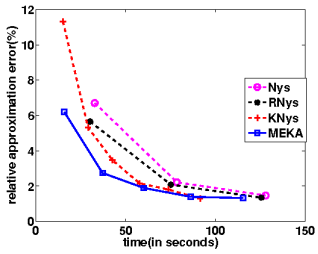
(c) covtype ($\gamma = 1$), memory vs approx. error.



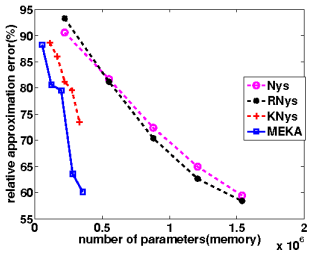
(d) pendigit ($\gamma = 2$), time vs approximation error.



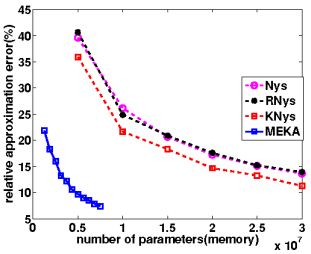
(e) ijcnn1 ($\gamma = 1$), time vs approximation error.



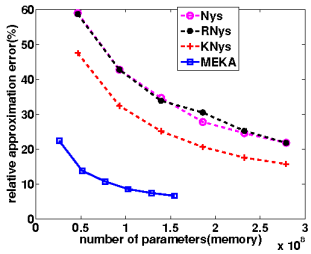
(f) covtype ($\gamma = 1$), time vs approximation error.



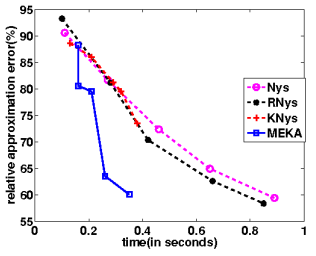
(g) pendigit ($\gamma = 10$), memory vs approximation error.



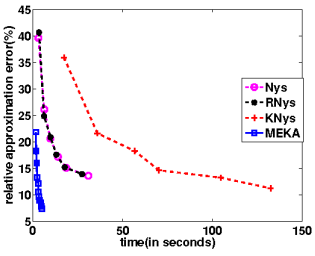
(h) ijcnn1 ($\gamma = 10$), memory vs approximation error.



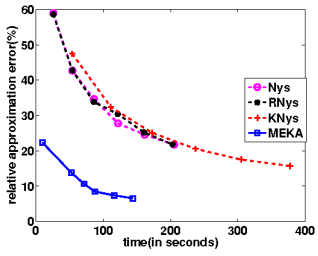
(i) covtype ($\gamma = 10$), memory vs approximation error.



(j) pendigit ($\gamma = 10$), time vs approximation error.



(k) ijcnn1 ($\gamma = 10$), time vs approximation error.



(l) covtype ($\gamma = 10$), time vs approximation error.

Figure 7: Low-rank Gaussian kernel approximation results. Methods with approximation error above the top of y-axis are not shown.

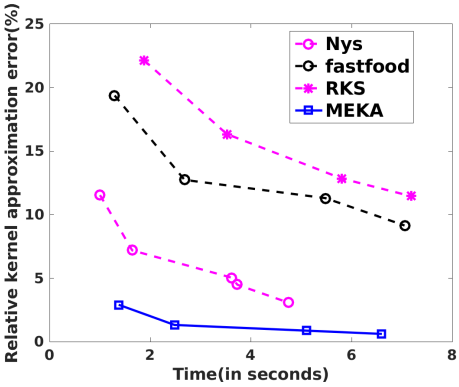
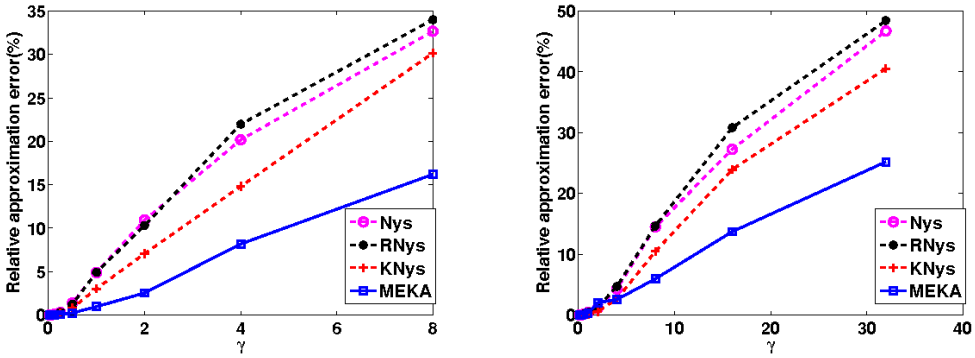


Figure 8: Comparison between Nyström based methods (MEKA and standard Nyström) and random feature based methods (RKS and fastfood). We can see that Nyström based methods perform much better than random feature based methods for kernel approximation.



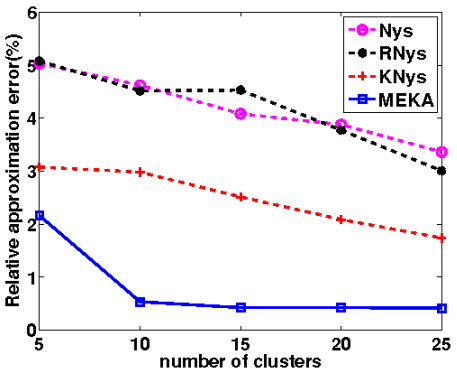
(a) ijcnn1. (b) cadata.

Figure 9: The kernel approximation errors for different Gaussian scaling parameter γ.

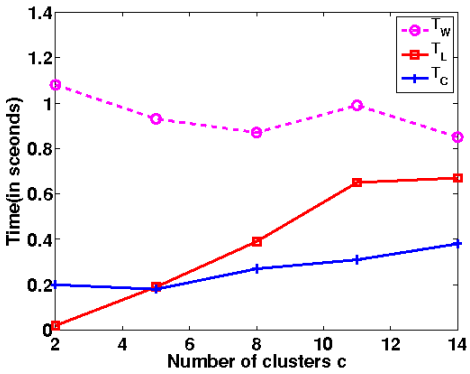
ijcnn1 dataset when varying the number of clusters c. Here the parameter γ is set to be 1. From Figure 10b, we observe that when the number of clusters c is small, T_W will dominate the whole process. As c increases, the time for computing the link matrix L, T_L, increases. This is because the number of off-diagonal blocks increases quadratically with c. Since the time complexity for k-means is O(ncd), T_C increases linearly with c.

6.2 The Performance of MEKA for Approximating the Laplacian Kernel

As we mentioned in Section 5, our algorithm is suitable for approximating stationary kernels. Besides Gaussian kernel, we can apply MEKA to approximate other kernels, e.g., Laplacian kernel ($K(x, y) = e^{-\gamma\|x-y\|_1}$). Figure 11 compares our proposed method with the standard Nyström(Nys), Randomized Nyström(RNys), and Kmeans Nyström(KNys) for approximating the Laplacian kernel on ijcnn1 data. Similar to Gaussian kernel, we observe that MEKA is more memory efficient and faster than other methods for approximating the Laplacian kernel.

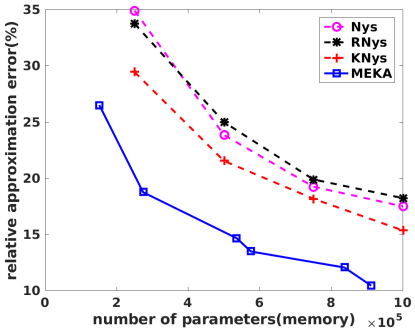


(a) kernel approximation under different numbers of clusters c in MEKA. For standard Nyström(Nys), Randomized Nyström(RNys), and Kmeans Nyström(KNys), we use the same memory with MEKA.

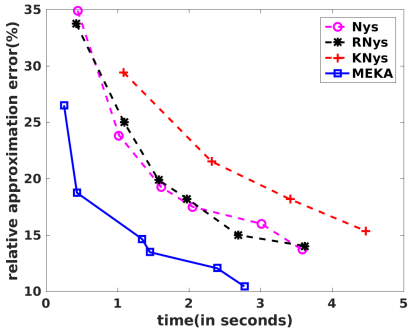


(b) Time for each step of MEKA when varying the number of clusters c. T_C is the time for performing k-means clustering; T_W is the time to form the "basis" W from the diagonal blocks; and T_L is the time to compute L from off-diagonal blocks.

Figure 10: The kernel approximation errors and time cost for each step of MEKA when varying c on ijcnn1 dataset.



(a) memory vs approximation error.



(b) time vs approximation error.

Figure 11: Low-rank Laplacian kernel approximation results for ijcnn1 dataset.

6.3 Kernel Ridge Regression

Next we compare the performance of various methods on kernel ridge regression (Saunders, Gammerman, and Vovk, 1998):

$$\max_{\boldsymbol{\alpha}} \quad \lambda \boldsymbol{\alpha}^T \boldsymbol{\alpha} + \boldsymbol{\alpha}^T G \boldsymbol{\alpha} - 2\boldsymbol{\alpha}^T \boldsymbol{y}, \tag{19}$$

where G is the kernel matrix formed by training samples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l\}$, and $\boldsymbol{y} \in \mathbb{R}^l$ are the targets. For each kernel approximation method, we first form the approximated kernel $\tilde{G}$, and then solve (19) by conjugate gradient (CG). The main computation in CG is the matrix vector product $\tilde{G}\boldsymbol{v}$. Using low-rank approximation, this can be computed using $O(nk)$ flops. For our proposed method, we compute $WLW^T\boldsymbol{v}$, where $W^T\boldsymbol{v} = \sum_{s=1}^{c} W^{(s)}\boldsymbol{v}^{(s)}$ requires

Dataset	k	γ	c	λ	Nys	RNys	KNys	ENys	RKS	fastfood	MEKA
wine	128	2^{-10}	3	2^{-4}	0.7514	0.7555	0.7568	0.7732	0.7459	0.7509	**0.7375**
cadata	128	2^{2}	5	2^{-3}	0.1504	0.1505	0.1386	0.1462	0.1334	0.1502	**0.1209**
cpusmall	256	2^{2}	5	2^{-4}	8.8747	8.6973	6.9638	9.2831	9.6795	10.2601	**6.1130**
census	256	2^{-4}	5	2^{-5}	0.0679	0.0653	0.0578	0.0697	0.0727	0.0732	**0.0490**
covtype	256	2^{2}	10	2^{-2}	0.8197	0.8216	0.8172	0.8454	0.8011	0.8026	**0.7106**
mnist2m	256	2^{-5}	40	2^{-5}	0.2985	0.2962	0.2725	0.3018	0.3834	na	**0.2667**

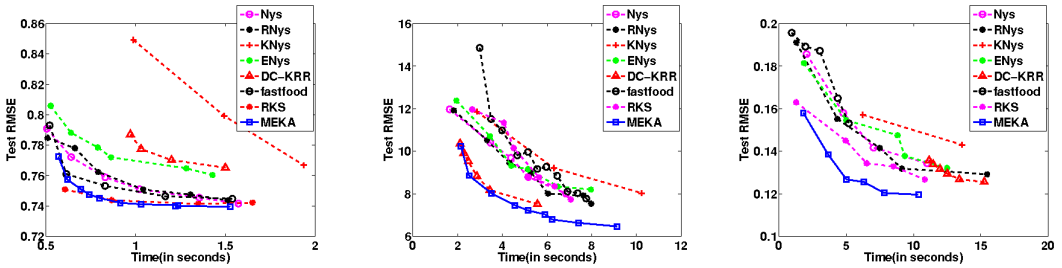
Table 8: Comparison of our proposed method with six other state-of-the-art kernel approximation methods on real datasets for kernel ridge regression, where λ is the regularization constant. All the parameters are chosen by cross validation, and every method has roughly the same amount of memory as in Table 7. All results show test RMSE for regression for each k. Note that k for fastfood needs to be larger than d, so we cannot test fastfood on mnist2m when $k = 256$.

$O(nk)$ flops, $L(W\boldsymbol{v})$ requires $O(\|L\|_0)$ flops, and $W(LW^T\boldsymbol{v})$ requires $O(nk)$ flops. Therefore, the time complexity for computing the matrix vector product for both MEKA and low-rank approximation methods is proportional to the memory for storing the approximate kernel matrices. Besides kernel approximation algorithms, we compare our method with another divide-and-conquer based kernel ridge regression method (denoted DC-KRR)(Zhang et al., 2013). The basic idea in Zhang et al. (2013) is to randomly partition the data into c parts and then train a kernel ridge regression model in each partition. To test a new data point, it will be tested on each submodel and the final prediction is the average of c predictions.

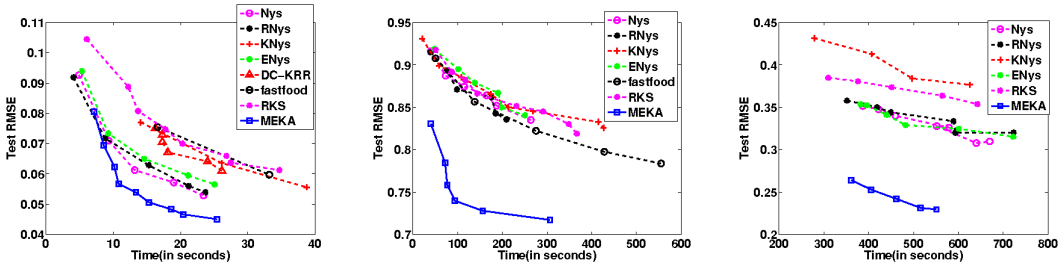
The parameters are chosen by five fold cross-validation and shown in Table 8. The rank for these algorithms is varied from 100 to 1000. The test root mean square error (test RMSE) is defined as $\|\boldsymbol{y}^{te} - G^{te}\boldsymbol{\alpha}\|$, where $\boldsymbol{y}^{te} \in \mathbb{R}^u$ is testing labels and $G^{te} \in \mathbb{R}^{u\times l}$ is the approximate kernel values between testing and training data. The covtype and mnist2m data sets are not originally designed for regression, and here we set the target variables to be 0 and 1 for mnist2m and -1 and 1 for covtype. Table 8 compares the kernel ridge regression performance of our proposed scheme with six other methods given the same amount of memory or same k in terms of test RMSE. It shows that our proposed method consistently performs better than other methods. Figure 12 shows the time usage of different methods for regression by varying the memory or rank k. As we can see that using the same amount of time, our proposed algorithm always achieves the lowest test RMSE. The total running time consists of the time for obtaining the low-rank approximation and time for regression. The former depends on the time complexity for each method, and the latter depends on the memory requirement to store the low-rank matrices. As shown in the previous experiment, MEKA is faster than the other methods while achieving lower approximation error and using less memory. As a consequence, it achieves lower test RMSE in less time compared to other kernel approximation methods.

7. Conclusions and Discussions

In this paper, we have proposed a novel framework, Memory Efficient Kernel Approximation (MEKA) for approximating shift-invariant kernel matrices. We observe that the structure of the shift-invariant kernel matrix changes from low rank to block diagonal as the scale



(a) wine, time vs regression error. (b) cpusmall, time vs regression error. (c) cadata, time vs regression error.



(d) census, time vs regression error. (e) covtype, time vs regression error. (f) mnist2m, time vs regression error.

Figure 12: Kernel ridge regression results for various data sets. Methods with regression error above the top of y-axis are not shown. All results are averaged over five independent runs.

parameter is changed. Our method exploits both low-rank and block structure present in the kernel matrix, and thus performs better than previously proposed low-rank based methods in terms of approximation and regression error, speed and memory usage. The code for MEKA is available at `www.cs.utexas.edu/~ssi/meka/`. We will discuss next about some typical problems encountered when using MEKA and how to deal with them.

7.1 Dealing with the Non-PSD Issue

If for each off-diagonal block, we sample all the entries, the resulting approximate matrix in MEKA will be positive semidefinite(PSD). The reason is as follow: if we use all the all-diagonal blocks to compute the link matrix L, the approximation will be $G \approx WLW^T$ with $L = W^T G W$. Since G is PSD, so it is with L, which proves WLW^T will be PSD.

However, due to the sampling procedure in MEKA, the resulting approximate matrix might not be PSD, which will cause some problems when PSD is required for some applications, e.g., kernel SVM with hinge loss. There are two simple and effective ways to solve this issue: (1) The first method (MEKA-PSD) is to set negative eigenvalues to 0. The procedure is first to perform eigen-decomposition on the small $ck \times ck$ "link" matrix $L = USU^T$(where U and S are the eigenvector and eigenvalue matrices for L respectively) in the MEKA representation, and then shrink its negative eigenvalues to be 0, which forms the new eigenvalue matrix $\bar{S}$ for L. After that, the new MEKA approximation will become: $\bar{G} = WU\bar{S}U^TW^T$. Since $\bar{S}$ is now PSD, so it is with $WU\bar{S}U^TW^T$. Therefore, $\bar{G}$ will be PSD after above shrinking operation. Due to eigen-decomposition on L and the following

shrinking operation, both the kernel approximation error and computation time will increase slightly. We show the comparison of Nys, MEKA and MEKA-PSD on ijcnn1 dataset in Figure 13. We can see that to achieve similar approximation, MEKA-PSD is slightly slower than MEKA, but still performs better than Nys, which also generates PSD kernel approximation. (2) The second method is to directly add a small value ϵ to the diagonal of $\bar{G}$, which is equivalent to using regularization term when applying MEKA for kernel methods. We show the kernel ridge regression results in Table 8, where we add regularization term to MEKA. Note that to make sure the resulting MEKA approximation is PSD for the second method, ϵ should be equal or larger than the absolute value of the smallest negative eigenvalue of $\bar{G}$.

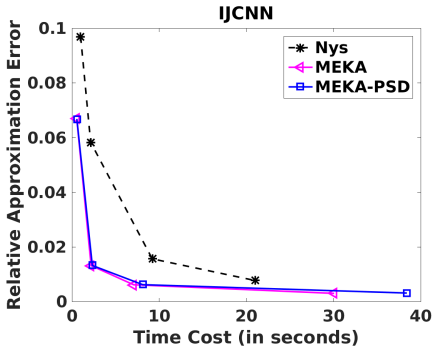


Figure 13: MEKA-PSD kernel approximation results for ijcnn1 dataset.

7.2 Dealing with the Instability Issue

There are two steps in Algorithm 1 that might cause the approximation unstable. One is due to the basis formed from each diagonal block– W_i. If the kernel matrix has strong block-diagonal structure, choosing columns from diagonal blocks to form W will perform well; on the contrary, if the kernel matrix does not have very strong block structure, other sampling strategies can be involved, for example, sampling columns from the rows corresponding to each partition to form the basis. Another step which might cause unstable result is due to insufficient entries sampled when forming the "link" matrix L. One way to solve this issue is to sample more entries from each off-diagonal block to form L when the computation time is not the main concern.

7.3 Other Applications using MEKA

In the experiment section, we apply MEKA for kernel approximation and kernel ridge regression, and besides that we could use our MEKA framework for many other machine learning applications: such as speeding up the computation of the inverse of the kernel matrix.

For many machine learning applications, e.g., Gaussian Process, we need to compute the matrix inverse $(G + \lambda I)^{-1}$, where computing the inverse of a dense matrix becomes the bottleneck for these applications. By approximating kernel matrix G in MEKA form

$G \approx WLW^T$, the inverse operation can be done in a faster fashion using Woodbury formula.

$$\begin{aligned}(\lambda I + G)^{-1} &\approx (\lambda I + WLW^T)^{-1} \\ &= \frac{1}{\lambda}(I - W(\lambda L^{-1} + W^T W)^{-1} W^T)\end{aligned}$$

We can see that after the approximation, we only need to inverse the $ck \times ck$ matrix which reduce the time complexity for inverse of the dense matrix from $O(n^3)$ to $O(n(ck)^2 + (ck)^3)$.

Besides speeding up the matrix inverse, MEKA can also be used to approximate the eigendecomposition of kernel matrix and used in various machine learning applications, for instance, manifold learning and kernelized dimensionality reduction.

Acknowledgments

This research was supported by NSF grants CCF-1320746, IIS-1546452 and CCF-1564000. Cho-Jui Hsieh also acknowledges support from XSED E startup resources.

References

D. Achlioptas, F. McSherry, and B. Schölkopf. Sampling techniques for kernel methods. In *Advances in Neural Information Processing Systems*, 2001.

F. R. Bach and M. I. Jordan. Predictive low-rank decomposition for kernel methods. In *International Conference on Machine Learning*, 2005.

K. Bache and M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

Y.-W Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J Lin. Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research*, 11:1471–1490, 2010.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

A. Cotter, J. Keshet, and N. Srebro. Explicit approximations of the Gaussian kernel. *arXiv:1109.47603*, 2011.

F. Cucker and S. Smale. On the mathematical foundations of learning. *Bulletin of the American Mathematical Society*, 39:1–49, 2001.

S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

A. Gittens and M. W. Mahoney. Revisiting the Nyström method for improved large-scale machine learning. In *International Conference on Machine Learning*, 2013.

N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

R. Hamid, Y. Xiao, A. Gittens, and D. DeCoste. Compact random feature maps. In *International Conference on Machine Learning*, 2014.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *International Conference on Machine Learning*, 2008.

C.-J. Hsieh, S. Si, and I. S. Dhillon. A divide-and-conquer solver for kernel support vector machines. In *International Conference on Machine Learning*, 2014a.

C.-J. Hsieh, S. Si, and I. S. Dhillon. Fast prediction for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2014b.

P. Kar and H. Karnick. Random feature maps for dot product kernels. In *International Conference on Machine Learning*, 2012.

S. Kumar, M. Mohri, and A. Talwalkar. Ensemble Nyström methods. In *Advances in Neural Information Processing Systems*, 2009.

S. Kumar, M. Mohri, and A. Talwalkar. Sampling methods for the Nyström method. *Journal of Machine Learning Research*, 13:981–1006, 2012.

Q. V. Le, T. Sarlos, and A. J. Smola. Fastfood – approximating kernel expansions in loglinear time. In *International Conference on Machine Learning*, 2013.

M. Li, J. T. Kwok, and B.-L. Lu. Making large-scale Nyström approximation possible. In *International Conference on Machine Learning*, 2010.

M. W. Mahoney and P. Drineas. CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Science*, 2009.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007.

A. Rahimi and B. Recht. Weighted sums of random kitchen sinks: replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems*, 2008.

C. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *International Conference on Machine Learning*, 1998.

B. Savas and I. S. Dhillon. Clustered low rank approximation of graphs in information science applications. In *SIAM Conference on Data Mining*, 2011.

B. Schölkopf and A. J. Smola. *Learning with kernels*. MIT Press, 2002.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

S. Si, C.-J. Hsieh, and I. S. Dhillon. Memory efficient kernel approximation. In *International Conference on Machine Learning*, 2014a.

S. Si, D. Shin, I. S. Dhillon, and Beresford N. Parlett. Multi-scale spectral decomposition of massive graphs. In *Advances in Neural Information Processing Systems*, 2014b.

S. Si, C.-J. Hsieh, and I. S. Dhillon. Computationally efficient Nyström approximation using fast transforms. In *International Conference on Machine Learning*, pages 2655–2663, 2016.

A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *International Conference on Machine Learning*, 2000.

G.W. Stewart and Sun Ji-Guang. *Matrix Perturbation Theory*. Academic Press, Boston, 1990.

Rashish Tandon, Si Si, Pradeep Ravikumar, and Inderjit Dhillon. Kernel ridge regression via partitioning. *arXiv preprint arXiv:1608.01976*, 2016.

U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4), 2007.

S. Wang, C. Zhang, H. Qian, and Z. Zhang. Improving the modified Nyström method using spectral shifting. In *ACM SIGKDD Conferences on Knowledge Discovery and Data Mining*, 2014.

Z. Wang, N. Djuric, K. Crammer, and S. Vucetic. Trading representability for scalability: Adaptive multi-hyperplane machine for nonlinear classification. In *ACM SIGKDD Conferences on Knowledge Discovery and Data Mining*, 2011.

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

J. Yang, V. Sindhwani, H. Avron, and M. W. Mahoney. Quasi-Monte Carlo feature maps for shift-invariant kernels. In *International Conference on Machine Learning*, 2014.

T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou. Nyström method vs random Fourier features: A theoretical and empirical comparison. In *Advances in Neural Information Processing Systems*, 2012.

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

K. Zhang, I. W. Tsang, and J. T. Kwok. Improved Nyström low rank approximation and error analysis. In *International Conference on Machine Learning*, 2008.

K. Zhang, L. Lan, Z. Wang, and F. Moerchen. Scaling up kernel SVM on limited resources: A low-rank linearization approach. In *International Conference on Artificial Intellignence and Statistics*, 2012.

Yuchen Zhang, John C. Duchi, and Martin J. Wainwright. Divide and conquer kernel ridge regression. In *Conference on Learning Theory*, 2013.