Distributed Kernel Matrix Approximation and Implementation Using Message Passing Interface

Taher A. Dameh
School of Computing Science
Simon Fraser University
BC, Canada

Wael Abd-Almageed
Institute for Advanced Computer Studies
University of Maryland at College Park
MD, USA

Mohamed Hefeeda
School of Computing Science
Simon Fraser University
BC, Canada

***Abstract*—We propose a distributed method to compute similarity (also known as kernel and Gram) matrices used in various kernel-based machine learning algorithms. Current methods for computing similarity matrices have quadratic time and space complexities, which make them not scalable to large-scale data sets. To reduce these quadratic complexities, the proposed method first partitions the data into smaller subsets using various families of locality sensitive hashing, including random project and spectral hashing. Then, the method computes the similarity values among points in the smaller subsets to result in approximated similarity matrices. We analytically show that the time and space complexities of the proposed method are subquadratic. We implemented the proposed method using the Message Passing Interface (MPI) framework and ran it on a cluster. Our results with real large-scale data sets show that the proposed method does not significantly impact the accuracy of the computed similarity matrices and it achieves substantial savings in running time and memory requirements.**

I. Introduction

The current information explosion has resulted in an increasing number of applications that need to deal with large volumes of data. While traditional algorithm analysis assumes that the data fits in main memory, it is unreasonable to make such assumptions when dealing with massive data sets such as multimedia contents and web page repositories. Kernel methods are one of the techniques that deal with high volume of data. Algorithms capable of operating with kernels include support vector machines (SVMs) [2], Gaussian processes [19], kernel Fisher discriminant (KFD) [16], kernel principal component analysis KPCA [16] and many others. Kernel-based algorithms are used for applications in many areas and fields, for example: web documents and web images clustering [1] and DNA and protein analysis [23].

The significant limitations of many such algorithms is that the kernel function $k(x, y)$ must be evaluated for all possible pairs x and y, which is computationally expensive. To overcome such limitation, one should think about distributed and approximation algorithms to process large-scale data sets.

The approximation algorithms are done by observing the eigen-spectrum of the kernel function. It is a Radial Basis function, which is a real-valued function whose value depends on the Euclidean distance. Thus, the kernel function is monotonically decreasing with the Euclidean distance between the input points. Substantial memory requirements reductions can be gained by computing the kernel function only between close points, i.e., preserving the large similarities of the Gram matrix, and filtering out the small similarities. So the problem is to find the close points in the space in a fast way, i.e., in a time complexity that is less than the one of computing the Gram matrix itself. This can be done using spatial indexing or spatial hashing that has locality preserving property.

In this work, we present a novel approach to scale kernel-based methods, using the locality sensitive hashing (LSH) to hash points in the space so that the probability of collision is high for close points. The contributions of this paper are:

- We propose an approximation algorithm for computing large scale Gram matrices. Gram matrices are the core component of the kernel based machine learning algorithms. The method achieves substantial memory and computation savings.
- The algorithm is designed such that it is independent of the subsequently used machine learning algorithm. Thus, it can be used to scale many kernel-based machine learning algorithms.
- We present a distributed implementation of the proposed method using MPI framework, our distributed implementation solves the scalability challenge of the kernel methods.
- We conduct rigorous empirical evaluation study based on our implementation and deployment on a multi node computer cluster. We use real data from the US census of 1990. Our results show an achievable accuracy of more than 90% comparing to the full Gram matrix.

II. Related Work

A. Low Rank Matrix Approximation Based on Nystrom Theorem

In low rank matrix approximation, we need to solve the problem of approximating a matrix K with another matrix $\tilde{K}$ which has a specific rank r. In the case that the approximation is based on minimizing the Frobenius norm of the difference between K and $\tilde{K}$ under the constraint that $rank(\tilde{K}) = r$. i.e.:

$$minimize \left\| K - \tilde{K} \right\|_F, \quad subject\ to\ rank(\tilde{K}) = r. \tag{1}$$

978-0-7695-5144-9/13 $31.00 © 2013 IEEE
DOI 10.1109/ICMLA.2013.17

It turns out that the solution is given by the Singular Value Decomposition (SVD) of K:
Let K be a square $n * n$ matrix with n linearly independent eigenvectors, $q_i \, (i = 1, \ldots, n)$. Then K can be factorized as:

$$K = U \Lambda U^T, \tag{2}$$

where U is the square $n * n$ matrix whose i^{th} column is the eigenvector q_i of K and Λ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, i.e., $\Lambda_{ii} = \lambda_i$.
And hence the low rank matrix $\tilde{K}$ is constructed like:

$$\tilde{K} = U \tilde{\Lambda} U^T, \tag{3}$$

where $\tilde{\Lambda}$ is the same matrix as Λ except that it contains only the r largest singular values (the other singular values are replaced by zeros), so that we say $\tilde{K}$ has the low rank r. This is known as the Eckart_Young theorem, as it was proved by those two authors in 1936 [20].
The Eigen decomposition is $O(n^3)$ computations, which is expensive. Nystrom theorem is used to approximate the eigen decomposition, by the following:
For some $m \ll n$ build $U \in R^{n \times m}$ by choosing m rows/columns of U and let $\Lambda_{m \times n} = diag(\lambda_1, \ldots, \lambda_m)$. where in this way we reduce the matrix computation complexity down to $O(mn)$.
For sampling the m rows or columns in Nystrom method, the most popular sampling scheme is random sampling, which leads to fast versions of kernel machines [22] [15], and spectral clustering [7]. In [18], several variants of multidimensional scaling are shown to be related to Nystrom approximation. Other methods [6] use randomized algorithms by sampling the columns of the Gram matrix based on a pre-computed distribution using the norms of the columns. The reconstruction of the Gram matrix is also normalized by the sampling distribution.
In spite of low-rank approximation algorithms based on Nystrom theorem reduce the computation down to $O(mn)$ where $m \ll n$, they fail to reduce the space complexity, still we need $O(n^2)$ to store the Gram matrix.
On the other hand, the following methods reduce both computations and space by using efficient way to compute the Gram matrix.

B. Efficient Implementation for Computing the Gram Matrix

In these methods, the basic idea is to compute the kernel function only between close points (points with high similarity), based on the fact that the kernel functions are radial basis functions, i.e., their values depend on the Euclidean distance between the points. The question that arises is how to find the close points in a fast way? This can be done either by spatial indexing or spatial hashing where the preprocessing step (hashing or indexing) should be smaller than quadratic, which is the complexity of computing the Gram matrix. Hussein and Abd-Almageed in [12] use the Z-curve to order the points in the space, then by using a sliding window of size w, the kernel function is computed only within this window, i.e, we compute the kernel function only between each point and the points that are at most $w/2$ far from it on the Z-order in both directions.
The drawbacks of such methods are:

1) their accuracy depends on the size w,
2) they fail for high dimensional space,
3) the sorting step is $O(n \log n)$,
4) and more importantly, they are hard to distribute, as we need to process the approximated Gram matrix as one.

To overcome these drawbacks, we propose a new algorithm using the locality sensitive hashing. We hash the points in space into m buckets. Then the kernel function is computed only between the points that reside in the same bucket. We show that we choose m to be $O(\log n)$ which reduces our computations and space down to sub quadratic. Our proposed algorithm has the following advantages:

1) linear time of preprocessing,
2) and more importantly, it is distributable. We have sub-problems to solve, which are the buckets of the hash table, where each is considered as a sub-Gram matrix. We compute the kernel function on each bucket independently. Moreover, we run the kernel method such as clustering [10] on each bucket independently.

III. Distributed Kernel Matrix Approximation (DKMA) Algorithm

In this section, we present our proposed distributed kernel matrix approximation algorithm, we start with a background information on different locality sensitive hashing families. In section III-B, we outline our proposed algorithm, then we present the analysis of the algorithm in section III-C.

A. Locality Sensitive Hashing

1) Definition: Locality sensitive hashing (LSH) is a method of performing probabilistic dimension reduction of high-dimensional data. The basic idea is to hash the input items so that similar items are mapped to the same bucket with high probability.

2) p-Stable Distributions LSH (Random Projection): Indyk and Motwani [4] use p-stable distribution to hash points in high-dimentional space. The hash function family is locality sensitive, so if two vectors (v_1, v_2) are close (small $\|v_2 - v_2\|_p$), then they should collide (hash to the same index) with high probability, and if they are far away, they should collide with small probability.
Each hash function $h_{a,b}(\mathbf{v}) : R^d \rightarrow N$ maps a d-dimensional vector v onto a set of integers. Each hash function in the family is indexed by a choice of random a and b where a is a d-dimensional vector with entries chosen independently from a p-stable distribution, and b is a real number chosen uniformly from the range $[0, W]$.
For a fixed a and b, the hash function $h_{a,b}$ is given by:

$$h_{a,b}(v) = \left\lfloor \frac{a.v + b}{W} \right\rfloor. \tag{4}$$

3) Spectral Hashing: Random projection technique mentioned in the previous section has strong theoretical guarantees. Unfortunately, it does not learn the hash values from the data set, becuase every bit in the hash value is calculated by a random linear projection followed by a random threshold. In practice this method can lead to inefficient hash values. Rather than using random projection to define bits in a hash value, other families have been proposed to learn the hash values from the data. One good example is the spectral hashing proposed by Weiss et al. [21].

In summary, given a training set of points x_i and a desired number of bits k, the spectral hashing algorithm works by [21]:

- Finding the principal components of the data using Principle Component Analysis (PCA).
- Calculating the k smallest single-dimension analytical eigenfunctions of L_p using a rectangular approximation along every PCA direction. This is done by evaluating the k smallest eigenvalues for each direction, thus creating a list of dk eigenvalues, and then sorting this list to find the k smallest eigenvalues.
- Thresholding the analytical eigenfunctions at zero, to obtain binary codes.

Spectral hashing is linear time, because the Principle Component Analysis step is done in linear time, by first finding the covariance matrix of the data set, and then finding the eigen solution of that matrix in linear time using the Lanczas algorithm [11]. Golub and van Loan give very good description of the various forms of Lanczos algorithms in their book Matrix Computations [11].

B. Proposed DKMA Algorithm using MPI

Our approximation algorithm depends on the observation that the kernel function is Radial Basis, whose value depends only on the distance. We use the locality sensitive hashing to hash the points that reside in the space. LSH preserves the locality by making the close points in the space collide with high probability. We compute the kernel function between points that collide in the same bucket. Kernel methods are distributable using our method, where we run the method on each bucket independently.

The Algorithm has four steps, see Figure 1:

1) **Pre-processing:** The data set is divided into small segments, where each segment can be handled by one process.
2) **MPI_PartitionData:** We initialize the MPI program with number of processes equals to the number of the segments. We initialize each process with same k hash functions. Each process reads single segment and hashes it to local bucket files.
3) **Mid-processing:** We combine the local buckets from the different processes that have the same index number into one bucket.
4) **MPI_ReduceData:** We initialize number of processes equals to the total number of buckets generated after hashing. Each process reads one bucket and uses a kernel method such as clustering [10] to cluster it. All processes write to a shared clusters file.

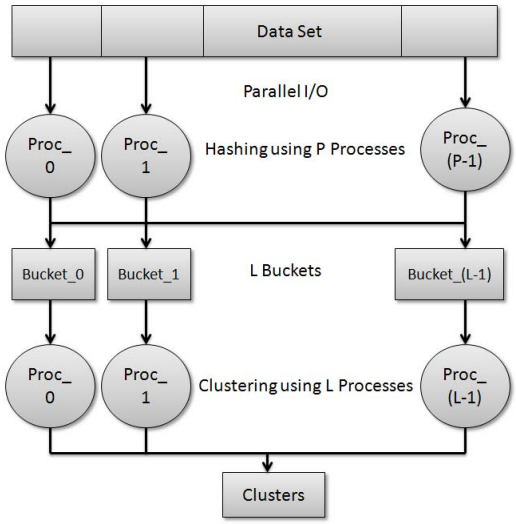

Fig. 1. The flow chart of clustering using the DKMA algorithm.

C. Analysis and Complexity

1) Time Complexity of Computing the Gram Matrix: We show that using the random projection locality sensitive hashing, the quadratic complexity of the brute force approach for computing the gram matrix is reduced down to sub quadratic. Johnson-Lindenstrauss lemma [14] states that any n points subset of Euclidean space can be embedded in $k = O(\log n/\epsilon^2)$ dimensions without distorting the distances between any pair of points by more than a factor of $(1 \pm \epsilon)$, for any $0 < \epsilon < 1$. In other words, any set of n points in d-dimensional Euclidean space can be embedded into k-dimensional Euclidean space - where k is logarithmic in n and independent of d - so that all pairwise distances are maintained within an arbitrary small factor. All known constructions of such embedding involve projecting the n points onto a random k-dimensional hyperplane. The proof of Johnson-Lindenstrauss lemma was subsequently simplified by Frankl and Maehara [9]. The proof given by Dasgupta and Gupta [3] uses elementary probabilistic techniques to obtain the result. Indyk and Motwani [13] have also given similar proofs of the theorem using simple randomized algorithm for their p-stable LSH, they show that the value of hash bits k is $\log_{1/p} n$, where p is the low probability for two far points to collide in the same bucket.

Given $k = \log_{1/p} n$ and since the hash value is a k-bit value, our hash table size m is $2^k = 2^{\log_{1/p} n}$. Assuming all buckets have the same size. Also assuming that we have enough nodes such that each node will handle one process at most. Thus, computing the Gram matrix using random projection LSH will

need:

$$\frac{n^2}{m^2} = \frac{n^2}{2^{2\log_{1/p} n}}, \quad (5)$$

$$= \frac{n^2}{2^{\frac{2\log_2 n}{\log_2 1/p}}}, \quad (6)$$

Thus,

$$\frac{n^2}{m^2} = \frac{n^2}{n^{\frac{2}{\log_2 1/p}}} = O(n^{2-c}). \quad (7)$$

where $c = 2/\log_2(1/p)$, and assuming the low probability $0 < p < 0.25$, then $0 < c < 1$. Hence, the complexity is sub-quadratic.

In our implementation using MPI, we divide the data set into p segments to be processed by p processes as a pre-processing step, which takes $O(n)$. Also in a mid-processing step before the clustering and after hashing, we combine all buckets of the same index number into one bucket, this takes $O(n)$ as well. Therefore, the total time complexity is $O(n^{2-c})$.

2) Clustering Complexity: When using the full Gram matrix, the complexity of the affinity propagation clustering algorithm, for example, is $O(n^2 \log n)$ [10], where n is the data size. Using our algorithm, and given that the bucket size is n/m, where $m = 2^k$ is the hash table size, and assuming each node will handle one process at most, the complexity will be reduced down to:

$$(\frac{n}{m})^2 \log \frac{n}{m} = \frac{n^2}{2^{2k}} \log \frac{n}{2^k} = \frac{n^2}{2^{2\log_{1/p} n}} \log \frac{n}{2^{log_{1/p} n}} \quad (8)$$

$$= \frac{n^2}{2^{\frac{2\log_2 n}{\log_2(1/p)}}} \log \frac{n}{2^{\frac{\log_2 n}{\log_2(1/p)}}} \quad (9)$$

$$= \frac{n^2}{n^{2/log_2(1/p)}} \log \frac{n}{n^{2/2log_2(1/p)}} \quad (10)$$

$$= O(n^{2-c} \log n^{1-c/2}), \quad (11)$$

where $c = 2/\log_2(1/p)$, and given the low probability $0 < p < 0.25$, then $0 < c < 1$.

IV. Experimental Evaluation

In this section, we present the experimental evaluation of the DKAM algorithm. We start with the performance metrics we use to evaluate our algorithm comparing to the full Gram matrix and comparing to the method using Hilbert curve. Then we describe the data set we use which is the US census of 1990, and the setup of our experiments. We show the results in section IV-D for the accuracy and the memory consumption, as well as the results of using large-scale data set of size one million items.

A. Performance Metrics (Frobenoius Norm)

The Frobenius norm, sometimes also called the Euclidean norm, is a matrix norm of an $m \times n$ matrix K defined as the square root of the sum of the absolute squares of its elements, i.e., for the matrix K, the Frobenius norm is:

$$\|K\|_F = \sqrt{\Sigma_{i=1}^{m} \Sigma_{j=1}^{n} |k_{ij}|^2} \quad (12)$$

In the Gram matrix, the Frobenius norm reflects the large similarities; as a larger value of an element has more effect on the Frobenoius norm value. If we have two approximated matrices of the same size, in terms of number of elements, then the one with larger Frobenoius norm value is the closer to the full Gram matrix.

B. Data set

We use The USCensus1990 data set [8], it is a discretized version of the USCensus1990raw data set. The USCensus1990raw data set contains a one percent sample of the Public Use Microdata Samples person records (PUMS) drawn from the full 1990 census sample.

There are 68 categorical attributes. Many of the less useful attributes in the original data set have been dropped, the few continuous variables have been discretized and the few discrete variables that have a large number of possible values have been collapsed to have fewer possible values.

C. Setup

We setup an MPICH cluster, where five machines are interconnected through a gigabit Ethernet switch. Each machine has Intel(R) Core(TM)2 Duo CPU E6550A processor and a 2 GB of RAM. An Ubuntu Linux operating system is installed on each. A master folder is shared among all machines using the Network File System (NFS). NFS allows us to create a folder on one machine and have it synced on all other machines. This folder is used to store programs and data to be used by all machines.

Our implementation uses LSHKIT which is a C++ Locality Sensitive Hashing Library written by Dong [5], fast Hilbert curves without recursion written in C by Moore [17].

We write one file code and store it along with the data set file on the shared folder. By parameterizing our code, each process will run on a separate data that is assigned to it by the code. If the number of processes is larger than the number of available nodes, each node will run more than one process. The number of processes is fixed in advance before the run time.

D. Results

In our first set of experiments, we use 4000 items from US census 1990 data set. We compare three methods: random projection LSH denoted by DKMA-RP in the figures, spectral hashing (DKMA-SH) and Hilbert curve. To unify the comparison, we compare the results using the same size of the approximated Gram matrix. To do so, we set the Hilbert curve window width as $n/2^k$, where n is data set size and k is the number of hash function bits in LSH. The approximated Gram matrix size using k for LSH is $n^2/2^k$, where 2^k is the number of buckets, assuming all buckets have the same size. At the same time, the approximated Gram matrix size using Hilbert curve is $n \times winWidth = n \times (n/2^k) = n^2/2^k$.

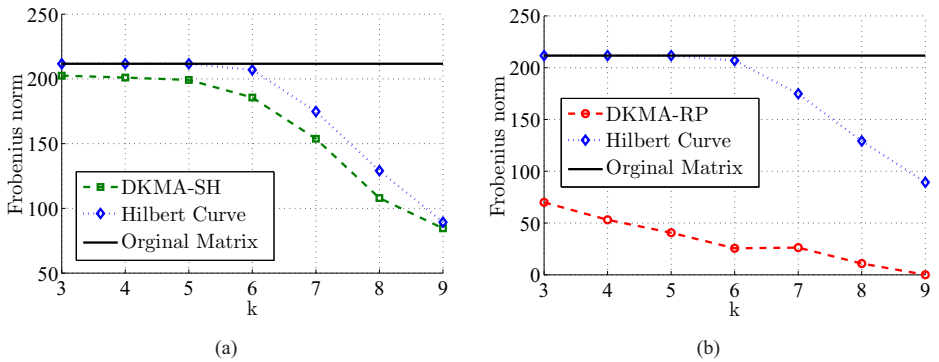

Fig. 2. Results for accuracy on DKMA Algorithm, it achieves high accuracy using spectral hashing and Hilbert curve.

E. Results for Accuracy

Figure 2 shows the results using the 4000 data points. In Figures 2(a) and 2(b) the Frobenius norm is obtained using different values of k. As k increases, the accuracy decreases. Using Hilbert curve or DKMA-SH, and for $k < 6$, we maintain a reasonable accuracy that is more than 90% of the Frobenius norm of the full Gram matrix. If we use Johnson-Lindenstrauss lemma, we find that $\log_{1/p} 4000 < 6$, and hence $p < 0.25$ the low probability for two far points to collide. Moreover, from the figures, Hilbert curve and DKMA-SH out perform DKMA-RP.

F. Results for Memory Consumption

We have shown previously that our proposed DKMA algorithm reduces the quadratic complexity for computing the Gram matrix, down to sub-quadratic. In Figure 3, the approximated Gram matrix size is drawn in log scale, it is the same for all methods at the same value of k, given that $W_{H-curve} = n/2^{k_{LSH}}$. At $k = 5$, we use a space for the approximated Gram matrix that is 3.57% of the full Gram matrix size, and at $k = 6$ we use only 1.78%, both $k = 5$ and $k = 6$ give a Frobenius norm more than 90% of the full Gram matrix using spectral hashing or Hilbert curve.
The advantage of using LSH over Hilbert curve is the distributed property of LSH. Each process handles a bucket independently, and in this case, our method using LSH is scalable. If we have enough nodes, such that, each node will handle one bucket at most, and if we ignore the communication and the synchronization overheads, we will get a speedup up to 2^k, where k is the number of hash bits, and 2^k is the total number of buckets.

G. Results for Large-Scale Data

In Figure 4, the results of processing one million data items from US census data 1990, this is done using MPI on cluster as been explained before. This size is not scalable using the brute-force way, 1 million $\times$ 1 million = 1 trillion of space and computations are required for the Gram matrix using the brute-force. Hilbert curve is not scalabe as well, as it needs the whole approximated Gram matrix stored in place. In LSH, we have independent sub-problems to be processed, which are the buckets. Since the naive way and the method using Hilbert curve are not scaled to this size, we show only the results for our method using locality sensitive hashing. From the figure, spectral hashing out performs random projection, and both give us significant reduction im memory requirements..

V. Conclusions and Future Work

A. Conclusions

Kernel-based machine learning algorithms require $O(n^2)$ to compute and store the Gram matrix, where n is the number of input points. The Gram matrix is the similarity matrix that uses a kernel function to compute the similarity between each pair of points. This complexity is infeasible and unscalable when dealing with massive data sets. Our proposed method is distributed that solves the scalability problem. The idea is to hash the input points using LSH, then generate sub-Gram matrix of each bucket.
We have implemented our method on a cluster using the Message Passing Interface (MPI) framework. Our implementation has minimum synchronization and communication overheads among processes, because we divide the data into buckets which are processed independently. The accuracy is dependent on the value of the number of hash bits k, as k increases accuracy decreases, but we showed that we can fix k at a maximum value of $\log_{1/p} n$, where n is the data set size, and p is the low probability for two far points to collide. In the same time, we maintain an accuracy that is more than 90%.

B. Future Work

We have explored and succeeded in processing million of data items, we used data from the US census of 1990 data set, this data set is relatively a low dimensional (< 100). One of the next steps is to explore the high dimensional challenge. In particular, the bio-informatics data or the text and image documents, where we may have thousands of dimensions. Moreover, exploring extremely large scale data sets is one of the next steps as well, this scale could range from billions to trillions of data items.

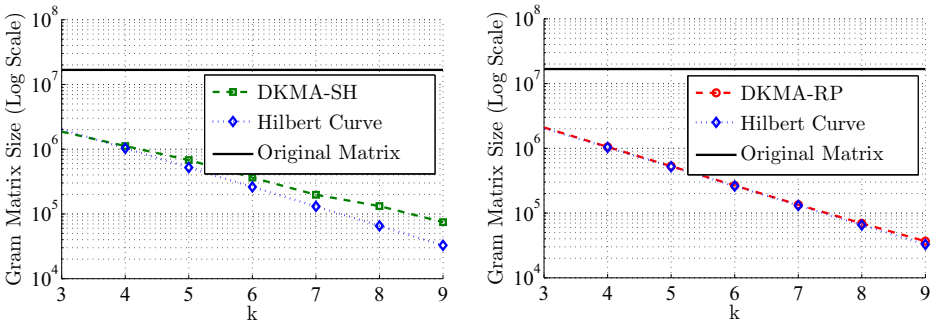

Fig. 3. Results for memory consumption. The approximated Gram matrix size is a small fraction of the original matrix.

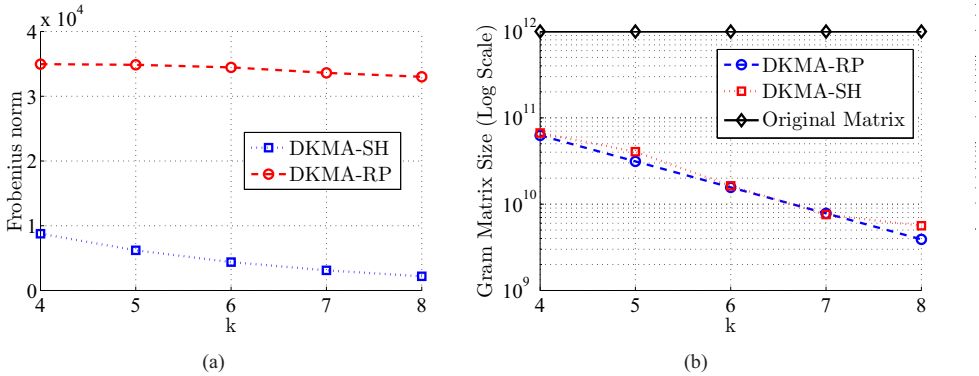

Fig. 4. Results for large-scale data set on DKMA algorithm, it scales well and it achieves substantial memory saving.

REFERENCES

[1] N. O. Andrews and E. A. Fox. Recent developments in document clustering. 2007.

[2] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, Sept. 1995.

[3] S. Dasgupta and A. Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures and Algorithms*, 22(1):60–65, 2003.

[4] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. of the twentieth annual symposium on Computational geometry*, pages 253–262, New York, NY, USA, 2004. ACM.

[5] W. Dong. Lshkit: A c++ locality sensitive hashing library. http://lshkit.sourceforge.net/, 2009.

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

[7] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the nystrom method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(2):214 –225, feb. 2004.

[8] A. Frank and A. Asuncion. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2010.

[9] P. Frankl and H. Maehara. The johnson-lindenstrauss lemma and the sphericity of some graphs. *J. Comb. Theory Ser. A*, 44:355–362, June 1987.

[10] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

[11] G. H. Golub and C. F. V. Loan. *Matrix Computations*. JHU Press, 1996.

[12] M. Hussein and W. Abd-Almageed. Efficient band approximation of gram matrices for large scale kernel methods on gpus. In *Proc. of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–10, New York, NY, USA, 2009. ACM.

[13] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, New York, NY, USA, 1998. ACM.

[14] W. Johnson and J. Lindenstauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[15] N. D. Lawrence, M. Seeger, and R. Herbrich. Fast sparse gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems 15*, pages 609–616. MIT Press, 2003.

[16] S. Mika, B. Schölkopf, A. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. Kernel pca and de-noising in feature spaces. In *Proc. of the 1998 conference on Advances in neural information processing systems II*, pages 536–542, Cambridge, MA, USA, 1999. MIT Press.

[17] D. Moore. Fast hilbert curve generation, sorting, and range queries. http://web.archive.org/web/20041028171141/http://www.caam.rice.edu/~dougm/twiddle/Hilbert/.

[18] J. C. Platt. Fastmap, metricmap, and landmark mds are all nystrom algorithms. In *Proc. of 10th International Workshop on Artificial Intelligence and Statistics*, pages 261–268, 2005.

[19] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[20] G. W. Stewart. On the early history of the singular value decomposition. *SIAM Review*, 35(4):pp. 551–566, 1993.

[21] Y. Weiss, A. B. Torralba, and R. Fergus. Spectral hashing. In *Neural Information Processing Systems*, pages 1753–1760, 2008.

[22] C. Williams and M. Seeger. Using the nystrm method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001.

[23] A. Zien, G. Ratsch, S. Mika, B. Scholkopf, T. Lengauer, and K.-R. Muller. Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 16(9):799–807, 2000.