

Kronecker Factorization for Speeding up Kernel Machines

Gang Wu, Zhihua Zhang, and Edward Chang
Electrical and Computer Engineering, UCSB

{gwu@engineering, zzhang@mddb.ece, echang@ece}.ucsb.edu

Abstract

In kernel machines, such as kernel principal component analysis (KPCA), Gaussian Processes (GPs), and Support Vector Machines (SVMs), the computational complexity of finding a solution is $O(n^3)$, where n is the number of training instances. To reduce this expensive computational complexity, we propose using *Kronecker factorization*, which approximates a positive definite kernel matrix by the *Kronecker product* of two smaller positive definite matrices. This approximation can speed up the calculation of the kernel-matrix inverse or eigendecomposition involved in kernel machines. When the two factorized matrices have about the same dimensions, the computational complexity is improved from $O(n^3)$ to $O(n^2)$. We propose two methods to carry out Kronecker factorization and apply them to speed up KPCA. In Experiments show that our methods can drastically reduce the computation time of kernel machines without any significant degradation in their effectiveness.

1 Introduction

The kernel matrix, such as the *Gram matrix* in Kernel PCA (KPCA) [5] and SVMs, and the covariance matrix in Gaussian Processes (GPs), plays an important role in kernel machines. In general, these machines are required to calculate the inverse of the kernel matrix or to make the eigendecomposition on it. Both operations take $O(n^3)$ where n denotes the number of training instances. Specifically, KPCA requires an eigendecomposition on an $n \times n$ Gram matrix, SVMs need to resolve a quadratic programming problem that involves an $n \times n$ Gram matrix, and GPs need to invert an $n \times n$ covariance matrix. Several methods have been proposed to address this problem of high computational cost, such as randomized techniques [1], sparse greedy approximation [6], and the Nyström method [8], etc. All these methods are based on sampling techniques. In this paper, we tackle the same computational challenge by a different route, using the *Kronecker product* of the matrices.

Suppose that two matrices $\mathbf{B} = [b_{ij}]$ and $\mathbf{C} = [c_{ij}]$ are $m_1 \times n_1$ and $m_2 \times n_2$, respectively. The Kronecker product of \mathbf{B} and \mathbf{C} (denoted by $\mathbf{B} \otimes \mathbf{C}$) is an $m_1 m_2 \times n_1 n_2$

matrix, defined as the following block matrix

$$\mathbf{B} \otimes \mathbf{C} = \begin{bmatrix} b_{11}\mathbf{C} & \cdots & b_{1n_1}\mathbf{C} \\ \vdots & \ddots & \vdots \\ b_{m_1 1}\mathbf{C} & \cdots & b_{m_1 n_1}\mathbf{C} \end{bmatrix}.$$

Conversely, let \mathbf{A} be an $m \times n$ matrix with $m = m_1 m_2$ and $n = n_1 n_2$. Our current problem is to find two matrices \mathbf{B} ($m_1 \times n_1$) and \mathbf{C} ($m_2 \times n_2$) so that $\mathbf{B} \otimes \mathbf{C}$ approximates \mathbf{A} . We denote this problem as the *Kronecker factorization* of the matrix \mathbf{A} . A generic structure of the Kronecker factorization and its applications are detailed in [7]. In this paper, we focus on a special case in that the matrices \mathbf{A} , \mathbf{B} and, \mathbf{C} are all symmetric positive definite.

1.1 Related Work

The topic of speeding up kernel machines has been an important and actively research one. However, the scalability of kernel machines is still a serious problem if they are to be used for large-scale problems. Many sampling based methods have been proposed to reduce the $O(n^3)$ computational time of kernel machines. Take SVMs as an example. (Since SVMs involve solving a quadratic programming optimization problem, its computational complexity is $O(n^3)$.) Sampling based algorithms such as Osuna's Decomposition algorithm [3] and SMO [4] are able to reduce the training time to $O(n^2)$. Other sampling based algorithms for KPCA and GPs such as [1, 4, 6, 8] (details are presented shortly) can also achieve speedup of an order of magnitude. However, large-scale applications demand even faster algorithms. The Kronecker factorization enjoys two advantages over the sampling based methods. First, the factorization can be recursively applied to a large matrix. Second, this divide-and-conquer approach can be parallelized and take advantage of the new generation *multi-core processor architecture* (multiple processors included on one chip). On the contrary, traditional iterative methods (e.g., SMO) can hardly be parallelized because of strong data dependencies between iterations.

Due to the space limitations, we will leave GPs and SVMs to a future, comprehensive treatment. In this paper, we focus on speeding up Kernel methods that require an eigendecomposition on kernel matrix such as

KPCA and KICA. For this purpose, Williams et al [8] propose to use the Nyström method to approximate the eigendecomposition of the Gram matrix \mathbf{K} . Specifically, the authors use a reduced-rank kernel matrix generated from a set of randomly sampled training data of size $m < n$. Then, the Nyström method is applied on the eigenspectrum (eigenvalues and eigenvectors) of the reduced-rank kernel matrix to recover the corresponding eigenspectrum of the large-size \mathbf{K} .

Smola et al [6] propose a sparse greedy approximation technique to construct a lower-rank representation of the Gram matrix \mathbf{K} . It turns out that the form of the Nyström approximation is almost identical to the sparse greedy matrix approximation [8]. The only difference comes from how to sample a set of training data so as to form a reduced-rank approximation of the Gram matrix. The Nyström method randomly samples training data, but the sparse greedy matrix method searches over the column or basis function space incrementally until a selection rule is satisfied. In [6], the authors defined three selection rules. For example, one rule could be the Frobenius norm of the difference between the approximated $\tilde{\mathbf{K}}$ and the original \mathbf{K} , $\|\mathbf{K} - \tilde{\mathbf{K}}\|_F^2$. However, as argued by the authors, there exists a tradeoff between the quality of the selection and the amount of computational resources needed to compute the best set of columns or basis functions. For a given m (the number of selected training data), the sparse greedy method has a better approximation to \mathbf{K} , but with more computational cost than the Nyström approximation. Considering both issues, in this paper, we use the Nyström method as a baseline to compare with our proposed Kronecker factorization method on speeding up KPCA.

One more method for speeding up KPCA is proposed by Achlioptas et al [1]. Similar with the Nyström and sparse greedy matrix method, this one is also based on sampling techniques. The basic idea is to randomly sample the kernel matrix \mathbf{K} according to some pre-defined probability so as to form a sparse kernel matrix. Such a sparse \mathbf{K} is then employed to accelerate the eigendecomposition in KPCA.

1.2 Contribution Summary

In summary, almost all traditional methods attempt to find a lower-rank matrix to approximate the kernel matrix by sampling the training dataset. The difference of these methods mainly lies in the way sampling is done. In this paper, we propose speeding up kernel machines by factorizing a large-size kernel matrix \mathbf{K} into multiple smaller ones and approximate the original \mathbf{K} using their Kronecker product. Since both eigendecomposition and inverse of \mathbf{K} can be recovered from the corresponding operations on multiple factorized smaller matrices, the computation of kernel machines can be parallelized.

Considering that the *multi-core architecture* is the trend of future processors, we believe that the Kronecker factorization approach is a viable one to allow kernel machines to scale. Our experiments show that our methods can reduce the computation time of kernel machines without significant degradation in their effectiveness.

2 Mathematical Background

Given a positive definite (p.d.) matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ with $n = n_1 n_2$, we have the following factorization problem

$$\mathbf{A} \triangleq \mathbf{B} \otimes \mathbf{C},$$

where $\mathbf{B} \in \mathbb{R}^{n_1 \times n_1}$ and $\mathbf{C} \in \mathbb{R}^{n_2 \times n_2}$ are both positive definite. In other words, we factorize a large-size p.d. matrix into two small-size p.d. matrices with the Kronecker product. We call this formulation the *Kronecker factorization* of the positive definite matrix. Some theoretical properties of the Kronecker products [2] are listed below, which provide the foundation to apply the Kronecker factorization in our paper. For more details, such as the Kronecker products on multiple matrices, please refer to [2].

THEOREM 2.1. *If \mathbf{B} and \mathbf{C} are $n_1 \times n_1$ and $n_2 \times n_2$ respectively, then*

- (a) $(\mathbf{B} \otimes \mathbf{C})' = \mathbf{B}' \otimes \mathbf{C}'$;
- (b) $\text{tr}(\mathbf{B} \otimes \mathbf{C}) = \text{tr}(\mathbf{B})\text{tr}(\mathbf{C})$;
- (c) $|\mathbf{B} \otimes \mathbf{C}| = |\mathbf{B}|^{n_2} \cdot |\mathbf{C}|^{n_1}$; and
- (d) *If \mathbf{B} and \mathbf{C} are nonsingular, then $(\mathbf{B} \otimes \mathbf{C})^{-1} = \mathbf{B}^{-1} \otimes \mathbf{C}^{-1}$.*

THEOREM 2.2. *Let $\sigma(\mathbf{B})$, $\sigma(\mathbf{C})$ and $\sigma(\mathbf{B} \otimes \mathbf{C})$ be the spectrums (or set of eigenvalues) of \mathbf{B} , \mathbf{C} , and $\mathbf{B} \otimes \mathbf{C}$, respectively. If $\lambda \in \sigma(\mathbf{B})$ and \mathbf{b} is the corresponding eigenvector of \mathbf{B} ; and if $\mu \in \sigma(\mathbf{C})$ and \mathbf{c} is the corresponding eigenvector of \mathbf{C} , then $\lambda\mu \in \sigma(\mathbf{B} \otimes \mathbf{C})$ and $\mathbf{b} \otimes \mathbf{c}$ is the corresponding eigenvector of $\mathbf{B} \otimes \mathbf{C}$. Furthermore, if $\sigma(\mathbf{B}) = \{\lambda_1, \dots, \lambda_{n_1}\}$ and $\sigma(\mathbf{C}) = \{\mu_1, \dots, \mu_{n_2}\}$, then $\sigma(\mathbf{B} \otimes \mathbf{C}) = \{\lambda_i \mu_j : i = 1, \dots, n_1, j = 1, \dots, n_2\}$.*

For an $n \times n$ matrix \mathbf{A} , the computational complexity is $O(n^3)$ to invert it or to make its eigendecomposition. The memory usage is $O(n^2)$. By means of the Kronecker factorization, the complexity of the same operations is reduced to $O(n_1^3) + O(n_2^3)$, following the above theorems. Moreover, the memory usage is reduced to $O(n_1^2) + O(n_2^2)$. Thus, the Kronecker factorization can effectively reduce the computational complexity. Especially when $n_1 \approx n_2$, the computation complexity of kernel machines is greatly reduced to $O(n^{\frac{3}{2}})$. Please note that this $O(n^{\frac{3}{2}})$ does not consider the factorization cost. When that cost is added, as we will show in Section 3, the computation complexity is $O(n^2)$.

3 Kronecker Factorization and KPCA

Given \mathbf{A} , our problem is to estimate $\mathbf{B} = [b_{ij}]$ and $\mathbf{C} = [c_{ij}]$. To achieve this goal, we propose two

iterative methods based on two different criteria, i.e., the least-squares (LS) error and the Kullback-Leibler (KL) divergence. At the end of this section, we show how these methods are applied to speed up KPCA. Because of the space limitation, we refer the reader to [9] for detailed derivations.

Partition \mathbf{A} into the block matrices, where \mathbf{A}_{ij} 's ($i, j = 1, \dots, n_1$) are all $n_2 \times n_2$ and $\hat{\mathbf{A}}_{ij}$'s ($i, j = 1, \dots, n_2$) are all $n_1 \times n_1$. Since \mathbf{A} is symmetric positive definite, it is clear that $\mathbf{A}_{ji}^T = \mathbf{A}_{ij}$ and $\hat{\mathbf{A}}_{ji}^T = \hat{\mathbf{A}}_{ij}$. Moreover, \mathbf{A}_{ii} 's and $\hat{\mathbf{A}}_{ii}$'s are positive definite.

3.1 Least-Squares Method

The first method is derived from minimizing the least-squares error defined in [7], which is expressed as follows:

$$\begin{aligned} e_{\mathbf{A}}(\mathbf{B}, \mathbf{C}) &= \|\mathbf{A} - \mathbf{B} \otimes \mathbf{C}\|_F^2 \\ (3.1) \quad &= \text{tr}((\mathbf{A} - \mathbf{B} \otimes \mathbf{C})'(\mathbf{A} - \mathbf{B} \otimes \mathbf{C})), \end{aligned}$$

We attempt to find a p.d. \mathbf{B} ($n_1 \times n_1$) and a p.d. \mathbf{C} ($n_2 \times n_2$), which minimize $e_{\mathbf{A}}(\mathbf{B}, \mathbf{C})$. Practically, we use a *separable least squares framework*, which consists of two successive parts, to solve this minimization problem. The computational cost of this LS-based factorization algorithm is $O(n_1^2 n_2^2) + O(n_2^2 n_1^2) \approx O(n^2)$ and the memory usage $O(n_1^2 + n_2^2)$.

3.2 Kullback-Leibler Divergence Method

The second method is derived from minimizing the KL divergence between \mathbf{A} and $\mathbf{B} \otimes \mathbf{C}$, which is defined as follows:

$$\begin{aligned} d_{\mathbf{A}}(\mathbf{B}, \mathbf{C}) &= \text{tr}(\mathbf{A}(\mathbf{B} \otimes \mathbf{C})^{-1}) - \log |\mathbf{A}(\mathbf{B} \otimes \mathbf{C})^{-1}| - n \\ (3.2) \quad &= \text{tr}(\mathbf{A}(\mathbf{B}^{-1} \otimes \mathbf{C}^{-1})) - \log |\mathbf{A}| \\ &\quad - n_2 \log |\mathbf{B}^{-1}| - n_1 \log |\mathbf{C}^{-1}| - n. \end{aligned}$$

Similar to $e_{\mathbf{A}}(\mathbf{B}, \mathbf{C})$ in Equation 3.1, $d_{\mathbf{A}}(\mathbf{B}, \mathbf{C})$ arrives at its minimum 0 iff $\mathbf{A} = \mathbf{B} \otimes \mathbf{C}$. We attempt to find positive definite \mathbf{B} and \mathbf{C} that minimize $d_{\mathbf{A}}(\mathbf{B}, \mathbf{C})$.

To solve the optimization problem of minimizing $d_{\mathbf{A}}(\mathbf{B}, \mathbf{C})$ with respect to \mathbf{B} and \mathbf{C} , we also use a separable framework and devise an iterative algorithm. Both \mathbf{B} and \mathbf{C} are positive definitive. For the proof and the detail of the algorithm, please consult [9]. The computational cost of this method is $O(n_2^3 + n_1^2 n_2^2) + O(n_1^3 + n_2^2 n_1^2) \approx O(n^2)$ and the memory usage is $O(n_1^2 + n_2^2)$. Compared with that of the LS-based method presented in Section 3.1, the computational cost of our proposed method is slightly higher.

3.3 Speeding up KPCA

According to Theorem 2.2, the eigen-spectrum of a large-size matrix can be calculated from the eigen-spectrums of its factorized small-size matrices. This enables us to apply the Kronecker factorization in KPCA for reducing its computational complexity. Figure 1 illustrates the

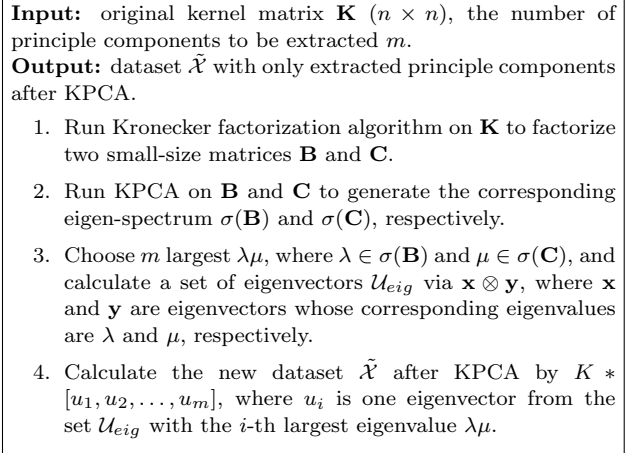


Figure 1: Kronecker Factorization for KPCA

detailed algorithm. Note that in general, KPCA works with a centered kernel matrix. In this case, the centered kernel matrix becomes singular, so theoretically we are not allowed to implement the Kronecker factorization on the matrix. However, they can still work in practice, since the two factorization algorithms involve the trace operations of only sub-block matrices of the original matrix. Another alternative way is to first implement the Kronecker factorization on the original kernel matrix and then to centerize the two factorized matrices. In the experiments of this paper, we choose the latter method.

4 Experiments

We used both artificial and real-world datasets to examine our proposed Kronecker factorization algorithms and compare them with one representative sampling-based method, the Nyström method, for speeding up KPCA.

The datasets used in our experiments are eight toy datasets (TOY), the USPS handwritten digit dataset, and the ABALONE dataset from the UCI Repository. Each of eight toy datasets has binary classes and ten features. The first feature was generated according to the Gaussian distribution $N(1, 1)$ for one class and $N(-1, 1)$ for the other class. The other nine features were generated according to $N(0, 3)$ so as to make the datasets noisy. Eight toy datasets are of size 100, 400, 900, 1600, 2500, 3600, 4900, and 6400, respectively. For each toy dataset, we randomly extracted about 67% as training data and the rest 33% as test data. The USPS dataset has 7291 training instances and 2007 test instances, each of which has 256 features. This dataset has 10 classes, corresponding to the digits 0, ..., 9. The ABALONE dataset has 3000 training instances and 1177 test instances, each of which has 8 dimensions. ABALONE has 29 classes. In the experiments, we set its first 10 classes to be positive and the remaining 19 classes to be negative so as to form a binary dataset.

To achieve the maximum performance of our algo-

rithms, we chose the sizes of two factorized matrices close to \sqrt{n} . When n couldn't be factorized, we simply removed some training instances. Our experiments showed that for a large-size dataset, doing so had almost no influence on the performance of KPCA and GPs. Following the experimental setup in [6, 8], we used the Gaussian kernel, $k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{\sigma^2}\right)$. For TOY datasets, we chose $\sigma = 10.0$. For USPS and ABALONE, we followed the parameter settings in [6, 8] and then chose $\sigma^2 = 0.5d$, where d is the dimensionality of the data. In our experiments, we initialize $\mathbf{C}(0) = \mathbf{I}$ and chose 5 as the running iterations in our algorithms. This heuristic number of iterations came from our observations in the experiments that the Kronecker factorization algorithms became convergent after about 5 iterations.

For a better illustration of comparison, we used a modified measurement for each algorithm to report the experimental results. Instead of just using $e_{\mathbf{A}}(\mathbf{B}, \mathbf{C})$ in Eqn. 3.1 and $d_{\mathbf{A}}(\mathbf{B}, \mathbf{C})$ in Eqn. 3.2, we used the average least-square error defined as $\frac{e_{\mathbf{A}}(\mathbf{B}, \mathbf{C})}{n^2}$ for the factorization method based on least-square error, and used the decimal logarithm $\log_{10} d_{\mathbf{A}}(\mathbf{B}, \mathbf{C})$ for the method based on Kullback-Leibler divergence,

4.1 Experiments on KPCA

In this experiment, we examined the effectiveness and efficiency of our proposed Kronecker factorization on KPCA. We compared with the regular KPCA without any speedup and the Nyström method, which is one representative sampling-based method for speeding up KPCA and GPs. Moreover, besides examining Kronecker factorization applied at one level, we also examined the factorization applied at two levels, which was to run factorization one more time on the factorized \mathbf{B} and \mathbf{C} so that $\mathbf{B} = \mathbf{B}_1 \otimes \mathbf{B}_2$ and $\mathbf{C} = \mathbf{C}_1 \otimes \mathbf{C}_2$. From Theorem 2.2, the eigen-spectrum $\sigma(\mathbf{B}_1 \otimes \mathbf{B}_2 \otimes \mathbf{C}_1 \otimes \mathbf{C}_2)$ can be exactly recovered from the eigen-spectrum of the smaller-size matrices, which can be conducted in parallel¹.

Since running KPCA on the whole USPS can be very time-consuming on our hardware platform², we followed the setting reported in [5] by randomly sampling 5,000 out of 7291 instances as training data, and 1400 out of 2,007 as test data. We set the size of matrices \mathbf{B} and \mathbf{C} to be 80, and the sizes of \mathbf{B}_1 , \mathbf{B}_2 , \mathbf{C}_1 , and \mathbf{C}_2 to be 8, 10, 8, and 10, respectively. For ABALONE, to make the Kronecker factorization feasible, we removed

¹Since we do not have a hardware platform to conduct the parallelized experiments, we just sequentially conducted the KPCA on each small matrix, which could not show the benefits of two-level factorization.

²All experiments were done on a PIII 900MHZ Workstation with 1GB memory.

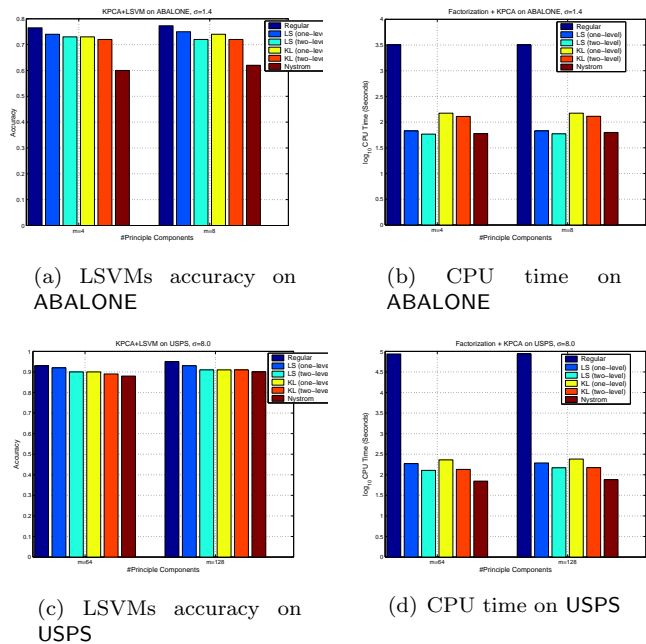


Figure 2: Comparison of Speeding Up KPCA: For each figure, x -axis denotes the number of principle components which is pre-defined before running KPCA, y -axis denotes the accuracy of running LSVMs after KPCA in (a) and (c), or the CPU time (in the decimal logarithm) of running factorization (if possible) and KPCA together. In each figure, from left to right, the six bars represent the results by running regular KPCA (denoted as Regular in the legend), KPCA after a one-level factorization based on least square error (LS (one-level)), KPCA after a two-level factorization based on least square error (LS (two-level)), KPCA after a one-level factorization based on KL divergence (KL (one-level)), KPCA after a two-level factorization based on KL divergence (KL (two-level)), and KPCA approximated by the Nyström method (Nyström).

one instance³ from the training dataset so as to form two feasible small matrices, a 58×58 matrix \mathbf{B} and a 72×72 matrix \mathbf{C} ($58 \times 72 = 4176$). Then, we set the sizes of \mathbf{B}_1 , \mathbf{B}_2 , \mathbf{C}_1 , and \mathbf{C}_2 to be 2, 29, 8, and 9, respectively. As for the Nyström method, we selected the number of sampled instances to be 80 for USPS and 60 for ABALONE so that its computational cost could be almost the same or even less than that of our algorithms.

In order to quantitatively evaluate the performance of each approximate method for KPCA, we ran linear SVMs (LSVMs) on the dimensionality-reduced dataset

³To achieve the maximum performance of our algorithms, we choose the sizes of two factorized matrices to \sqrt{n} . When n cannot be factorized, we simply remove some training instances. Our experiments show that doing so does not impact the performance of KPCA and GPs.

via KPCA following the setting in [5]. Figures 2(a) and (c) give the classification accuracy on ABALONE and USPS, respectively. Figures 2(b) and (d) show the corresponding running time of KPCA (in log-scale), which includes the time of running the factorization for our method and of running the approximation for the Nystrom method. From the figures, we make three observations. First, compared to the regular KPCA using the full kernel matrix, all our methods, one-level and two-level factorization based on LS or KL, can reduce the wall-clock time of running KPCA by hundreds of times while maintaining the same level of accuracy. (Again, Figures 2(b) and (d) are plotted in the log scale.) Second, the Nyström method, which we intentionally forced to have almost the same computational time as our methods, did not approximate well to the regular KPCA, especially for ABALONE. This means that more instances need to be sampled for Nyström, which will increase its computational time. Third, compared to one-level factorization, two-level factorization indeed helps speeding up the KPCA, and still enjoys a good approximation to the regular KPCA. For our experiment, we also found that the factorization part took up about 70% of the total running time, including the time for conducting factorization and running kernel machines. We thus believe that running KPCA on multiple smaller matrices in parallel can further reduce the 30% part of computational time (which is not reflected in the figures).

5 Concluding Remarks

In this paper, we have presented the idea of *Kronecker factorization* of the positive definite matrix to speed up kernel machines. Specifically, we factorize a large-size matrix \mathbf{A} into two considerably smaller matrices \mathbf{B} and \mathbf{C} , and we approximate \mathbf{A} with the Kronecker product of \mathbf{B} and \mathbf{C} . Our empirical studies showed that the proposed method can substantially speed up KPCA while maintaining high class-prediction accuracy. Please refer to [9] for detailed mathematical derivations and also the success of applying *Kronecker factorization* to GPs. Our future research plans to parallelize our proposed algorithms and apply the factorization recursively when the matrix dimension is very large.

References

- [1] D. Achlioptas, F. McSherry, and B. Schölkopf. Sampling techniques for kernel methods. *Advances in Neural Information Processing Systems 14*, 2002.
- [2] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, UK, 1991.
- [3] E. Osuna and F. Girosi. Reducing runtime complexity of svms. Technical Report Proceedings of the 14th International Conference on Pattern Recognition, 1998.

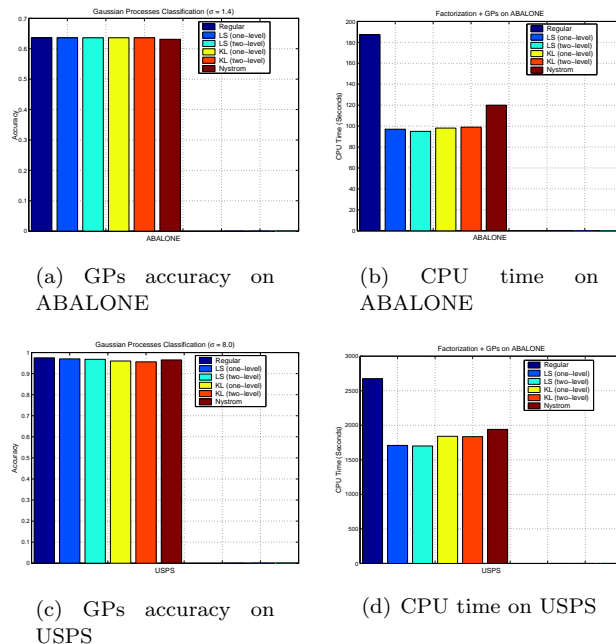


Figure 3: Comparison of Speeding Up GPs. (a) and (c) illustrate the test accuracy on Gaussian processes with different approximate methods. (b) and (d) then illustrate the corresponding cpu time (in seconds) of running factorization and GPs together. In each figure, from the left to the right, the six bars represent the results by running regular GPs (denoted as Regular in the legend), GPs after a one-level factorization based on least square error (LS (one-level)), GPs after a two-level factorization based on least square error (LS (two-level)), GPs after a one-level factorization based on KL divergence (KL (one-level)), GPs after a two-level factorization based on KL divergence (KL (two-level)), and GPs approximated by the Nyström method (Nystrom).

- [4] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, 1998.
- [5] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.
- [6] A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *The 17th International Conference on Machine Learning*, 2000.
- [7] C. F. Van Loan and N. Pitslanis. Approximation with kronecker products. In M. Moonen, G. Golub, and B. de Moor, editors, *Linear Algebra for Large Scale and Real-Time Applications*, pages 293–314. Kluwer Academic Publisher, Dordrecht, 1993.
- [8] C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, 2001.
- [9] G. Wu, Z. Zhang, and E. Chang. Kronecker factorization for speeding up kernel machines (extended version). *UCSB Technical Report*, June 2004.