Study of kernel SVM approximation methods
DC-Pred++ and LDKL

Dhruv Singal
Department of Computer Science
Indian Institute of Technology Kanpur
dhruv@iitk.ac.in

Pranav Maneriker
Department of Computer Science
Indian Institute of Technology Kanpur
mpranav@iitk.ac.in

Abstract

In this project, we analyze two state of the art kernel SVM approximation algorithms - LDKL[1] and DC-Pred++[2] in detail. We trace the ideas which influenced these algorithms strongly and present a brief overview of the history of contributions culminating in these landmark algorithms. We also present the novel ideas proposed in these algorithms in a lucid and comprehensive manner.

1 Introduction

Kernel methods form the backbone of machine learning. Kernels expand the horizons of standard algorithms used in machine learning like SVM, Ridge Regression, PCA, k-means, etc. and allow these algorithms to be used in settings which are non-linear by nature. For example, SVM outputs hyperplanes as separators in the space of input vectors for the binary classification problem. However, in general, a lot of practical applications have input vectors which are not linearly separable. In those cases, kernels help generalize the SVM algorithm by mapping the vectors from the d-dimesional input space $\mathcal{X}$ to a D-dimesional feature space $\mathcal{F}$ via a possibly non-linear function $\phi : \mathcal{X} \longrightarrow \mathcal{F}$. The algorithm now works in the feature space instead of the input space, which makes the algorithm more robust and applicable to a wider range of situations.

A key point to note in kernel methods is that any algorithm whose training and prediction depend only on *inner products* between vectors of the input space, can be kernelized. The kernel matrix G is the Gramian matrix which stores the inner products of the training vectors (in the features space), i.e. $G_{ij} =< \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) >= K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, where K is the *kernel function*. Since G is a Gramian matrix, it is symmetric and positive semidefinite.

The SVM algorithm is one of the most essential algorithms used in the setting of binary classification. It is a robust method which applies to a wide range of situations with nice error bounds. A major point in its favor is that it has been shown that kernels apply very successfully to SVMs.

Almost surely, every kernelized algorithm uses G in its entirety. However, if the number of input points n (with dimension d) becomes very large, it becomes difficult to compute and store G. It takes $\mathcal{O}(n^2 d)$ time to calculate G (if we assume that it takes $\mathcal{O}(1)$ time to compute $K(\cdot, \cdot)$). Also, it takes $\mathcal{O}(n^2)$ space to store G. In case of the kernel SVM algorithm, if $\bar{n} = \#\text{SVs}$, it takes $\mathcal{O}(\bar{n}d)$ time for prediction. Given a test sample $\boldsymbol{x}$, the prediction value is calculated as $\sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i)$.

Approximation of the kernel SVM becomes important to improve the time and space efficiency, while not compromising on accuracy. LDKL [1] and DC-Pred++ [2] are the state of the art algorithms in approximate kernel SVM methods. We will analyze these algorithms in detail in the subsequent sections.

2 DC-Pred++

DC-Pred++ is the name given to the algorithm proposed by Hsieh, Si and Dhillon in [2]. The algorithm is based on the classical method of Nyström method which has been used in numerical analysis. It improves the state of the art in kernel SVM approximation by matching benchmark accuracy and upgrading on the training time and prediction time significantly.

2.1 Nyström Method

2.1.1 History

Nyström method for approximating kernels has been a topic of rigorous research in the last few decades. This thread of research began when it was shown by Williams and Seeger, 2001 that Nyström method can be used to approximate Gram matrices [3]. As a result, subsequent papers demonstrated the efficacy of this method and improved bounds and error guarantees. Drineas and Mahoney, 2005 [4] are responsible for introducing the method as it is used in contemporary literature. The state of the art in Nyström methods was achieved by Kumar et al., 2009 who proposed an ensemble model of Nyström approximations [5].

2.1.2 A brief peek

As introduced in [4], the Nyström method for approximating a gram matrix G starts off by taking $m << n$ *landmark* points, $\{\boldsymbol{u}_j\}_{j=1}^m$ such that $\boldsymbol{u}_j \in \mathcal{X}$ are sampled from the set of input vectors. We then set up the matricies $C \in \mathbb{R}^{n\times m}$ and $W \in \mathbb{R}^{m\times m}$ such that $C_{ij} = K(\boldsymbol{x}_i, \boldsymbol{u}_j)$ and $W_{ij} = K(\boldsymbol{u}_i, \boldsymbol{u}_j)$. As is evident from definitions of C and W, these are submatrices of the original Gram matrix G. Now, as shown in [4], the matrix G is approximated as $\bar{G}$ using the matrices C and W as follows:

$$G \approx \bar{G} = CW^{\dagger}C^T \tag{1}$$

An important property of this method is that $\|\bar{G} - G\|_\xi,\ \xi = 2, F$ has a (small) bounded value [4]. This implies that the approximation of $\bar{G}$ hence obtained is quite close to the original matrix G. Given C, W and the model α, the prediction value for a test sample x is calculated as

$$\boldsymbol{c}(W^{\dagger}C^T\boldsymbol{\alpha}) = \boldsymbol{c}\boldsymbol{\beta} \tag{2}$$

where, $\boldsymbol{c} = [K(\boldsymbol{x}, \boldsymbol{u}_1), \ldots, K(\boldsymbol{x}, \boldsymbol{u}_m)]$. Hence, β can be precomputed and stored in only $\mathcal{O}(n)$ space.

2.1.3 Pros and cons

As is apparent from the discussion above, the improvements of this approximation over the traditional kernel SVM results in terms of both time and space compexity involved. It now takes $\mathcal{O}(m^2nd)$ (compared to $\mathcal{O}(n^2d)$) to calculate the (approximate) kernel matrix. Also, the information contained in the matrix can be retrieved using only $\mathcal{O}(mn)$ space (compared to $\mathcal{O}(n^2)$). Most importantly, the prediction time also reduces to $\mathcal{O}(md)$. Inherently, the fast approximation of the kernel matrix reduces the training time tremendously, since only the landmark points are required for training, compared to the whole data set.

In practice, however, it is observed that typically $m > 100$ is needed for reasonable accuracy [2]. This results in a tradeoff between prediction (and training) time and approximation error. For large values of m, while the approximation error goes down, the prediction time increases linearly. On the other hand, for small values of m, while the prediction time is less, the approximation error blows up. DC-Pred++ tries to resolve this tradeoff using a novel approach, as discussed below.

2.2 Algorithm

DC-Pred++ differs from the classical Nyström method on its use of three novel propositions, which work in tandem to help it achieve better results than the state of the art techniques. The propositions are the following:

- Adding *pseudo-landmark points* to resolve the tradeoff between approximation error and prediction time
- Using *weighted k-means* to give better approximation with pseudo-landmark points
- Implementing *divide and conquer* approach for improving the prediction and training time dramatically

We will go over each of these ideas in the subsequent sections. While the idea of pseudo-landmark points is applicable to kernel approximation in any setting, the weighted k-means and divide and conquer approach is valid only for kernel SVM and kernel ridge regression.

2.2.1 Pseudo-landmark points

We know from a previous section, that adding landmark points decreases the approximation error, while sacrificing the prediction time. To resolve this tradeoff, DC-Pred++ proposes adding pseudo-landmark points instead.

We consider p pseudo landmark points $\{\boldsymbol{v}_t\}_{t=1}^{p}$ such that $\boldsymbol{v}_t \in \mathbb{R}^d$ are sampled from the whole input space $\mathbb{R}^d$, instead of just the set of input vectors. Now, we know that the Nyström method requires two matrices C and W. For modifying the Nyström method to include the pseudo-landmark points, we are required to calculate $\{K(\boldsymbol{x}, \boldsymbol{v}_t)\}_{t=1}^{p}$ for any vector $\boldsymbol{x}$. Here, insteading of calculating this value using the kernel function K as in the case of landmark points, DC-Pred++ *estimates* it as $K(\boldsymbol{x}, \boldsymbol{v}_t) \approx f_t(\boldsymbol{c})$ using a function $f_t : \mathbb{R}^m \longrightarrow \mathbb{R}$ using $\boldsymbol{c} = [K(\boldsymbol{x}, \boldsymbol{u}_1), \ldots, K(\boldsymbol{x}, \boldsymbol{u}_m)]$.

For stationary kernels, the functions $\{f_t\}_{t=1}^{p}$ are obtained by using the triangle inequality, using the fact that stationary kernel values only depend on $||\boldsymbol{x} - \boldsymbol{v}_t||$. For any general kernel, the functions are obtained by using a regression based approach via (low degree) polynomial basis functions. The details of these approaches can be found in [2].

The modification in the Nyström approach using the pseudo-landmark points is formulated in the following manner. First, we obtain $\bar{C} = [C, C']$ by augmenting the matrix C with the estimated values of $K(\boldsymbol{x}_i, \boldsymbol{v}_t)$ using f_t. Now, using this matrix $\bar{C}$, the kernel matrix G is approximated as:

$$G \approx \bar{G} = \bar{C}\bar{W}\bar{C}^T \text{ and } \bar{W} = \bar{C}^{\dagger}G(\bar{C})^T \tag{3}$$

Note, the value of $\bar{W}$ is now calculated using $\bar{C}$ and G itself, as the pseudoinverse can not be used since the values in matrix C are now approximate values (due to the use of functions f_t). It can be shown that taking $\bar{W} = \bar{C}^{\dagger}G(\bar{C})^T$ minimizes $\|\bar{G} - G\|_F$ if $\bar{G}$ is restricted to the range space of $\bar{C}$.

Now, to overcome the challenging fact that G itself is used to estimate G, in the RHS of the equation, we use a submatrix G_{sub} of the whole kernel matrix G and use the corresponding vectors only in $\bar{C}$. This speeds up the calculation and does not require the use of the whole kernel matrix.

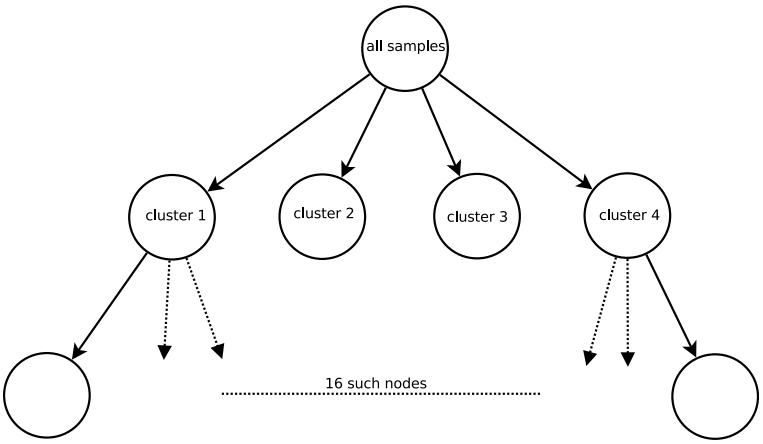
2.2.2 Weighted k-means

It is shown in [2] that it suffices to minimize kernel approximation error on $\{i\}$ with large $|\alpha_i^*|$, instead of all the input samples, as the vectors with small $|\alpha_i^*|$ do not contribute to the error significantly in case of kernel SVM.

For stationary kernels, a novel approach is suggested which involves using weighted k-means. First, perform weighted k-means on the input vectors using p as the number of clusters, $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ as the distance measure and $\{\bar{\alpha}_i^*\}_{i=1}^{n}$ (which are approximate values of $\{\alpha_i^*\}_{i=1}^{n}$ obtained using any fast approximate kernel SVM solver) as weights. Finally, we use the p cluster centroids hence obtained as the pseudo-landmark points, instead of choosing any points in $\mathbb{R}^d$. It is shown that this approach minimizes the error on $\{i\}$ with large $|\alpha_i^*|$ for kernel SVM.

2.2.3 Divide and conquer

To reduce the prediction time, the algorithm makes use of the approach proposed in [6]. It involves performing normal k-means i.e. unweighted k-means on the input vectors using some constant l as number of clusters and L2-norm (i.e. Euclidean norm) on the input space as the distance metric. This k-means is performed recursively to form a hierarchy of such clusters. An example of such clustering with $l = 4$ and two levels of hierarchy is given below:



Now, the pseudo-landmark points i.e. the weighted k-means cluster centroids earlier obtained by the weighted k-means are assigned to the nearest clusters (based on L2-norm). Now, a Nyström approximation model is trained on every cluster (at the lowest level), using these local pseudo-landmark points. The approximate values of $\{\alpha_i^*\}_{i=1}^n$ for the higher levels can be obtained using the *adaptive clustering* idea of [6]. For the prediction phase, the *early prediction* idea of [6] is used, which essentially returns the prediction of the local cluster model (at some pre-determined level), instead of the global model, for the test sample. As the value of c is now calculated only for that particular cluster, the prediction time is significantly reduced, while the approximation error is still low.

3 Local Deep Kernel Learning (LDKL)

3.1 Multiple Kernel Learning

The main idea behind multiple kernel learning is that it might be possible to improve errors by using some combination of kernels instead of a single kernel. The paper by Lanckriet et al. [7] tried to formulate the problem of learning a kernel using a semidefinite program. In a section of the paper, the authors also show that using combinaitons of kernels can lead to lower errors. This idea of combining kernels had alread been utilized in a few domain specific papers such as the ones by Pavlidis et al [8] and Ben Hur, Noble [9]. These papers use an unweighted sum of kernels.
In general, it may be better to learn the kernel as a convex combination of kernels but a simple algorithm for optimising such a kernel was only introduced by Bach et al [10].
These kernel combiations are global, in the sense that once we learn a set of weights corresponding to a set of kernels, we use them in the entire feature space. While these may be good, they do not utilize the information given by the data locally. For example, some points in a dataset may be more likely to be among one class than others in some region of the space, but learning a global kernel can only focus on minimizing error over the whole space and not make local optimizations.

3.2 Localized Multiple Kernel Learning (LMKL)

3.2.1 Algorithm

LMKL[11] utilizes the idea that assigning different weights to kernels in different regions may help improve classification accuracy. The mathematical model for this algorithm is:

$$y(\mathbf{x}) = sign(\sum_k p(\boldsymbol{w}_k|\mathbf{x})\mathbf{w}_k^t\phi_k(\mathbf{x}) + b) \tag{4}$$

$$p(\mathbf{w}_k|\mathbf{x}) = \frac{e^{\boldsymbol{\theta}_k^t\boldsymbol{x}+\theta_{0k}}}{\sum_m e^{\boldsymbol{\theta}_m^t\boldsymbol{x}+\theta_{0m}}} \tag{5}$$

$$\Theta = \{(\boldsymbol{\theta}_k, \theta_{0k})\} \tag{6}$$

Note that this is not a convex model due to the presence of the $p(\boldsymbol{W}_k|\boldsymbol{x})$ term. The algorithm to learn this is similar to the one in Rakotoamonjy et al [12]. It optimises the function by first optimising the kernel for a gating model and then updating the model. This steps alternate until convergence.

3.2.2 Pros and Cons

Experimental results show:

- Distinct kernels
  - Accuracy unchanged, support vectors ↓
- Same kernels
  - Accuracy ↑, support vectors ↓

3.3 Localized Deep Kernel Learning(LDKL)

LDKL[1] is based on the generalization of the idea of LMKL. It learns a non linear kernel as the matrix product of a local and a global kernel.

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = K_L(\boldsymbol{x}_i, \boldsymbol{x}_j) K_G(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

The mathematical model is:

$$y(\mathbf{x}) = sign(W^t(\boldsymbol{x})\phi_G(\boldsymbol{x})) \tag{7}$$

$$w_k = \sum_i \alpha_i y_i \phi_{L_k}(\boldsymbol{x}_i)\phi_G(\boldsymbol{x}_i), \phi_L \in \mathbb{R}^M \tag{8}$$

$$W = [\boldsymbol{w}_1, ..., \boldsymbol{w}_M] \tag{9}$$

$$W(\boldsymbol{x}) = W\phi_L(\boldsymbol{x}) \tag{10}$$

3.3.1 Local Kernel

The local kernel is a high dimensional tree structured kernel. For a deep representation, the kernel is chosen as follows:
ϕ_{L_k} stands for the k^{th} dimension of the feature space representation.

$$\phi_{L_k}(\boldsymbol{x}) = I_k(\boldsymbol{x}) f_{k_0}(\boldsymbol{x}, f_{k_1}(\boldsymbol{x}, ...(f_{k_R}(\boldsymbol{x}, 1)))) \tag{11}$$

where each k_i is the i^{th} ancestor of k and

$$I_k(\boldsymbol{x}) = \Pi_{l \in Ancestors(k)} \frac{1}{2}(sign(\boldsymbol{\theta}_l^t \boldsymbol{x}) + (-1)^{C(l)}) \tag{12}$$

$C(l) = 0$ if node l is its parents left child and $C(l) = 1$ if it is its parents right child

Note that LDKL would generate a piecewise smooth boundary if f_{k_r} would be a smooth function of $\boldsymbol{\theta}_{k_r}^t \boldsymbol{x}$ with $f_{k_r}(\boldsymbol{x}, z) = 0$ whenever $\boldsymbol{\theta}_{k_r}^t \boldsymbol{x} = 0$. This ensures that when ϕ_{L_k} tends to zero, so do its ancestors. In particular, the choice of the function for the results in the paper is taken as:

$$\phi_{L_k}(\boldsymbol{x}) = tanh(\sigma \boldsymbol{\theta}_k^{'t} \boldsymbol{x}) I_k(\boldsymbol{x}) \tag{13}$$

The indicator function is designed so that the kernel feature space is a tree shaped one. Since we only work with a path in the kernel, the overall prediction time for the algorithm is $O(DlogM)$ where D is the dimensionality of the vector $\boldsymbol{x}$ and M is the dimensionaliry of the local kernel space. (Assuming a linear global kernel
For a vector $\boldsymbol{x}$, the indicator function can be understood with the aid of the following series of diagrams:

Since the nodes that are visited are 0, 1 and 4, we will only pick up these components from the deep kernel function. Thus, if our local kernel has dimension m, the number of non zero dimensions in the feature space representation would be $log(m)$

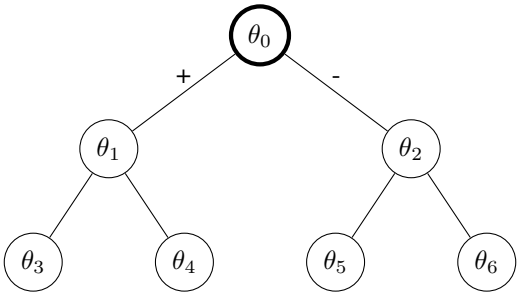


Figure 1: $\boldsymbol{\theta}_0\boldsymbol{x} > 0$

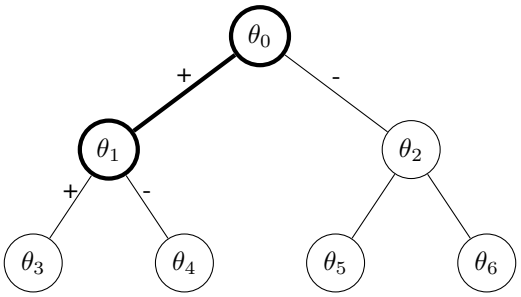


Figure 2: $\boldsymbol{\theta}_0\boldsymbol{x} > 0, \boldsymbol{\theta}_1\boldsymbol{x} < 0$

3.3.2 Algorithm

The primal problem for jointly learning Θ, Θ' and W

$$\min_{W,\Theta,\Theta'} P(W,\Theta,\Theta') = \frac{\lambda_W}{2}Tr(W^tW) + \frac{\lambda_\Theta}{2}Tr(\Theta^t\Theta) + \frac{\lambda_{\Theta'}}{2}Tr(\Theta'^t\Theta') + \sum_{i=1}^{N} L(y_i, \phi_L^t(\boldsymbol{x}_i)W^t\boldsymbol{x}_i)$$

where L is the hinge loss for binary classification, $L = max(0, 1 - y_i\phi_L^t(\boldsymbol{x_i})W^t\boldsymbol{x}_i)$

The algorithm used for optimizing this is a primal stochiastic sub-gradient descent. Note that the algorithm optimizes the primal function directly. The update rules for this algorithm are:

$$W^{j+1} = W^j - \eta_j \nabla_W P(W^j, \Theta^j, \Theta'^j, \boldsymbol{x}_i) \tag{14}$$

$$\Theta^{j+1} = \Theta^j - \eta_j \nabla_\Theta P(W^j, \Theta^j, \Theta'^j, \boldsymbol{x}_i) \tag{15}$$

$$\Theta'^{j+1} = \Theta'^j - \eta_j \nabla'_\Theta P(W^j, \Theta^j, \Theta'^j, \boldsymbol{x}_i) \tag{16}$$

where η_j is the step size at iteration j. The various gradients are defined as follows:

$$\nabla_{\boldsymbol{w}_k} P(\boldsymbol{x}_i) = \lambda_W \boldsymbol{w}_k - \delta_i y_i \phi_{L_k}(\boldsymbol{x}_i)\boldsymbol{x}_i \tag{17}$$

$$\nabla_{\boldsymbol{\theta}_k} P(\boldsymbol{x}_i) = \lambda_\Theta \boldsymbol{\theta}_k - \delta_i y_i \sum_l tanh(\sigma \boldsymbol{\theta}_l'^t \boldsymbol{x}_i)\nabla_{\boldsymbol{\theta}_k} I_l(\boldsymbol{x}_i)\boldsymbol{w}_l^t\boldsymbol{x}_i \tag{18}$$

$$\nabla_{\boldsymbol{\theta}'_k} P(\boldsymbol{x}_i) = \lambda_\Theta \boldsymbol{\theta}'_k - \delta_i y_i \sigma(1 - tanh^2(\sigma \boldsymbol{\theta}_k'^t \boldsymbol{x}_i))I_k(\boldsymbol{x}_i)\boldsymbol{w}_k^t\boldsymbol{x}_i\boldsymbol{x}_j \tag{19}$$

To make the optimisation tractable, and for ∇I to exist, we use a $tanh(.)$ parametrised by a scale parameter which is adaptively scaled to tend to sign(.) by the time convergence is reached. The relaxed I_k :

$$I_k(\boldsymbol{x}) = \prod_{I \in Ancestors(k)} \frac{1}{2}(tanh(s_i \boldsymbol{\theta}_l^t \boldsymbol{x}) + (-1)^{C(l)}) \tag{20}$$

The reason for choosing the primal is that the dual optimisation would have been a two stage alternating process which was found by the authors to be significantly more expensive.

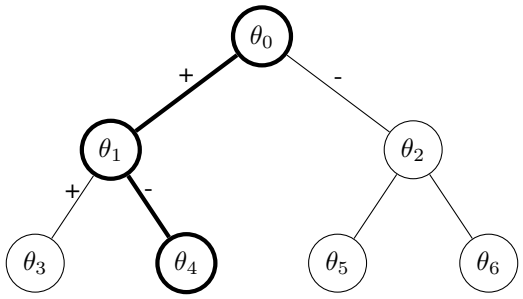


Figure 3: $\boldsymbol{\theta}_0\boldsymbol{x} > 0, \boldsymbol{\theta}_1\boldsymbol{x} < 0, \boldsymbol{\theta_4}$

4 Performance comparison

Data Set	Linear SVM	RBF-SVM	DC-Pred++	LDKL
CovType Train=522,910 Test=58,102 Dims=54	A = 76.32%	A = 91.21% P = 131,785x	A = 95.19% P = 18.8x T = 372s	A = 88.21% P = 32x T = 4095s
Letter Train=12,000 Test=6,000 Dims=16	A = 73.02%	A = 97.20% P = 1548x	A = 95.90% P = 12.8x T = 1.2s	A = 96.30% P = 33x T = 243s

Figure 4: A = Accuracy(%), P = Prediction Time(times Linear SVM), T = Training Time(s) Source: Jose et al., 2013 [1] and Hsieh et al., 2014 [2]

We see from the results that DC-Pred++ in general tends to perform the best among all these algorithms. The fast prediction times are a result of the early prediction idea (return the local prediction). Also, it seems that DC-Pred++ performs better than RBF-SVM on the covertype dataset. This unusual behaviour may be explained by the locality displayed by both the covertype dataset and the algorithm itself. A cover of a certain kind (land, water etc.) will tend to cluster locally and hence it lends itself to the cluster based prediction strategy used by DC-Pred++.

References

[1] Cijo Jose, Prasoon Goyal, Parv Aggrwal, and Manik Varma. Local deep kernel learning for efficient non-linear svm prediction. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 486–494, 2013.

[2] Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. Fast prediction for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pages 3689–3697, 2014.

[3] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. (EPFL-CONF-161322):682–688, 2001.

[4] Petros Drineas and Michael W Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *The Journal of Machine Learning Research*, 6:2153–2175, 2005.

[5] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Ensemble nystrom method. In *Advances in Neural Information Processing Systems*, pages 1060–1068, 2009.

[6] Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. A divide-and-conquer solver for kernel support vector machines. *arXiv preprint arXiv:1311.0914*, 2013.

[7] Gert RG Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I Jordan. Learning the kernel matrix with semidefinite programming. *The Journal of Machine Learning Research*, 5:27–72, 2004.

[8] Paul Pavlidis, Jason Weston, Jinsong Cai, and William Noble Grundy. Gene functional classification from heterogeneous data. In *Proceedings of the fifth annual international conference on Computational biology*, pages 249–255. ACM, 2001.

[9] Asa Ben-Hur and William Stafford Noble. Kernel methods for predicting protein–protein interactions. *Bioinformatics*, 21(suppl 1):i38–i46, 2005.

[10] Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004.

[11] Mehmet Gönen and Ethem Alpaydin. Localized multiple kernel learning. In *Proceedings of the 25th international conference on Machine learning*, pages 352–359. ACM, 2008.

[12] Alain Rakotomamonjy, Francis Bach, Stéphane Canu, and Yves Grandvalet. More efficiency in multiple kernel learning. In *Proceedings of the 24th international conference on Machine learning*, pages 775–782. ACM, 2007.