Scalable Dyadic Kernel Machines

Lester Mackey
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720
lmackey AT eecs DOT berkeley DOT edu

May 6, 2008

Abstract

In the dyadic data prediction (DDP) problem, we observe labeled pairs (dyads) drawn from a finite Cartesian product $M \times U$ and form predictions for the labels of unseen dyads. This results in a sparse, non-linear prediction problem, for which kernel machines, like the Support Vector Machine, are well suited. However, the release of the 100 million dyad Netflix dataset has brought the issue of DDP scalability to the forefront. Most kernel-machine solvers scale superlinearly in the number of data examples, making large-scale DDP infeasible. In this work, we explore techniques for enhancing the scalability of kernel machines for DDP. *En route*, we develop two natural reformulations of the kernel machine framework, designed to reflect and exploit the underlying structure of the DDP problem.

1 Introduction

This paper explores the use of kernel machines for prediction on large-scale *dyadic* data sets. Here, dyadic data is characterized by two finite sets of objects $M = \{m_1, ..., m_{n_M}\}$ and $U = \{u_1, ..., u_{n_U}\}$, a set of valid labels $R \subset \mathbb{R}$, and a training set of N observations $S = \{(p_1, r_1), ..., (p_N, r_N)\}$, where $r_i \in R$ is the observed label of the dyad (ordered pair) $p_i \in P = M \times U$. The goal of dyadic data prediction (DDP) is to predict the labels of unobserved dyads given the training set, S.

Instances of the DDP formulation have been studied extensively in a number of domain-specific settings including

1. *Collaborative filtering*: e.g. Predict the preferences or ratings that users will assign to items ([26])
2. *Bioinformatics*: e.g. Predict the allele-binding affinities of peptides ([22])
3. *Social networking*: e.g. Predict the unobserved relationships between members of a social network ([25])

Additionally, DDP is frequently the object of domain-independent study, with approaches ranging from statistical mixture models ([20]) to non-parametric Bayesian matrix factorization ([27]) to modified Support Vector Machines ([19]).

As a concrete example of DDP, consider the movie ratings prediction problem: given a collection of ratings assigned to movies by users, predict the ratings that users will give to other movies. In this setting, U represents a set of users, M is a set of movies, and R is the set of valid ratings a user can assign to movie. Typically, R is equal to the finite, ordinal set $\{1, ..., c\}$ for some maximum rating, c. In recent years, this problem has received a great deal of attention in the machine learning

literature ([1, 32, 31, 24]), due to the release of the 2.6 million dyad EachMovie dataset, the 100,000 and 1 million dyad MovieLens datasets ([29]), and, most recently, the 100 million dyad Netflix Prize dataset ([30]). We utilize the ratings prediction problem as a working example throughout the paper, paying particular attention to the issues of scalability raised by such large datasets.

The Support Vector Machine ([7]) and its cousins, the kernel machines, represent a challenging choice for large-scale DDP. On the one hand, kernel machines exhibit state-of-the-art performance in non-linear classification, regression, and ranking and are well-suited to settings with sparse, high-dimensional data. On the other hand, kernel machine solvers are plagued by poor scaling: typical solvers exhibit runtimes and/or memory requirements that scale superlinearly in the number of training examples. To combat this superlinear growth, we will consider several techniques for enhancing the scalability of kernel-machine learning and exploit the special structure of the DDP problem for further efficiency.

The remainder of the paper is structured as follows. In Section 2, we define the general kernel machine framework, discuss its limitations, and consider several methods for enhancing scalability. Section 3 introduces two reformulations of the kernel machine which benefit from the underlying structure of the DDP. Incorporating the enhancements of Section 2, we outline simple gradient-based algorithms for learning these Dyadic Kernel Machines. In Section 4 we discuss the details of kernel selection, and finally, in Section 5 we summarize our results and conclusions.

2 Kernel Machines

2.1 Defining kernel machines

In the prediction setting, we wish to select a function f which maps data points in P to labels in $\mathbb{R}$. The *kernel machine* is a procedure for selecting a prediction rule f by balancing the empirical loss of f on a training set with the complexity of f. More specifically, given a Mercer kernel ([28]) $k : P \times P \to \mathbb{R}$, a loss function $\ell : \mathbb{R}^2 \to \mathbb{R} \cup \{\infty\}$, and a real parameter $\lambda > 0$, we define the kernel machine over any training set $S \subseteq P \times \mathbb{R}$ as the following optimization problem

$$\min_{f \in \mathcal{H}} \frac{\lambda}{2} ||f||^2_{\mathcal{H}} + \frac{1}{|S|} \sum_{(p,r) \in S} \ell(r, f(p)) \tag{1}$$

where $\mathcal{H}$ is the unique Reproducing Kernel Hilbert Space (RKHS) associated with k ([2]). Note that the kernel machine has three parameters: the kernel k, the loss ℓ, and the trade-off parameter λ. The kernel determines the space of candidate predictors f, the loss measures the empirical performance of the predictor, and λ determines the relative importance of the two goals of minimizing empirical error and minimizing predictor complexity.

Typically, the loss is chosen to reflect the particular prediction task (classification, regression, ordinal regression) and the nature of labels to be predicted (discrete, continuous, ordinal). By varying the loss, we can recover a variety of well-known machine learning techniques as special examples of the kernel machine framework:

- *L-1 Soft-margin SVM* ([11]) ($\ell(r, f(p)) = |1 - r * f(p)|_+$) for binary classification[1]
- *L-2 Soft-margin SVM* ($\ell(r, f(p)) = (1 - r * f(p))^2_+$) for binary classification
- *Support Vector Regression* ([14]) ($\ell(r, f(p)) = |f(p) - r|_\epsilon$) for regression
- *Regularization Network* ([17]) ($\ell(r, f(p)) = (f(p) - r)^2$) for regression

Thus, the kernel machine framewok offers a rich class of regularized non-linear prediction methods.

A key property contributing to the appeal of the SVM in the non-linear prediction setting is its ability to find a non-linear, potentially infinite dimensional predictor in a feature space without explicitly mapping points into that feature space. This property is in fact enjoyed by all members of the kernel machine class and is a result of the following Representer Theorem ([33]):

[1] We recover the Hard-margin SVM in the limit as $\lambda \to 0$.

Theorem 2.1. Let f^* be a solution of Eq. 1 and $N = |S|$. Then f^* admits a representation of the form

$$f^* = \sum_{i=1}^{N} \alpha_i k(p_i, .) \tag{2}$$

Thus, the optimal predictor as determined by Eq. 1 can be written as a linear combination of the kernel function evaluated at each of the training data points. Our minimization over $f \in \mathcal{H}$ therefore reduces to a minimization over $\vec{\alpha} \in \mathbb{R}^N$. Indeed, if we let K^S be an empirical kernel matrix with $(K^S)_{ij} = k(p_i, p_j)$ and ith row $K_{i,.}$, then we may rewrite Eq. 1 as

$$\min_{\vec{\alpha} \in \mathbb{R}^N} \frac{\lambda}{2} \vec{\alpha}' K^S \vec{\alpha} + \frac{1}{N} \sum_{i=1}^{N} \ell(r_i, K^S_{i,.} \vec{\alpha}) \tag{3}$$

where we have implicitly utilized the reproducing property of k in reformulating the norm of f.

2.2 Complexity of kernel machine solvers

The abundance of algorithms developed for solving the SVM and its cousins is a testament to their popularity and power in solving prediction problems. This section gives a highlight of the most popular methods and their computational complexities. For simplicity, we focus on SVM-solvers. For a more detailed survey, see for example [34].

Interior point methods find the SVM solution by solving a sequence of unconstrained optimization problems with Newton or Quasi-Newton methods. Storage of the kernel matrix demands $O(N^2)$ memory, while matrix inversion or factorization leads to $O(N^3)$ running time ([18]).

Primal Newton methods apply Newton method updates to learn $\vec{\alpha}$ in Eq. 3 in $O(N * N_{sv} + N_{sv}^3)$ running time where N_{sv} is the maximum number of "support vectors" (training examples with non-zero loss) on any iteration. This complexity result assumes an $O(1)$ number of Newton updates are needed ([10]).

Decomposition methods operate on the dual representation of the SVM by grouping variables into sets and solving subproblems on each of the sets. The majority of decomposition methods yield superlinear scaling: e.g. the algorithm of [21] yields $O(N^2)$ time complexity. Recently, Joachims demonstrated in [23] that by reformulating the SVM problem and employing the cutting plane algorithm on top of a conventional decomposition solver, a linear time complexity could be achieved for *linear* SVMs. Unfortunately, when applied to kernelized SVMs, this method scales once more at $O(N^2)$.

Gradient-based primal methods learn the optimal predictors by performing gradient descent in α on Eq. 3 (or gradient descent in f on Eq. 1 in the linear SVM case). One particularly promising result by Shalev-Shwartz, Singer, and Srebro solves the *linear* SVM to within ϵ accuracy in $O(1/\epsilon)$ iterations. Since each iteration has time complexity independent of N, this yields a convergence bound with no dependence on the number of training examples. In the kernel setting, however, convergence does depend on the number of non-zero coefficients α_i on each iteration. In the worst case, this yields a $O(1/\epsilon^3)$ bound on time complexity which, though still independent of N, is decidedly worse.

In summary, while state-of-the-art techniques for linear SVMs do scale well in the number of training examples, non-linear kernel machines are penalized by their use of the kernel trick. This penalty stems from replacing f by its distributed representation in Eq. 2. Every evaluation of $f(p)$ in the expanded representation requires N_{bv} kernel evaluations, where N_{bv} represents the number of *basis vectors*: points p_i with non-zero kernel coefficients α_i. In the worst case, $N_{bv} = N$, leading to the observed $O(N)$ and at times $O(N^2)$ scale-ups in complexity. If we can reduce or bound the number of non-zero kernel coefficients to a small number independent of N, we can conceivably recapture the linear (or sublinear) scaling of linear SVMs. We explore the theme of sparsity inducing scalability in more detail in the next section.

2.3 Enhancements for scalability

Improving the scalability of kernel methods by inducing sparse representations for the kernel predictor is a subject which receives a great deal of attention in machine learning literature. In this section, we discuss a number of the more recent and popular methods.

2.3.1 Kernel Machines on a Budget

Budget methods, primarily studied in the context of the kernelized perceptron, maintain sparsity by explicitly capping the number of basis vectors in the kernel predictor expansion. When the number of basis vectors is small, these methods operate like typical, unconstrained algorithms. However, when a budget method receives a request to exceed its budget, it typically expels an older basis vector to make room. Thus, budget methods are primarily distinguished by their deletion criteria. The Forgetron ([13]) evicts the oldest basis vector while the algorithm of Cesa-Bianchi et. al. ([9]) evicts a random vector. The budget techniques of Crammer et. al. ([12]) and Weston et. al. ([35]) utilize greedy criteria for expulsion. With their Budget Perceptron, Crammer et. al. test how well each basis vector is classified after removal from the basis and evict the vector with the largest margin. The Tighter Budget Perceptron of [35] removes the example which minimizes the increase in training error. Moreover, in [35] it is shown that the greedy strategy of [12] is unstable in noisy settings. The remaining algorithms demonstrate good performance both in noisy and noiseless settings, although the randomized algorithm of [9] must often be averaged over several runs. Among these publications, the Forgetron is the only to offer a relative mistake bound for its algorithm, while [9] provides a bound on its model's expected number of mistakes.

2.3.2 Frequent Deletion

Frequent deletion techniques do not impose a hard limit on the number of basis vectors but instead attempt to maintain sparsity by frequently deleting redundant examples from the basis. This category is exemplified by the variable-cache perceptron of Cesa-Bianchi et. al. ([9]) and the online SVM approximator, LASVM ([6]), both of which alternate between adding and deleting vectors.

2.3.3 Exploiting Low-dimensional Structure

In many cases, the kernel matrix is of low rank c or can be approximated well by a c-rank decomposition. In these instances, the data feature vectors lie (nearly) in a c-dimensional subspace and can be accurately represented by a set of c spanning vectors. If we can discover such a spanning set, we need only maintain c coefficients for our predictor.

Online basis selection methods attempt to find a spanning set among the data points by iteratively constructing a basis and testing if new points fall into the span of that basis. Examples of this procedure include the Kernel Recursive Least Squares algorithm ([16]) and the Sparse Online Greedy Support Vector Regression method ([15]). Both methods utilize a least squares test of almost linear dependence to determine whether to introduce a training point as a new basis vector or to represent it as a linear combination of previous vectors.

Kernel matrix decomposition methods directly solve for the low-rank factorization of K^S and typically produce factorizations of the form $K^S = GG'$ with $G \in \mathbb{R}^{N \times c}$. The most common technique is the incomplete Cholesky decomposition with pivoting [18], a numerically stable factorization algorithm which decomposes K^S in $O(Nc^2)$ time with $O(Nc)$ storage. A more recent formulation by Bach and Jordan ([3]) incorporates side information into the factorization allowing for a more discriminative decomposition while exhibiting the same time complexities.

3 Dyadic Kernel Machines

In the previous section, we surveyed methods for improving the general scalability of kernel machine solvers. Additionally, we can consider problem specific methods of reducing time complexity. Dyadic data sets are especially rich in structure and redundancy, due to finite class sizes and the Cartesian product constraint. A natural way to capture this structure is to represent the dyad kernel, k, as a function of object kernels, k^M and k^U, which operate on pairs from M and pairs from

U respectively. This simple but general reparameterization frees us to independently evalute and construct object-specific kernels with memory requirements $|M|^2 + |U|^2$ instead of the prohibitive N^2 for the dyad kernel. When object class sizes make storing these kernel matrices prohibitive, we utilize the budget caps and basis selection techniques of Section 2.3 to find low-dimensional representations of the object kernels. Notably, if c_M, c_U, and c_N are the number of basis vectors extracted for each kernel, the factorization of the matrices K^M and K^U will only require $O(|M|c_M^2)$ and $O(|U|c_U^2)$ time instead of the $O(|N|c_N^2)$ time needed for decomposing k.

In the following subsections we will consider specific mappings from the object Hilbert spaces to the dyad RKHS and detail the Dyadic Kernel Machine induced by each such mapping.

3.1 Direct Sum DKM

To evaluate the kernel between dyads, the Direct Sum Dyadic Kernel Machine computes

$$k(p_i, p_j) = k^M(a_i, a_j) + k^U(b_i, b_j). \tag{4}$$

That is, each dyad kernel evaluation is a sum over the corresponding M-object and U-object evaluations. Abstractly, this corresponds to a direct sum between the M and U Hilbert spaces: $\mathcal{H} = \mathcal{H}^M \oplus \mathcal{H}^U$. Intuitively, this represents appending the U-feature vector onto the end of the M-feature vector and computing a full vector inner product (admittedly with different kernels for U and M). Utilizing this reparameterization, we can recast the classical kernel machine optimization problem into a form more suitable for our constrained setting.

The Representer Theorem ensures that any optimal predictor satisfies

$$f^* = \sum_{i=1}^{N} \alpha_i k(p_i, .) = \sum_{i=1}^{N} \alpha_i k^M(a_i, .) + \alpha_i k^U(b_i, .) = \sum_{a \in M} \alpha_a^M k^M(a, .) + \sum_{b \in U} \alpha_b^U k^U(b, .) \tag{5}$$

for $\alpha_a^M = \sum_{i=1}^{N} \alpha_i \mathbf{I}[a_i = a]$ and $\alpha_b^U = \sum_{i=1}^{N} \alpha_i \mathbf{I}[b_i = b]$.

The Direct Sum DKM optimization problem thus becomes

$$\min_{\vec{\alpha} \in \mathbb{R}^N} \frac{\lambda}{2} \vec{\alpha^M}' K^M \vec{\alpha^M} + \frac{\lambda}{2} \vec{\alpha^U}' K^U \vec{\alpha^U} + \frac{1}{N} \sum_{i=1}^{N} \ell(r_i, K^M_{a_i,.} \vec{\alpha^M} + K^U_{b_i,.} \vec{\alpha^U}) \tag{6}$$

where K^M and K^U are kernel matrices over M and U respectively.

Note that while the number of parameters has not decreased ($\vec{\alpha^M}$ and $\vec{\alpha^U}$ are implicitly functions of $\vec{\alpha}$), time complexity has diminished, as kernel evaluations now depend independently on K^U and K^M.

A second direct sum DKM formulation arises when we choose to ignore the constraints on α_a^M and α_b^U. That is, we can treat α_a^M and α_b^U as arbitrary vectors in $\mathbb{R}^{|M|}$ and $\mathbb{R}^{|U|}$. This allows us to narrow our search over $\vec{\alpha} \in \mathbb{R}^N$ to searches over $\vec{\alpha^M} \in \mathbb{R}^{|M|}$ and $\vec{\alpha^U} \in \mathbb{R}^{|U|}$. Since f^* admits this factored form, the optimal solutions of the two problems are equivalent. The unconstrained Direct Sum DKM optimization problem is thus

$$\min_{\vec{\alpha^M} \in \mathbb{R}^{|M|}, \vec{\alpha^U} \in \mathbb{R}^{|U|}} \frac{\lambda}{2} \vec{\alpha^M}' K^M \vec{\alpha^M} + \frac{\lambda}{2} \vec{\alpha^U}' K^U \vec{\alpha^U} + \frac{1}{N} \sum_{i=1}^{N} \ell(r_i, K^M_{a_i,.} \vec{\alpha^M} + K^U_{b_i,.} \vec{\alpha^U}) \tag{7}$$

The unconstrained formulation lends itself to a family of simple parameter update algorithms based on alternating gradient (or subgradient) descent. First we fix $\vec{\alpha^M}$ and compute the gradient of the objective function at $\vec{\alpha^U}$:

$$\nabla_{\vec{\alpha^U}} = \lambda K^U \vec{\alpha^U} + \frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial t_2} \ell(r_i, K^M_{a_i,.} \vec{\alpha^M} + K^U_{b_i,.} \vec{\alpha^U}) K^U_{.,b_i} \tag{8}$$

where $\frac{\partial}{\partial t_2} \ell$ is the partial derivative of the loss with respect to its second argument. We then perform the update $\vec{\alpha^U} \leftarrow -\eta \nabla \vec{\alpha^U}$ for some suitable learning parameter. Next, we fix $\vec{\alpha^U}$ and update $\vec{\alpha^M}$

accordingly. We repeat these steps until the change in objective function or parameter values is neglible.

Note that in the unconstrained direct sum DKM, the two parameter vectors are coupled only by the loss term, whereas in the constrained variant, the parameter vectors are both functions of the input α. One potential disadvantage of the unconstrained direct sum DKM is its inability to explicitly capitalize on this additional structural information.

3.2 Tensor Product DKM

The Tensor Product DKM explores a second natural mapping from object Hilbert spaces to dyad space:

$$k(p_i, p_j) = k^M(a_i, a_j)k^U(b_i, b_j). \tag{9}$$

Here, the dyad kernel evaluation is a product of the object kernel evaluations. Underlyingly, this corresponds to a tensor product between the M and U Hilbert spaces: $\mathcal{H} = \mathcal{H}^M \otimes \mathcal{H}^U$. Tensor products are a popular and natural method for comparing pairs of datapoints: Basilico and Hofmann suggest the use of the tensor product for kernelized perceptron training in ([4]), while Jacob and Vert utilize the tensor product in SVM prediction of allele binding ([22]).

In our setting, the tensor product reparameterization guarantees the following form for any optimal predictor

$$f^* = \sum_{i=1}^{N} \alpha_i k(p_i, .) = \sum_{i=1}^{N} \alpha_i k^M(a_i, .)k^U(b_i, .) = \sum_{(a,b)\in M\times U} \alpha_{ab} k^M(a, .)k^U(b, .) = K^M A K^U.$$

where $\alpha_{ab} = \alpha_i$ when $p_i = (a, b)$ and $\alpha_{ab} = 0$ otherwise, and $A \in \mathbb{R}^{|M|\times|U|}$ with $(A)_{ab} = \alpha_{ab}$.

In place of our original sparse vector, $\vec{\alpha}$, we now have the sparse matrix A to learn. Conceptually, the matrix $K^M A K^U$ contains the predictions of our model for every (a, b). To make a prediction for a particular (a, b) we simply choose the a, bth entry (ath row and bth column of $K^M A K^U$).

Under this formulation, the norms of optimal f^* admit the form

$$\begin{aligned}
||f||^2 &= \sum_{i,j} \alpha_i \alpha_j k(p_i, p_j) = \sum_{i,j} \alpha_i \alpha_j k^M(a_i, a_j)k^U(b_i, b_j) \\
&= \sum_{(a_1,b_1)\in M\times U} \sum_{(a_2,b_2)\in M\times U} \alpha_{a_1 b_1}\alpha_{a_2 b_2} k^M(a_1, a_2)k^U(b_1, b_2) \\
&= \sum_{(a_1,b_1)\in M\times U} \alpha_{a_1 b_1} f((a_1, b_1)) = \sum_{a\in M} K^M_{a,.} A K^U (A_{a,.})' \\
&= Tr(K^M A K^U A')
\end{aligned}$$

where we have implicitly used the kernel reproducing property.

Our Tensor Product DKM minimization problem is therefore:

$$\min_{A\in\mathbb{R}^{|M|\times|U|}} \frac{\lambda}{2} Tr(K^M A K^U A') + \frac{1}{N}\sum_{i=1}^{N} \ell(r_i, K^M_{a_i,.} A K^U_{.,b_i}) \tag{10}$$

If we ignore the constraint on A that unobserved entries are all zero, Eq. 10 readily yields a family of (sub)gradient descent algorithms for learning the optimal parameter matrix. Below, we list the general gradient formula as well as the gradient under a number of specific loss functions.

- **General:** $\nabla_A = \lambda K^M A K^U + \frac{1}{N}\sum_{i=1}^{N} \frac{\partial}{\partial t_2}\ell(r_i, K^M_{a_i,.} A K^U_{.,b_i}) K^M_{.,a_i} K^U_{b_i,.}$
- **L1-SVM:** $\nabla_A = \lambda K^M A K^U + \frac{1}{N}\sum_{i=1}^{N} \mathbf{I}[r_i K^M_{a_i,.} A K^U_{.,b_i} - 1 > 0] r_i K^M_{.,a_i} K^U_{b_i,.}$
- **L2-SVM:** $\nabla_A = \lambda K^M A K^U + \frac{1}{N}\sum_{i=1}^{N} 2|r_i K^M_{a_i,.} A K^U_{.,b_i} - 1|_+ r_i K^M_{.,a_i} K^U_{b_i,.}$

- **SVR:** $\nabla_A = \lambda K^M A K^U + \frac{1}{N}\sum_{i=1}^{N}(\mathbf{I}[K^M_{a_i,.} A K^U_{.,b_i} - r_i > \epsilon] - \mathbf{I}[K^M_{a_i,.} A K^U_{.,b_i} - r_i < -\epsilon])K^M_{.,a_i} K^U_{b_i,.}$
- **RN:** $\nabla_A = \lambda K^M A K^U + \frac{1}{N}\sum_{i=1}^{N}(K^M_{a_i,.} A K^U_{.,b_i} - r_i)K^M_{.,a_i} K^U_{b_i,.}$

The updates induced by these gradient equations are appealing in their simplicity but fundamentally impractical: each iteration requires an update of $|M||U|$ matrix entries, and this number can be far larger than N in practice. To make learning in the tensor product DKM practical, we will need to revisit the ideas of Section 2.3. We can employ a budget-capped basis selection or matrix factorization method to one or both of the kernels matrices to obtain low rank approximations. Indeed, assume we have found low-rank decompositions: $K^M = G^M G^{M\prime}$ and $K^U = G^U G^{U\prime}$ where $c^M = rank(G^M)$ and $c^U = rank(G^U)$, and let $B = G^{M\prime} A G^U$. Then the Representer Theorem tells us that an optimal f^* can be expressed as $G^M B G^{U\prime}$. Moreover, we may rewrite our DKM optimization problem to reflect this factorization

$$\min_{B \in \mathbb{R}^{c^M \times c^U}} \frac{\lambda}{2} Tr(BB') + \frac{1}{N}\sum_{i=1}^{N} \ell(r_i, G^M_{a_i,.} B G^U_{.,b_i}) \tag{11}$$

Thus in the factorized setting, our tensor product DKM is transformed from an optimization problem over the prohibitely large space $\mathbb{R}^{|M|\times|U|}$ to a problem over the decidedly tractable space $\mathbb{R}^{c^M \times c^U}$. Notably, this drastic reduction in complexity was achieved by first exploiting the problem-specific structure of the DDP and then employing general techniques for enhanced scalability. All of the gradient based algorithms discussed earlier carry over easily to this setting. In particular the general gradient equation for the factorized tensor product DKM has an even simpler form

$$\nabla_B = \lambda B + \frac{1}{N}\sum_{i=1}^{N} \frac{\partial}{\partial t_2} \ell(r_i, G^M_{a_i,.} B G^U_{.,b_i})(G^M_{a_i,.})'(G^U_{.,b_i})'$$

On each iteration of gradient descent, we need only update the $c^M c^U$ elements of B.

4 Kernel Selection

4.1 Kernel Types

A final requirement for the successful deployment of kernel machines in the large-scale DDP setting is the selection of appropriate kernels. Several classes of kernels recommend themselves to the DDP problem:

- **Classical kernels** include the linear kernel ($k(x,y) =< x, y >$), the polynomial kernel ($k(x,y) =< x, y >^d$), and the Gaussian kernel or radial basis function ($k(x,y) = \exp(\frac{-||x-y||^2}{\sigma^2})$). These kernels have been extensively studied in the literature and have demonstrated impressive discrimination across a variety of domains and prediction tasks.
- **Domain-specific kernels** refer to kernels well suited to discrimination in a particular domain. In [4], the authors suggest a Pearson correlation kernel ($k(x,y) = corr(x,y)$) and a squared correlation kernel ($K^{corrSqd} = K^{corr} K^{corr}$) for the collaborative filtering setting. The squared correlation measures similarities between the correlation structures of two objects.
- **The identity kernel** ($\delta(x,y) = \mathbf{I}[x = y]$) maximally distinguishes objects and induces separability.
- **Attribute kernels** allow for the seamless integration of external data into the kernel machine model. For exameple, in the movie ratings prediction setting, we might want to incorporate demographic information about the users or genre information about the movies into our model.

4.2 Combining Kernels

While any of the kernels mentioned could be utilized in isolation, combinations of these kernels may lead to greater predictive power. In [4], the authors recommend selecting several kernels for each object class and combining them additively into a final object kernel. The authors report sizeable reductions in generalization error when following this procedure. Additionally, a well-known technique for incorporating a bias term into kernel machines consists of adding a kernel of all ones to an existing kernel ([34]). This corresponds to extending each training vector in feature space with an additional constant feature. Finally, the L-2 penalized soft-margin C-SVM with kernel k is known to be equivalent to the hard-margin SVM with kernel $k + \frac{\delta}{C}$, where δ is the identity kernel ([5]). Thus, combination with the identity kernel can induce data separability and expand the applicability of algorithms designed for the hard-margin setting.

5 Results and Conclusions

Results of experiments on the Netflix Prize dataset are forthcoming.

In summary, we have studied the problem of kernel machine scalability for dyadic data prediction. We have surveyed a number of effective complexity reduction techniques from the literature and introduced two Dyadic Kernel Machines, reformulations of the standard kernel machine which take advantage of the special structure inherent to the DDP. We have demonstrated that the unification of these problem-specific and general techniques for scalability results in significant reductions in problem complexity, placing large-scale dyadic data prediction well within the reach of kernel machine learning.

References

[1] D. Agarwal, S. Merugu: Predictive discrete latent factor models for large scale dyadic data. KDD 2007: 26-35.

[2] N. Aronszajn, Theory of Reproducing Kernels, Transactions of the American Mathematical Society, volume 68, number 3, pages 337-404, 1950.

[3] F. R. Bach and M. I. Jordan. Predictive low-rank decomposition for kernel methods. In ICML, 2005.

[4] J. Basilico and T. Hofmann. Unifying Collaborative and Content-Based Filtering. International Conference on Machine Learning (ICML), 2004.

[5] A. Bordes and L. Bottou: The Huller: a simple and efficient online SVM, Machine Learning: ECML 2005, 505-512, Lecture Notes in Artificial Intelligence, LNAI 3720, Springer Verlag, 2005.

[6] A. Bordes, S. Ertekin, J. Weston and L. Bottou: Fast Kernel Classifiers with Online and Active Learning, Journal of Machine Learning Research, 6:1579-1619, September 2005.

[7] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, 5th Annual ACM Workshop on COLT, pages 144-152, Pittsburgh, PA, 1992. ACM Press.

[8] Boyd, S., and Vandenberghe, L. (2004). Convex optimization. Cambridge University Press.

[9] N. Cesa-Bianchi, C. Gentile: Tracking the Best Hyperplane with a Simple Budget Perceptron. COLT 2006: 483-498.

[10] O. Chapelle. Training a Support Vector Machine in the Primal, Neural Computation, in press.

[11] C. Cortes and V. Vapnik, "Support-Vector Networks, Machine Learning, 20, 1995.

[12] K. Crammer, J. Kandola and Y. Singer. Online Classification on a Budget. Proceedings of the Sixteenth Annual Conference on Neural Information Processing Systems (NIPS), 2003.

[13] , O. Dekel, S. Shalev-Shwartz, and Y. Singer. The Forgetron: A Kernel-Based Perceptron on a Budget. SIAM Journal on Computing, 37(5):1342-1372, 2008.

[14] H. Drucker, C. Burges, L. Kaufman, A. Smola and V. Vapnik (1997). "Support Vector Regression Machines". Advances in Neural Information Processing Systems 9, NIPS 1996, 155-161, MIT Press.

[15] Y. Engel, S. Mannor, and R. Meir. Sparse online greedy support vector regression. In 13th European Conference on Machine Learning, 2002.

[16] Y. Engel, S. Mannor, and Ron Meir. The Kernel Recursive Least Squares Algorithm (IEEE Trans. on Signal Processing, 52(8):2275-2285, 2004).

[17] T. Evgeniou, M. Pontil, and T. Poggio. Regularization Networks and Support Vector Machines. Advances in Computational Mathematics 13 (2000) 1, pages 1-50.

[18] Fine, S., and Scheinberg, K. (2001). Efficient svm training using low-rank kernel representations. JMLR, 2, 242264.

[19] S. Hochreiter and K. Obermayer. Support vector machines for dyadic data, Neural Computation, v.18 n.6, p.1472-1510, June 2006.

[20] T. Hofmann, J. Puzicha, and M. I. Jordan (1999). Unsupervised learning from dyadic data. In Advances in Neural Information Processing Systems, Vol. 11, MIT Press.

[21] Hush, D., Kelly, P., Scovel, C., and Steinwart, I. (2006). Qp algorithms with guaranteed accuracy and run time for support vector machines. JMLR.

[22] L. Jacob and J.-P. Vert, "Efficient peptide-MHC-I binding prediction for alleles with few known binders", Bioinformatics, 24(3):358-366, 2008.

[23] Joachims, T. (2006). Training linear svms in linear time. KDD.

[24] R. Bell and Y. Koren, "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights", IEEE International Conference on Data Mining (ICDM'07), 2007.

[25] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In Proceedings of the Twelfth International Conference on Information and Knowledge Management, 2003, 556-559.

[26] B. Marlin. Collaborative filtering: A machine learning perspective. Master's thesis, University of Toronto, 2004.

[27] E. Meeds, Z. Ghahramani, R. Neal, and S. Roweis. Modeling dyadic data with binary latent factors. In Advances in Neural Information Processing Systems, 2007.

[28] J. Mercer, "Functions of positive and negative type, and their connection with the theory of integral equations" Proc. Roy. Soc. London Ser. A , 83 (1908) pp. 6970.

[29] MovieLens Datasets: http://www.grouplens.org/data/.

[30] Netflix Prize Dataset: http://www.netflixprize.com/download/.

[31] J. Rennie, N. Srebro. Fast maximum margin matrix factorization for collaborative prediction, Proceedings of the 22nd international conference on Machine learning, p.713-719, August 07-11, 2005.

[32] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann machines for collaborative filtering. ICML 2007: 791-798.

[33] B. Scholkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In Proceedings of the 14th Annual Conference on Computational Learning Theory, pages 416426, 2001.

[34] S. Shalev-Shwartz, Y. Singer, and N. Srebro. "Pegasos: Primal Estimated sub-GrAdient SOlver for SVM" ICML 2007.

[35] J. Weston, A. Bordes and L. Bottou. Online (and Offline) on an Even Tighter Budget, Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, Jan 6-8, 2005, Savannah Hotel, Barbados, 413-420, Edited by Robert G. Cowell and Zoubin Ghahramani, Society for Artificial Intelligence and Statistics, 2005.