Hash-SVM: Scalable Kernel Machines for Large-Scale Visual Classification

Yadong Mu*, Gang Hua†, Wei Fan*, Shih-Fu Chang‡
*Noah's Ark Lab, Huawei Hong Kong †Stevens Institute of Technology ‡Columbia University
Email: muyadong@gmail.com, ghua@stevens.edu, david.fanwei@huawei.com, sfchang@ee.columbia.edu

Abstract

This paper presents a novel algorithm which uses compact hash bits to greatly improve the efficiency of non-linear kernel SVM in very large scale visual classification problems. Our key idea is to represent each sample with compact hash bits, over which an inner product is defined to serve as the surrogate of the original nonlinear kernels. Then the problem of solving the nonlinear SVM can be transformed into solving a linear SVM over the hash bits. The proposed Hash-SVM enjoys dramatic storage cost reduction owing to the compact binary representation, as well as a (sub-)linear training complexity via linear SVM. As a critical component of Hash-SVM, we propose a novel hashing scheme for arbitrary non-linear kernels via random subspace projection in reproducing kernel Hilbert space. Our comprehensive analysis reveals a well behaved theoretic bound of the deviation between the proposed hashing-based kernel approximation and the original kernel function. We also derive requirements on the hash bits for achieving a satisfactory accuracy level. Several experiments on large-scale visual classification benchmarks are conducted, including one with over 1 million images. The results show that Hash-SVM greatly reduces the computational complexity (more than ten times faster in many cases) while keeping comparable accuracies.

1. Introduction

Kernel based methods [23] are powerful machine learning tools for visual classification tasks. Despite the enhanced data separability through feature dimension uplifting, non-linear kernel machines suffer from the high time/space complexities associated with the need to operate the kernel matrix. A naive kernel SVM solver is prohibitively expensive either by loading pre-computed kernel matrix into computer memory (*e.g.*, 1 million data requires several tera-bytes memory) or tremendous kernel function execution. The scalability issue of kernel machines has inevitably become a bottleneck for advancing the frontiers of large-scale visual classification systems, which motivates recent research interest in developing more scalable solvers.

In this work, we propose a scalable kernel learning algorithm named Hash-SVM, which demonstrates striking superiority in terms of both training time and memory requirement particularly in a large-scale setting. The idea is applicable to general kernel methods. Without loss of generality we choose kernel *support vector machine* (SVM) [6] as the exemplar application. Though parallel systems prove useful for scaling up large-scale learning techniques after properly tailoring the machine learning algorithms [3], we target single machines with limited memory capacity which do not require the expensive cost (communication, energy, hardware etc.) of parallel systems. The proposed algorithm represents the first algorithm that explores the idea of approximating arbitrary nonlinear kernels using compact hash bits towards accelerated kernel machine optimization. The idea is a natural marriage of the hashing technique and kernel-based methods. Our work presents the following key contributions:

Hash Bits for Surrogate Kernel Function: Recent years have witnessed the success of accelerated similarity search by using binary hash bits as the signatures of the original samples [1]. As a new exploration of the hashing technique, we propose to define a linear function over these hash bits, which serves as a surrogate for the original highly non-linear kernel function. This idea benefits the SVM optimization in two-folds. First, the linear surrogate kernel function transforms the problem to a simple linear SVM. Therefore the proposed Hash-SVM can avoid the super-linear time complexity typically required by classic kernel SVM solvers.

Meanwhile, after obtaining the compact hash bits for the data, we need not access the original (possibly ultra high-dimensional) data. It immediately enables loading all data into the main memory in a single-machine setting, avoiding the time-consuming data swap between memory and disk as in classic kernel SVM. Our theoretic analysis shows that the required number of hash bits for a specific level of accuracy is irrelevant to the intrinsic dimension of the kernel-induced Hilbert space. In most cases several thousand bytes are sufficient to represent a sample, which resolves the memory bottleneck in conventional kernel learning.

Similarity-Preserving Kernel Hashing Scheme: A crucial ingredient of Hash-SVM is a hashing scheme which operates in the kernel space and provably preserves the data similarity defined by the original kernel function. Such hashing schemes have been critically missing in the literature and we propose a novel solution to fill the gap. Conventional kernel hashing algorithms (*e.g.*, KLSH [15]) rely on estimation of the covariance matrix in the kernel-induced Hilbert space. Such estimation is highly unreliable considering the gap between the huge number of free parameters to estimate and the limited number of samples that KLSH uses. In contrast, our proposed *kernelized random subspace hashing* (KRSH) makes the least assumption about the data.

The remainder of this paper is organized as the following: after a brief survey of related work in Section 2, we present the idea of Hash-SVM as well as the proposed hashing scheme in Section 3, followed by theoretic analysis in Section 4 and experimental evaluation in Section 5. Final conclusions are included in Section 6.

2. Related Work

Related work can largely be categorized into two lines. We briefly summarize each of them.

SVM Optimization by Kernel Linearization: Scaling up non-linear kernel SVM is a long-standing topic in machine learning and optimization. A variety of solutions have been proposed in the past decade [21, 2, 26, 30, 3]. On the other hand, training linear SVM [14, 25, 32, 24, 12, 10] is favored in practice owing to its low computation/storage overhead and being free from storing support vectors. The linear or sublinear complexity of these algorithms enables efficient training of linear SVM on gigantic samples, which motivates the adventure of linearizing non-linear kernels to convert the otherwise nonlinear SVM problem into linear SVM. Examples include the algorithms for shift-invariant kernels (*e.g.*, Gaussian, Cauchy or Laplacian) [22] or homogeneous additive kernels [27]. The work in [17] provides a convergent analytic series to approximate χ_2 kernel defined on histogram feature. Besides the afore-mentioned random features derived from specific kernel properties, another strong competitor is the Nyström method [9, 16], which relies on landmark sampling for kernel approximation [29, 31].

Hashing Technique for Large-Scale Optimization: The original motivation of locality-Sensitive hashing (LSH) [13, 5, 8, 1] is to find an approximate nearest neighbor for a query point in sub-linear time. It is only recently discovered the power of the hashing techniques for large-scale optimization. For example, Li et al. [18] use b-bit minwise hashing to compress high-dimensional sparse text feature. Mu et al. [20] identify two common operations in a family of large-scale learning algorithms and make acceleration by concomitant statistics. These works are both limited to special kernels (Jaccard index and cosine similarity respectively), leaving the hashing scheme for general kernels an open problem in the field. Existing hashing schemes for general kernels either lack theoretic guarantee [11, 19] or suffer from unreliable estimate of the covariance matrix in high-dimensional (even infinite) Hilbert space (*e.g.*, kernelized locality sensitive hashing [15]).

3. The Proposed Algorithm

This section details our proposed Hash-SVM. We first present the idea of hash bit based kernel approximation (Section 3.1), which is a principled method and the kernel approximation error is highly dependent on the choice of hashing functions. Section 3.2 shows our proposed kernel hashing algorithm, which is surprisingly simple yet effective. Finally Section 3.3 elaborates on how the idea is seamlessly incorporated into optimizing kernel SVMs.

Notations: Let $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$ denote two points in a specific feature space, whose similarity can be measured by a well-defined kernel function $\kappa(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_2)$. Here $\phi(\cdot)$ defines a mapping function to a high-dimensional reproducing kernel Hilbert space (RKHS) in $\mathbb{R}^D$ ($D \gg d$). $\phi(\cdot)$ is often implicitly defined and we need not specify it. $\|\phi(\mathbf{x})\|_{\mathcal{H}}$ represents the vector norm $\sqrt{\langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle}$ in RKHS. In a supervised learning setting, let $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be the training data which are associated with ground truth binary labels $\mathbf{y} = (y_1, \ldots, y_n)^\top$. For notation simplicity we also use $\mathbf{\Phi} = [\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)]$ and $\mathbf{K} = \mathbf{\Phi}^\top \mathbf{\Phi} \in \mathbb{R}^{n \times n}$ to represent the data collection and kernel matrix respectively.

3.1. Surrogate Kernel Function with Hash Bits

Review of LSH: A binary hashing function defines a mapping $h(\mathbf{x})$ from $\mathbb{R}^D$ to the discrete set $\{0, 1\}$. In practice, each sample will be fed into a number of independent random hashing functions. Let $H_k(\mathbf{x}) = (h_1(\mathbf{x}), \ldots, h_k(\mathbf{x}))$ be the k-bit hash code for $\mathbf{x}$. Hash code dissimilarity is measured by the Hamming distance $D_{ham}(H_k(\mathbf{x}_1), H_k(\mathbf{x}_2))$ which returns the count of different bits.

LSH [13, 1] is a special type of hashing algorithms with the *locality-sensitive* property, which basically states that if two samples are similar in the original feature space, their corresponding hash codes shall be alike. A key notion to quantify this property is *collision probability*, which is defined as $Pr(h(\mathbf{x}_1) = h(\mathbf{x}_2))$ (the expectation of identical hash bit over all possible hashing functions). By definition, the normalized Hamming distance $D_{ham}(H_k(\mathbf{x}_1), H_k(\mathbf{x}_2))/k$ converges to $1 - Pr(h(\mathbf{x}_1) = h(\mathbf{x}_2))$ when k approaches infinity. The locality-sensitive property also implies that the collision probability should be monotonically increasing with respect to the pairwise data similarity.

Surrogate Kernel Function: It is easily seen that an arbi-

trary kernel function can be written as:

$$\begin{aligned} & \kappa(\mathbf{x}_1, \mathbf{x}_2) \\ = \; & \|\phi(\mathbf{x}_1)\|_{\mathcal{H}} \times \|\phi(\mathbf{x}_2)\|_{\mathcal{H}} \times \cos(\theta_{\mathbf{x}_1,\mathbf{x}_2}), \end{aligned} \quad (1)$$

where $\theta_{\mathbf{x}_1,\mathbf{x}_2}$ is the angle between $\phi(\mathbf{x}_1), \phi(\mathbf{x}_2)$ in RKHS.

The corner stone of our proposed Hash-SVM is a surrogate function $\widehat{\kappa}(\mathbf{x}_1, \mathbf{x}_2)$ to Equation (1). To ensure the surrogate kernel function is in linear form, a simple way is to assume that it is a linear function with respect to the normalized Hamming distance, parameterized by a and b:

$$\begin{aligned} \widehat{\kappa}(\mathbf{x}_1, \mathbf{x}_2) = & \|\phi(\mathbf{x}_1)\|_{\mathcal{H}} \times \|\phi(\mathbf{x}_2)\|_{\mathcal{H}} \\ & \times \left(a + b \cdot \frac{D_{ham}(H_k(\mathbf{x}_1), H_k(\mathbf{x}_2))}{k} \right). \end{aligned} \quad (2)$$

To calibrate the values of a, b, note that when the normalized Hamming distance in Equation (2) achieves the value of 1 (or 0), $\cos(\theta_{\mathbf{x}_1,\mathbf{x}_2})$ in Equation (1) reaches the value of -1 (or 1). Such correspondences immediately give us the choice that $a = 1$ and $b = -2$. In fact, the surrogate kernel function always induces positive semi-definite (p.s.d.) kernel matrix.

Theorem 3.1. *The surrogate kernel function in Equation (2) on binary hash bits defines a linear p.s.d. kernel.*

Proof. The claim trivially holds by verifying that

$$\begin{aligned} & 1 - \frac{2}{k} D_{ham}(H_k(\mathbf{x}_1), H_k(\mathbf{x}_2)) \\ = \; & \frac{1}{k} \sum_{i=1 \ldots k} (2h_i(\mathbf{x}_1) - 1) \cdot (2h_i(\mathbf{x}_2) - 1), \end{aligned} \quad (3)$$

which introduces the following explicit feature mapping:

$$\mathbf{h}(\mathbf{x}) = \left(\frac{2h_1(\mathbf{x}) - 1}{\sqrt{k}}, \ldots, \frac{2h_k(\mathbf{x}) - 1}{\sqrt{k}} \right). \quad (4)$$

The feature mapping is linear with respect to the hash bits and it is well known that all kernel matrices for linear kernels are p.s.d.. □

3.2. Kernelized Random Subspace Hashing

To ensure $\kappa(\mathbf{x}_1, \mathbf{x}_2) \approx \widehat{\kappa}(\mathbf{x}_1, \mathbf{x}_2)$ for arbitrary points $\mathbf{x}_1, \mathbf{x}_2$, the hashing scheme should be deliberately designed to properly approximate $\cos(\theta_{\mathbf{x}_1,\mathbf{x}_2})$, which is a non-trivial task. Given the assumption that only the information of the kernel value $\kappa(\mathbf{x}_1, \mathbf{x}_2)$ is accessible, it is impossible to apply the classical random vector based hashing [8]. The most popular work named KLSH proposed by Kulis et al. [15] enables drawing Gaussian random vector in RKHS by applying *central limit theorem* (CLT) to the training data. However, KLSH needs to estimate the covariance matrix of the data in possibly infinite-dimensional RKHS using only a partial set of the training data, which is computationally unreliable. Most other kernel hashing schemes [19, 11] learn the hashing functions in a data-driven manner and lack theoretic collision analysis, which are thus not satisfactory.

Algorithm 1 Algorithmic Pipeline of Kernelized Random Subspace Hashing (KRSH)

Input: data set $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, sizes $m < n$ (random subspace dimension) and k (hash bit number).
Output: bases matrix and Gaussian random vector pair $(\mathbf{V}_i, \mathbf{w}_i)$, $i = 1 \ldots k$ used for defining the hashing functions, and binary hash bits $h_1(\mathbf{x}), \ldots, h_k(\mathbf{x})$ for any new sample $\mathbf{x}$;
Phase of Hash Function Generation
Use the kernelized Gram-Schmidt process in Section 3.2 to obtain bases matrix $\mathbf{V}$ in RKHS;
Draw random vector $\mathbf{w} \in \mathbb{R}^m$ from the normal distribution to define the hashing function;
Phase of Hashing New Samples
for $i = 1$ **to** k **do**
  $h_i(\mathbf{x}) = 1$ **if** $\mathbf{w}_i^\top \mathbf{V}_i^\top \phi(\mathbf{x}) \geq 0$, **otherwise** 0;
end for

We propose a strikingly simple scheme named *kernelized random subspace hashing* (KRSH) to circumvent the aforementioned challenge in kernel hashing. It is applicable to arbitrary kernels. The key idea is to randomly generate the orthogonal bases for an m-dimensional subspace in kernel-induced RKHS from the subspace spanned by all training data, *i.e.*, **span**$(\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n))$. Afterwards each sample is projected into this random subspace, obtaining a new representation with m dimensions. The final data hashing operates on the new representation by employing the classical hashing scheme in [5]. The pseudo-code is found in Algorithm 1.

The proposed KRSH is inspired by the seminal work of KPCA [23]. The key operation of KRSH is maintaining a set of landmarks $\{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$ which defines a random subspace. The set is initialized to be empty and iteratively expanded via Gram-Schmidt process in RKHS. In other words, after $k-1$ iterations, we will obtain $k-1$ orthogonal bases in RKHS. Denote them by $\mathbf{V}^{(k-1)} = [\mathbf{v}_1, \ldots, \mathbf{v}_{k-1}]$, where the i-th basis $\mathbf{v}_i \in \mathbb{R}^D$ is a linear combination of $\phi(\mathbf{z}_1), \ldots, \phi(\mathbf{z}_i)$ and $\mathbf{v}_i^\top \mathbf{v}_j = 0$ if $i \neq j$. To obtain $\mathbf{v}_k$, a new sample $\mathbf{x}$ is drawn from the training samples. We first calculate the residual vector $\delta_{\mathbf{x}}$ for $\mathbf{x}$ (which is orthogonal to any $\mathbf{v}_i \in \mathbf{V}^{(k-1)}$) by subtracting its projections onto the bases in $\mathbf{V}^{(k-1)}$. The computation can be accomplished merely using the kernel function $\kappa(\cdot, \cdot)$, whose details are omitted due to space limit. If the residual vector has negligible magnitude in RKHS, it will be discarded for numerical stability. Otherwise we set $\mathbf{z}_m \leftarrow \mathbf{x}$, $\mathbf{v}_k \leftarrow \delta_{\mathbf{x}} / \sqrt{\langle \delta_{\mathbf{x}}, \delta_{\mathbf{x}} \rangle}$ to expand the landmark set.

After obtaining the random subspace, we perform the hashing procedure proposed by Datar et al. [8] within the

subspace. Specifically, for a new sample $\mathbf{x}$, $h(\mathbf{x}) = 1$ if $\mathbf{w}^\top \mathbf{V}^\top \phi(\mathbf{x}) \geq 0$, otherwise 0, where $\mathbf{w} \in \mathbb{R}^m$ denotes a random vector whose elements are drawn from the normal distribution (zero-mean and unit-variance 1-D Gaussian).

3.3. Integrating Hash Bits with Linear SVM

Given the training data $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and corresponding binary labels $\mathbf{y} = \{y_1, \ldots, y_n\}$, according to Theorem 3.1 the approximate solution of kernel SVM can be pursued through linear SVM:

$$\min_{\omega} \quad \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{n} \max\left\{1 - y_i \cdot \sqrt{\kappa(\mathbf{x}_i, \mathbf{x}_i)} \cdot \omega^\top \mathbf{h}(\mathbf{x}_i),\ 0\right\}, \quad (5)$$

where $\mathbf{h}(\mathbf{x}_i)$ is the transformed hash bit vector for sample $\mathbf{x}_i$ whose definition is found in Equation (4). The factor $\sqrt{\kappa(\mathbf{x}_i, \mathbf{x}_i)}$ is to compensate the vector norms in Equation (2), which must be kept when samples have non-uniform lengths in RKHS. The hyperplane vector ω will be learned and used for determining the label of a new sample according to the prediction function $f(\mathbf{x}) = \sqrt{\kappa(\mathbf{x}, \mathbf{x})} \cdot \omega^\top \mathbf{h}(\mathbf{x})$. We have a fully-optimized implementation in C++ for the above formulation based on the dual coordinate descent method [12], which is one of state-of-the-art solvers for linear SVM.

4. Algorithm Analysis

Our proposed Hash-SVM trades accuracy for scalability at several aspects. The approximation stems from three sources: random subspace projection as described in Section 3.2, finite number of hash bits due to budgeted memory, and hashing-based kernel approximation (*i.e.*, $\widehat{\kappa}(\mathbf{x}_1, \mathbf{x}_2) \approx \kappa(\mathbf{x}_1, \mathbf{x}_2)$ in Section 3.1). This section elaborates on the detailed analysis of the algorithm, particularly on the accuracy-scalability tradeoff and space/time complexities. Our analysis adopts the methodology of investigating one factor with others fixed. For example, the theoretic observation in Theorem 4.1 has assumed infinite hash bits are used to ensure the collision bound is rigourously reached.

Hash-SVM v.s. Original Kernel SVM: Let $\theta_{\mathbf{x}_1,\mathbf{x}_2}$ be the angle between two samples $\mathbf{x}_1, \mathbf{x}_2$ in RKHS. In the ideal case, if the normalized Hamming distance of a hashing scheme converges to $(1 - \cos(\theta_{\mathbf{x}_1,\mathbf{x}_2}))/2$ when the hash bit number k approaches infinity, it can be verified from Equations (1)(2) that $\widehat{\kappa}(\mathbf{x}_1, \mathbf{x}_2) = \kappa(\mathbf{x}_1, \mathbf{x}_2)$. In other words, the surrogate kernel function is an exact one. It is the very case for minHash which is designed for Jaccard index similarity [18]. However, for the general cases the existence of such hashing scheme remains an open issue. In our proposed Hash-SVM algorithm, $\widehat{\kappa}(\mathbf{x}_1, \mathbf{x}_2) \approx \kappa(\mathbf{x}_1, \mathbf{x}_2)$ with upper bound of the approximation error.

To compare the solutions obtained by Hash-SVM and the original kernel SVM, recall that the dual form of kernel SVM is as below:

$$\min_{\alpha \in \mathbb{R}^n} \quad \frac{1}{2}\alpha^\top \mathbf{K} \odot \left(\mathbf{y}\mathbf{y}^\top\right)\alpha - \mathbf{1}^\top \alpha, \quad (6)$$
$$s.t. \quad \alpha \geq \mathbf{0},\ \ \alpha^\top \mathbf{y} = 0,\ \ 0 \leq \alpha_i \leq C,\ \forall i.$$

where $\mathbf{K}(i,j) = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ and $\odot$ is the Hadamard product. The proposed Hash-SVM can be re-formulated into similar form, except that it utilizes a different kernel matrix $\widehat{\mathbf{K}}$ with $\widehat{\mathbf{K}}(i,j) = \widehat{\kappa}(\mathbf{x}_i, \mathbf{x}_j)$. Both $\mathbf{K}, \widehat{\mathbf{K}}$ are p.s.d. matrices (by definition of $\kappa(\cdot,\cdot)$ or Theorem 3.1 respectively). We have an observation on the approximation error:

Theorem 4.1. *Let $\alpha^*, \widehat{\alpha}^*$ be the optimal solutions by plugging $\mathbf{K}, \widehat{\mathbf{K}}$ into Problem (6), respectively. For test data, their labels can be determined by the induced prediction functions, e.g., $f(\mathbf{x}) = \langle \mathbf{\Phi}\alpha^*, \phi(\mathbf{x})\rangle$ for original kernel SVM and $\hat{f}(\mathbf{x}) = \langle \mathbf{\Phi}\widehat{\alpha}^*, \phi(\mathbf{x})\rangle$ for Hash-SVM, where $\mathbf{\Phi} = (\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n))$ are the training samples in RKHS. When $\kappa(\mathbf{x}, \mathbf{x})$ is uniform for any sample $\mathbf{x}$, the following approximation error bound holds for any $\mathbf{x}$ ($\mathbf{x}$ is not necessarily training sample) with high probability:*

$$\left|f(\mathbf{x}) - \hat{f}(\mathbf{x})\right| \leq C_0\left(\epsilon\|\mathbf{K}\|_2 + D\right)^{1/4}\left[1 + C_1\left(\epsilon\|\mathbf{K}\|_2 + D\right)^{1/4}\right], \quad (7)$$

where ϵ is a tiny positive scalar. C_0, C_1 are both constant; $\|\mathbf{K}\|_2$ is the spectral norm of $\mathbf{K}$, which returns the largest singular value of $\mathbf{K}$. $D = \sigma_{m+1} + n_0 s_0$, where σ_{m+1} is the $m+1$-th largest singular value of $\mathbf{K}$, $s_0 = 1 - \frac{1}{\pi}\arcsin\frac{2}{\pi} - \frac{\sqrt{\pi^2-4}}{\pi}$ and n_0 is the cardinality of $supp(\alpha^*) \cup supp(\widehat{\alpha}^*)$, where $supp(\alpha^*), supp(\widehat{\alpha}^*)$ denote the sets of nonzero elements in $\alpha^*, \widehat{\alpha}^*$ respectively.

The above theorem states that the prediction function learned by our proposed surrogate kernel function deviates the original one by a factor of $\mathcal{O}\left(\sqrt{\epsilon\|\mathbf{K}\|_2 + D}\right)$. The bound takes both the fixed-rank random subspace projection and the hashing binarization into account. The proof is deferred to the supplemental material due to space limit.

KRSH Collision Probability: The proposed KRSH is essentially a kernelized extension of the seminal work by Charikar [5]. In the ideal case that the m-dimensional subspace learned in Algorithm 1 sufficiently reconstructs the original data space, the collision probability (CP) of two arbitrary samples is [5],

$$(\textbf{Ideal CP}) : Pr(h(\mathbf{x}_1) = h(\mathbf{x}_2)) = 1 - \frac{\theta_{\mathbf{x}_1,\mathbf{x}_2}}{\pi}, \quad (8)$$

where $\theta_{\mathbf{x}_1,\mathbf{x}_2} = \frac{\phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_2)}{\|\phi(\mathbf{x}_1)\|_{\mathcal{H}} \|\phi(\mathbf{x}_2)\|_{\mathcal{H}}}$. Intuitively the non-collision probability is proportional to the inter-vector angle. In our proposed KRSH, the random subspace projection is a critical operation to enable kernel-based hashing, although it introduces some additional approximation error. Let $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$ be the projection bases of the random subspace learned in Algorithm 1, and $\theta'_{\mathbf{x}_1,\mathbf{x}_2} = \frac{(\mathbf{V}^\top \phi(\mathbf{x}_1))^\top (\mathbf{V}^\top \phi(\mathbf{x}_2))}{\|\mathbf{V}^\top \phi(\mathbf{x}_1)\|_{\mathcal{H}} \|\mathbf{V}^\top \phi(\mathbf{x}_2)\|_{\mathcal{H}}}$ be the angle between $\mathbf{x}_1, \mathbf{x}_2$ after projecting to the m-dimensional random subspace. The collision probability of KRSH is,

$$(\textbf{KRSH CP}) : Pr(h(\mathbf{x}_1) = h(\mathbf{x}_2)) = 1 - \frac{\theta'_{\mathbf{x}_1,\mathbf{x}_2}}{\pi}. \quad (9)$$

Though many kernels are of high (even infinite) dimensions in Hilbert space, the landmark count m is not necessarily huge to ensure $\theta'_{\mathbf{x}_1,\mathbf{x}_2} \approx \theta_{\mathbf{x}_1,\mathbf{x}_2}$ since the kernels often have long-tailed singular value distributions. To quantify the approximation error in KRSH, it is desired to bound the deviation of $\theta'_{\mathbf{x}_1,\mathbf{x}_2}$ away from $\theta_{\mathbf{x}_1,\mathbf{x}_2}$. A closed-form bound for $\theta'_{\mathbf{x}_1,\mathbf{x}_2}$ is only available for specific data distributions. Let us consider the simplest case that samples are uniformly distributed on the hypersphere in RKHS, which is approximately achievable by whitening operation. Based on the concentration arguments in proving Johnson-Lindenstrauss lemma [7], we show that for any ϵ in $(0, 1)$, with probability at least $1 - 3\exp(-\frac{m\epsilon^2}{4}) - 3\exp(-\frac{m(\epsilon^2/2 - \epsilon^3/3)}{2})$,

$$\frac{\cos\theta_{\mathbf{x}_1,\mathbf{x}_2} - \epsilon}{1+\epsilon} \leq \cos\theta'_{\mathbf{x}_1,\mathbf{x}_2} \leq \frac{\cos\theta_{\mathbf{x}_1,\mathbf{x}_2} + \epsilon}{1-\epsilon}. \quad (10)$$

The proof is found in the supplemental material. Above inequalities essentially address that $\cos\theta'_{\mathbf{x}_1,\mathbf{x}_2}$ approaches $\cos\theta_{\mathbf{x}_1,\mathbf{x}_2}$ with sufficient m and near-uniform data distribution. We also provide empirical investigation of inter-vector angle preservation on real-world data in Section 5.

How Many Hash Bits Are Needed?: The collision probability converges to the theoretic result when infinite hash bits are independently generated. And more hash bits tend to boost the level of solution accuracy. However, practitioners always adopt finite k hash bits due to budgeted memory consumption. It is important to investigate the "optimal" k that well balances the storage cost and accuracy loss. Given the independence of multiple hashing functions, the Hamming distance for k-bit code follows a binomial distribution:

$$Pr(D_{ham}(H_k(\mathbf{x}_1), H_k(\mathbf{x}_2)) = t) = \binom{k}{t} \left(1 - \frac{\theta_{\mathbf{x}_1,\mathbf{x}_2}}{\pi}\right)^{k-t} \left(\frac{\theta_{\mathbf{x}_1,\mathbf{x}_2}}{\pi}\right)^{t}. \quad (11)$$

Deriving from binomial distribution, the expectation and standard deviation of $1 - 2D_{ham}(H_k(\mathbf{x}_1), H_k(\mathbf{x}_2))/k$ in Equation (2) are known to be $(\mu,\ \sigma) \sim (1 - 2\theta_{\mathbf{x}_1,\mathbf{x}_2}/\pi,\ \sqrt{(1 - \theta_{\mathbf{x}_1,\mathbf{x}_2}/\pi) \cdot \theta_{\mathbf{x}_1,\mathbf{x}_2}/\pi}/k\ )$. It indicates that our proposed surrogate kernel function in Equation (2) converges to $1 - 2\theta_{\mathbf{x}_1,\mathbf{x}_2}/\pi$, which is an approximation to $\cos(\theta_{\mathbf{x}_1,\mathbf{x}_2})$ in Equation (1). Moreover, the standard deviation drastically drops at linear rate with respect to k.

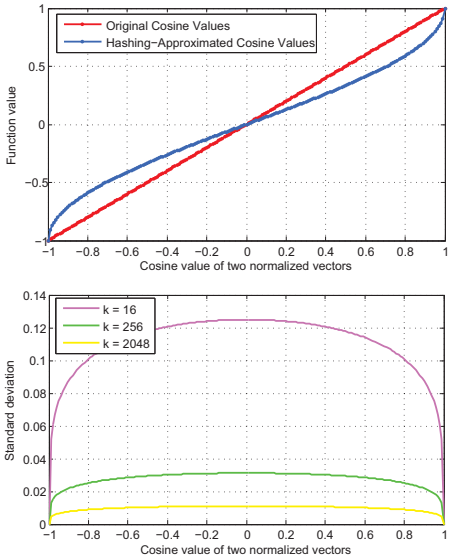

Figure 1. Top: approximating $\cos\theta_{x_1,\mathbf{x}_2}$ in original kernel function with $1 - 2\theta_{\mathbf{x}_1,\mathbf{x}_2}/\pi$ in our proposed surrogate function. Bottom: standard deviation of hashing-based approximation to $\cos\theta_{x_1,\mathbf{x}_2}$ with varying hash bits. k denotes the hash bit number.

Figure 1 graphically illustrates the original and hashing-approximated inter-vector cosine values whose implications can be found at Equations (1)(2), respectively, as well as the effect of different hash bits on the estimation variations. It is also worth noting that the standard deviation is irrelevant to the random subspace parameter m, which is a desired property since m is often large in order to capture sufficient data variations. Empirically even only a small number of bytes (*e.g.*, 1024 bytes for $k = 8096$) are able to achieve reasonable level of accuracy, which is much more compact than other competing algorithms.

Time/Storage Complexity: In the proposed KRSH, learning the m-dimensional random subspace is accomplished by a Gram-Schmidt process, which requires a computational complexity of $\mathcal{O}(m^3)$ and storage $\mathcal{O}(m^2)$ for bases matrix $\mathbf{V}$. Each sample is represented by k hash bits, or equivalently $k/8$ bytes. Given a new sample, the production of k-bit hash code requires executing m kernel functions between the new sample and landmark points, and $\mathcal{O}(mk)$ algebraic operations. For the Hash-SVM implementation, we adopt the framework of dual-coordinate descent method [12], which enjoys a linear storage complexity with respect to the sample number and at least linear convergence rate towards the global optimum.

Dataset	Train/Test Size	#Dim	#Class	Kernel
MED11	30,000/16,904	5,000	25	χ_2
IJCNN	49,990/91,701	22	2	RBF
WebSpam	280,000/70,000	254	2	RBF
CIFAR10	50,000/10,000	800	10	RBF
ImageNet	1,261,406/50,000	1,000	1,000	RBF

Table 1. Summary of the benchmarks used in the experiments.

5. Experiments

This section reports the comparison between our proposed Hash-SVM and other competing algorithms.

Dataset Description: We adopt five benchmarks which cover a variety of tasks in multiple data scales: *TRECVID MED11*, which is a video corpus collected by NIST to foster the research on detecting semantic events from videos. Each video either contains one of 15 pre-defined events or is "null" video, *ImageNet* which consists of more than 1 million photographs with the presence or absence of 1000 object categories (10 categories are randomly chosen for our evaluation), and *CIFAR10* which is comprised of images from ten semantic categories. To make the experiments more comprehensive, we also include two widely-used non-vision machine learning benchmarks: time-series data *IJCNN* and annotated spam/nonspam data *WebSpam*.

Table 1 summarizes the important information of the experimental data. Regarding the features, both *TRECVID MED11* and *ImageNet* adopt SIFT bag-of-words representation with or without spatial pyramid. *CIFAR10* uses a variant of convolutional neuron features. For non-vision benchmarks *IJCNN* and *WebSpam*, we extract standard time-series feature and uni-gram feature, respectively. We would like to highlight that most features may not bring state-of-the-art recognition accuracy since our focus is optimizing the way to learn the classification models instead of just driving for higher recognition accuracy. On most benchmarks we adopt RBF kernel, except for MED11 which adopts histogram χ_2 kernel owing to its empirical superiority. The kernel width parameter σ in RBF kernel is empirically estimated from training data. For fair comparison, on each benchmark σ is identical for all non-linear learning algorithms. Five independent trials are performed for all algorithms with randomized operation.

Baseline Algorithms: We report the mean accuracies averaged over all classes for the proposed Hash-SVM and competitors, including 1) *LibSVM and LibLinear*: the most popular SVM solvers based on decomposition method [2] and dual coordinate descent [12], respectively. 2) *Core vector machine (CVM)* [26]: which accelerates SVM training using core-set approximation on very large scale data sets. Our experiments adopt the implementation from the authors. 3) *Adaptive Multi-hyperplane Machine (AMM)* [28]: which utilizes multiple linear classifiers to approximate decision boundary for non-linear data. 4) *Low-rank linearization SVM (LLSVM)* [31]: it utilizes similar idea of Nyström method to generate data-dependant kernel linearization. The resultant linear features are fed into the Selective block minimization (SBM) based linear SVM solver as described in [4]. 5) *Random features followed by linear SVM* (denoted as RF-SVM): for the shift-invariant RBF kernel, random Fourier features are generated according to [22]. For histogram χ_2 kernel, there are two popular approximation algorithms, *i.e.*, [27] and [17]. We use the Chebyshev approximation based random feature [17] since it is a more recent work and already contains an extensive comparison with [27]. 6) *KLSH-SVM*: we also plug the hash bits obtained by KLSH into our proposed framework.

Accuracy and Speed: For all baselines, we use the code provided by the authors and properly tune the parameters. In training visual classification model on large scale data, the key bottleneck often lies in the tremendous memory requirement rather than CPU. For example, it is known that the performance of LL-SVM will be significantly improved if more landmarks are used. However, even an moderate landmark set of 4000 points implies more than 32G bytes for loading the transformed features on ImageNet. In light of this, we set the feature storage to be the key factor for fairly comparing those randomized algorithms (RF-SVM, LLSVM and Hash-SVM). Specifically, all aforementioned algorithms are allocated 2,048 bytes for storing each sample's feature, which implies a 512-dimensional random feature representation for RF-SVM, LLSVM[1] and 16K hash bits for Hash-SVM. In addition, note that the performances of LL-SVM and Hash-SVM are heavily affected by the number of landmarks sampled from the training set, or the random subspace parameter m in the proposed Hash-SVM. For fairness we sampled 4,096 landmarks for all algorithms.

Table 2 presents the computing time (including data I/O, training and testing time) and accuracies. For multi-class data sets, we train a one-vs-rest classifier for each class and calculate the average accuracy over all classes. Table 2 shows that kernel SVM consistently outperforms linear SVM in terms of classification accuracy, yet suffering from the curse of high feature dimension and large data scale. The proposed Hash-SVM proves to be a good remedy to the slow optimization of kernel SVM. Regarding the prediction accuracy, the proposed Hash-SVM consistently outperforms other competing randomized algorithms (RF-SVM, LLSVM) on all benchmarks. It also delivers comparable accuracy to the best one achieved by LibSVM (except for MED11, which has huge intrinsic dimension in RKHS and can hardly be embedded into any 4096-

[1] It is assumed that these features are stored in single-precision floating format (4 bytes per dimension). For LL-SVM, we sample 4,096 landmarks in accord with the setting of Hash-SVM, but only keep the first 512 principal components of the kernel matrix to fulfill the 2K-bytes budget.

	MED11		CIFAR10		IJCNN		WebSpam		ImageNet	
	time	acc.(%)	time	acc.(%)	time	acc.(%)	time	acc.(%)		
Linear SVM	396	17.37	906	60.28	0.78	91.79	9.77	92.63	1044	12.68
LibSVM	47160	25.59	32849	69.24	66.5	98.16	4860	98.54	–	–
CVM	–	–	9861	63.42	159	98.22	6660	98.83	–	–
AMM	–	–	–	–	4.95	92.18	154	91.96	–	–
RF-SVM	383	5.08	924.7	38.04	4	93.28	67	93.84	131	4.84
LL-SVM	7740	20.90	292	60.35	68	98.20	505	96.82	775	13.12
KLSH-SVM	7745	20.94	1424	62.74	1260	98.48	2304	98.36	14221	14.29
Hash-SVM	7308	21.74	1137	63.13	85	98.51	405	98.50	14976	14.70

Table 2. Experimental results in terms of computing time and test accuracies. The marks '–' indicate that the program crashes, or fails to converge within reasonable time, or is unable to report accuracy in specific format (*e.g.*, AMM on MED11 and CIFAR10). The unit of the reported time is in seconds. Experiments for CVM and AMM are conducted on a Windows machine (quad-core Intel CPU and 16GB RAM). The rest are on a Ubuntu-OS machine (12-core Intel CPU and 64 GB RAM).

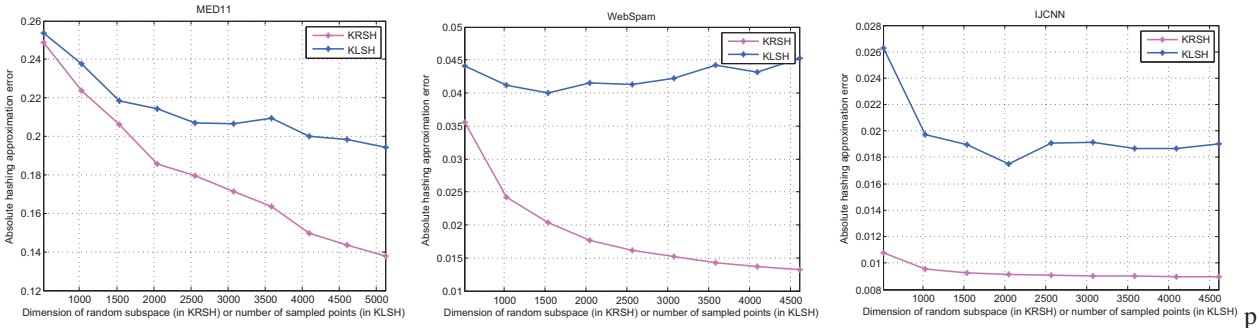

Figure 2. Investigation on the angle-preserving property of KRSH and KLSH. We record $|\cos\theta_{\mathbf{x}_1,\mathbf{x}_2} - \cos\theta'_{\mathbf{x}_1,\mathbf{x}_2}|$. The numbers of random subspace in KRSH or the sampler count in KLSH vary over a large range. All estimations are obtained using 16K hash bits.

dimensional random subspace with high fidelity) but enjoys much less training time and lower complexity of induction to new samples. Considering that only a small portion of 4096 landmarks are used in building the final classifiers, the performance of Hash-SVM is quite striking.

Angle Preservation: One critical index of the effectiveness of the proposed hashing scheme is angle preservation, *i.e.*, $\theta_{\mathbf{x}_1,\mathbf{x}_2} \approx \theta'_{\mathbf{x}_1,\mathbf{x}_2}$ such that the surrogate kernel function induces less approximation errors. For clarity, let us first review how $\theta_{\mathbf{x}_1,\mathbf{x}_2}$, $\theta'_{\mathbf{x}_1,\mathbf{x}_2}$ are calculated. By definition, $\theta_{\mathbf{x}_1,\mathbf{x}_2}$ is computed as $\arccos\kappa(\mathbf{x}_1,\mathbf{x}_2)/\sqrt{\kappa(\mathbf{x}_1,\mathbf{x}_1)}\sqrt{\kappa(\mathbf{x}_2,\mathbf{x}_2)}$. According to KRSH collision probability in Equation (9),

$$\theta'_{\mathbf{x}_1,\mathbf{x}_2} = \lim_{k\to\infty} \frac{1}{k} D_{ham}(H_k(\mathbf{x}_1), H_k(\mathbf{x}_2)). \tag{12}$$

In other words, $\theta'_{\mathbf{x}_1,\mathbf{x}_2}$ shall be estimated from sufficient hash bits.

In Figure 2, we demonstrate how the inter-vector angles in kernel-induced Hilbert space are preserved. To quantitatively evaluate the discrepancy between them, the residual $|\cos\theta_{\mathbf{x}_1,\mathbf{x}_2} - \cos\theta'_{\mathbf{x}_1,\mathbf{x}_2}|$ is estimated from 4 million random sample pairs. For contrastive study, we also show the performance of the prominent kernel hashing algorithm KLSH [15] since it also effectively preserves pairwise angles in kernel space. Other kernel hashing algorithms are not included, since most of them rarely provide any theoretic analysis on angle preservation or collision probability. We use the KLSH code provided by the authors and perform a fine tuning on its key parameters such as number of Gaussian approximation elements. The results on three out of the five benchmarks are shown due to space limit. We observe consistent superiority of KRSH under identical key parameters (the dimension m of random subspaces in KRSH, and the number of active samples used for covariance estimation in KLSH), which partially supports our concerns about KLSH's unreliable covariance parameter estimation and the application of CLT in high-dimensional Hilbert space.

Parameter Sensitivity: Finally an investigation of parameter sensitivity is conducted. In Figure 3 the evaluation on CIFAR10 is shown. For the evaluation the angle residue $|\cos\theta_{\mathbf{x}_1,\mathbf{x}_2} - \cos\theta'_{\mathbf{x}_1,\mathbf{x}_2}|$ is used as the criterion since it is tightly related to the final performance. The values are plotted as a surface parameterized by random subspace dimension and hash bit number in KRSH. It is observed that the approximation errors of RKHS continue to drop when the dimension of random subspace is increased. It is also worth to note that enlarging the parameter m produces more and more marginal gains, which indicates that a moderate value of m is sufficient for reasonable performance. We can also

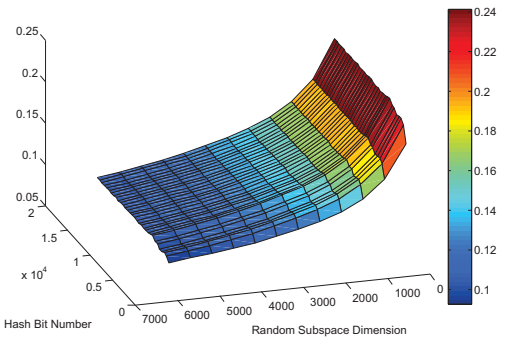

Figure 3. Parameter sensitivity in terms of angle preservation. The vertical axis denotes the angle residual $|\cos\theta_{\mathbf{x}_1,\mathbf{x}_2} - \cos\theta'_{\mathbf{x}_1,\mathbf{x}_2}|$.

observe that a parsimony choice of the hash bit number still brings reasonable accuracy level, which justifies the analysis in Section 4.

6. Limitations and Future Work

In this paper we present a novel large-scale kernel SVM solver based on the locality-sensitive hashing technique. It conveys some fresh ideas on general kernel approximation and SVM optimization. We provide extensive theoretic analysis as well as comprehensive empirical study for the proposed Hash-SVM. The major limitations of Hash-SVM lie in 1) our current implementation of SVM optimization is based on dual coordinate descent, which does not effectively utilize the special property of our hashing features (*e.g.*, the feature is binary) for further acceleration; and 2) long hash bit vectors are required to ensure reasonable performance, which unfortunately may increase the risk of overfitting. We will address these limitations in the future work.

Acknowledgement: The work is partly supported by a grant from China 973 Fundamental R&D Program (No.2014CB340304). Gang Hua is partly supported by US National Science Foundation Grant IIS 1350763, China National Natural Science Foundation Grant 61228303, a collaborative gift grant from Adobe, GHs start-up funds from Stevens Institute of Technology, a Google Research Faculty Award, a gift grant from Microsoft Research, and a gift grant from NEC Labs America.

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008. 1, 2

[2] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:1–27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm. 2, 6

[3] E. Y. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui. Parallelizing support vector machines on distributed computers. In *NIPS*, 2007. 1, 2

[4] K.-W. Chang and D. Roth. Selective block minimization for faster convergence of limited memory large-scale linear models. In *SIGKDD*, 2011. 6

[5] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, 2002. 2, 3, 4

[6] C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, 1995. 1

[7] S. Dasgupta and A. Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, 2003. 5

[8] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *STOC*, 2004. 2, 3

[9] P. Drineas and M. W. Mahoney. Approximating a gram matrix for improved kernel-based learning. In *COLT*, 2005. 2

[10] E. Hazan, T. Koren, and N. Srebro. Beating sgd: Learning svms in sublinear time. In *NIPS*, 2011. 2

[11] J. He, W. Liu, and S.-F. Chang. Scalable similarity search with optimized kernel hashing. In *SIGKDD*, 2010. 2, 3

[12] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *ICML*, 2008. 2, 4, 5, 6

[13] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998. 2

[14] T. Joachims. Training linear svms in linear time. In *SIGKDD*, 2006. 2

[15] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 34(6):1092–1104, 2012. 2, 3, 7

[16] S. Kumar, M. Mohri, and A. Talwalkar. Sampling methods for the nyström method. *J. Mach. Learn. Res.*, 98888:981–1006, 2012. 2

[17] F. Li, G. Lebanon, and C. Sminchisescu. Chebyshev approximations to the histogram chi-square kernel. In *CVPR*, 2012. 2, 6

[18] P. Li, A. Shrivastava, J. L. Moore, and A. C. König. Hashing algorithms for large-scale learning. In *NIPS*, 2011. 2, 4

[19] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *CVPR*, 2010. 2, 3

[20] Y. Mu, J. Wright, and S.-F. Chang. Accelerated large scale optimization by concomitant hashing. In *ECCV*, 2012. 2

[21] J. C. Platt. Using analytic qp and sparseness to speed training of support vector machines. In *NIPS*, 1999. 2

[22] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*, 2007. 2, 6

[23] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001. 1, 3

[24] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML*, 2007. 2

[25] A. Smola, S. Vishwanathan, and Q. Lenicta. Bundle methods for machine learning. In *NIPS*, 2008. 2

[26] I. Tsang, J. Kwok, and P.-M. Cheung. Core vector machines: Fast svm training on very large data sets. *J. Mach. Learn. Res.*, 6:363–392, 2005. 2, 6

[27] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):480–492, 2012. 2, 6

[28] Z. Wang, N. Djuric, K. Crammer, and S. Vucetic. Trading representability for scalability: adaptive multi-hyperplane machine for nonlinear classification. In *SIGKDD*, 2011. 6

[29] C. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *NIPS*, 2001. 2

[30] L. Zanni, T. Serafini, and G. Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7:1467–1492, 2006. 2

[31] K. Zhang, L. Lan, Z. Wang, and F. Moerchen. Scaling up kernel svm on limited resources: A low-rank linearization approach. *Journal of Machine Learning Research - Proceedings Track*, 22:1425–1434, 2012. 2, 6

[32] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML*, 2004. 2