SIAM J. MATRIX ANAL. APPL.
Vol. 38, No. 4, pp. 1116–1138

FASTER KERNEL RIDGE REGRESSION USING SKETCHING AND PRECONDITIONING*

HAIM AVRON†, KENNETH L. CLARKSON‡, AND DAVID P. WOODRUFF§

Abstract. Kernel ridge regression is a simple yet powerful technique for nonparametric regression whose computation amounts to solving a linear system. This system is usually dense and highly ill-conditioned. In addition, the dimensions of the matrix are the same as the number of data points, so direct methods are unrealistic for large-scale datasets. In this paper, we propose a preconditioning technique for accelerating the solution of the aforementioned linear system. The preconditioner is based on random feature maps, such as random Fourier features, which have recently emerged as a powerful technique for speeding up and scaling the training of kernel-based methods, such as kernel ridge regression, by resorting to approximations. However, random feature maps only provide crude approximations to the kernel function, so delivering state-of-the-art results by directly solving the approximated system requires the number of random features to be very large. We show that random feature maps can be much more effective in forming preconditioners, since under certain conditions a not-too-large number of random features is sufficient to yield an effective preconditioner. We empirically evaluate our method and show it is highly effective for datasets of up to one million training examples.

Key words. kernel ridge regression, preconditioning, random features

AMS subject classifications. 15-04, 68T05, 68W20, 68Q87, 65F08, 62G08

DOI. 10.1137/16M1105396

1. Introduction.

Kernel ridge regression (KRR) is a simple yet powerful technique for nonparametric regression whose computation amounts to solving a linear system. Its underlying mathematical framework is as follows. A kernel function, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, is defined on the input domain $\mathcal{X} \subseteq \mathbb{R}^d$. The kernel function k may be (nonuniquely) associated with an embedding of the input space into a high-dimensional reproducing kernel Hilbert space $\mathcal{H}_k$ (with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}_k}$) via a feature map, $\Psi : \mathcal{X} \to \mathcal{H}_k$ such that $k(\mathbf{x}, \mathbf{z}) = \langle \Psi(\mathbf{x}), \Psi(\mathbf{z}) \rangle_{\mathcal{H}_k}$. Given training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y}$ and ridge parameter λ, we perform linear ridge regression on $(\Psi(\mathbf{x}_1), y_1), \ldots, (\Psi(\mathbf{x}_n), y_n)$. Ultimately, the model has the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} c_i k(\mathbf{x}_i, \mathbf{x}) \,,$$

where $c_1, \ldots, c_n$ can be found by solving the linear equation

$$(\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{c} = \mathbf{y} \,. \tag{1}$$

Here $\mathbf{K} \in \mathbb{R}^{n \times n}$ is the *kernel matrix* or *Gram matrix* defined by $\mathbf{K}_{ij} \equiv k(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{c} = [c_1 \cdots c_n]^{\mathrm{T}}$, and $\mathbf{y} \equiv [y_1 \cdots y_n]^{\mathrm{T}}$. See Saunders, Gammerman, and Vovk [30] for details.

*Received by the editors November 28, 2016; accepted for publication (in revised form) by D. L. Boley July 18, 2017; published electronically October 18, 2017.
http://www.siam.org/journals/simax/38-4/M110539.html

Funding: The work of the authors was supported by the XDATA program of the Defense Advanced Research Projects Agency (DARPA), administered through Air Force Research Laboratory contract FA8750-12-C-0323.

†Department of Applied Mathematics, Tel Aviv University, Tel Aviv 6997801, Israel (haimav@post.tau.ac.il).

‡IBM Almaden Research Center, San Jose, CA 95120 (klclarks@us.ibm.com, dpwoodru@us.ibm.com).

§School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 (dwoodruf@cs.cmu.edu).

Compared to kernel support vector machines (KSVMs), the computations involved in KRR are conceptually much simpler: solving a single linear system as opposed to solving a convex quadratic optimization problem. However, KRR has been observed experimentally to often perform just as well as KSVM [18]. In this paper, we exploit the conceptual simplicity of KRR and using advanced techniques in numerical linear algebra design an efficient method for solving (1).

For widely used kernel functions, the kernel matrix $\mathbf{K}$ is fully dense, so solving (1) using standard direct methods takes $\Theta(n^3)$, which is prohibitive even for modest n. Although statistical analysis does suggest that iterative methods can be stopped early [8], the condition number of $\mathbf{K}$ tends to be so large that a large number of iterations are still necessary. Thus, it is not surprising that the literature has moved toward designing approximate methods.

One popular strategy is the Nyström method [34] and variants that improve the sampling process [22, 19]. We also note recent work by Yang, Pilanci, and Wainwright [37] that replace the sampling with sketching. More relevant to this paper is the line of research on randomized construction of approximate feature maps, originating from the seminal work of Rahimi and Recht [28]. The underlying idea is to construct a distribution D on functions from $\mathcal{X}$ to $\mathbb{R}^s$ (s is a parameter) such that

$$k(\mathbf{x}, \mathbf{z}) = \mathbb{E}_{\varphi \sim D}\left[\varphi(\mathbf{x})^{\mathrm{T}} \varphi(\mathbf{z})\right] .$$

One then samples a φ from D and uses $\tilde{k}(\mathbf{x}, \mathbf{z}) \equiv \varphi(\mathbf{x})^{\mathrm{T}} \varphi(\mathbf{y})$ as an alternative kernel. Training can be done in $O(ns^2 + T_\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_n))$, where $T_\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is the time required to compute $\varphi(\mathbf{x}_1), \ldots, \varphi(\mathbf{x}_n)$. This technique has been used in recent years to obtain state-of-the-art accuracies for some important datasets [21, 16, 6, 13]. We also note recent work on using random features to construct stochastic gradients [23].

One striking feature of the aforementioned papers is the use of a very large number of random features. Random feature maps typically provide only crude approximations to the kernel function, so to approach the full capacity of exact kernel learning (which is required to obtain state-of-the-art results) many random features are necessary. Nevertheless, even with a very large number of random features, we sometimes pay a price in terms of generalization performance. Indeed, in section 6.1 we show that in some cases, driving s to be as large as n is not sufficient to achieve the same test error rate as that of the full kernel method. Ultimately, methods that use approximations compromise in terms of performance in order to make the computation time manageable.

1.1. Contributions. We propose to use random feature maps as a means of forming a preconditioner for the kernel matrix. This preconditioner can be used to solve (1) to high accuracy using an iterative method. Thus, while training time still benefits from the use of high-quality random feature maps, there is no compromise in terms of modeling capabilities, modulo the decision to use kernel ridge regression and not some other learning method.

We provide a theoretical analysis that shows that at least for one kernel selection, the polynomial kernel, selecting the number of random features to be proportional to the so-called statistical dimension of the problem (this established quantity is also frequently referred to as the *effective degrees-of-freedom*) yields a high-quality preconditioner in the sense that the relevant condition number is bounded by a constant. Thus, the theoretical analysis can be viewed as a generalization of recent results on sharper bounds for linear regression and low-rank approximation with regularization [3]. In addition, we discuss a method for testing whether the preconditioner computed by

our algorithm is indeed such that the relevant condition number is bounded. While our analytical results are mostly of theoretical value (e.g., they are limited to only the polynomial kernel and are likely very pessimistic), they do expose an important connection between the statistical dimension and preconditioner size.

Finally, we report experimental results with a distributed-memory parallel implementation of our algorithm. We demonstrate the effectiveness of our algorithm for training high-quality models using both the Gaussian and polynomial kernel on datasets as large as one million training examples without compromising in terms of statistical capacity. For example, on one dataset with one million examples our code is able to solve (1) to relatively high accuracy in about an hour on resources readily available to researchers and practitioners (a cluster of EC2 instances).

An open-source implementation of the algorithm is available through the libSkylark library (http://xdata-skylark.github.io/libskylark/).

1.2. Related work. Devising scalable methods for kernel methods has long been an active research topic. In terms of approximations, one dominant line of work constructs low-rank approximations of the Gram matrix. Two popular variants of this approach, as mentioned earlier, are the randomized feature maps and the Nyström methods [34]. There are many variants of these approximation schemes, and it is outside the scope of this paper to mention all of them. Recent work has also focused on devising scalable methods that are capable of utilizing rather high rank approximations [16, 6, 23]. In contrast, our goal is to develop a method which is capable of using lower rank approximation without paying a price in terms of model quality.

Another approach is to approximate the kernel matrix so that it will be more amenable to matrix-vector products or linear system solution. One idea is to use the fast Gauss transform to accelerate the matrix-vector products of the kernel matrix by an arbitrary vector [29, 25]. Another approach is to use a tree code to efficiently perform matrix-vector products [25, 12]. Related is also the hierarchical matrix approach in which an hierarchical matrix approximation to the kernel matrix is built [11]. This representation is amenable to efficient implementation of wide range of operations on the approximate kernel matrix, including the matrix-vector product and linear system solution.

The preconditioning approach has also been explored in the literature. Srinivasan et al. propose to use a regularized kernel matrix as a preconditioner in a flexible Krylov method [32]. The regularized kernel matrix, which has a lower condition number, is solved using an inner conjugate gradient iteration. In parallel work to ours, Cutajar et al. recently discussed various preconditioning techniques for kernel matrices [15]. One of the methods they propose is using random features to form a preconditioner. However, unlike our work, they do not include any theoretical analysis of this preconditioning approach. Furthermore, we propose additional algorithmic enhancements (multiple level preconditioning, testing preconditioners). Finally, it is worth mentioning that Cutajar et al. only experiment with small-scale low-dimensional datasets, while we present experimental results with large-scale high-dimensional datasets.

For a broad discussion of scalable methods for kernel learning, including of ideas not mentioned here, see Bottou et al. [9].

2. Preliminaries.

2.1. Basic definitions and notation. We denote scalars using Greek letters or using $x, y, \ldots$. Vectors are denoted by $\mathbf{x}, \mathbf{y}, \ldots$ and matrices by $\mathbf{A}, \mathbf{B}, \ldots$. The $s \times s$

identity matrix is denoted $\mathbf{I}_s$. We use the convention that vectors are column-vectors. We use $\mathbf{nnz}(\cdot)$ to denote the number of nonzeros in a vector or matrix. We denote by $[n]$ the set $1, \ldots, n$. The notation $\alpha = (1 \pm \gamma)\beta$ means that $(1-\gamma)\beta \leq \alpha \leq (1+\gamma)\beta$.

A symmetric matrix $\mathbf{A}$ is positive semidefinite (PSD) if $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} \geq 0$ for every vector $\mathbf{x}$. It is positive definite (PD) if $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} > 0$ for every vector $\mathbf{x} \neq 0$. For any two symmetric matrices $\mathbf{A}$ and $\mathbf{B}$ of the same size, $\mathbf{A} \preceq \mathbf{B}$ means that $\mathbf{B} - \mathbf{A}$ is a PSD matrix. For a PD matrix $\mathbf{A}$, $\|\cdot\|_{\mathbf{A}}$ denotes the norm induced by $\mathbf{A}$, i.e., $\|\mathbf{x}\|_{\mathbf{A}}^2 \equiv \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$.

We denote the training set by $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^d \times \mathbb{R}$. Note that n denotes the number of training examples, and d denotes their dimension. We denote the kernel, which is a function from $\mathcal{X} \times \mathcal{X}$ to $\mathbb{R}$, by k. We denote the kernel matrix by $\mathbf{K}$, i.e., $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The associated reproducing kernel Hilbert space (RKHS) is denoted by $\mathcal{H}_k$ and the associated inner product by $(\cdot,\cdot)_{\mathcal{H}_k}$. We use λ to denote the ridge regularization parameter, which we always assume to be greater than 0.

2.2. Random feature maps. As explained in the introduction, a *random feature map* is a distribution D on functions from $\mathcal{X}$ to $\mathbb{R}^s$ such that

$$k(\mathbf{x}, \mathbf{z}) = \mathbb{E}_{\varphi \sim D}\left[\varphi(\mathbf{x})^{\mathrm{T}}\varphi(\mathbf{z})\right].$$

Throughout the paper, we use s to denote the number of random features. This quantity is a parameter of the various algorithms.

In recent years, diverse uses of random features have been seen for a wide spectrum of techniques and problems. However, the original motivation is the following technique, originally due to Rahimi and Recht [28], which we refer to as the *random features method*: sample a φ from D and use $\tilde{k}(\mathbf{x}, \mathbf{z}) \equiv \varphi(\mathbf{x})^{\mathrm{T}}\varphi(\mathbf{z})$ as an alternative kernel. For KRR the resulting model is

$$f(\mathbf{x}) = \varphi(\mathbf{x})^{\mathrm{T}}(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \lambda\mathbf{I}_s)^{-1}\mathbf{Z}^{\mathrm{T}}\mathbf{y},$$

where $\mathbf{Z} \in \mathbb{R}^{n \times s}$ has ith row $\mathbf{z}_i = \varphi(\mathbf{x}_i)$. Thus, training can be done in $O(ns^2 + T_\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_n))$, where $T_\varphi(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is the time required to compute $\mathbf{z}_1, \ldots, \mathbf{z}_n$.

Although our proposed method can be composed with any of the many feature maps suggested in recent literature, we discuss and experiment with two specific feature maps. The first is *random Fourier features* originally suggested by Rahimi and Recht [28]. The underlying observation that lead to this transform is that a shift-invariant kernel[1] k for which $k(\mathbf{x}, \mathbf{x}) = 1$ for all with $\mathbf{x} \in \mathcal{X}$, then $k(\mathbf{x}, \mathbf{z})$ can be expressed as

$$k(\mathbf{x}, \mathbf{z}) = \mathbb{E}_{\mathbf{w} \sim p, b \sim U(0, 2\pi)}\left[\cos(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)\cos(\mathbf{w}^{\mathrm{T}}\mathbf{z} + b)\right]$$

where p is some appropriate distribution that depends on the kernel function (e.g., Gaussian distribution for the Gaussian kernel). The existence of such a p for every shift-invariant kernel function k is a consequence of Bochner's theorem; see Rahimi and Recht [28] for details. The feature map is then a Monte-Carlo sample: $\varphi(\mathbf{x}) = s^{-1/2}[\cos(\mathbf{w}_1^{\mathrm{T}}\mathbf{x} + b_1) \ldots \cos(\mathbf{w}_s^{\mathrm{T}}\mathbf{x} + b_n)]^{\mathrm{T}}$, where $\mathbf{w}_1, \ldots, \mathbf{w}_s$ are sampled from p and $b_1, \ldots, b_n$ are sampled from a uniform distribution on $[0, 2\pi]$.

The second feature map we discuss in this paper is TensorSketch [26], which is designed to be used for the polynomial kernel $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^{\mathrm{T}}\mathbf{z})^q$. In order to describe

[1] That is, it is possible to write $k(\mathbf{x}, \mathbf{z}) = k_0(\mathbf{x} - \mathbf{z})$ for some positive definite function $k_0 : \mathbb{R}^d \to \mathbb{R}$.

TensorSketch we first describe CountSketch [10]. Suppose we want to sketch a d-dimensional vector to an s-dimensional vector. CountSketch is specified by a 2-wise independent hash function $h : [d] \to [s]$ and a 2-wise independent sign function $g : [d] \to \{+1, -1\}$. Suppose CountSketch is applied to a vector $\mathbf{x} \in \mathbb{R}^d$ to yield $\mathbf{z} \in \mathbb{R}^s$. The value of coordinate i of $\mathbf{z}$ is $\sum_{j|h(j)=i} g(j)x_j$. It is clear that this transformation can be represented as a $s \times d$ matrix in which the jth column contains a single nonzero entry $g(j)$ in the $h(j)$th row. Therefore, the distribution on h and g defines a distribution on $s \times d$ matrices.

TensorSketch implicitly defines a random linear transformation $\mathbf{S} \in \mathbb{R}^{s \times d^q}$. The transform is specified using q 3-wise independent hash functions $h_1, \ldots, h_q : [d] \to [s]$ and q 4-wise independent sign functions $g_1, \ldots, g_q : [d] \to \{+1, -1\}$. The TensorSketch matrix $\mathbf{S}$ is then a CountSketch matrix with hash function $H : [d]^q \to [s]$ and sign function $G : [d]^q \to \{+1, -1\}$ defined as follows:

$$H(i_1, \ldots, i_q) \equiv h_1(i_1) + h_2(i_2) + \cdots + h_q(i_q) \bmod m$$

and

$$G(i_1, \ldots, i_q) \equiv g_1(i_1) \cdot g_2(i_1) \cdots g_q(i_q)\,.$$

We now index the columns of $\mathbf{S}$ by $[d]^q$ and set column $(i_1, \ldots, i_q)$ to be equal to $G(i_1, \ldots, i_q) \cdot \mathbf{e}_{H(i_1,\ldots,i_q)}$, where $\mathbf{e}_j$ denotes the jth identity vector. Let $v_q : \mathbb{R}^d \to \mathbb{R}^{d^q}$ map each vector to the evaluation of all possible degree q monomials of the entries. Thus, $k(\mathbf{x}, \mathbf{z}) = v_q(\mathbf{x})^{\mathrm{T}} v_q(\mathbf{z}) \approx v_q(\mathbf{x})^{\mathrm{T}} \mathbf{S}^{\mathrm{T}} \mathbf{S} v_q(\mathbf{z})$. The feature map is then defined by $\varphi(\mathbf{x}) = \mathbf{S} v_q(\mathbf{x})$.

A crucial observation that makes this transformation useful is that via a clever application of the fast Fourier transform, $\varphi(\mathbf{x})$ can be computed in $O(q(\mathbf{nnz}(\mathbf{x}) + s \log s))$ (see Pagh [26] for details), which allows for a fast application of the transform.

2.3. Fast numerical linear algebra using sketching. Sketching has recently emerged as a powerful dimensionality reduction technique for accelerating numerical linear algebra primitives typically encountered in statistical learning such as linear regression, low rank approximation, and principal component analysis. The following description is only a brief semi-formal description of this emerging area. We refer the interested reader to a recent surveys [35, 36] for more information.

The underlying idea is to construct an embedding of a high-dimensional space into a lower-dimensional one and use this to accelerate the computation. For example, consider the classical linear regression problem

$$\mathbf{w}^\star = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2,$$

where $\mathbf{X} \in \mathbb{R}^{n \times d}$ is a sample-by-feature design matrix and $\mathbf{y} \in \mathbb{R}^n$ is the target vector. Sketching methods for linear regression define a distribution on s-by-n matrices, sample a matrix $\mathbf{S}$ from this distribution, and solve the approximate problem

$$\mathbf{w} = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{S}\mathbf{X}\mathbf{w} - \mathbf{S}\mathbf{y}\|_2\,.$$

If the distribution is an *oblivious subspace embedding (OSE)* for $\mathbf{Range}([\mathbf{X}\,\mathbf{y}])$, i.e., with high probability for all $\mathbf{x} \in \mathbf{Range}([\mathbf{X}\,\mathbf{y}])$ we have $\|\mathbf{S}\mathbf{x}\|_2 = (1 \pm \epsilon)\|\mathbf{x}\|_2$, then one can show that with high probability $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2 \le (1+\epsilon)\|\mathbf{X}\mathbf{w}^\star - \mathbf{y}\|_2$. See Drineas et al. [17].

The "sketch-and-solve" approach just described allows only crude approximations: the ϵ-dependence for OSEs is ϵ^{-2}. There is also an alternative "sketch-to-

precondition" approach, which enjoy a much better $\log(1/\epsilon)$ dependence for ϵ and so supports very high accuracy approximations.

For linear regression the idea is as follows. The sketched matrix is factored, $\mathbf{SX} = \mathbf{QR}$, and the unsketched original problem is solved using an iterative method with $\mathbf{R}$ serving as a preconditioner. This technique has been shown to be very effective in solving linear regression to high accuracy [4, 24].

2.4. Random feature maps as sketching. A random feature map φ can naturally be extended from defining a function $\mathcal{X} \to \mathbb{R}^s$ to one that defines a function $\mathbf{Sp}\left(\{k(\mathbf{x}_1, \cdot), \ldots, k(\mathbf{x}_n, \cdot)\}\right) \to \mathbb{R}^s$ via

$$\varphi\left(\sum_{i=1}^{n} \alpha_i k(x_i, \cdot)\right) \equiv \sum_{i=1}^{n} \alpha_i \varphi(\mathbf{x}_i)\,.$$

Thus, random feature maps can be viewed as a sketch that embeds

$$\mathbf{Sp}\left(\{k(\mathbf{x}_1, \cdot), \ldots, k(\mathbf{x}_n, \cdot)\}\right)$$

in $\mathbb{R}^s$. (Here $\mathbf{Sp}\,(\cdot)$ denotes the span of set of functions $\mathcal{X} \to \mathbb{R}$.) At least in one case, this embedding is an OSE, allowing stronger analysis of algorithms involving such feature maps: TensorSketch defines an OSE [5].

Viewed this way, the random features method is a sketch-and-solve approach. It is therefore not surprising that it produces suboptimal models. In this paper, we take the sketch-to-precondition approach to utilizing sketching.

3. Random features preconditioning.

3.1. Algorithm. We propose to use the random feature maps to form a preconditioner for $\mathbf{K} + \lambda \mathbf{I}_n$. In particular, let $\mathbf{Z} \in \mathbb{R}^{n \times s}$ have rows $\mathbf{z}_1^{\mathrm{T}}, \ldots, \mathbf{z}_n^{\mathrm{T}} \in \mathbb{R}^s$, where $\mathbf{z}_i = \varphi(\mathbf{x}_i)$. Here and throughout the rest of the paper, φ is a sample from the distribution defined by the random feature map. We assume that $s < n$. We solve $(\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{c} = \mathbf{y}$ using preconditioned conjugate gradients (PCG) with $\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n$ as a preconditioner.

Using $\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n$ as a preconditioner requires, in each iteration, the computation of $(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n)^{-1}\mathbf{x}$ for some vector $\mathbf{x}$. Let $\mathbf{L}\mathbf{L}^{\mathrm{T}} = \mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \lambda \mathbf{I}_s$ be a Cholesky decomposition of $\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \lambda \mathbf{I}_s$. The Woodbury formula implies that

$$(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n)^{-1} = \lambda^{-1}\left(\mathbf{I}_n - \mathbf{Z}\left(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \lambda \mathbf{I}_s\right)^{-1}\mathbf{Z}^{\mathrm{T}}\right) = \lambda^{-1}\left(\mathbf{I}_n - \mathbf{U}^{\mathrm{T}}\mathbf{U}\right),$$

where $\mathbf{U} = \mathbf{L}^{-\mathrm{T}}\mathbf{Z}^{\mathrm{T}}$. Thus, we can compute $\mathbf{L}$ and $\mathbf{U}$ before the PCG iterations to obtain an efficient method to apply the preconditioner in every iteration. A pseudocode description of the algorithm appears as Algorithm 1.

We remark that strictly speaking $\mathbf{U}$ is not necessary and we could apply the preconditioner using $\mathbf{L}$ and $\mathbf{Z}$ with the same asymptotic complexity. However, we empirically observed that forming $\mathbf{U}$ and using the above formula reduces the cost per iteration considerably and more than compensates for the additional time spent on pre-processing. Overall, the end result is a considerable improvement in running time in our implementation (even though both approaches have the same asymptotic complexity). One possible reason is that with our scheme we only need to do matrix-matrix products (GEMM operations) to apply the preconditioner, and we avoid triangular solves (TRSM operation) which tends not to exhibit good parallel scalability

Algorithm 1 Faster Kernel Ridge Regression using Sketching and Preconditioning.

Input: Data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$, kernel $k(\cdot,\cdot)$, feature map generating algorithm T, $\lambda > 0$, $s < n$, accuracy parameter $\epsilon > 0$.

Compute kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$.
Using T, create a feature map $\varphi : \mathbb{R}^d \to \mathbb{R}^s$.
Compute $\mathbf{z}_i = \varphi(\mathbf{x}_i), i = 1, \ldots, n$ and stack them in a matrix $\mathbf{Z}$.
Compute $\mathbf{Z}^{\mathrm{T}}\mathbf{Z}$, and Cholesky decomposition $\mathbf{L}^{\mathrm{T}}\mathbf{L} = \mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \lambda\mathbf{I}_s$.
Solve $\mathbf{L}^{\mathrm{T}}\mathbf{U} = \mathbf{Z}^{\mathrm{T}}$ for $\mathbf{U}$.
Stack $y_1, \ldots, y_n$ in a vector $\mathbf{y} \in \mathbb{R}^n$.
Solve $(\mathbf{K}+\lambda\mathbf{I}_n)\mathbf{c} = \mathbf{y}$ using PCG to accuracy ϵ with $\mathbf{Z}\mathbf{Z}^{\mathrm{T}}+\lambda\mathbf{I}_n$ as a preconditioner. In each iteration, in order to apply the preconditioner to some $\mathbf{x}$, compute $\lambda^{-1}(\mathbf{x}-\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{x})$.

return $\tilde{\mathbf{c}}$

(we observed gains mostly on large datasets where a large number of processes were used), although another possible reason might be specific features of the underlying library used for parallel matrix operations (Elemental [27]) and with another library it might be preferable to use $\mathbf{L}$ to solve for $\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda\mathbf{I}_n$.

Before analyzing the quality of the preconditioner we discuss the complexities of various operations associated with the algorithm. The cost of computing the kernel matrix $\mathbf{K}$ depends on the kernel and the sparsity of the input data $\mathbf{x}_1, \ldots, \mathbf{x}_n$. For the Gaussian kernel and the polynomial kernel the matrix can be computed in $O(n\sum_{i=1}^n \mathbf{nnz}(\mathbf{x}_i))$ time (although in many cases it might be beneficial to use the straightforward $\Theta(n^2 d)$ algorithm). The cost of computing $\mathbf{z}_1, \ldots, \mathbf{z}_n$ depends on the specific kernel and feature map used as well. For the Gaussian kernel with random Fourier features it is $O(s\sum_{i=1}^n \mathbf{nnz}(\mathbf{x}_i))$, and for the polynomial kernel with TensorSketch it is $O(q(\sum_{i=1}^n \mathbf{nnz}(\mathbf{x}_i) + ns\log s))$. Computing and decomposing $\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \lambda\mathbf{I}_s$ takes $\Theta(ns^2)$ and then another $\Theta(ns^2)$ for computing $\mathbf{U}$. The dominant cost per iteration is now multiplying the kernel matrix by a vector, which is $\Theta(n^2)$ operations.

3.2. Analysis. We now analyze the algorithm when applied to kernel ridge regression with the polynomial kernel and TensorSketch as the feature map. In particular, in the following theorem we show that if s is large enough, then PCG will converge in $O(1)$ iterations (for a fixed convergence threshold).

Theorem 1. *Let $\mathbf{K}$ be the kernel matrix associated with the q-degree homogeneous polynomial kernel $k(\mathbf{x},\mathbf{z}) = (\mathbf{x}^{\mathrm{T}}\mathbf{z})^q$. Let $s_\lambda(\mathbf{K}) \equiv \mathbf{Tr}\left((\mathbf{K}+\lambda\mathbf{I}_n)^{-1}\mathbf{K}\right)$. Let $\mathbf{z}_i = \varphi(\mathbf{x}_i) \in \mathbb{R}^s, i = 1, \ldots, n$, where φ is a* TensorSketch *map, and $\mathbf{Z} \in \mathbb{R}^{n\times s}$ have rows $\mathbf{z}_1^{\mathrm{T}}, \ldots, \mathbf{z}_n^{\mathrm{T}}$. Provided that*

$$s \geq 4(2+3^q)s_\lambda(\mathbf{K})^2/\delta \tag{2}$$

with probability of at least $1-\delta$, after

$$T = \left\lceil \frac{\sqrt{3}}{2}\ln(2/\epsilon) \right\rceil$$

iterations of PCG on $\mathbf{K}+\lambda\mathbf{I}_n$ starting from the all-zeros vector with $\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda\mathbf{I}_n$ as a

preconditioner we will find a $\tilde{\mathbf{c}}$ *such that*

$$\|\tilde{\mathbf{c}} - \mathbf{c}\|_{\mathbf{K}+\lambda\mathbf{I}_n} \leq \epsilon \|\mathbf{c}\|_{\mathbf{K}+\lambda\mathbf{I}_n} . \tag{3}$$

In the above, $\mathbf{c}$ *is the exact solution of the linear equation at hand (see* (1)*).*

Remark. The result is stated for the homogeneous polynomial kernel $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^{\mathrm{T}}\mathbf{z})^q$, but it can be easily generalized to the nonhomogeneous case $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^{\mathrm{T}}\mathbf{z} + c)^q$ by adding a constant feature to each training point.

Remark. While the theorem gives an explicit formula for the number of iterations, we do not recommend to actually use this formula and recommend instead the use of standard stopping criteria to declare convergence. (These usually involve determining that the residual norm has dropped below some tolerance.) The reason is that the iteration bound holds only with high probability. On the other hand, a higher probability bound bound holds when we consider a higher bound on the number of iterations. Thus, using standard stopping criteria renders the algorithm more robust. Furthermore, the bound holds only under exact arithmetic, while in practice PCG is used with inexact arithmetic.

The quantity $s_\lambda(\mathbf{K})$ is often referred to as the *statistical dimension* or *effective degrees-of-freedom* of the problem. It frequently features in the analysis of kernel regression [38, 8], low-rank approximations of kernel matrices [7], and analysis of sketching based approximate kernel learning [2, 37].

Estimating the statistical dimension is a nontrivial task that is outside the scope of this paper (and can sometimes be avoided: see section 5). Nevertheless, the bound does establish that the number of random features required for a constant number of iterations depends on the statistical dimension, which is always smaller than the number of training points. Since the kernel matrices often display quick decay in eigenvalues, it can be substantially smaller. In particular, the number of random features can be $o(n)$ when the statistical dimension is $o(n^{1/2})$. A discussion on how the statistical dimension behaves with regard to the training size is outside the scope of this paper. We refer the reader to a recent discussion by Bach on the subject [7].

Before proving the theorem we state some auxiliary definitions and lemmas.

DEFINITION 1. *Let* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *with* $n \geq m$ *and let* $\lambda \geq 0$. $\mathbf{A} = \mathbf{LQ}$ *is a* λ*-LQ factorization of* $\mathbf{A}$ *if* $\mathbf{Q}$ *is full rank,* $\mathbf{L}$ *is lower triangular and* $\mathbf{LL}^{\mathrm{T}} = \mathbf{AA}^{\mathrm{T}} + \lambda\mathbf{I}_m$.

A λ-LQ factorization always exists, and $\mathbf{L}$ is invertible for $\lambda > 0$. $\mathbf{Q}$ has orthonormal rows for $\lambda = 0$. The following proposition refines the last statement a bit.

PROPOSITION 2. *If* $\mathbf{A} = \mathbf{LQ}$ *is an* λ*-LQ factorization* $\mathbf{A} \in \mathbb{R}^{m\times n}$*, then* $\mathbf{QQ}^{\mathrm{T}} + \lambda\mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}} = \mathbf{I}_m$.

Proof. Multiply $\mathbf{LL}^{\mathrm{T}} = \mathbf{AA}^{\mathrm{T}} + \lambda\mathbf{I}_m$ from the left by $\mathbf{L}^{-1}$ and from the right by $\mathbf{L}^{-\mathrm{T}}$ to obtain the equality. ∎

PROPOSITION 3. *If* $\mathbf{A} = \mathbf{LQ}$ *is a* λ*-LQ factorization of* $\mathbf{A} \in \mathbb{R}^{m\times n}$*, then*

$$\|\mathbf{Q}\|_F^2 = s_\lambda\left(\mathbf{AA}^{\mathrm{T}}\right) \equiv \mathbf{Tr}\left(\left(\mathbf{AA}^{\mathrm{T}} + \lambda I_m\right)^{-1}\mathbf{AA}^{\mathrm{T}}\right).$$

Proof. Let $\sigma_1, \ldots, \sigma_m$ be the singular values of $\mathbf{A}$.

$$\begin{aligned}
\|\mathbf{Q}\|_F^2 = \mathbf{Tr}\left(\mathbf{Q}\mathbf{Q}^{\mathrm{T}}\right) &= \mathbf{Tr}\left(\mathbf{I}_m - \lambda \mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}}\right) \\
&= m - \lambda \mathbf{Tr}\left(\mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}}\right) \\
&= m - \lambda \mathbf{Tr}\left(\left(\mathbf{A}\mathbf{A}^{\mathrm{T}} + \mathbf{I}_m\right)^{-1}\right) \\
&= m - \sum_{i=1}^{m} \frac{\lambda}{\sigma_i^2 + \lambda} \\
&= \sum_{i=1}^{m} \frac{\sigma_i^2}{\sigma_i^2 + \lambda} \\
&= \mathbf{Tr}\left(\left(\mathbf{A}\mathbf{A}^{\mathrm{T}} + \lambda I_m\right)^{-1} \mathbf{A}\mathbf{A}^{\mathrm{T}}\right).
\end{aligned}$$

∎

In addition, we need the following lemma.

LEMMA 4 (see [5]). *Let $\mathbf{S} \in \mathbb{R}^{s \times d^q}$ be a* TENSORSKETCH *matrix, and suppose that $\mathbf{A}$ and $\mathbf{B}$ are matrices with d^q columns. For $s \geq (2 + 3^q)/(\nu^2 \delta)$ we have*

$$\Pr\left[\left\|\mathbf{A}\mathbf{S}^{\mathrm{T}}\mathbf{S}\mathbf{B}^{\mathrm{T}} - \mathbf{A}\mathbf{B}^{\mathrm{T}}\right\|_F^2 \leq \nu^2 \|\mathbf{A}\|_F^2 \|\mathbf{B}\|_F^2\right] \geq 1 - \delta\,.$$

We can now prove Theorem 1.

Proof of Theorem 1. We prove that with probability of at least $1 - \delta$

$$\frac{2}{3}\left(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n\right) \preceq \mathbf{K} + \lambda \mathbf{I}_n \preceq 2\left(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n\right). \tag{4}$$

Thus, with probability of $1 - \delta$ the relevant condition number is bounded by 3. For PCG, if the condition number is bounded by κ, we are guaranteed to reduce the error (measured in the matrix norm of the linear equation) to an ϵ fraction of the initial guess after $\lceil \sqrt{\kappa} \ln(2/\epsilon)/2 \rceil$ iterations [31]. This immediately leads to the bound in the theorem statement.

Let $\mathbf{V}_q \in \mathbb{R}^{n \times d^q}$ be the matrix whose row i corresponds to expanding $\mathbf{x}_i$ to the values of all possible q-degree monomials,[2] i.e., $v_q(\mathbf{x}_i)$ in the terminology of section 2.2. We have $\mathbf{K} = \mathbf{V}_q \mathbf{V}_q^{\mathrm{T}}$. Furthermore, there exists a matrix $\mathbf{S} \in \mathbb{R}^{s \times d^q}$ such that $\mathbf{Z} = \mathbf{V}_q \mathbf{S}^{\mathrm{T}}$, so (4) translates to

$$\frac{2}{3}\left(\mathbf{V}_q \mathbf{S}^{\mathrm{T}} \mathbf{S} \mathbf{V}_q^{\mathrm{T}} + \lambda \mathbf{I}_n\right) \preceq \mathbf{V}_q \mathbf{V}_q^{\mathrm{T}} + \lambda \mathbf{I}_n \preceq 2\left(\mathbf{V}_q \mathbf{S}^{\mathrm{T}} \mathbf{S} \mathbf{V}_q^{\mathrm{T}} + \lambda \mathbf{I}_n\right),$$

or equivalently,

$$\frac{1}{2}\left(\mathbf{V}_q \mathbf{V}_q^{\mathrm{T}} + \lambda \mathbf{I}_n\right) \preceq \mathbf{V}_q \mathbf{S}^{\mathrm{T}} \mathbf{S} \mathbf{V}_q^{\mathrm{T}} + \lambda \mathbf{I}_n \preceq \frac{3}{2}\left(\mathbf{V}_q \mathbf{V}_q^{\mathrm{T}} + \lambda \mathbf{I}_n\right). \tag{5}$$

Let $\mathbf{V}_q = \mathbf{L}\mathbf{Q}$ be a λ-LQ factorization of $\mathbf{V}_q$. It is well known that for $\mathbf{C}$ that is square and invertible, $\mathbf{A} \preceq \mathbf{B}$ if and only if $\mathbf{C}^{-1}\mathbf{A}\mathbf{C}^{-\mathrm{T}} \preceq \mathbf{C}^{-1}\mathbf{B}\mathbf{C}^{-\mathrm{T}}$. Applying this

[2] The letter $\mathbf{V}$ alludes to the fact that $\mathbf{V}_q$ can be thought of as a multivariate analogue of the Vandermonde matrix.

to the previous equation with $\mathbf{C} = \mathbf{L}$ implies that (5) holds if and only if

$$\frac{1}{2}\mathbf{I}_n \preceq \mathbf{Q}\mathbf{S}^{\mathrm{T}}\mathbf{S}\mathbf{Q}^{\mathrm{T}} + \lambda\mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}} \preceq \frac{3}{2}\mathbf{I}_n\,. \tag{6}$$

A sufficient condition for (6) to hold is that

$$\left\|\mathbf{Q}\mathbf{S}^{\mathrm{T}}\mathbf{S}\mathbf{Q}^{\mathrm{T}} + \lambda\mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}} - \mathbf{I}_n\right\|_2 \leq \frac{1}{2}\,. \tag{7}$$

According to Proposition 2 we have

$$\left\|\mathbf{Q}\mathbf{S}^{\mathrm{T}}\mathbf{S}\mathbf{Q}^{\mathrm{T}} + \lambda\mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}} - \mathbf{I}_n\right\|_2 = \left\|\mathbf{Q}\mathbf{S}^{\mathrm{T}}\mathbf{S}\mathbf{Q}^{\mathrm{T}} - \mathbf{Q}\mathbf{Q}^{\mathrm{T}}\right\|_2\,.$$

According to Lemma 4, if $s \geq 4(2+3^q)\|\mathbf{Q}\|_F^4/\delta$, then with probability of at least $1-\delta$ we have

$$\left\|\mathbf{Q}\mathbf{S}^{\mathrm{T}}\mathbf{S}\mathbf{Q}^{\mathrm{T}} - \mathbf{Q}\mathbf{Q}^{\mathrm{T}}\right\|_2 \leq \left\|\mathbf{Q}\mathbf{S}^{\mathrm{T}}\mathbf{S}\mathbf{Q} - \mathbf{Q}\mathbf{Q}^{\mathrm{T}}\right\|_F \leq \frac{1}{2}\,.$$

Now complete the proof using the equality $\|\mathbf{Q}\|_F^2 = s_\lambda(\mathbf{V}_q\mathbf{V}_q^{\mathrm{T}}) = s_\lambda(\mathbf{K})$ (Proposition 3). ∎

3.3. Other kernels and feature maps. Close inspection of the proof reveals that the crucial ingredient is the matrix multiplication lemma (Lemma 4). In the following, we generalize Theorem 1 to feature maps which have similar structural properties. The proof, which is mostly analogous to the proof of Theorem 1, is included in the appendix.

In the following, for finite ordered sets $\mathcal{U}, \mathcal{V} \subset \mathcal{H}_k$ we denote by $\mathbf{K}(\mathcal{U}, \mathcal{V})$ the Gram matrix associated with this two sets, i.e., $\mathbf{K}_{ij} = (\mathbf{u}_i, \mathbf{v}_j)_{\mathcal{H}_k}$ for $\mathbf{u}_i \in \mathcal{U}$ and $\mathbf{v}_j \in \mathcal{V}$. We assume that the feature map defines a transformation from $\mathcal{H}_k$ to $\mathbb{R}^s$, that is, a sample from the distribution is a function $\varphi : \mathcal{H}_k \to \mathbb{R}^s$. A feature map is linear if every sample φ it can take is linear.

Definition 2. *A linear feature map has an* approximate multiplication property *with $f(\nu, \delta)$ if φ with at least $f(\nu, \xi, \delta)$ random features has that for all finite ordered sets $\mathcal{U}, \mathcal{V} \subset \mathcal{H}_k$ the following holds with probability of at least $1-\delta$:*

$$\left\|\mathbf{Z}_{\mathcal{U}}\mathbf{Z}_{\mathcal{V}}^{\mathrm{T}} - \mathbf{K}(\mathcal{U}, \mathcal{V})\right\|_F^2 \leq \nu^2 \mathbf{Tr}\left(\mathbf{K}(\mathcal{U},\mathcal{U})\right)\mathbf{Tr}\left(\mathbf{K}(\mathcal{V},\mathcal{V})\right) + \xi^2\,,$$

where $\mathbf{Z}_{\mathcal{U}}$ (resp., $\mathbf{Z}_{\mathcal{V}}$) is the matrix whose row i corresponds to applying φ to $\mathbf{u}_i$ (resp., $\mathbf{v}_i$).

Theorem 5. *Suppose that φ is a sample from a feature map that has an approximate multiplication property with $f(\nu, \delta)$. Suppose that φ has $s \geq f(s_\lambda(\mathbf{K})^{-1}/2, 0, \delta)$ or $s \geq f(s_\lambda(\mathbf{K})^{-1}/2\sqrt{2}, 1/2\sqrt{2}, \delta)$ features. Let $\mathbf{z}_i = \varphi(k(\mathbf{x}_i, \cdot)) \in \mathbb{R}^s, i = 1, \ldots, n$ and $\mathbf{Z} \in \mathbb{R}^{n\times s}$ have rows $\mathbf{z}_1^{\mathrm{T}}, \ldots, \mathbf{z}_n^{\mathrm{T}}$. With probability of at least $1-\delta$, after*

$$T = \left\lceil \frac{\sqrt{3}}{2}\ln(2/\epsilon) \right\rceil$$

iterations of PCG on $\mathbf{K} + \lambda\mathbf{I}_n$ starting from the all-zeros vector with $\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda\mathbf{I}_n$ as a preconditioner we will find a $\tilde{\mathbf{c}}$ such that

$$\|\tilde{\mathbf{c}} - \mathbf{c}\|_{\mathbf{K}+\lambda\mathbf{I}_n} \leq \epsilon\|\mathbf{c}\|_{\mathbf{K}+\lambda\mathbf{I}_n}\,.$$

Currently, there is no proof that the approximate multiplication property holds for any feature map except for TensorSketch.

4. Multiple level sketching. We now show that the dependence on the statistical dimension can be improved by composing multiple sketching transforms. The crucial observation is that after the initial random feature transform, the training set is embedded in a Euclidean space. This suggests the composition of well-known transforms such as the subsampled randomized Hadamard transform (SRHT) and the Johnson–Lindenstrauss transform with the initial random feature map. A similar idea, referred to as *compact random features*, was explored in the context of the random features method by Hamid et al. [20].

First, we consider the use of the SRHT after the initial TensorSketch for the polynomial kernel. To that end we recall the definition of the SRHT. Let m be a power of 2. The $m \times m$ matrix of the Walsh–Hadamard Transform (WHT) is defined recursively as

$$\mathbf{H}_m = \begin{bmatrix} \mathbf{H}_{m/2} & \mathbf{H}_{m/2} \\ \mathbf{H}_{m/2} & -\mathbf{H}_{m/2} \end{bmatrix} \text{ with } \mathbf{H}_2 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix}.$$

Definition 3. *Let m be some integer which is a power of 2, and s an integer. A* subsampled randomized Walsh–Hadamard transform *is an $s \times m$ matrix of the form*

$$\mathbf{S} = \frac{1}{\sqrt{s}} \mathbf{PHD},$$

where $\mathbf{D}$ is a random diagonal matrix of size m whose entries are independent random signs, $\mathbf{H}$ is a Walsh–Hadamard matrix of size m, and $\mathbf{P}$ is a random sampling matrix.

The recursive nature of the WHT matrix allows for a quick multiplication of a SRHT matrix by a vector. In particular, if $\mathbf{S} \in \mathbb{R}^{s \times m}$ is an SRHT, then $\mathbf{Sx}$ can be computed in $O(m \log(s))$ [1].

The proposed algorithm proceeds as follows. First, we apply a random feature transform with s_1 features to obtain $\mathbf{Z}_1 \in \mathbb{R}^{n \times s_1}$. We now apply a SRHT $\mathbf{S}_2 \in \mathbb{R}^{s_2 \times s_1}$ to the rows of $\mathbf{Z}_1$, that is, compute $\mathbf{Z}_2 = \mathbf{Z}_1 \mathbf{S}_2^{\mathrm{T}}$, and use $\mathbf{Z}_2 \mathbf{Z}_2^{\mathrm{T}} + \lambda \mathbf{I}_n$ as a preconditioner for $\mathbf{K} + \lambda \mathbf{I}_n$.

The following theorem establishes that using this scheme it is possible for the polynomial kernel to construct a good preconditioner with only $O(s_\lambda(\mathbf{K}) \log(s_\lambda(\mathbf{K})))$ columns in $\mathbf{Z}_2$.

Theorem 6. *Let $\mathbf{K}$ be the kernel matrix associated with the q-degree homogeneous polynomial kernel $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^{\mathrm{T}} \mathbf{z})^q$. Let $s_\lambda(\mathbf{K}) \equiv \mathbf{Tr}\left((\mathbf{K} + \lambda \mathbf{I}_n)^{-1} \mathbf{K}\right)$. Assume that $\|\mathbf{K}\|_2 \geq \lambda$. Let $\varphi : \mathbb{R}^d \to \mathbb{R}^{s_1}$ be a* TensorSketch *mapping with*

$$s_1 \geq 32(2 + 3^q) s_\lambda(\mathbf{K})^2 / \delta,$$

and let $\mathbf{S}_2 \in \mathbb{R}^{s_2 \times s_1}$ be a SRHT with $s_2 = \Omega(s_\lambda(\mathbf{K}) \log(\mathbf{s}_\lambda(\mathbf{K}))/\delta^2)$. Let $\mathbf{z}_i = \mathbf{S}_2 \varphi(\mathbf{x}_i) \in \mathbb{R}^s, i = 1, \ldots, n$ and $\mathbf{Z}_2 \in \mathbb{R}^{n \times s}$ have rows $\mathbf{z}_1^{\mathrm{T}}, \ldots, \mathbf{z}_n^{\mathrm{T}}$. Then, with probability of at least $1 - \delta$, after

$$T = \left\lceil \frac{\sqrt{3}}{2} \ln(2/\epsilon) \right\rceil$$

iterations of PCG on $\mathbf{K}+\lambda\mathbf{I}_n$ *starting from the all-zeros vector with* $\mathbf{Z}\mathbf{Z}^{\mathrm{T}}+\lambda\mathbf{I}_n$ *as a preconditioner we will find a* $\tilde{\mathbf{c}}$ *such that*

$$\|\tilde{\mathbf{c}}-\mathbf{c}\|_{\mathbf{K}+\lambda\mathbf{I}_n} \le \epsilon\|\mathbf{c}\|_{\mathbf{K}+\lambda\mathbf{I}_n}. \tag{8}$$

In the above, $\mathbf{c}$ *is the exact solution of the linear equation at hand (see* (1)*).*

Proof. Let $\mathbf{V}_q$ and $\mathbf{Q}$ be as defined in the proof of Theorem 1. Let $\mathbf{S}_1$ be the matrix that corresponds to φ, and let $\mathbf{S}=\mathbf{S}_2\mathbf{S}_1$. We have $\mathbf{Z}=\mathbf{V}_q\mathbf{S}^{\mathrm{T}}$. Following the proof of Theorem 1 it suffices to prove that

$$\left\|\mathbf{Q}\mathbf{S}^{\mathrm{T}}\mathbf{S}\mathbf{Q}^{\mathrm{T}}-\mathbf{Q}\mathbf{Q}^{\mathrm{T}}\right\|_2 \le \frac{1}{2}.$$

Noticing that

$$\left\|\mathbf{Q}\mathbf{S}^{\mathrm{T}}\mathbf{S}\mathbf{Q}^{\mathrm{T}}-\mathbf{Q}\mathbf{Q}^{\mathrm{T}}\right\|_2 \le \left\|\mathbf{Q}\mathbf{S}_1^{\mathrm{T}}\mathbf{S}_2^{\mathrm{T}}\mathbf{S}_2\mathbf{S}_1\mathbf{Q}^{\mathrm{T}}-\mathbf{Q}\mathbf{S}_1^{\mathrm{T}}\mathbf{S}_1\mathbf{Q}^{\mathrm{T}}\right\|_2 + \left\|\mathbf{Q}\mathbf{S}_1^{\mathrm{T}}\mathbf{S}_1\mathbf{Q}^{\mathrm{T}}-\mathbf{Q}\mathbf{Q}^{\mathrm{T}}\right\|_2$$

it suffices to prove that each of the two terms is bounded by $\frac{1}{4}$ with probability $1-\delta/2$. Due to the lower bound on s_1 this holds for the right term as explained in the proof of Theorem 1.

For the left term, we used recent results by Cohen, Nelson, and Woodruff [14] that show that for a matrix $\mathbf{A}$ and a SRHT Π with $O(\tilde{r}(\mathbf{A})\log(\tilde{r}(\mathbf{A}))/\nu^2)$ rows, where $\tilde{r}(\mathbf{A})\equiv\|\mathbf{A}\|_F^2/\|\mathbf{A}\|_2^2$ is the *stable rank* of $\mathbf{A}$, we have

$$\left\|\mathbf{A}^{\mathrm{T}}\Pi^{\mathrm{T}}\Pi\mathbf{A}-\mathbf{A}^{\mathrm{T}}\mathbf{A}\right\|_2 \le \nu\|\mathbf{A}\|_2^2$$

with high probability.

We bound the left term by applying this result to $\mathbf{A}=\mathbf{S}_1\mathbf{Q}^{\mathrm{T}}$. First, we note that $\|\mathbf{Q}^{\mathrm{T}}\|_2^2=\|\mathbf{K}\|_2/(\|\mathbf{K}\|_2+\lambda)\ge 1/2$. Next, we note that

$$\begin{aligned}\left\|\mathbf{S}_1\mathbf{Q}^{\mathrm{T}}\right\|_2 &= \left\|\mathbf{Q}\mathbf{S}_1^{\mathrm{T}}\mathbf{S}_1\mathbf{Q}^{\mathrm{T}}\right\|_2^{1/2}\\ &\le \left\|\mathbf{Q}^{\mathrm{T}}\right\|_2 + \left\|\mathbf{Q}\mathbf{S}_1^{\mathrm{T}}\mathbf{S}_1\mathbf{Q}^{\mathrm{T}}-\mathbf{Q}\mathbf{Q}^{\mathrm{T}}\right\|_2^{1/2}.\end{aligned}$$

The second term has already been proved to be $O(1)$, while the first is bounded by one. Therefore, we conclude that $\|\mathbf{S}_1\mathbf{Q}^{\mathrm{T}}\|_2=O(1)$ so $O(\tilde{r}(\mathbf{S}_1\mathbf{Q}^{\mathrm{T}})\log(\tilde{r}(\mathbf{S}_1\mathbf{Q}^{\mathrm{T}})))$ row is sufficient. In addition, since $\mathbb{E}[\|\mathbf{S}_1\mathbf{Q}^{\mathrm{T}}\|_F^2]=\|\mathbf{Q}^{\mathrm{T}}\|_F^2$ we conclude that with high probability $\|\mathbf{S}_1\mathbf{Q}^{\mathrm{T}}\|_F^2=O(\|\mathbf{Q}^{\mathrm{T}}\|_F^2)=O(s_\lambda(\mathbf{K}))$. Finally, recall that $\|\mathbf{Q}^T\|_2^2\ge 1/2$ so $\tilde{r}(\mathbf{S}_1\mathbf{Q}^{\mathrm{T}})=O(s_\lambda(\mathbf{K}))$. ∎

Remark. Notice that for multiple level sketching we require that $\|\mathbf{K}\|_2\ge\lambda$. This assumption is very reasonable: from a statistical point of view it should be the case that $\lambda\ll\|\mathbf{K}\|_2$. (Otherwise, the regularizer dominates the data.) From a computational point of view, if $\lambda=\Omega(\|\mathbf{K}\|_2)$, then the condition number of $\mathbf{K}+\lambda\mathbf{I}_n$ is $O(1)$ to begin with (although the constant might be huge). In particular, if $\lambda\ge\|\mathbf{K}\|_2$, then the condition number is bounded by 2.

Remark. The dependence on δ in the previous theorem (and also in Theorem 1) can be improved to $O(\mathbf{poly}(\log(1/\delta)))$ by considering a fixed failure probability and repeating the algorithm $O(\log(1/\delta))$ times. If $\tilde{\mathbf{c}}_1,\ldots,\tilde{\mathbf{c}}_r$ are the candidate solutions for $r=O(\log(1/\delta))$, then a sufficiently good solution can be found by looking at the

Algorithm 2 Multiple Level Sketching Algorithm.

Input: Data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$, kernel $k(\cdot,\cdot)$, feature map generating algorithm T, $\lambda > 0$, $s_3 < s_2 < s_1 < n$, accuracy parameter $\epsilon > 0$.

Compute kernel matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$.
Using T, create a feature map $\varphi : \mathbb{R}^d \to \mathbb{R}^{s_1}$.
Compute $\mathbf{z}_i = \varphi(\mathbf{x}_i), i = 1, \ldots, n$ and stack them in a a matrix $\mathbf{Z}_1$.
Apply a subsampled randomized Hadamard transform $\mathbf{S}_2 \in \mathbb{R}^{s_2\times s_1}$ to the rows of $\mathbf{Z}_1$ to form $\mathbf{Z}_2 = \mathbf{Z}_1\mathbf{S}_2^{\mathrm{T}}$.
Let $\mathbf{S}_3 \in \mathbb{R}^{s_3\times s_2}$ be a random matrix whose entries independent standard normal random variables. Compute $\mathbf{Z}_3 = \frac{1}{\sqrt{s_3}}\mathbf{Z}_2\mathbf{S}_3^{\mathrm{T}}$.
Solve $\mathbf{L}^{\mathrm{T}}\mathbf{U} = \mathbf{Z}^{\mathrm{T}}$ for $\mathbf{U}$.
Stack $y_1, \ldots, y_n$ in a vector $\mathbf{y} \in \mathbb{R}^n$.
Solve $(\mathbf{K}+\lambda\mathbf{I}_n)\mathbf{c} = \mathbf{y}$ using PCG to accuracy ϵ with $\mathbf{Z}\mathbf{Z}^{\mathrm{T}}+\lambda\mathbf{I}_n$ as a preconditioner. In each iteration, in order to apply the preconditioner to some $\mathbf{x}$, compute $\lambda^{-1}(\mathbf{x}-\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{x})$.

return $\tilde{\mathbf{c}}$

differences $\|\tilde{\mathbf{c}}_i - \tilde{\mathbf{c}}_j\|_{\mathbf{K}+\lambda\mathbf{I}_n}$ for all i and j, and outputting an i for which, if we sort these differences for all j, the median value of the sorted list for i is the smallest. The analysis is based on Chernoff bounds and the triangle inequality, and it requires adjusting ϵ by a constant factor. We omit the details.

From a computational complexity point of view, in many cases one can set s_1 to be very large without increasing the asymptotic cost of the algorithm. For example, for random Fourier features, if $\sum_{i=1}^n \mathbf{nnz}(\mathbf{x}_i) = O(n \log s_2)$, then it is possible to set $s_1 = \Theta(n)$ without increasing the asymptotic cost of the algorithm.

It is also possible to further reduce the dimension to $O(s_\lambda(\mathbf{K}))$ using a dense random projection, i.e., multiplying $\mathbf{Z}_2$ from the right by a scaled random matrix with sub-Gaussian entries. We omit the proof as it is analogous to the proof of Theorem 6. Pseudocode description of the three-level sketching algorithm is given as Algorithm 2.

5. Adaptively setting the sketch size. It is also possible to adaptively set s. The basic idea is to successively form larger $\mathbf{Z}$'s until we have one that is good enough. The following theorem implies quickly testable conditions for this, again for the polynomial kernel and TensorSketch.

THEOREM 7. *Let $\mathbf{P} \in \mathbb{R}^{n\times n}$ be the orthogonal projection matrix on the subspace spanned by the eigenvectors of $\mathbf{Z}\mathbf{Z}^{\mathrm{T}}$ whose corresponding eigenvalues are bigger than 0.05λ, and let $\mathbf{V}_q$ have $\mathbf{K} = \mathbf{V}_q\mathbf{V}_q^{\mathrm{T}}$. Suppose that the following hold:*

1. $\|(\mathbf{I}_n - \mathbf{P})\mathbf{K}(\mathbf{I}_n - \mathbf{P})\|_2 \le 0.1\lambda$.
2. *For all* $\mathbf{x}$, $\|\mathbf{V}_q^{\mathrm{T}}\mathbf{P}\mathbf{x}\|_2^2 = (1 \pm 0.1)\|\mathbf{Z}^{\mathrm{T}}\mathbf{P}\mathbf{x}\|_2^2$.

Then the guarantees of Theorem 1 hold. Moreover, if $s \ge Cs_\lambda(\mathbf{K})^2/\delta$ for some sufficiently large constant C, then with probability of at least $1-\delta$ conditions 1 and 2 above hold as well.

Proof. We show that there exists constants m and M such that

$$m\left(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda\mathbf{I}_n\right) \preceq \mathbf{K} + \lambda\mathbf{I}_n \preceq M\left(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda\mathbf{I}_n\right).$$

This holds if there is a constant $c < 1$ such that for all $\mathbf{y}$.

$$\mathbf{y}^T(\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{y} = (1 \pm c)\mathbf{y}^{\mathrm{T}} \left(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n\right) \mathbf{y}\,.$$

Without loss of generality we restrict ourselves to $\|\mathbf{y}\|_2 = 1$. We have

$$\begin{aligned}
\mathbf{y}^{\mathrm{T}}(\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{y} &= \lambda\|\mathbf{y}\|_2^2 + \mathbf{y}^{\mathrm{T}}\mathbf{K}\mathbf{y} \\
&= \lambda + (\mathbf{P}\mathbf{y} + (\mathbf{I} - \mathbf{P})\mathbf{y})^{\mathrm{T}}\mathbf{K}(\mathbf{P}\mathbf{y} + (\mathbf{I} - \mathbf{P})\mathbf{y}) \\
&= \lambda + \mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{K}\mathbf{P}\mathbf{y} + \mathbf{y}^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{K}(\mathbf{I} - \mathbf{P})\mathbf{y} + 2\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{K}(\mathbf{I} - \mathbf{P})\mathbf{y} \\
&= (1 \pm 0.1)\lambda + (1 \pm 0.1)\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\mathbf{P}\mathbf{y} + 2\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{K}(\mathbf{I} - \mathbf{P})\mathbf{y} \pm 0.1\lambda \\
&= (1 \pm 0.1)\lambda + (1 \pm 0.1)\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\mathbf{P}\mathbf{y} + 2\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{K}(\mathbf{I} - \mathbf{P})\mathbf{y} \\
&\quad + \big[(1 \pm 0.1)\mathbf{y}^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{Z}\mathbf{Z}^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{y} + (1 \pm 0.1)\mathbf{y}^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\mathbf{P}\mathbf{y} \\
&\quad - (1 \pm 0.1)\mathbf{y}^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{Z}\mathbf{Z}^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{y}\big] \\
&= (1 \pm 0.1)\lambda + (1 \pm 0.1)\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\mathbf{P}\mathbf{y} + 2\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{K}(\mathbf{I} - \mathbf{P})\mathbf{y} \\
&\quad + \big[(1 \pm 0.1)\mathbf{y}^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{Z}\mathbf{Z}^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{y} \\
&\quad + (1 \pm 0.1)\mathbf{y}^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\mathbf{P}\mathbf{y} \pm (1 \pm 0.1)0.1\lambda\big] \\
&= (1 \pm 0.21)\lambda + (1 \pm 0.1)\mathbf{y}^{\mathrm{T}}\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\mathbf{y} + 2\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{K}(\mathbf{I} - \mathbf{P})\mathbf{y} \\
&= (1 \pm 0.21)\mathbf{y}^{\mathrm{T}} \left(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}\right) \mathbf{y} + 2\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{K}(\mathbf{I} - \mathbf{P})\mathbf{y} \\
&= (1 \pm 0.21)\mathbf{y}^{\mathrm{T}} \left(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}\right) \mathbf{y} \pm 2 \left\|\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{V}_q\right\|_2 \left\|\mathbf{V}_q^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{y}\right\|_2 \\
&= (1 \pm 0.21)\mathbf{y}^{\mathrm{T}} \left(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}\right) \mathbf{y} \pm 2 \left\|\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{V}_q\right\|_2 \sqrt{0.1\lambda}(1 \pm 0.1)\,.
\end{aligned}$$

The fourth equality is due to condition 2 (which bounds $\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{K}\mathbf{P}\mathbf{y}$ since $\mathbf{K} = \mathbf{V}_q\mathbf{V}_q^{\mathrm{T}}$) and condition 1 (which bounds $\mathbf{y}^{\mathrm{T}}(\mathbf{I}-\mathbf{P})\mathbf{K}(\mathbf{I}-\mathbf{P})\mathbf{y}$). In the fifth equality we note that because $\mathbf{P}$ is a projection on an eigenspace of $\mathbf{Z}\mathbf{Z}^{\mathrm{T}}$, $(\mathbf{I} - \mathbf{P})\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\mathbf{P} = 0$. In the sixth equality we use the definition of $\mathbf{P}$ to bound $\mathbf{y}^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{Z}\mathbf{Z}^{\mathrm{T}}(\mathbf{I} - \mathbf{P})\mathbf{y} \leq 0.1\lambda$. In the ninth equality we use Cauchy–Schwarz. And in the tenth equality we use condition 1 again.

If $\sqrt{0.1\lambda} \leq \|\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{Z}\|_2/3$, then the last term is at most $\frac{2}{3}1.1\|\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{V}_q\|_2^2 = \frac{11}{15}\mathbf{y}^{\mathrm{T}}\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\mathbf{y}$. Notice that now $(0.21 + 11/15) < 1$, so $\mathbf{y}^{\mathrm{T}}(\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{y} = (1 \pm c)\mathbf{y}^{\mathrm{T}}(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I})\mathbf{y}$ for some constant smaller than 1.

If $\sqrt{0.1\lambda} \geq \|\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{Z}\|_2/3$, then the last term is bounded $2\|\mathbf{y}^{\mathrm{T}}\mathbf{P}\mathbf{V}_q\|_2\sqrt{0.1\lambda}(1 \pm 0.1) \leq \frac{6}{11}\lambda \leq \frac{6}{11}\mathbf{y}^{\mathrm{T}}(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n)\mathbf{y}$ and again $\mathbf{y}^{\mathrm{T}}(\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{y} = (1 \pm c)\mathbf{y}^{\mathrm{T}}(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I})\mathbf{y}$ for some constant smaller than 1.

As for the other direction, if the conditions of Theorem 1 hold, then we showed in the proof of Theorem 1 that (4) holds with high probability. It can be easily seen that by adjusting the constant in front of (2) we can adjust the constants in (4) to be as small as needed. Equation (4) immediately implies condition 2. As for condition 1, if it was not true than there is a unit vector $\mathbf{y}$ such that $\mathbf{y}^{\mathrm{T}}(\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{y} \geq 1.1\lambda$ but $\mathbf{y}^{\mathrm{T}}(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n)\mathbf{y} \leq 1.05\lambda$. Thus, if the adjusted constant is small enough it is impossible for (the modified) (4) to hold (i.e., $1.1\lambda \leq (1 + \epsilon_0)1.05\lambda$ cannot hold for small enough ϵ_0). ∎

This theorem can be used in the following way. We start with some small s, generate $\mathbf{Z}$, and test it. If $\mathbf{Z}$ is not good enough, we double s and generate a new $\mathbf{Z}$. We continue until we have a good enough preconditioner (which will happen

when s is large enough per Theorem 1). Testing condition 1 can be accomplished using simple power-iteration. Testing of condition 2 can be done by constructing a subspace embedding to the range of $\mathbf{V}_q^{\mathrm{T}}\mathbf{P}$ using TensorSketch. We omit the technical details.

6. Experiments. In this section we report experimental results with an implementation of our proposed one-level sketching algorithm. To allow the code to scale to datasets of size one million examples and beyond, we designed our code to use distributed memory parallelism using MPI. For most distributed matrix operations we rely on the Elemental library [27]. We experiment on clusters composed on Amazon Web Services EC2 instances.

In our experiments, we use standard classification and regression datasets, using regularized least squares classification to convert the classification problem to a regression problem. We use two kernel-feature map combinations: the Gaussian kernel $k(\mathbf{x},\mathbf{z}) = \exp(-\|\mathbf{x}-\mathbf{z}\|_2^2/2\sigma^2)$ along with random Fourier features and the polynomial kernel $k(\mathbf{x},\mathbf{z}) = (\gamma\mathbf{x}^{\mathrm{T}}\mathbf{z}+c)^q$ along with TensorSketch. We declare convergence of PCG when the iterate $\mathbf{c}$ is such that $\|\mathbf{y}-(\mathbf{K}+\lambda\mathbf{I}_n)\mathbf{c}\|_2 \le \tau\|\mathbf{y}\|_2$. For multiple right-hand sides we require this to hold for all right-hand sides. We set $\tau = 10^{-3}$ for classification and $\tau = 10^{-5}$ for regression; while these are not particularly small, our experiments did reveal that for the tested datasets further error reduction does not improve the generalization performance (see also subsection 6.3).

6.1. Comparison to the random features method. In this section we compare our method to the random features method, as defined in section 2.2. Our goal is to demonstrate that our method, which solves the nonapproximate kernel problem to relatively high accuracy, is able to fully leverage the data and deliver better generalization results than is possible using the random features method.

We conducted experiments on the MNIST dataset. We use both the Gaussian kernel (with $\sigma = 8.5$) and the polynomial kernel $k(\mathbf{x},\mathbf{z}) = (0.01\mathbf{x}^{\mathrm{T}}\mathbf{z}+1)^3$. The regularization parameter was set to $\lambda = 0.01$. Running time were measured on a single c4.8xlarge EC2 instance.

Results are shown in Figure 1. Inspecting the error rates (left plots), we see that while the random features method is able to deliver close to optimal error rates, there is a clear gap between the errors obtained using our method and the ones obtained by the random features method even for a very large number of random features. The gap persists even if we set the number of random features to be as large as the training size!

In terms of the running time (right plots), our method is, as expected, generally more expensive than the sketch-and-solve approach of the random features method, at least when the number of random features is the same. However, in order to get close to the performance of our method, the random features method requires so many features that its running time eventually surpasses that of our method with the optimal number of features. (Recall that for our method the number of random features only affects the running time, not the generalization performance.)

It is worth noting that the optimal training time for our method on MNIST was actually quite small: less than 2 minutes for the Gaussian kernel and less than 4 minutes for the polynomial kernel, and this without sacrificing in terms of generalization performance by using an approximation.

In Figure 2 we further explore the complex interaction between algorithmic complexity, quantity of data, and predicative performance. In this set of experiments we use subsamples of the COVTYPE dataset (to a maximum sample of 80% of the data;

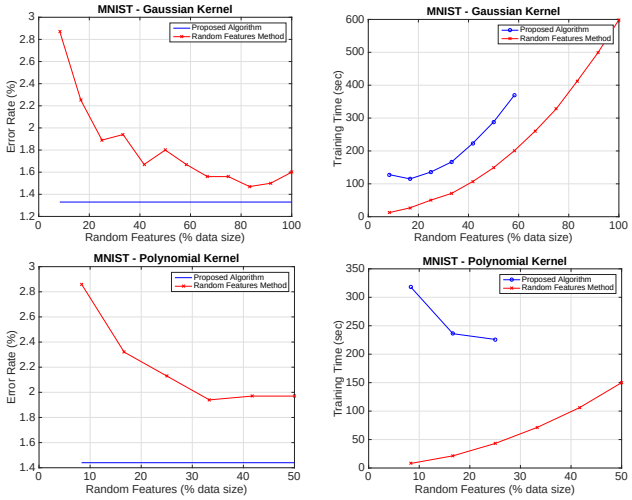

FIG. 1. *Comparison with random features on MNIST.*

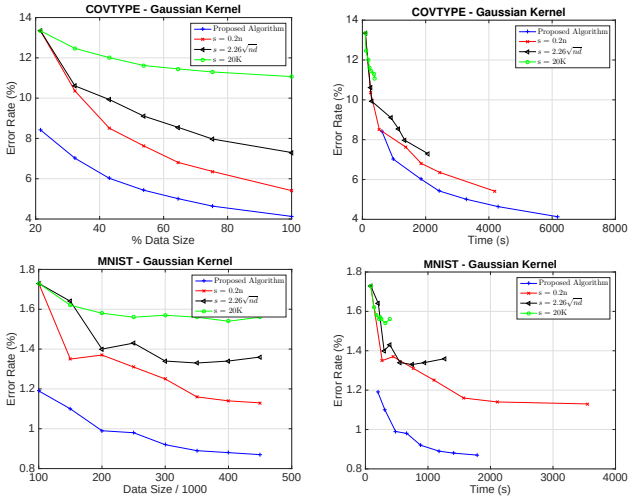

FIG. 2. *Error as a function of time and data size on MNIST and COVTYPE.*

the rest is used for testing and validation) and subsamples from the extended MNIST-8M dataset (to a maximum of 450K data points). We compare the performance of our method as the number of samples increases to the performance of the random features method with three different profiles for setting s: $s = 20000$ (an $O(n)$ training algorithm), $s = 2.26\sqrt{nd}$ (an $O(n^2 d)$ algorithm), and $s = 0.2n$ (an $O(n^3)$ time, $O(n^2)$ memory algorithm). For both datasets we plot both error as a function of the datasize and error as a function of training time. The graphs clearly demonstrate the superiority of our method.

6.2. Resources and running time on a cloud service. Our implementation is designed to leverage distributed processing using clusters of machines. The wide availability of cloud-based computing services provides easy access to such platforms on an on-demand basis.

To give an idea of the running time and resources (and as a consequence the cost) of training models using our algorithm on public cloud services, we applied our method to various popular datasets on EC2. The results are summarized in Table 1. We can observe that using our method it is possible to train high-quality models on datasets as large as one million data points in a few hours. We remark that while we did tune σ and λ somewhat, we did not attempt to tune them to the best possible values.

6.3. Additional experimental results. So far we described and experimented with using $\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda\mathbf{I}_n$ as a preconditioner. One straightforward idea is to try to use $\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda_p\mathbf{I}_n$ as a preconditioner, where λ_p is now a parameter that is not necessarily equal to λ. Although our current theory does not cover this case, we evaluate this idea empirically and report the results Table 2. Our experiments indicate that setting λ_p to be larger than λ often produces a higher quality preconditioner. In particular, $\lambda_p = 10\lambda$ seems like a reasonable rule-of-thumb for setting λ_p. We leave a theoretical analysis of this scheme to future work.

In Table 3 we compare our method to a high-performance distributed block ADMM-based solver using the random features method [6]. We use the same resource configuration as in Table 1. (We remark that the ADMM solver is rather memory efficient and can function with fewer resources.) The ADMM solver is more versatile in the choice of objective function, so we use hinge-loss (SVM). We use the same bandwidth (σ) and reguarization parameter (λ) as in Table 1. In general we set the number of random features (s) to be equal 25% of the dataset size. We clearly see that our method achieves better error rates, usually with better running times.

In Table 4 we examine whether the preconditioner indeed improves convergence and running time. We set the maximum iterations to 1000 and declare failure if the solver failed to converge to 10^{-3} tolerance for classification and 10^{-5} for regression. We remark the following on the items labeled FAIL:

- For MNIST-200K, without preconditioning CG failed to convergence but the error rate of the final model was just as good as our method.
- For MNIST-300K the error rate of the final model deteriorated to 1.33% (compare to 0.92%).
- For YEARMSD the final residual was 6.63×10^{-4} and the error deteriorated to 5.25×10^{-3} (compare to 4.58×10^{-3}).

Almost always (with two exception, one of them a tiny dataset) our method was faster than the nonpreconditioned method. More importantly our method is much more robust: the nonpreconditioned algorithm failed in some cases.

In Table 5 we examine how classification generalization quality is affected by choosing a more relaxed tolerance criteria. In general, setting tolerance to 10^{-3} is just barely better than 10^{-2} in terms of test error. Only in one case (MNIST) is the difference bigger than 0.01%. Running time for 10^{-3} is worse by a small factor and the results are almost the same. We set the tolerance to 10^{-3} to be consistent with the notion of exploiting data to the fullest, although in practice 10^{-2} seems to be sufficient.

7. Conclusions. Kernel ridge regression is a powerful nonparametric technique whose solution has a closed form that involves the solution of a linear system and thus is amenable to applying advanced numerical linear algebra techniques. A naive method for solving this system is too expensive to be realistic beyond "small data." In this paper we propose an algorithm that solves this linear system to high accuracy using a combination of sketching and preconditioning. Under certain conditions, the

TABLE 1
Running time and resources on EC2. The datasets in the top part are classification datasets, and on the bottom they are regression datasets.

Dataset	n	d	Parameters	Resources	s	Iterations	Time (sec)	Error rate
GISETTE	6,000	5,000	$\sigma = 3, \lambda = 0.01$	1 c4.large	500	6	8.1	3.50%
ADULT	32,561	123	$\sigma = 8, \lambda = 0.01$	1 c4.4xlarge	5,000	13	19.8	14.99%
IJCNN1	49,990	22	$\sigma = 0.3, \lambda = 0.01$	1 c4.8xlarge	12,500	120	85.1	1.39%
MNIST	60,000	780	$\sigma = 8.5, \lambda = 0.01$	1 c4.8xlarge	10,000	85	115	1.33%
MNIST-400K	400,000	780	$\sigma = 8.5, \lambda = 0.01$	8 r3.8xlarge	35,000	99	1580	0.89%
MNIST-1M	1,000,000	780	$\sigma = 8.5, \lambda = 0.01$	42 r3.8xlarge	35,000	189	3820	0.72%
EPSILON	400,000	2,000	$\sigma = 8, \lambda = 0.01$	8 r3.8xlarge	20,000	12	526	10.21%
COVTYPE	464,809	54	$\sigma = 0.1, \lambda = 0.01$	8 r3.8xlarge	55,000	555	6180	4.13%
YEARMSD	463,715	90	$\sigma = 3, \lambda = 0.001$	8 r3.8xlarge	15,000	15	289	4.58×10^{-3}

TABLE 2
Running time and resources on EC2 when using different λ in the preconditioner.

Dataset	Parameters	Resources	Precond λ (λ_p)	s	Iterations	Time (sec)
GISETTE	$\sigma = 3, \lambda = 0.01$	1 c4.large	0.1	500	6	8.1
ADULT	$\sigma = 8, \lambda = 0.01$	1 c4.4xlarge	0.1	5,000	17	20.7
IJCNN1	$\sigma = 0.3, \lambda = 0.01$	1 c4.8xlarge	0.1	10,000	52	55.5
MNIST	$\sigma = 8.5, \lambda = 0.01$	1 c4.8xlarge	0.1	10,000	37	76.3
MNIST-400K	$\sigma = 8.5, \lambda = 0.01$	8 r3.8xlarge	0.1	40,000	42	1060
MNIST-1M	$\sigma = 8.5, \lambda = 0.01$	42 r3.8xlarge	0.1	40,000	77	1210
EPSILON	$\sigma = 8, \lambda = 0.01$	8 r3.8xlarge	0.1	20,000	18	547
COVTYPE	$\sigma = 0.1, \lambda = 0.01$	8 r3.8xlarge	0.2	40,000	206	2960
YEARMSD	$\sigma = 3, \lambda = 0.001$	8 r3.8xlarge	0.01	15,000	20	289

TABLE 3
Comparison to high-performance random features solver based on block ADMM [6].

Dataset	Resources	ADMM - s	ADMM - time	ADMM - error	Proposed - time	Proposed - error
MNIST	1 c4.8xlarge	15,000	102	1.95%	115	1.33%
MNIST-400K	8 r3.8xlarge	100,000	1017	1.10%	1580	0.89%
EPSILON	8 r3.8xlarge	100,000	1823	11.58%	526	10.21%
COVTYPE	8 r3.8xlarge	115,000	6640	5.73%	6180	4.13%
YEARMSD	8 r3.8xlarge	115,000	958	5.01×10^{-3}	289	4.58×10^{-3}

TABLE 4
Comparison to CG without preconditioning. See text for details on cells marked FAIL.

Dataset	No precond - its	No precond - time	s	Proposed - its	Proposed - time
GISETTE	1	3.66	500	6	8.1
ADULT	369	52.9	5,000	13	19.1
IJCNN1	764	230	10,000	120	85.1
MNIST	979	500	10,000	85	115
MNIST-200K	FAIL	3890	25,000	90	1090
MNIST-300K	FAIL	5320	25,000	117	1400
EPSILON	194	854	20,000	12	526
COVTYPE	913	5111	55,000	555	6180
YEARMSD	FAIL	1900	15,000	15	289

TABLE 5

Comparison between setting tolerance to 10^{-2} *and* 10^{-3}.

Dataset	10^{-2} - its	10^{-2} - Error rate	10^{-3} - its	10^{-3} - Error rate
GISETTE	3	3.50	6	3.50%
ADULT	10	14.99%	13	14.99%
IJCNN1	69	1.38%	120	1.39%
MNIST	54	1.37%	85	1.33%
MNIST-200K	54	1.00%	90	0.99%
MNIST-300K	73	0.92%	117	0.92%
EPSILON	7	10.21%	12	10.22%
COVTYPE	253	4.12%	555	4.13%

running time of our algorithm is somewhere between $O(n^2)$ and $O(n^3)$, depending on properties of the data, kernel, and feature map. Empirically, it often behaves like $O(n^2)$. As we show experimentally, our algorithm is highly effective on datasets with as many as one million training examples.

Obviously the main limitation of our algorithm is the $\Theta(n^2)$ memory requirement for storing the kernel matrices. There are a few ways in which our algorithm can be leveraged to allow learning on much larger datasets. One nonalgorithmic software-based idea is to use an out-of-core algorithm, i.e., use SSD storage, or even a magnetic drive, to hold the kernel matrix. From an algorithmic perspective there are quite a few options. One idea is to use boosting to design a model that is an ensemble of a several smaller models based on nonuniform sampling of the data. Huang et al. recently showed that this can be highly effective in the context of kernel ridge regression [21]. Another idea is to use our solver as the block solver in the block coordinate descent algorithm suggested by Tu et al. [33]. We leave the exploration of these techniques to future work.

More importantly, one should note that even in the era of Big Data, it is not always the case that for a particular problem we have access to a very big training set. In such cases it is even more important to fully leverage the data and produce the best possible model. Our algorithm and implementation provide an effective way to do so.

Appendix A. Proof of Theorem 5.

PROPOSITION 8. *Let* $\mathbf{K} + \lambda \mathbf{I}_n = \mathbf{L}\mathbf{L}^{\mathrm{T}}$ *be a Cholesky decomposition of* $\mathbf{K} + \lambda \mathbf{I}_n$. *Let* $\mathbf{M} = \mathbf{L}^{-1}$. *Let* $\mathcal{Q} = (\mathbf{q}_1, \ldots, \mathbf{q}_n) \subset \mathcal{H}_k$ *defined by* $\mathbf{q}_i = \sum_{j=1}^{n} \mathbf{M}_{ij} k(\mathbf{x}_j, \cdot)$. *We have*

$$\mathbf{K}(\mathcal{Q}, \mathcal{Q}) + \lambda \mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}} = \mathbf{I}_n .$$

Proof. Since $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = (k(\mathbf{x}_i, \cdot), k(\mathbf{x}_j, \cdot))_{\mathcal{H}_k}$ and inner products are bilinear, we have $\mathbf{K}(\mathcal{Q}, \mathcal{Q}) = \mathbf{L}^{-1}\mathbf{K}\mathbf{L}^{-\mathrm{T}}$. So,

$$\mathbf{I}_n = \mathbf{L}^{-1}(\mathbf{K} + \lambda \mathbf{I}_n)\mathbf{L}^{-\mathrm{T}} = \mathbf{K}(\mathcal{Q}, \mathcal{Q}) + \lambda \mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}} .$$

∎

PROPOSITION 9. *Under the conditions of the previous proposition,*

$$\mathbf{Tr}\left(\mathbf{K}(\mathcal{Q}, \mathcal{Q})\right) = s_\lambda(\mathbf{K}) .$$

Proof. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of $\mathbf{K}$.

$$\begin{aligned}
\mathbf{Tr}\left(\mathbf{K}(\mathcal{Q},\mathcal{Q})\right) &= \mathbf{Tr}\left(\mathbf{I}_n - \lambda \mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}}\right) \\
&= n - \lambda \mathbf{Tr}\left(\mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}}\right) \\
&= n - \lambda \mathbf{Tr}\left((\mathbf{K}+\mathbf{I}_n)^{-1}\right) \\
&= n - \sum_{i=1}^{n} \frac{\lambda}{\lambda_i + \lambda} \\
&= \sum_{i=1}^{n} \frac{\lambda_i}{\lambda_i + \lambda} \\
&= \mathbf{Tr}\left((\mathbf{K}+\lambda I_n)^{-1}\mathbf{K}\right) .
\end{aligned}$$

∎

Proof of Theorem 5. We prove that with probability of at least $1-\delta$

$$\frac{2}{3}\left(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n\right) \preceq \mathbf{K} + \lambda \mathbf{I}_n \preceq 2\left(\mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n\right) , \tag{9}$$

or equivalently,

$$\frac{1}{2}(\mathbf{K} + \lambda \mathbf{I}_n) \preceq \mathbf{Z}\mathbf{Z}^{\mathrm{T}} + \lambda \mathbf{I}_n \preceq \frac{3}{2}(\mathbf{K} + \lambda \mathbf{I}_n) . \tag{10}$$

Thus, with probability of $1-\delta$ the relevant condition number is bounded by 3. For PCG, if the condition number is bounded by κ, we are guaranteed to reduce the error (measured in the matrix norm of the linear equation) to an ϵ fraction of the initial guess after $\lceil \sqrt{\kappa} \ln(2/\epsilon)/2 \rceil$ iterations [31]. This immediately leads to the bound in the theorem statement.

Let $\mathbf{K} + \lambda \mathbf{I}_n = \mathbf{L}\mathbf{L}^{\mathrm{T}}$ be a Cholesky decomposition of $\mathbf{K} + \lambda \mathbf{I}_n$. Let $\mathbf{M} = \mathbf{L}^{-1}$. Let $\mathcal{Q} = (\mathbf{q}_1, \ldots, \mathbf{q}_n) \subset \mathcal{H}_k$ defined by $\mathbf{q}_i = \sum_{j=1}^{n} \mathbf{M}_{ij} k(\mathbf{x}_j, \cdot)$. Let $\mathbf{Z}_{\mathcal{Q}} \in \mathbb{R}^{n \times s}$ be the matrix whose row i is equal to $\varphi(k(\mathbf{x}_i, \cdot))$. Due to the linearity of φ, we have $\varphi(\mathbf{q}_i) = \sum_{j=1}^{n} \mathbf{M}_{ij} \varphi(k(\mathbf{x}_j, \cdot)) = \sum_{j=1}^{n} \mathbf{M}_{ij} \mathbf{z}_j$ so $\mathbf{Z}_{\mathcal{Q}} = \mathbf{L}^{-1}\mathbf{Z}$.

It is well known that for $\mathbf{C}$ that is square and invertible $\mathbf{A} \preceq \mathbf{B}$ if and only if $\mathbf{C}^{-1}\mathbf{A}\mathbf{C}^{-T} \preceq \mathbf{C}^{-1}\mathbf{B}\mathbf{C}^{-T}$. Applying this to the previous equation with $\mathbf{C} = \mathbf{L}$ yields

$$\frac{1}{2}\mathbf{I}_n \preceq \mathbf{Z}_{\mathcal{Q}}\mathbf{Z}_{\mathcal{Q}}^{\mathrm{T}} + \lambda \mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}} \preceq \frac{3}{2}\mathbf{I}_n . \tag{11}$$

A sufficient condition for (11) to hold is that

$$\left\| \mathbf{Z}_{\mathcal{Q}}\mathbf{Z}_{\mathcal{Q}}^{\mathrm{T}} + \lambda \mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}} - \mathbf{I}_n \right\|_2 \leq \frac{1}{2} . \tag{12}$$

According to Proposition 8 we have

$$\left\| \mathbf{Z}_{\mathcal{Q}}\mathbf{Z}_{\mathcal{Q}}^{\mathrm{T}} + \lambda \mathbf{L}^{-1}\mathbf{L}^{-\mathrm{T}} - \mathbf{I}_n \right\|_2 = \left\| \mathbf{Z}_{\mathcal{Q}}\mathbf{Z}_{\mathcal{Q}}^{\mathrm{T}} - \mathbf{K}(\mathcal{Q},\mathcal{Q}) \right\|_2 .$$

Now, the approximate multiplication property along with the requirement that $s \geq f(s_\lambda(\mathbf{K})^{-1}/2, 0, \delta)$ or $s \geq f(s_\lambda(\mathbf{K})^{-1}/2\sqrt{2}, 1/2\sqrt{2}, \delta)$ guarantees that with probability of at least $1-\delta$ we have

$$\left\| \mathbf{Z}_{\mathcal{Q}}\mathbf{Z}_{\mathcal{Q}}^{\mathrm{T}} - \mathbf{K}(\mathcal{Q},\mathcal{Q}) \right\|_2 \leq \left\| \mathbf{Z}_{\mathcal{Q}}\mathbf{Z}_{\mathcal{Q}}^{\mathrm{T}} - \mathbf{K}(\mathcal{Q},\mathcal{Q}) \right\|_F \leq \frac{1}{2} .$$

Now complete the proof using the equality $\mathbf{Tr}\left(\mathbf{K}(\mathcal{Q},\mathcal{Q})\right) = s_\lambda(\mathbf{K})$ (Proposition 9). ∎

REFERENCES

[1] N. Ailon and E. Liberty, *Fast dimension reduction using Rademacher series on dual BCH codes*, Discrete Comput. Geom., 42 (2008), pp. 615–630, https://doi.org/10.1007/s00454-008-9110-x.

[2] A. E. Alaoui and M. W. Mahoney, *Fast randomized kernel ridge regression with statistical guarantees*, in Proceedings of Neural Information Processing Systems (NIPS), 2015.

[3] H. Avron, K. L. Clarkson, and D. P. Woodruff, *Sharper bounds for regularized data fitting*, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, Schloss Dagstuhl, Dagstuhl, Germany, 2017, pp. 27:1–27:22.

[4] H. Avron, P. Maymounkov, and S. Toledo, *Blendenpik: Supercharging LAPACK's least-squares solver*, SIAM J. Sci. Comput., 32 (2010), pp. 1217–1236, https://doi.org/10.1137/090767911.

[5] H. Avron, H. Nguyen, and D. Woodruff, *Subspace embeddings for the polynomial kernel*, Advances in Neural Information Processing Systems 27 (NIPS 2014).

[6] H. Avron and V. Sindhwani, *High-performance kernel machines with implicit distributed optimization and randomization*, Technometrics, 58 (2016), pp. 341–349, https://doi.org/10.1080/00401706.2015.1111261.

[7] F. R. Bach, *Sharp analysis of low-rank kernel matrix approximations*, in Proceedings of the 26th Annual Conference on Learning Theory (COLT), Princeton, NJ, 2013, pp. 185–209.

[8] G. Blanchard and N. Krämer, *Optimal learning rates for kernel conjugate gradient regression*, Advances in Neural Information Processing Systems 23 (NIPS 2010).

[9] L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, *Large-Scale Kernel Machines (Neural Information Processing)*, MIT Press, Cambridge, MA, 2007.

[10] M. Charikar, K. Chen, and M. Farach-Colton, *Finding frequent items in data streams*, Theoret. Comput. Sci., 312 (2004), pp. 3–15.

[11] J. Chen, H. Avron, and V. Sindhwani, *Hierarchically compositional kernels for scalable nonparametric learning*, J. Mach. Learn. Res., 18 (2017), 66.

[12] J. Chen, L. Wang, and M. Anitescu, *A fast summation tree code for matérn kernel*, SIAM J. Sci. Comput., 36 (2014), pp. A289–A309, https://doi.org/10.1137/120903002.

[13] J. Chen, L. Wu, K. Audhkhasi, B. Kingsbury, and B. Ramabhadrari, *Efficient one-vs-one kernel ridge regression for speech recognition*, in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2016, pp. 2454–2458, https://doi.org/10.1109/ICASSP.2016.7472118.

[14] M. B. Cohen, J. Nelson, and D. P. Woodruff, *Optimal approximate matrix product in terms of stable rank*, in Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, 2016, pp. 11:1–11:14, https://doi.org/10.4230/LIPIcs.ICALP.2016.11.

[15] K. Cutajar, M. Osborne, J. Cunningham, and M. Filippone, *Preconditioning kernel matrices*, in Proceedings of the 33nd International Conference on Machine Learning (ICML), New York, 2016, pp. 2529–2538; also available online from http://jmlr.org/proceedings/papers/v48/cutajar16.html.

[16] B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M.-F. Balcan, and L. Song, *Scalable kernel methods via doubly stochastic gradients*, Advances in Neural Information Processing Systems 27 (NIPS 2014).

[17] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós, *Faster least squares approximation*, Numer. Math., 117 (2011), pp. 219–249, https://doi.org/10.1007/s00211-010-0331-6.

[18] G. Fung and O. L. Mangasarian, *Proximal support vector machine classifiers*, in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2001.

[19] A. Gittens and M. Mahoney, *Revisiting the Nyström method for improved large-scale machine learning*, in Proceedings of the 30th International Conference on Machine Learning (ICML), Atlanta, GA, 28 (2013), pp. 567–575.

[20] R. Hamid, Y. Xiao, A. Gittens, and D. DeCoste, *Compact random feature maps*, in Proceedings of the 31st International Conference on Machine Learning (ICML), Beijing, China, 32 (2014), pp. 19–27.

[21] P. Huang, H. Avron, T. Sainath, V. Sindhwani, and B. Ramabhadran, *Kernel methods match deep neural networks on TIMIT*, in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2014, pp. 205–209.

[22] S. Kumar, M. Mohri, and A. Talwalkar, *Sampling methods for the Nyström method*, J. Mach. Learn. Res., 13 (2012), pp. 981–1006.

[23] Z. Lu, A. May, K. Liu, A. Bagheri Garakani, D. Guo, A. Bellet, L. Fan, M. Collins, B. Kingsbury, M. Picheny, and F. Sha, *How to Scale Up Kernel Methods to Be as Good as Deep Neural Nets*, ArXiv e-prints, 2014, https://arxiv.org/abs/1411.4000.

[24] X. Meng, M. A. Saunders, and M. W. Mahoney, *LSRN: A parallel iterative solver for strongly over- or underdetermined systems*, SIAM J. Sci. Comput., 36 (2014), pp. C95–C118, https://doi.org/10.1137/120866580.

[25] V. I. Morariu, B. V. Srinivasan, V. C. Raykar, R. Duraiswami, and L. S. Davis, *Automatic online tuning for fast Gaussian summation*, Advances in Neural Information Processing Systems 21 (NIPS 2008).

[26] R. Pagh, *Compressed matrix multiplication*, ACM Trans. Comput. Theory, 5 (2013), pp. 9:1–9:17, https://doi.org/10.1145/2493252.2493254.

[27] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero, *Elemental: A new framework for distributed memory dense matrix computations*, ACM Trans. Math. Software, 39 (2013), 13, https://doi.org/10.1145/2427023.2427030.

[28] A. Rahimi and B. Recht, *Random features for large-scale kernel machines*, Advances in Neural Information Processing Systems 20 (NIPS 2007).

[29] V. C. Raykar and R. Duraiswami, *Fast large scale Gaussian process regression using approximate matrix-vector products*, in Proceedings of the Learning workshop, 2007.

[30] C. Saunders, A. Gammerman, and V. Vovk, *Ridge regression learning algorithm in dual variables*, in Proceedings of the 15th International Conference on Machine Learning (ICML), Madison, WI, (1998), pp. 515–521.

[31] J. R. Shewchuk, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, Technical report, Carnegie Mellon University, Pittsburgh, PA, 1994.

[32] B. V. Srinivasan, Q. Hu, N. A. Gumerov, R. Murtugudde, and R. Duraiswami, *Preconditioned Krylov solvers for kernel regression*, ArXiv e-prints, 2014, http://arxiv.org/abs/1408.1237.

[33] S. Tu, R. Roelofs, S. Venkataraman, and B. Recht, *Large scale kernel learning using block coordinate descent*, ArXiv e-prints, 2016, http://arxiv.org/abs/1602.05310.

[34] C. K. I. Williams and M. Seeger, *Using the Nyström method to speed up kernel machines*, Advances in Neural Information Processing Systems 13 (NIPS 2000).

[35] D. P. Woodruff, *Sketching as a tool for numerical linear algebra*, Found. Trends Theor. Comput. Sci., 10 (2014), pp. 1–157, https://doi.org/10.1561/0400000060.

[36] J. Yang, X. Meng, and M. W. Mahoney, *Implementing randomized matrix algorithms in parallel and distributed environments*, Proc. IEEE, 104 (2016), pp. 58–92, https://doi.org/10.1109/JPROC.2015.2494219.

[37] Y. Yang, M. Pilanci, and M. J. Wainwright, *Randomized sketches for kernels: Fast and optimal nonparametric regression*, Ann. Statist., 45 (2017), pp. 991–1023, https://doi.org/10.1214/16-AOS1472.

[38] T. Zhang, *Learning bounds for kernel regression using effective data dimensionality*, Neural Comput., 17 (2005), pp. 2077–2098, https://doi.org/10.1162/0899766054323008.