Communication Efficient Distributed Kernel Principal Component Analysis

Maria-Florina Balcan*
School of Computer Science
Carnegie Mellon University
ninamf@cs.cmu.edu

Yingyu Liang
Computer Science
Princeton University
yingyul@cs.princeton.edu

Le Song
College of Computing
Georgia Institute of Technology
lsong@cc.gatech.edu

David Woodruff
Almaden Research Center
IBM Research
dpwoodru@us.ibm.com

Bo Xie
College of Computing
Georgia Institute of Technology
bo.xie@gatech.edu

ABSTRACT

Kernel Principal Component Analysis (KPCA) is a key machine learning algorithm for extracting nonlinear features from data. In the presence of a large volume of high dimensional data collected in a distributed fashion, it becomes very costly to communicate all of this data to a single data center and then perform kernel PCA. Can we perform kernel PCA on the entire dataset in a distributed and communication efficient fashion while maintaining provable and strong guarantees in solution quality?

In this paper, we give an affirmative answer to the question by developing a communication efficient algorithm to perform kernel PCA in the distributed setting. The algorithm is a clever combination of subspace embedding and adaptive sampling techniques, and we show that the algorithm can take as input an arbitrary configuration of distributed datasets, and compute a set of global kernel principal components with relative error guarantees independent of the dimension of the feature space or the total number of data points. In particular, computing k principal components with relative error ϵ over s workers has communication cost $\tilde{O}(s\rho k/\epsilon + sk^2/\epsilon^3)$ words, where ρ is the average number of nonzero entries in each data point. Furthermore, we experimented the algorithm with large-scale real world datasets. The experimental results showed that the algorithm produces a high quality kernel PCA solution while using significantly less communication than alternative approaches.

Keywords

Kernel method; Principal Component Analysis; distributed computing

*The authors are listed in alphabetical order.

KDD '16, August 13-17, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2939672.2939796

1. INTRODUCTION

Kernel Principal Component Analysis (KPCA) is a key machine learning algorithm for extracting nonlinear features from complex datasets, such as image, text, healthcare and biological data [27, 26, 28]. The original kernel PCA algorithm is designed for a batch setting, where all data points need to fit into a single machine. However, nowadays large volumes of data are being collected increasingly in a distributed fashion, which poses new challenges for running kernel PCA. For instance, a large network of distributed sensors can collect temperature readings from geographically distant locations; a system of distributed data centers in an Internet company can process user queries from different countries; a fraud detection system in a bank needs to perform credit checks on people opening accounts from different branches; and a network of electronic healthcare systems can store patient records from different hospitals. It is very costly in terms of network bandwidth and transmission delays to communicate all of the data collected in a distributed fashion to a single data center, and then run kernel PCA on the central node. In other words, communication now becomes the bottleneck to the nonlinear feature extraction pipeline. How can we leverage the aggregated computing power in a large distributed system? Can we perform kernel PCA on the entire dataset in a distributed and communication efficient fashion while maintaining provable and strong guarantees in solution quality?

While recent work shows how to do linear PCA in a communication efficient and distributed fashion [8], the kernel setting is significantly more challenging. The main problem with previous work is that it achieves communication proportional to the dimension of the data points, which if implemented straightforwardly in the kernel setting would give communication proportional to the dimension of the feature space which can be very large or even infinite. Kernel PCA uses the kernel trick to avoid going to the potentially infinite dimensional kernel feature space explicitly, so intermediate results are often represented by a function (*e.g.*, a weighted combination) of the feature mapping of some data points. Communicating such intermediate results requires communicating all the data points they depend on. To lower the communication, the intermediate results should only depend on a small number of data points. A distributed algorithm then needs to be carefully designed to meet this constraint.

In this paper, we propose a communication efficient algorithm for distributed KPCA in a master-worker setting where the dataset is arbitrarily partitioned and each portion sits in one worker, and the workers can communicate only through the master. Our key idea is to design a communication efficient way of generating a small representative subset of the data, and then performing kernel PCA based on this subset. We show that the algorithm can compute a rank-k subspace in the kernel feature space using just a representative subset of size $O(k/\epsilon)$ built in a distributed fashion. For polynomial kernels, it achieves a $(1+\epsilon)$ relative-error approximation to the best rank-k subspace, and for shift-invariant kernels (such as the Gaussian kernel), it achieves $(1+\epsilon)$-approximation with an additive error term that can be made arbitrarily small. In both cases, the total communication for a system of s workers is $\tilde{O}(s\rho k/\epsilon + sk^2/\epsilon^3)$ words, where ρ is the average number of nonzero entries in each data point, and is always bounded by the dimension of the data d and independent of the dimension of the kernel feature space. This for constant ϵ nearly matches the lower bound $\Omega(sdk)$ for linear PCA [8]. As far as we know, this is the first algorithm that can achieve provable approximation with such communication bounds.

As a subroutine of our algorithm, we have also developed an algorithm for the distributed Column Subset Selection (CSS) problem, which can select a set of $O(k/\epsilon)$ points whose span contains $(1+\epsilon)$-approximation, with communication $O(s\rho k/\epsilon + sk^2)$. This is the first algorithm that addresses the problem for kernels, and it nearly matches the communication lower bound $\Omega(s\rho k/\epsilon)$ for this problem in the linear case [10]. The column subset selection problem has various applications in big data scenarios, so this result could be of independent interest.

Furthermore, our algorithm also leads to some other distributed kernel algorithms: the data can then be projected onto the subspace found and processed by downstream applications. For example, an immediate application is for distributed spectral clustering, that first computes KPCA to rank-k/ϵ and then does k-means on the data projected on the subspace found by KPCA (*e.g.*, [17]). This can be done by combining our algorithm with any efficient distributed k-means algorithms (*e.g.*, [6]).

We evaluate our algorithm on datasets with millions of data points and hundreds of thousands of dimensions where non-distributed algorithms such as batch KPCA are impractical to run. Furthermore, comparing to other distributed algorithms, our algorithm requires less communication and fewer representation data points to achieve the same approximation error.

2. RELATED WORK

There has been a surge of recent work on distributed machine learning, *e.g.*, [5, 30, 20, 6]. In this setting, the data sets are typically large, and small error rate is required. This is because if only a coarse error is needed then there is no need to use large-scale data sets; a small subset of the data will be sufficient. Furthermore, one prefers relative error rates instead of additive error rates, since the latter is worse and harder to interpret without knowing the optimum. Our algorithm can achieve small relative error with limited communication.

Since there exist communication efficient distributed linear PCA algorithms [6, 20], it is tempting to adopt the random feature approach for distributed kernel PCA: first construct m random features and then solve PCA in the primal form, *i.e.*, apply distributed linear PCA on the random features. However, the communication of this method is too high. One needs $m = \tilde{O}(d/\epsilon^2)$ random features to preserve the kernel values up to additive error ϵ, leading to a communication of $O(skm/\epsilon) = O(skd/\epsilon^3)$. Another drawback of using random features is that it only produces a solution in the space spanned by the random features, but not a solution in the feature space of the kernel.

The Nyström method is another popular tool for large-scale kernel methods: sample a subset of data points uniformly at random, and use them to construct an approximation of the original kernel matrix. However, it also suffers from high communication cost, since one needs $O(1/\epsilon^4)$ sampled points to achieve additive ϵ error in the Frobenius norm of the kernel matrix [21]. A closely related method is incomplete Cholesky decomposition [3], where a few pivots are greedily chosen to approximate the kernel matrix. It is unclear how to design a communication efficient distributed version since it requires as many rounds of communication as the number of pivots, which is costly.

Leverage score sampling is a related technique for low-rank approximation [29]. A prior work of Boutsidis et al. [8] gives the first distributed protocol for column subset selection. [11] gives a distributed PCA algorithm with optimal communication cost, but only for linear PCA. In comparison, our work is the first communication efficient distributed algorithm for low rank approximation in the kernel space.

3. BACKGROUNDS

For any vector v, let $\|v\|$ denote its Euclidean norm. For any matrix $M \in \mathbb{R}^{d\times n}$, let $M_{i:}$ denote its i-th row and $M_{:j}$ its j-th column. Let $\|M\|_F$ denote its Frobenius norm, and $\|M\|_2$ denote its spectral norm. Let its rank be $r \le \min\{n,d\}$, and denote its SVD as $M = U\Sigma V^\top$ where $U \in \mathbb{R}^{d\times r}, \Sigma \in \mathbb{R}^{r\times r}$, and $V \in \mathbb{R}^{n\times r}$. Let $[M]_k$ denote its best rank-k approximation. Finally, denote its number of nonzero entries as $\mathrm{nnz}(M)$.

In the distributed setting, there are s workers that are connected to a master processor. Worker i has a local data set $A^i \in \mathbb{R}^{d\times n_i}$, and the global data set $A \in \mathbb{R}^{d\times n}$ is the concatenation of the local data ($n = \sum_{i=1}^s n_i$).

Kernels and Random Features. For a kernel $\kappa(x,x')$, let $\mathcal{H}$ denote its feature space, i.e., there exists a feature mapping $\phi(\cdot) \in \mathcal{H}$ such that $\kappa(x,x') = \langle \phi(x), \phi(x')\rangle_{\mathcal{H}}$. Let $\phi(A) \in \mathcal{H}^n$ denote the matrix obtained by applying ϕ on each column of A and concatenating the results. Throughout the paper, we regard any $M \in \mathcal{H}^n$ as a matrix whose columns are elements in $\mathcal{H}$ and define matrix operations accordingly. For example, for any $M \in \mathcal{H}^n$ and $N \in \mathcal{H}^m$, let $B = M^\top N \in \mathbb{R}^{n\times m}$ where $B_{ij} = \langle M_{:i}, N_{:j}\rangle_{\mathcal{H}}$, and let $\|M\|_{\mathcal{H}}^2 = \mathrm{tr}\left(M^\top M\right)$. When there is no ambiguity, we omit the subscript $\mathcal{H}$.

The random feature approach is a recent technique to scale up kernel methods. Many kernels can be approximated by $\frac{1}{m}\sum_{i=1}^m \xi_{\omega_i}(x)\xi_{\omega_i}(y)$ where ω_i's are randomly sampled. These include Gaussian RBF kernels and other shift-invariant kernels, inner product kernels, etc ([24, 15]). For example, Gaussian RBF kernels, $\kappa(x,y) = \exp(-\|x-y\|^2/2\sigma^2)$, can be approximated by $\frac{1}{m}\sum_{i=1}^m z_{\omega_i,b_i}(x) z_{\omega_i,b_i}(y)$ where $z_{\omega,b}(x) = \sqrt{2}\cos(\omega^\top x + b)$ and ω_i is from a Gaussian

distribution with density proportional to $\exp(-\sigma^2 \|\omega\|^2/2)$ and b_i is uniform over $[0, 2\pi]$.

In this paper, we provide guarantees for shift-invariant kernels using Fourier random features (the extension to other kernels/random features is straightforward). We assume the kernel satisfies some regularization conditions: it is defined over bounded compact domain in $\mathbb{R}^d$, with $\kappa(0) \leq 1$ and bounded $\nabla^2 k(0)$ [24]. Such conditions are standard in practice, and thus we assume them throughout the paper.

Kernel PCA. An element $u \in \mathcal{H}$ is an eigenfunction of $\phi(A)\phi(A)^\top$ with the corresponding eigenvalue λ if $\|u\| = 1$ and $\phi(A)\phi(A)^\top u = \lambda u$. Given eigenfunctions $\{u_i\}$ of $\phi(A)\phi(A)^\top$ and eigenvectors $\{v_i\}$ of $\phi(A)^\top\phi(A)$, $\phi(A)$ has the singular decomposition $U\Sigma_k V^\top + U_\perp \Sigma_\perp V_\perp^\top$, where U, V are the lists of top k eigenfunctions/vectors, Σ_k is a diagonal matrix with the corresponding singular values, $U_\perp$, $V_\perp$ are the lists of the rest of the eigenfunctions/vectors, and $\Sigma_\perp$ is a diagonal matrix with the rest of the singular values. Kernel PCA aims to identify the top k subspace U, since the best rank-k approximation $[\phi(A)]_k = U\Sigma_k V^\top = UU^\top\phi(A)$. Typically, the goal is to find a good approximation to this subspace. Formally,

DEFINITION 1. *A subspace $L \in \mathcal{H}^k$ is a rank-k $(1+\epsilon, \Delta)$-approximation for kernel PCA on A if $L^\top L = I_k$ and*

$$\left\|\phi(A) - LL^\top\phi(A)\right\|^2 \leq (1+\epsilon)\left\|\phi(A) - [\phi(A)]_k\right\|^2 + \Delta.$$

Kernel PCA leads to immediate solutions for some other nonlinear component analysis (e.g., kernel CCA), and provides needed subroutines for tasks like spectral clustering.

Subspace Embeddings. Subspace embeddings are a useful technique that can improve the computational and space costs by embedding data into lower dimension while preserving interesting properties. They have been extensively studied in recent years [25, 1, 13]. The recent fast sparse subspace embeddings [13] and its optimizations [22, 23] are particularly suitable for large-scale sparse datasets, since their running time is linear in the number of non-zero entries in the data matrix. They also preserve the sparsity of the input data. Formally,

DEFINITION 2. *An ϵ-subspace embedding of $M \in \mathbb{R}^{m\times n}$ is a matrix $S \in \mathbb{R}^{t\times m}$ such that for any x,*

$$\|SMx\| = (1 \pm \epsilon)\|Mx\|.$$

Subspace embeddings can also be done on the right hand side, i.e., *$S \in \mathbb{R}^{n\times t}$ and $\|x^\top MS\| = (1\pm\epsilon)\|x^\top M\|$.*

Mx is in the column space of M and SMx is its embedding, so the definition means that the norm of any vector in the column space of M is approximately preserved. This then provides a way to do dimensional reduction for problems depending on inner products of vectors. Our algorithm repeatedly makes use of subspace embeddings. In particular, the embedding we use is the concatenation of the following known sketching matrices: CountSketch and i.i.d. Gaussians (or the concatenation of CountSketch, fast Hadamard and i.i.d. Gaussians). The details can be found in [29]; we only need the following fact.

LEMMA 1. *For $M \in \mathbb{R}^{d\times n}$, there exist sketching matrices $S \in \mathbb{R}^{t\times d}$ with $t = O(n/\epsilon^2)$ that are ϵ-subspace embeddings. Furthermore, SM can be successfully computed in time $\tilde{O}(\mathrm{nnz}(M))$ with probability at least $1-\delta$.*

The work of [2] shows that a fast computational approach, TensorSketch, is indeed a subspace embedding for the polynomial kernel. However, there are no previously known subspace embeddings for other kernels. We develop efficient and provable embeddings for a large family of kernels including Gaussian kernel and other shift invariant kernels. These embeddings will be a key tool used by our algorithm.

4. OVERVIEW

In view of the limitations of the related work, we instead take a different approach, which first selects a small subset of points whose span contains an approximation with relative error rate ϵ, and then find a low rank approximation in their span. It is important to keep the size of the subset small and also guarantee that their span contains a good approximation (this is also called kernel column subset selection). A well known technique is to sample according to the statistical leverage scores.

Challenges. However, this immediately raises the following technical challenges.

I. Computing the statistical leverage scores is prohibitively expensive. Naïvely computing them requires communicating all data points. There exist non-trivial fast algorithms [18], but they are designed for the non-distributed setting. Using them in the distributed setting leads to communication linear in the number of data points, or linear in the number of random features if one uses random features and computes the leverage scores for them.

Our key idea is that it is sufficient to compute the (generalized) leverage scores of the data points, i.e., the leverage scores of another matrix whose row space approximates that of the original data matrix. So the problem is reduced to designing kernel subspace embeddings that can approximate the row space of the data.

II. Even given the embedded data, it is unclear how to compute its leverage scores in a communication efficient way. Although the dimension of the embedded data is small, existing algorithms will lead to communication linear in the number of data points, which is impractical.

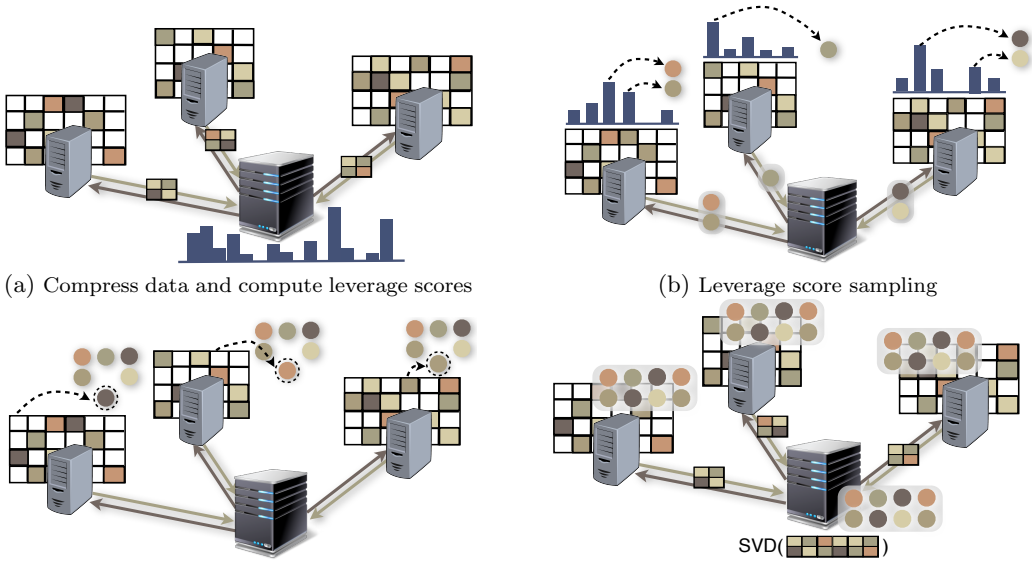
III. Simply sampling according to the generalized leverage scores does not give the desired results: a good approximation can only be obtained using a much larger rank, specifically, $O(k/\epsilon)$.

IV. After selecting the small subset of points, we need to design a distributed algorithm to compute a good low rank approximation in their span.

Algorithm. We have designed a distributed kernel PCA algorithm that computes an approximated solution with relative error rate ϵ using low communication. The algorithm operates in following key steps, each of which addresses one of the challenges mentioned above (See Figure 1):

I. Kernel Subspace Embeddings. To approximate the subspace of the original data matrix, we propose subspace embeddings for a large family of kernels. For polynomial kernels we improve the prior work by reducing the embedding dimension and thus lowering the communication. Furthermore, we propose new subspace embeddings for kernels with random feature expansions, allowing PCA for these kernels to be computed in a communication efficient manner. See Section 5.1 for the details.

II. Distributed Leverage Scores. To compute the leverage scores, sampling with constant approximations is sufficient. We can thus drastically reduce the number of data points:



(a) Compress data and compute leverage scores

(b) Leverage score sampling

(c) Adaptive sampling

(d) Project data and compute KPCA

Figure 1: Algorithm overview. The black machine at the center is the master and the gray machines are the workers. Each worker stores its portion of the dataset, and the algorithm computes the top k principle components on the whole dataset. The arrows between the machines denote the direction of communications. In each round, the communication always starts from the workers to the master (lighter arrows) and then from the master to the workers (darker arrows). (a) Each worker compresses its data by using (kernel) subspace embeddings and sends it to the master. The master aggregates the data and computes intermediate results for leverage scores and sends back to the workers. (b) Each worker computes the leverage scores, samples data points (denoted by circles) and then sends them to the master. The master distributes back the union of the sampled data points. (c) Each worker conducts adaptive sampling and sends newly sampled points to the master. The master distributes back the union of all sampled points. (d) Each worker projects its data onto the subspace spanned by the sampled data points and sends the compressed projections to the master. The master computes coefficients for the top k principle components by running SVD, and then sends them back to the workers. (best viewed in color)

first do another (non-kernel) subspace embeddings on the embedded data, and then send the result to the master for computing the scores. See Figure 1(a) for an illustration and Section 5.2 for the details.

III. Sampling Representative Points. We take a two-step approach as leverage scores alone is not good enough : first sample according to generalized leverage scores, and then sample additional points according to their distances to the span of the points sampled in the first step. The first step gains some coarse information about the data, and the second step use it to get the desired samples. The two steps are illustrated in Figure 1(b) and 1(c), respectively, while the details are in Section 5.3.

IV. Computing an Approximation. After projecting the data to the span of the representative points, we sketch the projections by (non-kernel) subspace embeddings. We then send the compressed projections to the master and compute the solution there. See Figure 1(d) for an illustration and Section 5.4 for the details.

Main Theoretical Results. Given as input the local datasets, the rank k and error parameters ϵ, Δ, our algorithm outputs a $(1+\epsilon, \Delta)$-approximation to the optimum with large probability. Formally, we have the following theorem.

THEOREM 1. *Algorithm 4 produces a subspace L for kernel PCA on A that with probability ≥ 0.99 satisfies:*

1. *L is a rank-k $(1+\epsilon, 0)$-approximation when applied to polynomial kernels.*
2. *L is a rank-k $(1+\epsilon, \Delta)$-approximation when applied to shift-invariant kernels with regularization.*

The total communication is $\tilde{O}(\frac{s\rho k}{\epsilon} + \frac{sk^2}{\epsilon^3})$ words, where ρ is the average number of nonzero entries in one data point.

The constant success probability can be boosted up to any high probability $1-\delta$ by repetition, which adds only an extra $O(\log \frac{1}{\delta})$ term to communication and computation.

The output subspace L is represented by $\tilde{O}(k/\epsilon)$ sampled points Y from A (*i.e.*, $L = \phi(Y)C$ for some coefficient matrix C), so L can be easily communicated and the projection of any point on L can be easily computed by the kernel trick. The communication has linear dependence on the dimension and the number of workers, and has no dependence on the number of data points, which is crucial for big data scenarios. Moreover, it does not depend on Δ (but the computation does), so the additive error can be made arbitrarily small with more computation.

The theorem also holds for other properly regularized kernels with random feature expansions (see [24, 15] for more such kernels); the extension of our proof is straightforward.

We also make the following contributions: (*i*) Subspace embedding techniques for many kernels. (*ii*) Distributed algorithm for computing generalized leverage scores with low communication. (*iii*) Distributed algorithm for kernel column subset selection.

5. DISTRIBUTED KERNEL PRINCIPAL COMPONENT ANALYSIS

Our algorithm first computes the (generalized) leverage scores that measure the non-uniform structure, then samples

a subset of points whose span contains a good approximated solution, and finally finds such a solution in the span.

Leverage scores are critical for importance sampling in many fast randomized algorithms. The leverage scores are defined as follows.

DEFINITION 3. *For $E \in \mathbb{R}^{t\times n}$ with SVD $E = U\Sigma V^\top$, the leverage score ℓ_j for its j-th column is $\ell_j = \|V_{j:}\|^2$.*

Their importance is reflected in the following fact: suppose E has rank at most k, and suppose P is a subset of $O(\frac{k\log k}{\epsilon^2})$ columns obtained by repeatedly sampled from the columns of E according to their leverage scores, then the span of P contains an $(1+\epsilon, 0)$-approximation subspace for E with probability ≥ 0.99 (see,*e.g.*, [19]). Here, sampling one column according to the leverage scores ℓ_j means to define sampling probabilities p_j such that $p_j \geq \frac{\ell_j}{4\sum_j \ell_j}$ for all j, and then pick one column where the j-th column is picked with probability p_j. Note that setting $p_j = \frac{\ell_j}{\sum_j \ell_j}$ is clearly sufficient, but a constant variance of p_j is allowed at the expense of an extra constant factor in the sample size. This means that it is sufficient to compute constant approximations $\tilde{\ell}_j$ for ℓ_j, and then sample according to $p_j = \frac{\tilde{\ell}_j}{\sum_j \tilde{\ell}_j}$.

However, even computing constant approximations of the leverage scores are non-trivial: naïve approaches require SVD, which is expensive. Actually, SVD is more expensive than the task of PCA itself. Even ignoring computation cost, naïve SVD is prohibitive in the distributed setting due to its high communication cost. Fortunately, it turns out that the leverage scores are an over kill for our purpose; it suffices to compute the generalized leverage scores, i.e., the leverage scores of a proxy matrix.

DEFINITION 4. *If E has rank q and can approximate the row space of M up to $(1+\epsilon, \Delta)$,* i.e.*, there exists X with*

$$\|XE - M\|_F \leq (1+\epsilon)\|M - [M]_k\|_F + \Delta,$$

then the leverage scores of E are called the generalized leverage scores of M with respect to rank q.

This generalizes the definition in [18] by allowing the rank of E to be larger than k and allowing additive error Δ, which are important for our application. The generalized leverage scores can act as the leverage scores for our purpose in the following sense.

LEMMA 2. *Let P be $O(\frac{q\log q}{\epsilon^2})$ columns sampled from M according to their generalized leverage scores w.r.t. rank q. Then with probability ≥ 0.99, the span of P has a rank-s $(1+2\epsilon, 2\Delta)$-approximation subspace for M.*

PROOF. It follows from combining Theorem 5 in [19] and the definition of the generalized leverage scores. □

Computing the generalized scores with respect to rank q could be much more efficient, since the intrinsic dimension now becomes q, which can be much smaller than the ambient dimension (the number of points or the dimension of the feature space). However, as noted in the overview, there are still a few technical challenges.

- Efficiently find a smaller matrix E that can approximate the row space of the original data.
- Compute the leverage scores of E in a communication efficient way.
- The approximation solution in the span of P has the same rank as E, which is $O(k/\epsilon)$ when we use kernel subspace embedding to obtain E. This is not satisfying since our final goal is to compute a rank-k solution.
- Find a good approximation in the span of $\phi(Y)$ with low communication.

Our final algorithm consists of four key steps, each of which addresses one of the above challenges. They are elaborated in the following four subsections respectively, and the final subsection presents the overall algorithm.

5.1 Kernel Subspace Embeddings

Recall that a subspace embedding S for a matrix M is such that $\|SMx\| \approx \|Mx\|$, *i.e.*, the norm of any vector in the column space of M is approximately preserved. Subspace embeddings can also be generalized for the feature mapping of kernels, simply by setting $M = \phi(A)$, S a linear mapping from $\mathcal{H} \mapsto \mathbb{R}^t$ and using the corresponding inner product. If the data after the kernel subspace embedding is sufficient for solving the problem under consideration, then only $S\phi(A)$ in much lower dimension is needed. This is especially interesting for distributed kernel methods, since directly using the feature mapping or the kernel trick in this setting will lead to high communication cost, while the data after embedding can be smaller and lead to lower communication cost.

A sufficient condition for solving many problems (in particular, kernel PCA) is to preserve the low rank structure of the data. More precisely, the row space of $S\phi(A)$ is a good approximation to that of $\phi(A)$, where the error is comparably to the best rank k approximation error. Then $S\phi(A)$ can be used to compute the generalized leverage scores for $\phi(A)$, which can then be utilized to compute kernel PCA as mentioned above.

More precisely, we would like $S\phi(A)$ to approximate the row space of $\phi(A)$ up to $(1+\epsilon, \Delta)$, as required in the definition of the generalized leverage scores. We give such embeddings a particular name.

DEFINITION 5. *S is called a $(1+\epsilon, \Delta)$-good subspace embedding for $\phi(A) \in \mathcal{H}^n$, if there exists X such that*

$$\|X(S\phi(A)) - \phi(A)\|^2 \leq (1+\epsilon)\|\phi(A) - [\phi(A)]_k\|^2 + \Delta.$$

We now identify the sufficient conditions for $(1+\epsilon, \Delta)$-good subspace embeddings, which can then be used in constructing such embeddings for various kernels.

LEMMA 3. *S is a $(1+\epsilon, \Delta)$-good subspace embedding for $\phi(A) \in \mathcal{H}^n$ if it satisfies the following.*

P1 *(Subspace Embedding): For any orthonormal $V \in \mathcal{H}^k$* (i.e.*, $V^\top V$ is the identity), for all $x \in \mathbb{R}^k$,*

$$\|SVx\| = (1 \pm c)\|Vx\|$$

where c is a sufficiently small constant.

P2 *(Approximate Product): for any $M \in \mathcal{H}^n, N \in \mathcal{H}^k$,*

$$\left\|(SN)^\top(SM) - N^\top M\right\|_F^2 \leq \frac{\epsilon}{k}\|N\|^2\|M\|^2 + \Delta.$$

Polynomial Kernels. For polynomial kernels, there exists an efficient algorithm TENSORSKETCH to compute the embedding [2]. However, the embedding dimension has a

Algorithm 1 Distributed Leverage Scores: $\{\tilde{\ell}_j^i\} = \mathbf{disLS}(\{E^i\}_{i=1}^s, k)$

1: Each worker i: do $\frac{1}{4}$-subspace embedding $E^iT^i \in \mathbb{R}^{t\times p}$ with $p = O(t)$; send E^iT^i to Master.
2: Master: QR-factorize $\left[E^1T^1, \ldots, E^sT^s\right]^\top = UZ$; send Z to all workers.
3: Each worker i: compute $\tilde{\ell}_j^i = \left\|\left((Z^\top)^{-1}E^i\right)_{:j}\right\|_2^2$.

quadratic dependence on the rank k, which will increase the communication. Fortunately, subspace embedding can be concatenated, so we can further apply another known subspace embedding such as one of those in Lemma 1 which, though not fast for feature mapping, is fast for the already embedded data and has lower dimension. In this way, we can enjoy the benefits of both approaches.

The guarantee of TensorSketch in [2] and the property of the subspace embeddings in Lemma 1 can be combined to verify **P1** and **P2**. So we have

Lemma 4. *For polynomial kernels $\kappa(x, y) = (\langle x, y\rangle)^q$, there exists a $(1+\epsilon, 0)$-good subspace embedding matrix $S : \mathbb{R}^{d^q} \mapsto \mathbb{R}^t$ with $t = O(k/\epsilon)$.*

Kernels with Random Feature Expansions. Polynomial kernels have finite dimensional feature mappings, for which the sketching seems natural. It turns out that it is possible to extend subspace embeddings to kernels with infinite dimensional feature mappings. More precisely, we propose subspace embeddings for kernels with random feature expansions, *i.e.*, $\kappa(x, y) = \mathbb{E}_\omega\left[\xi_\omega(x)\xi_\omega(y)\right]$ for some function $\xi(\cdot)$. Therefore, one can approximate the kernel by using m features $z_\omega(x)$ on randomly sampled ω. Such random feature expansion can be exploited for subspace embeddings: view the expansion as the "new" data points and apply a sketching matrix on top of it. Compared to polynomial kernels, the finite random feature expansion leads to an additional additive error term. Our analysis shows that bounding the additive error term only requires sufficiently large sampled size m, which affects the computation but does not affect the final embedding dimension and thus the communication.

In summary, the embedding is $S\phi(x) = TR(\phi(x))$, where $R(\phi(x)) \in \mathbb{R}^m$ is m random features for x and $T \in \mathbb{R}^{t\times m}$ is an embedding as in Lemma 1. The properties **P1** and **P2** can be verified by combining Lemma 1 and the guarantees of random features.

Lemma 5. *For a continuous shift-invariant kernels $\kappa(x, y) = \kappa(x - y)$ with regularization, there exists a $(1 + \epsilon, \Delta)$-good subspace embedding $S : \mathcal{H} \mapsto \mathbb{R}^t$ with $t = O(k/\epsilon)$.*

5.2 Computing Leverage Scores

Given the matrix E obtained from kernel subspace embedding, we would like to compute the leverage scores of E. First note that this cannot be done simply in a local manner: the leverage score of a column in E^i is different from the leverage score of the same column in E. Furthermore, though data in E have low dimension, communicating all points in E to the master is still impractical, since it leads to communication linear in the total number of points.

Fortunately, we only need to compute constant approximations of the scores, which allows us to use subspace embedding on E to greatly reduce the number of data points.

Algorithm 2 Sampling Representative Points: $Y = \mathbf{RepSample}(\{A^i\}_{i=1}^s, \{\tilde{\ell}_j^i\}, k, \epsilon)$

1: Workers: sample $O(k \log k)$ points according to $\{\tilde{\ell}_j^i\}$; send to Master;
2: Master: send all the sampled points P to the workers;
3: Workers: sample $O(k/\epsilon)$ points $\tilde{Y}$ according to the square distances to P in the feature space; send to Master;
4: Master: send $Y = \tilde{Y} \cup P$ to all the workers.

In particular, we apply a $\frac{1}{4}$-subspace embedding T^i (*e.g.*, one of those in Lemma 1) on each local data set E^i, and then send them to the master. Let ET denote all the embedded data, and do QR factorization $(ET)^\top = UZ$. Now, the rows of $U^\top = \left(Z^\top\right)^{-1} ET$ are a set of basis for ET. Then, think of $U^\top T^\dagger = \left(Z^\top\right)^{-1} E$ as the basis for E, so it suffices to compute the norms of the columns in $\left(Z^\top\right)^{-1} E$.

The details are described in Algorithm 1 and Figure 1(a) shows an illustration. The algorithm is guaranteed to output constant approximations of the leverage scores of E.

Lemma 6. *Let ℓ_j^i be the true leverage scores of E. Then Algorithm 1 outputs $\tilde{\ell}_j^i = (1 \pm 1/2)\ell_j^i$.*

Proof. The algorithm can be viewed as applying an embedding $T = \operatorname{diag}\left(T^1, \ldots, T^s\right)$ on E to approximate the scores while saving the costs. Each T^i is an $\frac{1}{4}$-subspace embedding matrix, then for any x,

$$\begin{aligned}\left\|x^\top ET\right\|^2 &= \left\|[x^\top E^1T^1, x^\top E^2T^2, \ldots, x^\top E^sT^s]\right\|^2 \\ &= \sum_{i=1}^s \left\|x^\top E^iT^i\right\|^2 = \sum_{i=1}^s (1 \pm 1/4)^2 \left\|x^\top E^i\right\|^2 \\ &= (1 \pm 1/4)^2 \left\|x^\top E\right\|^2.\end{aligned}$$

So T is also $\frac{1}{4}$-subspace embedding. Such a scheme of using embedding for approximating the scores has been analyzed (Lemma 6 in [18]), and the lemma follows. □

We note that though a constant approximation is sufficient for our purpose, but the algorithm can output $\tilde{\ell}_j^i = (1 \pm \epsilon)\ell_j^i$ by doing an $\frac{\epsilon}{2}$-subspace embedding (instead of $\frac{1}{4}$), which can be useful for other applications.

5.3 Sampling Representative Points

Sampling directly to the leverage scores can produce a set of points P such that the span of $\phi(P)$ contains a $(1+\epsilon, \Delta)$-approximation to $\phi(A)$. However, the rank of that approximation can be as high as $O(k/\epsilon)$, since its rank is the same as that of the embedded data (see Lemma 2), which will be $O(k/\epsilon)$ to achieve ϵ error. To get a rank-k approximation and also enjoy the advantage of leverage scores, we propose to combine leverage score sampling and the adaptive sampling algorithm in [16, 9].

The details are presented in Algorithm 2. We first sample a set P of $O(k \log k)$ points according to the leverage scores, so that the span of $\phi(P)$ contains a $(2, \Delta)$-approximation. Then we use the adaptive sampling method: sample $O(k/\epsilon)$ points according to the square distances from the points to their projections on P and then add them to P to get the

Algorithm 3 Computing an Approximation: $L = \mathbf{disLR}(\{A^i\}_{i=1}^s, Y, k, \epsilon, \Delta)$

1: Each worker i: compute the basis Q for $\phi(Y)$ and $\Pi^i = Q^\top \phi(A^i)$; do an ϵ-subspace embedding $\Pi^i T^i \in \mathbb{R}^{|Y| \times w}$ with $w = O(|Y|/\epsilon^2)$, and send $\Pi^i T^i$ to Master;
2: Master: concatenate $\Pi T = \left[\Pi^1 T^1, \ldots, \Pi^s T^s\right]$ and send the top k singular vectors W of ΠT to the workers.
3: Each worker i: set $L = QW$.

desire set Y of representative points. Figure 1(b) and 1(c) demonstrate the two steps of the algorithm.

Adaptive sampling has the following guarantee:

LEMMA 7. *Suppose there is a $(2, \Delta)$-approximation for $\phi(A)$ in the span of $\phi(P)$. Then with probability ≥ 0.99, the span of $\phi(Y)$ has a rank-k $(1+\epsilon, \Delta)$-approximation.*

Therefore, we solves the column subset selection problem for kernels in the distributed setting, with $O(k \log k + k/\epsilon)$ selected columns and with a communication of only $O(s\rho k/\epsilon + sk^2)$. This also provides the foundation for kernel PCA task.

5.4 Computing an Approximation

To compute a good approximation in the span of $\phi(Y)$, the naïve approach is to project the data to the span and compute SVD there. However, the communication will be linear in the number of data points. Subspace embedding can be used to sketch the projected data, so that the number of data points is greatly reduced.

Algorithm 4 describes the details and Figure 1(d) shows an illustration. To compute the best rank-k approximation for the projected data Π, we do a subspace embedding on the right hand side, *i.e.*, compute $\Pi T = \left[\Pi^1 T^1, \ldots, \Pi^s T^s\right]$. Then the algorithm computes the best rank-k approximation W for ΠT, which is then a good approximation for Π and thus $\phi(A)$. It then returns L, the representation of W in the coordinate system of $\phi(A)$. The output L is guaranteed to be a good approximation.

LEMMA 8. *If there is a rank-k $(1+\epsilon, \Delta)$-approximation subspace in the span of $\phi(Y)$, then*

$$\left\|LL^\top \phi(A) - \phi(A)\right\|^2 \leq (1+\epsilon)^2 \left\|\phi(A) - [\phi(A)]_k\right\|^2 + (1+\epsilon)\Delta.$$

Proof Sketch. For our choice of w, T^i is an ϵ-subspace embedding matrix for Π^i. Then their concatenation B is an ϵ-subspace embedding for Π, the concatenation of Π^i. Then we can apply the idea implicit in [20].

By Pythagorean Theorem, the error can be factorized into

$$\underbrace{\left\|LL^\top \phi(A) - QQ^\top \phi(A)\right\|^2}_{T1} + \underbrace{\left\|\phi(A) - QQ^\top \phi(A)\right\|^2}_{T2}.$$

Since $LL^\top = QWW^\top Q^\top$,

$$T1 = \left\|WW^\top Q^\top \phi(A) - Q^\top \phi(A)\right\|^2.$$

Note that $\Pi = Q^\top \phi(A)$, and W is the best rank-k subspace for its embedding ΠT. By property of T (Theorem 7 in [20]), it is also a good approximation for Π. So

$$T1 \approx \left\|[Q^\top \phi(A)]_k - Q^\top \phi(A)\right\|^2 = \left\|Q[Q^\top \phi(A)]_k - QQ^\top \phi(A)\right\|^2.$$

Algorithm 4 Distributed Kernel PCA: $L = \mathbf{disKPCA}(\{A^i\}_{i=1}^s, k, \epsilon, \Delta)$

1: Each worker i: do a $(1/4, \Delta)$-good subspace embedding $E^i = S(\phi(A^i)) \in \mathbb{R}^{t \times n_i}, t = O(k)$;
2: Compute the leverage scores: $\{\tilde{\ell}_j^i\} = \mathbf{disLS}(\{E^i\}_{i=1}^s, k)$;
3: Sample points: $Y = \mathbf{RepSample}(\{A^i\}_{i=1}^s, \{\tilde{\ell}_j^i\}, k, \epsilon)$;
4: Output $L = \mathbf{disLR}(\{A^i\}_{i=1}^s, Y, k, \epsilon, \Delta)$.

Table 1: Dataset specification: d is the original feature dimension, n is the number of data points, and s is the total number of workers storing the dataset distributedly. Among them, bow and 20news are sparse datasets. All datasets except mnist8m are taken from UCI repository [4] and [7].

Dataset	d	n	s
bow	100,000	8,000,000	200
higgs	28	11,000,000	200
mnist8m	784	8,000,000	100
susy	18	5,000,000	100
yearpredmsd	90	463,715	10
ctslice	384	53,500	10
20news	61,118	11,269	5
protein	9	41,157	5
har	561	10,299	5
insurance	85	9,822	5

Combining this with $T2$, and applying Pythagorean Theorem again, we know that the error is roughly

$$\left\|Q[Q^\top \phi(A)]_k - \phi(A)\right\|^2.$$

Now, by assumption, there is a rank-k $(1+\epsilon, \Delta)$-approximation subspace X in the span of $\phi(Y)$. Since $[Q^\top \phi(A)]_k$ is the best rank-k approximation to $Q^\top \phi(A)$,

$$\begin{aligned}
&\left\|Q[Q^\top \phi(A)]_k - \phi(A)\right\|^2 \\
=&\left\|Q[Q^\top \phi(A)]_k - QQ^\top \phi(A)\right\|^2 + \left\|QQ^\top \phi(A) - \phi(A)\right\|^2 \\
\leq&\left\|X - QQ^\top \phi(A)\right\|^2 + \left\|QQ^\top \phi(A) - \phi(A)\right\|^2 \\
=&\left\|X - \phi(A)\right\|^2.
\end{aligned}$$

The lemma then follows.

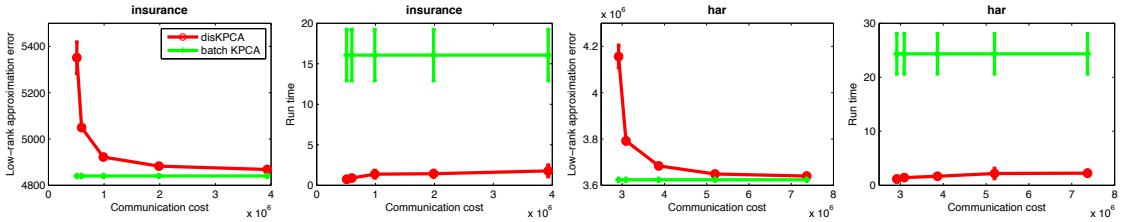
5.5 Overall Algorithm

Now, putting things together, we obtain our final algorithm for distributed kernel PCA (Algorithm 4). Our main result, Theorem 1, follows by combining all the lemmas in the previous subsections (with properly adjusted ϵ and Δ).

6. EXPERIMENTS

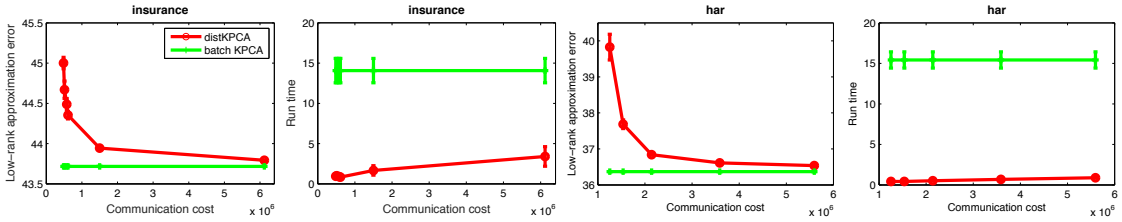
6.1 Datasets

We use ten datasets to evaluate our algorithm. They contain both sparse and dense data and come from a variety of different domains, such as text, images, high energy physics and biology. We use two smaller ones to benchmark against the single-machine batch KPCA algorithm while the rest are large-scale datasets with up to tens of millions of data points and hundreds of thousands dimensions. Refer to Table 1 for detailed specifications.



(a) error on insurance (b) runtime on insurance (c) error on har (d) runtime on har

Figure 2: KPCA for polynomial kernels on small datasets: low-rank approximation error and runtime



(a) error on insurance (b) runtime on insurance (c) error on har (d) runtime on har

Figure 3: KPCA for Gaussian kernels on small datasets: low-rank approximation error and runtime

Each dataset is partitioned on different workers according to the power law distribution with exponent 2 to simulate the distribution of the data over large networks [14]. Depending on the size of the dataset, the number of workers used ranges from 5 to 200 (see Table 1 for details).

6.2 Experiment Settings

Since our key contribution is sampling a small set of data points intelligently, the natural alternative is uniformly sampling. We compare with two variants of uniform sampling algorithms: 1) uniformly sampling representative points and use Algorithm 3 to get KPCA solution (denoted as uniform+disLR); 2) uniformly sampling data points and apply batch KPCA (denoted as uniform+batch KPCA).

For both algorithms, we compare the tradeoff of low rank approximation error and communication cost. Particularly, we compare the communication needed to achieve the same error. Each method is run 5 times and the mean and the standard deviation are reported.

For polynomial kernel, the degree is $q = 4$ and for Gaussian RBF kernel, the kernel bandwidth σ is set to 0.2 of the median pairwise distance among a subset of 20000 randomly chosen data points (a.k.a, the "median trick"). For Gaussian random feature expansion, we use 2000 random features.

In all experiments, we set the number of principle components $k = 10$, which is the same number for k-means. The algorithm specific parameters are set as follows: 1) The subspace embedding dimension for the feature expansion t is 50; 2) The subspace embedding dimension for the data points p is 250; 3) We vary the number of adaptively sampled points $|\tilde{Y}|$ from 50 to 400 to simulate different communication cost; 4) The subspace embedding dimension w is set to equal $|Y|$.

6.3 Comparison with Batch Algorithm

We compare to the "ground-truth" solutions produced by batch KPCA on two small datasets where it is feasible. The experiment results for the polynomial kernel and the Gaussian RBF kernel are presented in Figures 2 and 3, respectively. In both cases, the approximation error of disKPCA

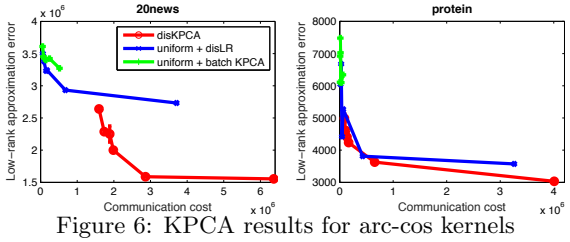


Figure 6: KPCA results for arc-cos kernels

decreases as more communication is allowed. It can nearly match the optimum low-rank approximation error with much fewer data points. In addition, it is much faster: we gain a speed up of 10× by using five workers.

6.4 Communication Efficiency

In these experiments, we compare the tradeoff between communication cost and approximation accuracy on large-scale datasets. The alternative, uniform + batch KPCA, is stopped short in many experiments due to its excessive computation cost for larger number of sampled data points.

Figure 4 demonstrates the performance on polynomial kernels on four large datasets. On all four datasets, our algorithm outperforms the alternatives by significant margins. Especially on bow, which is a sparse dataset, the usage of kernel embeddings takes advantage of the sparsity structure and leads to much smaller error. On other datasets, uniform + disLR cannot match the error achieved by our algorithm even when using much more communication.

Figure 5 shows the performance on Gaussian kernels. On mnist8m, the error for uniform + batch KPCA is so large (almost twice of the errors in the figure) that it is not shown. On other datasets, disKPCA achieves significant smaller error. For example, on higgs dataset, to achieve the same error, uniform + disLR requires more than 5 times communication. Since it does not have the communication of computing leverage scores, this means that it needs to sample much more points to get similar performance. Therefore, our algorithm is very efficient in communication.

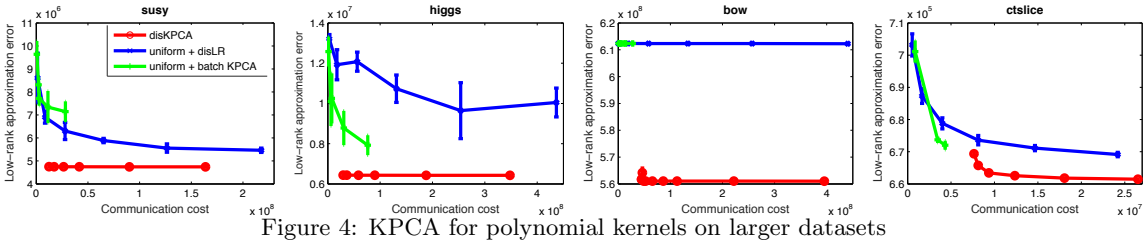


Figure 4: KPCA for polynomial kernels on larger datasets

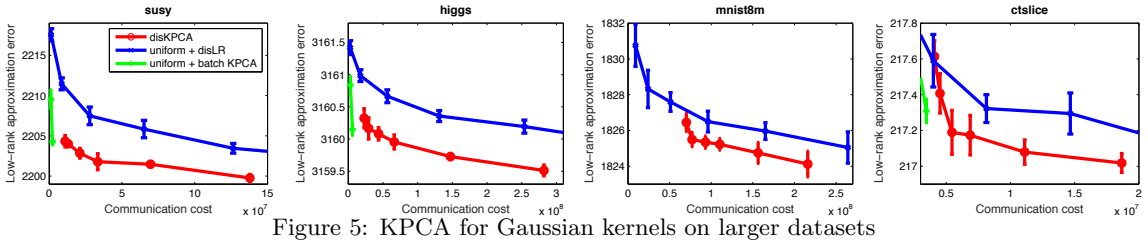


Figure 5: KPCA for Gaussian kernels on larger datasets

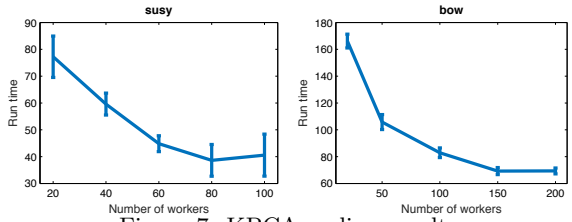


Figure 7: KPCA scaling results

Besides polynomial and Gaussian kernels, we have also conducted experiments using arc-cos kernel [12]. The arc-cosine kernels have random feature bases similar to the Rectified Linear Units (ReLU) used in deep learning. We use degree $n = 2$ and Figure 6 shows the results. Our algorithm consistently achieves better tradeoff between communication and approximation and the benefit is especially more pronounced on sparser dataset such as 20news.

6.5 Scaling Results

In Figure 7, we present the scaling results for disKPCA. In these experiments, we vary the number of workers and record the corresponding computation time (communication time excluded). On both datasets, the runtime decreases as we use more workers, and it eventually plateaus. Our algorithm gains about $2\times$ speedup by using $4\times$ more workers. Note that our algorithm is designed to strike a good balance between communication and approximation. Even though computation complexity is not our first priority, the experiments show disKPCA still enjoys favorable scaling property.

6.6 Distributed Spectral Clustering

We have also experimented a form of spectral clustering (KPCA followed by k-means clustering). We project the data onto the top k principle components and then apply a distributed k-means clustering algorithm [6]. The evaluation criterion is the k-means objective, *i.e.*, average distances to the corresponding centers, in the feature space.

Figure 8(a) presents results for polynomial kernels on the 20news and susy and Figure 8(b) presents results for Gaussian kernels on ctslice and yearpredmsd. Our algorithm compares favorably with the other methods and achieves a better tradeoff of communication and error. This means that although the other methods require similar communication, they need to sample more data points to achieve the same loss, demonstrating the effectiveness of our algorithm.

7. CONCLUSION

This paper proposes a communication efficient distributed algorithm for kernel Principal Component Analysis with theoretical guarantees. It computes a relative-error approximation compared to the best rank-k subspace, using communication that nearly matches that of the state-of-the-art algorithms for distributed linear PCA. This is the first distributed algorithm that can achieve such provable approximation and communication bounds. The experimental results show that it can achieve better performance than the baseline using the same communication budget.

8. ACKNOWLEDGMENTS

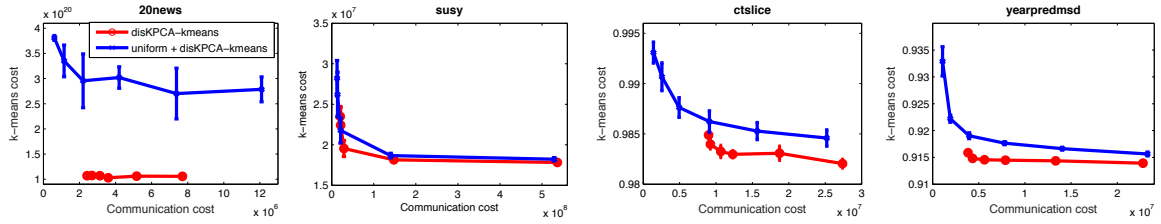
M.-F. B. and Y. L. were supported in part by NSF grants CCF-0953192 and CCF-1101283, ONR N00014-09-1-0751, AFOSR grant FA9550-09-1-0538, a Microsoft Faculty Fellowship, and a Google Research Award. Y. L. was also supported in part by NSF grants CCF-1527371, DMS-1317308, Simons Investigator Award, Simons Collaboration Grant, and ONR-N00014-16-1-2329. L. S. and B. X. were supported in part by NSF/NIH BIGDATA 1R01GM108341, ONR N00014-15-1-2340, NSF IIS-1218749, and NSF CAREER IIS-1350983. D.W. acknowledges the support from XDATA program of the Defense Advanced Research Projects Agency (DARPA), administered through Air Force Research Laboratory contract FA8750-12-C-0323.

9. REFERENCES

[1] N. Ailon and B. Chazelle. The fast johnson-lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on Computing*, 39(1):302–322, 2009.

[2] H. Avron, H. Nguyen, and D. Woodruff. Subspace embeddings for the polynomial kernel. In *Advances in Neural Information Processing Systems*, pages 2258–2266, 2014.

[3] F. Bach and M. Jordan. Predictive low-rank decomposition for kernel methods. In *Proceedings of the International Conference on Machine Learning*, 2005.



(a) Polynomial kernels (b) Gaussian kernels

Figure 8: KPCA + k-means clustering

[4] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[5] M.-F. Balcan, A. Blum, S. Fine, and Y. Mansour. Distributed learning, communication complexity and privacy. *COLT*, 2012.

[6] M.-F. Balcan, V. Kanchanapally, Y. Liang, and D. Woodruff. Improved distributed principal component analysis. In *Advances in Neural Information Processing Systems 27*, pages 3113–3121. Curran Associates, Inc., 2014.

[7] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 2014.

[8] C. Boutsidis, M. Sviridenko, and D. P. Woodruff. Optimal distributed principal component analysis. In *manuscript*, 2015.

[9] C. Boutsidis and D. P. Woodruff. Optimal cur matrix decompositions. *arXiv preprint arXiv:1405.7910*, 2014.

[10] C. Boutsidis and D. P. Woodruff. Communication-optimal distributed principal component analysis in the column-partition model. *CoRR*, abs/1504.06729, 2015.

[11] C. Boutsidis, D. P. Woodruff, and P. Zhong. Communication-optimal distributed principal component analysis in the column-partition model. In *ACM Symposium on Theory of Computing*, 2015.

[12] Y. Cho and L. K. Saul. Kernel methods for deep learning. In *NIPS*, pages 342–350, 2009.

[13] K. L. Clarkson and D. P. Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 2013.

[14] A. Clauset, C. R. Shalizi, and M. E. Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[15] B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M.-F. F. Balcan, and L. Song. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems*, pages 3041–3049, 2014.

[16] A. Deshpande and S. Vempala. Adaptive sampling and fast low-rank matrix approximation. *Algorithms and Techniques in Approximation, Randomization, and Combinatorial Optimization*, pages 292–303, 2006.

[17] I. S. Dhillon, Y. Guan, and B. Kulis. Kernel kmeans, spectral clustering and normalized cuts. In *Conference on Knowledge Discovery and Data Mining*, 2004.

[18] P. Drineas, M. Magdon-Ismail, M. Mahoney, and D. Woodruff. Fast approximation of matrix coherence and statistical leverage. *The Journal of Machine Learning Research*, 13(1):3475–3506, 2012.

[19] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Relative-error cur matrix decompositions. *SIAM Journal on Matrix Analysis and Applications*, 30(2):844–881, 2008.

[20] R. Kannan, S. Vempala, and D. Woodruff. Principal component analysis and higher correlations for distributed data. In *Proceedings of The 27th Conference on Learning Theory*, pages 1040–1057, 2014.

[21] S. Kumar, M. Mohri, and A. Talwalkar. Sampling methods for the nyström method. *Journal of Machine Learning Research*, 13:981–1006, 2012.

[22] X. Meng and M. W. Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the Annual ACM symposium on Symposium on Theory of Computing*, 2013.

[23] J. Nelson and H. L. Nguyên. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *IEEE Annual Symposium on Foundations of Computer Science*, 2013.

[24] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

[25] T. Sarlós. Improved approximation algorithms for large matrices via random projections. In *IEEE Symposium on Foundations of Computer Science*, 2006.

[26] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[27] B. Schölkopf, A. J. Smola, and K.-R. Müller. Kernel principal component analysis. In *Artificial Neural Networks ICANN'97*, volume 1327, pages 583–588, Berlin, 1997.

[28] B. Schölkopf, K. Tsuda, and J.-P. Vert. *Kernel Methods in Computational Biology*. MIT Press, Cambridge, MA, 2004.

[29] D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Theoretical Computer Science*, 10(1-2):1–157, 2014.

[30] Y. Zhang, M. J. Wainwright, and J. C. Duchi. Communication-efficient algorithms for statistical optimization. In *Advance in Neural Information Processing Systems*, 2012.